ASCILITE 2023

People, Partnerships and Pedagogies

AI-powered peer review process: An approach to enhance computer science students' engagement with code review in industry-based subjects

Eduardo Araujo Oliveira, Shannon Rios and Zhuoxuan Jiang
The University of Melbourne

Code review is a common type of peer review in Computer Science (CS) education. It's a peer review process that involves CS students other than the original author examining source code and is widely acknowledged as an effective method for reducing software errors and enhancing the overall quality of software projects. While code review is an essential skill for CS students, they often feel uncomfortable to share their work or to provide feedback to peers due to concerns related to coding experience, validity, reliability, bias, and fairness. An automated code review process could offer students the potential to access timely, consistent, and independent feedback about their coding artifacts. We investigated the use of generative Artificial Intelligence (genAI) to automate a peer review process to enhance CS students' engagement with code review in an industry-based subject in the School of Computing and Information System, University of Melbourne. Moreover, we evaluated the effectiveness of genAI at performing checklist-based assessments of code. A total of 80 CS students performed over 36 reviews in two different weeks. We found our genAI-powered reviewing process significantly increased students' engagement in code review and, could also identify a larger number of code issues in short times, leading to more fixes. These results suggest that our approach could be successfully used in code reviews, potentially helping to address issues related to peer review in higher education settings.

Keywords: peer review, code review, automated review, feedback, self-efficacy

Introduction

Peer review of student's work has long been established as an effective teaching technique (Kottke, 1988) with a wide range of benefits and uses, although it is primarily thought of as an alternative mechanism for providing formative feedback (Liu & Carless, 2006). In the context of this study, peer review is defined as a process whereby a student conducts a structured critical analysis of another student's work and then provides that analysis to the other student, either synchronously or asynchronously. This may be done one-on-one, in small groups or be completely open; it also may be anonymous or personal. There are many approaches to implementing peer review in the classroom (Pearce et al, 2009), some of these include: having students share work in a forum for discussion, asking students to provide critical feedback on a presentation, demonstration, or piece of writing, or by grouping students to reciprocally review and discuss each other's work.

Peer review has the potential to help students in the development of self-efficacy towards 21st century skills as they apply knowledge rather than simply recalling concepts (Novakovich, 2016). By participating in peer review activities, students have a chance to engage and apply critical thinking, problem-solving and decision-making skills to their tasks. Additionally, as students document and provide constructive feedback to each other on reviewed artifacts, they develop social and communication skills, enhance analytical and evaluative abilities, and foster collaboration on course-related content (Boud and Falchikov, 2008). Harland and colleagues (2017) observed that when students are presented with feedback from multiple sources (i.e., other students, teachers, and/or industry partners), they often make use of "whichever comment made sense to them". One significant challenge of using peer review is getting students to see the value in the process, and to participate honestly and effectively (Reddy et al., 2021).

Code review is a common type of peer review in Computer Science (CS) education. It's a process in which developers other than the original author examine source code and is widely acknowledged as an effective method for reducing software errors and enhancing the overall quality of software projects (Bacchelli and Bird, 2013). In educational contexts, CS students review each other's work to provide constructive feedback about source codes. Providing and receiving feedback during code review promotes self-reflection among students and drives improvement in the quality of their work (Pearce et a., 2009). While code review is an essential skill for

CS students, students and educators face challenges that can hinder the effectiveness and overall outcomes of this process (Indriasari et al., 2020): (i) students may lack technical knowledge or skills to participate in the reviews, making it difficult for them to perform thorough and accurate code reviews; (ii) students may not fully engage in reviewing processes, which can negatively impact the outcomes of it; (iii) students may perform poor review quality, which refers to cases where the students' assessments and feedback are inconsistent or unreliable, undermining the credibility of the review process and limiting its effectiveness in improving code quality. Addressing some of these issues could enhance the effectiveness and outcomes of peer code reviews, enabling students to benefit from this collaborative and instructive process.

In this context, we investigated potential use of generative Artificial Intelligence (genAI), which is a sub-branch of AI that uses machine learning trained models to create various types of media contents, to automate a peer review process to enhance CS students' engagement with code review in industry-based subjects. Moreover, we evaluated the effectiveness of genAI at performing checklist-based assessments of code. The goal of our automated code review process is not to replace developers during code reviews but to support them identifying code quality issues. Code reviews are particularly relevant for CS capstone courses with real industry partners where software solutions become available to stakeholders beyond subject boundaries.

Our study designed, compared, and evaluated peer-to-peer and genAI code review processes from 80 CS students involved in a capstone subject. We found our genAI-powered reviewing process significantly increased students' engagement in code review. As an outcome, more code issues were identified, and addressed in the subject.

Background literature

Peer code review in higher education

When undertaking code review developers work together, synchronously, to carefully examine and evaluate proposed changes before incorporating them into the project's code repository (Wurzel et al., 2023). The code review process is simple, but time consuming. In a typical example, the original author of a software code invites a few team members (reviewers) to review artifacts or documents prior to a synchronous meeting. Reviewers inspect coding artifacts and document their observations considering criteria like: (i) are there semantic errors in the code? (ii) is the code well-documented? (iii) does the code conform to existing coding styles? (iv) do existing unit tests need to be rewritten to account for the changes in this code? Additionally, reviewers identify the severity of identified issues in the inspected code. Once all reviewers complete their code review and document their feedback to the original author of the software code, everyone meets, in real time, and discusses all identified issues. As an outcome, the original author of the code will have clarity on what artifacts must be reworked and team members share knowledge and reflect on the process together.

Although there are a lot of positive traits associated with code review practices in the software development lifecycle, the development of software with high code quality in university environments is a challenge for educators (Krusche et al., 2016). Attempts to better promote code quality among software engineering students may require curricula redesign and new, more authentic assessments. Academic fabricated projects have a clear scope, problem, and expected outcome. They are especially popular among programming courses because they can be scaled to large classrooms and graded with a limited teaching staff (Combefis, 2022). However, as fabricated projects are not expected to become available to broader audiences nor are fully aligned with real-world problems, they don't always provide an opportunity for software engineering students to demonstrate their true capacity to integrate and apply their knowledge and skills in the design of software solutions, resulting in lower learning experiences (Krusche et al., 2016). One approach to mitigate this challenge is with the use of industry based capstone projects.

Code review in capstone projects

Computer science, software engineering and other IT-related courses are industry-oriented disciplines. Their curricula need to prepare students to be industry-ready, which means teaching should promote the development of authentic technical and professional skills in classroom environments (Chen et al., 2009). To address this issue, several universities have designed capstone courses where all the knowledge acquired during the students' learning journey is applied in practice.

Moore and Potts (1994), one of the first authors to report initiatives involving collaborations between students and real industry partners in software engineering courses, designed The Real-World Lab experience to emulate industrial experiences among students. Projects involved collaboration with industry partners who acted as clients in those projects. They provided students with project direction and resources. Students were able to negotiate and change project scope, development methods, and techniques during project execution time. Johns-Boast and Flint (2013), Khakurel and Porras (2020) and many others reported on their initiatives and positive observed results in promoting technical and professional skills among students through the adoption of capstone project with real industry partners.

Although there are a lot of positive traits associated with teaching code review in capstone subjects, previous studies showed students feel uncomfortable to share their work or to provide feedback to peers due to concerns related to coding experience, validity, reliability, bias, and fairness (Mulder et al, 2014; Indriasari et al., 2020). In this sense, an automated code review process could offer students the potential to access timely, consistent, scalable, evidence-based, and independent feedback about their coding artifacts. By (partially) delegating code review responsibilities to AI-based tools, students can review and reflect on their coding practices without feeling exposed. Several previous studies investigated the use of automated code review in software engineering (Indriasari et al., 2020; Kaufmann et al., 2022; Farah et al., 2022), however, to the best of our knowledge, no previous work has investigated the use of generative AI combined with assessment checklists in educational code review processes.

Current study

The current study aimed to understand the use of generative Artificial Intelligence to automate a peer review process to enhance CS students' engagement with code review in industry-based subjects. We investigated 2 research questions related to this aim: RQ1) Does a genAI-powered peer review process improve CS students' engagement and participation in code review?; RQ2) How effective was genAI at performing a checklist-based assessment of code?

Methods

An experimental research design was used in our investigation. Our method included research design, data collection and data analysis. These steps are presented in Figure 1.

Research design

The Master of Information Technology (MIT) hosted by the University of Melbourne is designed for those interested in a career in technical IT. The course is organised in specialisations in key areas of Information Technology: (i) Computing; (ii) Distributed Computing; (iii) Human-Computer Interaction; (iv) Artificial Intelligence; and (v) Cyber Security. In the second and final year of the course, students from all streams (or specialisation areas) can choose to complete a software project as a capstone subject.

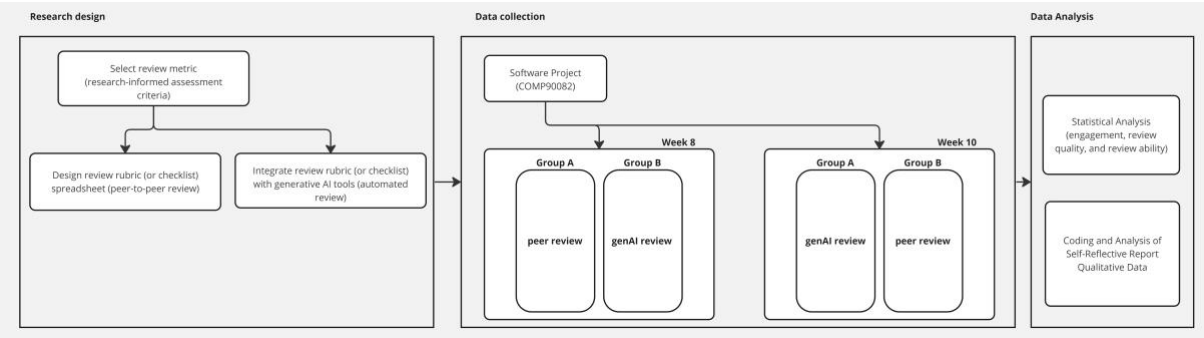

Figure 1. Research Procedure

The software project subject gives students in the MIT experience in analysing, designing, implementing, managing, and delivering a software project related to their stream of IT specialty. The aim of the subject is to guide students in becoming an independent member working within a team over the major phases of IT

development, giving hands-on practical opportunity to apply the topics seen throughout their degree. The subject also gives students a concrete understanding of teamwork processes and tools that underpin the practical aspects of developing software. Students work in teams of five to conceive, analyse, design, implement, test, and maintain a software product for a real-world industry partner. In this subject, students use GitHub, a cloud-based source code repository, to store, track and collaborate on software projects. Additionally, quality assurance processes that include code reviews are taught and assessed in the subject.

Checklist-based code review is one popular approach that can be an effective way to assist inexperienced students to learn code reviews since this process guides students to focus on well-known coding issues or defects (Dunsmore et al., 2003). Checklists can serve as control mechanisms to mitigate the variability in review processes. Without it, peer review could be influenced by individual biases, knowledge gaps, or inconsistencies in approach. In this sense, we adopted checklists to support the design of an automated code review process that is consistent and explainable. Our code review checklist is adapted from Mäntylä & Lassenius (2008), and it provides a comprehensive qualitative classification of code review (Figure 2a). Evolvability defects were classified in three main categories: (i) documentation defects, (ii) visual representation defects, and (iii) structure defects. Functional defects were classified in five groups: (i) new functionality, (ii) resource defects, (iii) check defects, (iv) interface defects and, (v) logic defects.

After designing the code review criteria list, we created a spreadsheet with selected criteria list to support students documenting peer-to-peer review and engineered a prompt (Figure 2b) to automate the process using the OpenAI GPT Large Language Model (LLM) which is a genAI tool. The reference to genAI in our work means the use of OpenAI GPT model. In the peer-to-peer review, students must complete the spreadsheet with their notes (or reviews). To maximise the efficiency of our suggested automated code review process, we integrated our designed prompt with GitHub Actions so students would be able to perform code reviews whenever they decided to, without changing their existing coding process (or having the need to learn new technologies). This process is detailed in Figure 3. The GitHub Actions, which are scripts that support the design and creation of automated workflows on GitHub, was developed as part of this project and utilise OpenAI's v2 GPT API (gpt3.5-turbo).

Students were not asked to perform any additional task in our suggested genAI-powered code review other than existing ones: (i) write source code; (ii) commit and push source code to GitHub repository (submit code to cloud); (iii) approve pull request. Pull requests let software engineers tell others about changes in the source code they've pushed (submitted) to a repository on GitHub. Once a pull request is opened, students within the same team can discuss and review the potential changes and add follow-up commits before submitted changes are merged with other codes. In this subject, once students approved pull requests (or integrations between new code with existing ones), we initiated the code review process with generative AI.

Category	Subcategory	Description
Documentation Defects	Naming	Defects on the software element names
	Comment	Defects on a code comment
Visual Representation Defects	Bracket Usage	Defects on incorrect or missing necessary brackets
	Indentation	Defects on incorrect indentation
	Long Line	Defects on a long code statement which cause the readability
Structure Defects	Dead Code	Defects on code statements that do not serve any meaningful purpose
	Duplication	Defects on duplicate code statements
New Functionality	Use Standard Method	Defects on a piece of source code that does not have a single purpose. Defects whether the standardized approach should be used.
Resource Defects	Variable Initialization	Defects on the variables that are uninitialized or incorrectly initialized.
	Memory Management	Defects on how program use and manage the memory.
Check Defects	Check User Input	Defects on the validity of user input.
Interface Defects	Paramenter	Defects on incorrect or missing parameters when calling another function or library.
Logic Defects	Compute	Defects on incorrect logic when the system runs.
	Performance	Defects on the efficiency of the algorithm.

(a)

```
Please evaluate the {code} below:

Use the following checklist to guide your analysis:
  1. Documentation Defects:
     a. Naming: Assess the quality of software element names.
     b. Comment: Analyze the quality and accuracy of code comments.
  2. Visual Representation Defects:
     a. Bracket Usage: Identify any issues with incorrect or missing brackets.
     b. Indentation: Check for incorrect indentation that affects readability.
     c. Long Line: Point out any long code statements that hinder readability.
  3. Structure Defects:
     a. Dead Code: Find any code statements that serve no meaningful purpose.
     b. Duplication: Identify duplicate code statements that can be refactored.
  4. New Functionality:
     a. Use Standard Method: Determine if a standardized approach should be used
     for single-purpose code statements.
  5. Resource Defects:
     a. Variable Initialization: Identify variables that are uninitialized or
     incorrectly initialized.
     b. Memory Management: Evaluate the program's memory usage and management.
  6. Check Defects:
     a. Check User Input: Analyze the validity of user input and its handling.
  7. Interface Defects:
     a. Parameter: Detect incorrect or missing parameters when calling functions
     or libraries.
  8. Logic Defects:
     a. Compute: Identify incorrect logic during system execution.
     b. Performance: Evaluate the efficiency of the algorithm used.

Provide your feedback in a numbered list for each category. At the end of your answer,
summarize the recommended changes to improve the quality of the code provided.
```

(b)

Figure 2. Code Review Criteria List (a) (adapted from Mäntylä & Lassenius (2008)) and prompt created to automate code review process with generative AI (b)

Participants

Participants were recruited via Canvas Learning Management System (LMS) and provided informed consent (Ethics approval #24272). The sample included a total of 80 (out of 170) students enrolled in a Software Project

subject, with 56 males and 24 females. All participant students worked in teams of five students (16 groups decided to participate in this experiment). Students who did not participate in the study had the same experience, but their data was not collected or analysed as part of this study.

To motivate students to engage in code review processes, in week 6 the subject coordinator delivered a standard lecture on the importance of software quality and the role of code review in software development. Two practical demonstrations of a code review process involving the use of generative AI (gpt3.5-turbo, in our study) and Github Actions, and peer group review were conducted. Lastly, the assessment checklist for development sprints 2 and 3 was explained to students and updated to include 2 points (out of possible 15 for each development sprint) for quality of conducted code reviews and reflections on them.

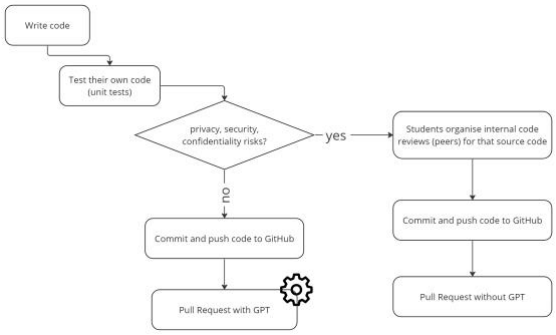

Figure 3. Code review process adopted in our subject.

Data collection

Participants were randomly assigned to two groups: Group A were asked to conduct a peer code review in week 8 and our an automated genAI-powered code review in week 10; Group B were asked to conduct our an automated genAI-powered code review in week 8 and a peer code review in week 10. Students asked to perform peer code review made use of the provided code review checklist to inspect their codes and documented the code review results on spreadsheet provided on the Learning Management System (LMS). Students using the automated process didn't have to fill in the spreadsheet as their code review results were documented and available on their GitHub repositories. At the end of the teaching semester (week 12), students were asked to submit a 400-words individual self-reflection report organised in two subsections: (i) Personal reflections on professional skills development (not included in this paper); (ii) Personal reflections on technical skills development through the use of GitHub Actions and GPT (our suggested development workflow): did you receive insightful/meaningful feedback from genAI? To what extent do you believe genAI helped you improve your coding skills? This report was worth 8 points (out of possible 20 for individual assessment component).

Data analysis

The data was analysed using an iterative analysis method, which moves from a general high-level analysis to more specific ones. In the current study, we evaluated the following interactions relevant to the research questions: overall number of code reviews performed, overall number and types of code defects identified across different code reviews' approaches, and severity of code defects identified across different approaches. A series of plots were created to examine the distribution of code reviews and how students interacted with different approaches across the two weeks of data collection for the research questions, which revealed patterns of engagement and effectiveness.

Results

The results are presented in three sections: overall number of code reviews performed, overall number and types of code defects identified and fixed across different code reviews' approaches, and severity of code defects identified across different approaches. We move from high-level analysis to more detailed ones in the subsections. Additionally, some reflections by students on the use of ChatGPT, which is beyond the prescribed code review methodology, are also provided.

Overall number of code reviews performed

Overall, there were 36 code reviews performed by 16 groups in the subject. During week 8 there were a total of 19 code reviews performed. 13 groups did that using genAI, 6 groups performed peer reviews. In week 10, there were 17 code reviews conducted of which 12 groups performed code reviews using genAI while 5 conducted peer reviews. The use of genAI for code review was significantly larger in the subject with 70% of all reviews conducted using genAI. We also noted that a few teams performed multiple code reviews using genAI, despite only being required to do a single review, while others continued to utilise ChatGPT outside of the GitHub action to better understand their code.

Based on student reflections, we believe students' felt encouraged to engage in code review activities as our genAI-powered code review process was easy-to-use and provided timely feedback:

> …Firstly, ChatGPT frees up our hands and gives us more time for other tasks. Our team conducted a code review using ChatGPT in sprint 2 and a peer review in sprint 3. This deeply impressed me with the power of AI, which can provide good answers in a very short time. We spent an afternoon in the peer review, and ChatGPT was completed in just a few seconds, which is incredible. It is also a patient mentor, its feedback is very gentle and inspiring, giving high recognition and appreciation to our code, which also makes our team feel very happy.

Overall number and types of code defects identified and fixed across different code reviews' approaches

After looking at overall number of code reviews performed during our study, we focused on types of code defect identified by students. GenAI-powered reviews were able to identify a significantly higher number of issues (Figure 4) with an average of 26.713 (SD=27.412) issues identified compared to 9.512 (SD=3.013) in peer reviews. A significant difference between groups was observed in our t-test (p-value=0.01). This difference suggests that genAI is able to provide a more thorough and detailed review of student's code, finding issues that human reviewers may overlook. It also enabled students to promptly identify, understand, discuss, and address code issues, fostering a more interactive and engaging learning process:

> …ChatGPT was instrumental in improving my coding skills. The AI was quick to provide feedback, suggestions, and alternatives to the code we wrote. ChatGPT is able to understand our code and explain the issues in our code. This not only helped me correct my mistakes, but also deepened my understanding of coding principles involved…

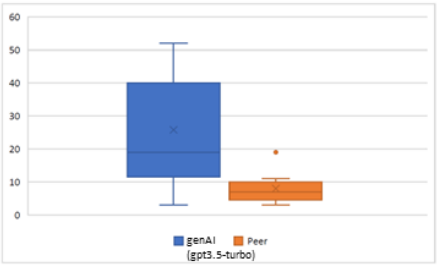

Figure 4. Total number of issues identified

Six out of the thirteen teams (week 8) reported going beyond the recommendations of the genAI review and implementing further enhancements to their code. Students commented in their self-reflection reports about the advice received from genAI on code defects identified, and evaluations of the accuracy and value of reviews provided by genAI:

> …During the code review process, I found ChatGPT's feedback somewhat beneficial. One notable instance was when it highlighted the security concerns about the JWT algorithm we were using. This feedback prompted us to research and implement a more secure algorithm for our website, effectively enhancing its security…

Figure 5 shows the average number of issues identified by both approaches, categorised by the type of issue

identified (consistent with Figure 2a). This classification was extracted from data provided by students and from responses generated by genAI. GenAI was able to identify more issues related to documentation, visual representation, and checks, while peer review was able to identify more structural and logic issues. Resource and interface issues were minimal for both methods.

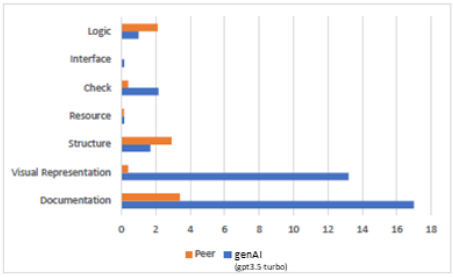

Figure 5. Mean number of issues identified in each code review approach.

One possible explanation for observed differences in logical and structural issues is due to the deeper understanding of the project context that human reviewers typically have, as previously discussed. Participants in the project team have a better understanding of the project's objectives, requirements, and overall context. This knowledge allows them to make better judgements whether or not new coding functionalities align with the project's direction. Moreover, code review often involves subjective judgments such as decisions about overall design. These judgments can vary based on the specific and contextual needs of a project. Peer reviewers, with a comprehensive understanding of the project, are usually better equipped to make these judgments.

Additionally, human reviewers are better at evaluating relationships between code structure and software performance. Software performance often depends on the efficiency of IT frameworks and can't be assessed by looking at code complexity in isolation. An AI tool based on genAI might not be able to fully assess the efficiency of an algorithm or the relevance of a code structure without broader project context. Peer reviews can provide a more holistic evaluation of code by using their understanding of project requirements, specific use cases, and the trade-offs between different algorithms and project technologies in use. Another way to categorise the issues identified is to consider the accuracy of the review. After extracting information from the student data, we could see genAI outperformed peer review in terms of the total number of issues rectified by authors, registering a mean of 9.165 (SD=14) by genAI compared to a mean of 5.667 (SD 2.16) in peer reviews. This data suggests that while human reviewers demonstrate greater expertise in spotting complex issues, genAI's swift and immediate feedback facilitates the faster rectification of a greater number of issues. Additionally, simpler issues are quicker to be addressed. This is further corroborated by the variance in the number of issues identified according to severity highlighted in the next section.

The large standard deviation in the number of issues fixed for genAI-powered reviews is mirrored in reflections from several students who found that feedback received from genAI was very generic:

> …I found the Peer-to-Peer code review to be more helpful than the ChatGPT code review. While we may be less knowledgeable in analyzing code compared to ChatGPT, I felt that I learnt more from discussing coding decisions as a team. When reviewing another member's code, I could listen to why my team member made a specific decision and provide my suggestions. I felt that the feedback received by ChatGPT was also very generic, such as adding more descriptive comments or optimising the efficiency if necessary. While this can be partially attributed to the code not having any logic defects for ChatGPT to comment on, I think it also struggles with providing specific feedback with limited context…

This issue is partially caused by the fact that students were unable to provide the entire project source code to genAI, which meant that genAI couldn't have enough context of the project to properly evaluate specific parts of the software solution separately and independently. Additionally, the rubric based prompt using Github Actions did not allow for students to provide additional context, however several students went beyond the prescribed use of genAI and used ChatGPT to further debug and understand their code in a more conversational way:

> …The use of ChatGPT also provided impressive feedback on coding practices. It offered not only suggestions for code improvements, debugging, and optimizations but also made our codes more compatible. The ability to transfer Python to Lua from Farmbot OS to Raspberry was brilliant and impressive. The help from ChatGPT provided perfect solutions for our problems and saved months of time in coding and debugging. The AI's ability to explanations and suggestions also helped me learn technical skills faster in web app development and server construction…

Despite this limitation, students observed ChatGPT was great for troubleshooting warning messages. In most cases, ChatGPT could help resolve errors when provided with a block of code and a warning message. ChatGPT was also useful in providing multiple code examples like tutorials or documentation summary related to specific inspected source code (maximising abstraction and minimising complexity to students).

Severity of code defects identified across different approaches

GenAI-powered reviews were able to identify a larger number of trivial issues but didn't perform as well in identifying critical ones (Figure 6). These results are consistent with previous findings that suggest peers have a more complete understanding of the project and its involved technology, while genAI performed reviews of parts of the whole, without context.

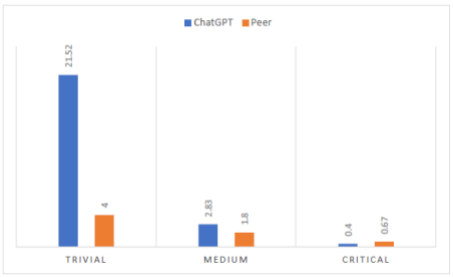

Figure 6. Mean number of issues identified by severity.

Similar to the previous categorisations of issues, the severity of the issue identified was evaluated by students and the researchers did not verify the students' assessment.

Other uses of ChatGPT beyond the rubric based review

Many students found the use of genAI review to be a transformative experience and continued to use ChatGPT independently to help them understand code and improve their skills. One of the uses by these students was to help understand complicated functions, especially those from pre-existing code:

> During the semester I was assigned with working on a pre-existing file that had legacy code. I wasn't exactly sure what a function within the source code was doing so I asked GPT, which thoroughly explained the code to me and suggested ways to document it so other members could also understand that function.

This last comment highlights student's tendency to trust ChatGPT's interpretation of code.

> I found great utility in ChatGPT when it came to deciphering code written by previous teams in the handover code. At times, the code was messy and used packages I was unfamiliar with. ChatGPT was able to identify these packages and when prompted, provide me with a concise explanation of what it does and how to use it. The way that ChatGPT was able to understand snippets of code easily was astounding and useful.

Discussion

In this paper we investigated how the use of an automated peer review process could enhance CS students' engagement with code review in industry-based subjects. The research questions allowed us to examine students' level of interaction with different code review approaches (RQ1), and the effectiveness of a genAI

performing a checklist-assessment of code (RQ2). We used a combination of statistical analysis and self-reported data to examine the research questions.

For the first research question, we identified meaningful patterns for engagement and participation in code review across both approaches. We found that genAI stimulated student engagement in the code review process, demonstrating a higher average number of identified issues compared to the peer review process and a higher average number of those issues identified by genAI were fixed. This aligns with results reported in previous studies that used automated peer feedback (Farah et al., 2022). Interestingly, inaccurate feedback and misunderstandings of project requirements encouraged students to revisit the inspected code and stimulated further critical discussion. GenAI feedback created a dynamic learning environment that encouraged students to understand, discuss and make decisions on how to address issues, fostering a more interactive learning process.

Our analysis also showed that students continued to engage with genAI beyond the code review process by asking ChatGPT to explain unclear code written by previous teams to them and by feeding in error and warning messages to ChatGPT. The use of genAI in code reviews has the potential to minimise challenges in peer review identified in previous studies that showed students feel uncomfortable to share their work or to provide feedback to peers (Mulder et al., 2014; Indriasari et al., 2020). By using genAI, students felt less exposed and more supported to engage in reviewing processes. This could be a useful tool to initially focus on students' technical code reviewing skills, and then adding layers related to teamwork skills later (e.g., how to provide and receive feedback).

In answering the second research question, we found that genAI-powered code reviews can identify a larger number of code issues in a very short time, leading to more fixes, but human reviewers demonstrate superior proficiency in identifying logical and structural issues. This is most likely the result of the human reviewer having a much broader understanding of the code, where the context of the project and the understanding of its specific requirements play a significant role in code review (Henderson, 2020). This suggests peer review and automated code review could complement each other well, as indicated in previous research using chatbots to support peer review (Farah et al., 2022). There is also potential to address this issue with the ability of GPT-4 to handle much larger input data so that the student can provide additional context to the system.

This study has two main limitations. First, genAI can only review individuals' own submitted codes. This process lacks the collaborative learning potential inherent in peer code reviews. Peer code reviews provide an opportunity for team members to understand sections of the code they did not directly work on. To address this limitation, next semester we plan to update our code review process to suggest students to review received feedback from genAI together. Second, we acknowledge that genAI-powered feedback is limited and can be generic or inaccurate. The value of genAI-powered review is not purely in the feedback it provides to students but more significantly in its capacity to prompt students to discuss and critically re-evaluate their code. Instead of merely accepting the feedback from genAI, students will be challenged to validate received feedback, promoting a more thorough and in-depth engagement with their code and the reviewing process. Despite its limitations, the use of genAI in code review can still serve as a valuable tool to drive student engagement. However, students must continue to lead and have control over this process and related project decisions.

Conclusion

This study provides a better understanding of the importance, capability, and meaningfulness of a genAI-powered peer review process in improving CS students' engagement and participation in code review. We found that the genAI review stimulated student engagement in the code review process, demonstrated by a higher number of genAI powered reviews being conducted and a higher average number of identified issues compared to the peer review process. Automated code reviews could identify a larger number of code issues in short time, leading to more fixes, but peer reviewers demonstrated superior proficiency in identifying logic and structure issues. Students had a better understanding of project context and involved technologies and can inspect code broadly.

This research contributes to the field of educational technology research and software engineering with the use of an approach that combines statistical and self-reported data to provide a meaningful interpretation of code reviews' results. This research has implications for both practice and research related to using generative AI for the automation of peer reviews based on the use of checklists, rubrics, and other assessment instruments.

References

Bacchelli, A., & Bird, C. (2013, May). Expectations, outcomes, and challenges of modern code review. In 2013 35th International Conference on Software Engineering (ICSE) (pp. 712-721). IEEE. https://doi.org/10.1109/ICSE.2013.6606617

Boud, D. and Falchikov, N. (2008). Rethinking assessment in higher education: Learning for the longer term. Routledge, London. https://doi.org/10.4324/9780203964309

Chen, J., Lu, H., An, L., & Zhou, Y. (2009, July). Exploring teaching methods in software engineering education. In 2009 4th International Conference on Computer Science & Education (pp. 1733-1738). IEEE. https://doi.org/10.1109/ICCSE.2009.5228269

Dunsmore, A., Roper, M., and Wood, M. (2003). The development and evaluation of three diverse techniques for object-oriented code inspection. IEEE Transactions on Software Engineering, 29(8):677–686. https://doi.org/10.1109/TSE.2003.1223643

Farah, J. C., Spaenlehauer, B., Sharma, V., Rodriguez-Triana, M. J., Ingram, S., and Gillet, D. (2022). Impersonating chatbots in a code review exercise to teach software engineering best practices. In 2022 IEEE Global Engineering Education Conference (EDUCON). https://doi.org/10.1109/EDUCON52537.2022.9766793

Harland, T., Wald, N., & Randhawa, H. (2017). Student peer review: Enhancing formative feedback with a rebuttal. Assessment & Evaluation in Higher Education, 42(5), 801-811. https://doi.org/10.1080/02602938.2016.1194368

Henderson, F. (2020). Software engineering at google. arXiv.org. Accessed: April 26, 2023.

Indriasari, T. D., Luxton-Reilly, A., and Denny, P. (2020). A review of peer code review in higher education. ACM Transactions on Computing Education, 20(3):1–25. https://doi.org/10.1145/3403935

Johns-Boast, L., & Flint, S. (2013, October). Simulating industry: An innovative software engineering capstone design course. In 2013 IEEE Frontiers in Education Conference (FIE) (pp. 1782-1788). IEEE. https://doi.org/10.1109/FIE.2013.6685145

Kaufmann, C., Pavão, J., & Wahl, H. (2022, June). Is there a Need for Automated Code Review to be Used in Teaching?: From the perspective of students. In 2022 17th Iberian Conference on Information Systems and Technologies (CISTI) (pp. 1-6). IEEE. https://doi.org/10.23919/CISTI54924.2022.9820030

Khakurel, J., & Porras, J. (2020, November). The effect of real-world capstone project in an acquisition of soft skills among software engineering students. In 2020 IEEE 32nd Conference on Software Engineering Education and Training (CSEE&T) (pp. 1-9). IEEE. https://doi.org/10.1109/CSEET49119.2020.9206201

Kottke, J. L. (1988). Students as peer critics of writing in a psychology course. Psychological Reports, 62(1), 337-338. https://doi.org/10.2466/pr0.1988.62.1.337

Krusche, S., Berisha, M., & Bruegge, B. (2016, May). Teaching code review management using branch based workflows. In Proceedings of the 38th International Conference on Software Engineering Companion (pp. 384-393). https://doi.org/10.1145/2889160.2889191

Liu, N. F., & Carless, D. (2006). Peer feedback: the learning element of peer assessment. Teaching in Higher education, 11(3), 279-290. https://doi.org/10.1080/13562510600680582

Luxton-Reilly, A. (2009). A systematic review of tools that support peer assessment. Computer Science Education, 19(4):209–232. https://doi.org/10.1080/08993400903384844

Mäntylä, M. V., & Lassenius, C. (2008). What types of defects are really discovered in code reviews?. IEEE Transactions on Software Engineering, 35(3), 430-448. https://doi.org/10.1109/TSE.2008.71

Moore, M., & Potts, C. (1994). Learning by doing: Goals and experiences of two software engineering project courses. In Software Engineering Education: 7th SEI CSEE Conference San Antonio, Texas, USA, January 5–7, 1994 Proceedings 7 (pp. 151-164). Springer Berlin Heidelberg. https://doi.org/10.1007/BFb0017611

Mulder, R. A., Pearce, J. M., & Baik, C. (2014). Peer review in higher education: Student perceptions before and after participation. Active Learning in Higher Education, 15(2), 157-171. https://doi.org/10.1177/1469787414527391

Novakovich, J. (2016). Fostering critical thinking and reflection through blog-mediated peer feedback. Journal of Computer Assisted Learning, 32(1), 16-30. https://doi.org/10.1111/jcal.12114

Pearce, J., Mulder, R., and Baik, C. (2009). Involving students in peer review: Case studies and practical strategies for university teaching. Centre for the Study of Higher Education, University of Melbourne, Parkville, Vic.

Reddy, K., Harland, T., Wass, R., & Wald, N. (2021). Student peer review as a process of knowledge creation through dialogue. Higher Education Research & Development, 40(4), 825-837. https://doi.org/10.1080/07294360.2020.1781797

Wurzel Gonçalves, P., Calikli, G., Serebrenik, A., & Bacchelli, A. (2023). Competencies for Code Review. Proceedings of the ACM on Human-Computer Interaction, 7(CSCW1), 1-33. https://doi.org/10.1145/3579471

Oliveira, E. A., Rios, S. & Jiang, Z. (2023). AI-powered peer review process: An approach to enhance computer science students' engagement with code review in industry-based subjects. In T. Cochrane, V. Narayan, C. Brown, K. MacCallum, E. Bone, C. Deneen, R. Vanderburg, & B. Hurren (Eds.), *People, partnerships and pedagogies*. Proceedings ASCILITE 2023. Christchurch (pp. 184 - 194). https://doi.org/10.14742/apubs.2023.482