

Master's Thesis Computer Science
University of Twente

**CodeQuizzer: Improving codebase
understanding for code review via a gamified
quiz taking system**

Filip Ivanov

July 2024

Supervisors

Vadim Zaytsev

Arjan van Hessen

Abstract

This research attempts to apply gamification to the code review process. In particular, it applies it to the challenge of gaining a better understanding of the surrounding codebases that changes presented for review belong to. This is done in order to improve the quality and efficiency of code reviews, but more importantly in order to get an answer to the question if gamification can be used to tackle this challenge and if so - how.

To do so, first a background research was conducted involving an examination of available literature, coupled with semi-structured expert interviews. This was done in order to set a foundation for what the code review process is, what it involves, what the expectations and outcomes are and what challenges are still present. Afterwards, a prototype for a gamified quiz taking system named CodeQuizzer was developed and an evaluation survey was conducted to see if and to what extent this prototype proves successful in increasing participants' understanding of a mock-up codebase via gamification.

The results obtained indicate that a majority of participants found CodeQuizzer effective in increasing their understanding of the mock-up codebase. Furthermore, it was established that the gamification elements included were motivational for the majority of participants as well. There were also other relevant findings about the specifics of CodeQuizzer's prototype such as its usability and its specific content, which provide several avenues for future research.

Based on the positive reception of CodeQuizzer, it can be concluded that gamification can indeed be used to tackle the challenge of understanding in code review. Furthermore, the system's effectiveness in this shows its implementation is a valid approach to take, answering the question of how exactly gamification can be used in this regard as well. There were several areas for improvement also identified such as making quizzes feel less like exams and expanding the usage of gamification elements further. These coupled with various feasibility concerns can serve as a good direction for future research built on top of the solid foundation that this paper provides.

Keywords: code review, gamification, codebase understanding, quizzes, learning tools

Acknowledgements

I would like to acknowledge and thank both of my supervisors for this thesis - Vadim and Arjan. They provided guidance and feedback throughout the entire period of this project and helped whenever uncertainties arose. I am in particular grateful for Vadim's technical expertise and vast knowledge of research which made ideation and background research much easier than they otherwise would have been. I am also very thankful to Arjan for his willingness to provide insights and recommendations about the user research done, as this contributed to the quality of essential results for this thesis.

Aside from my two supervisors, I would like to acknowledge every participant who took time out of their day to provide their opinions and thoughts. As the code review process is one that is very human-centric, this data is the backbone for every bit of work done here. In particular, I would like to thank all my colleagues working at El Niño for their support and eagerness to participate and provide pointers for improvement.

Finally, I want to thank my friends and family who provided invaluable moral and mental support throughout the process of this research. Without this I surely would not have achieved a good final result and would certainly have had a much worse work-life balance than I did throughout the duration of the thesis.

Table of contents

CodeQuizzer: Improving codebase understanding for code review via a gamified quiz taking system	1
Abstract	2
Acknowledgements	3
Table of contents	4
Introduction	7
Background research	9
Literature examination on code review	9
The code review process	9
Early formalizations of code review	10
Modern code review	11
Motivations and outcomes of code review	14
Motivations for doing code review	14
Outcomes of doing code review	16
Challenges in code review	17
Understanding the changes presented for review	18
Understanding the change context	19
Additional challenges	21
Expert interviews on code review	22
The code review process	23
Motivations for code review	24
Expected outcomes of code review	25
Challenges	26
Understanding	26
Understanding the change code	26
Understanding the codebase context	27
Understanding the ticket context	28
Additional challenges	29
Feedback	29
Change size and presentation	29
Tool related challenges	30
Process related challenges	30
Conclusions on code review	30
Literature examination on gamification	32

Gamification - a workable definition	32
Applying gamification to the problem of obtaining understanding	33
Which gamification elements to apply?	34
Existing applications of gamification	34
Conclusions on gamification	36
Prototype and evaluation	37
Prototype Design	37
Quizzes	38
Exercises	39
Multiple choice and open question exercises	40
Function flow exercises	40
Component diagram exercises	41
Functionality altering exercises	42
Variable role exercises	43
Gamification Elements	43
Quiz progress bar	44
Quiz completion message	44
Profile badges	45
Comprehension levels	46
Leaderboard	47
UI / UX Design	48
Evaluation Design	48
High Level Approach	49
Survey Design	49
Participant Context	50
Example Quiz	50
Exercise Types	51
Gamification	51
Usability	52
Evaluation results	53
Participant context	53
Example quiz	54
Exercise types	59
Multiple choice exercise	59
Open question exercise	61

Function flow exercise	63
Component diagram exercise	65
Functionality altering exercise	67
Variable role exercise	69
Conclusions on exercise types	70
Gamification	71
Quiz progress bar	71
Quiz completion message	73
Profile badges	74
Comprehension levels	76
Leaderboard	77
Conclusions on gamification	79
Usability	79
Conclusion	81
Discussion	82
Limitations and future research	84
Limitations	84
Future research	85
Appendix	86
Appendix A. Interview questions used during the expert interviews done as part of this paper's preliminary research phase	86
Appendix B. Survey questions presented to participants in CodeQuizzer's evaluation survey.	87
References	91

Introduction

The code review process is one of the most essential for the proper functioning of the software development industry. It has changed significantly since first being formalized and is currently viewed as a lightweight, asynchronous and iterative process. The goals, expectations and specific implementation of this process can vary among different companies, but its importance for ensuring good quality code and well designed solutions is acknowledged and agreed upon nonetheless.

Despite its widespread usage and the different ways it is put into practice code review still comes with a variety of challenges that can be addressed in the interest of increasing review quality and efficiency. One of the biggest such challenges identified by research is the challenge of understanding. This challenge is present in a variety of contexts such as understanding the code presented for review or understanding reviewers' feedback. For the purposes of this thesis, the type of understanding that is the focus is understanding the surrounding context of code presented for review. In particular, the codebase that the specific change is meant for. Hopefully through increasing understanding of it, code reviews can be done more efficiently as contextual knowledge can make them go faster, but also their quality can increase, as such knowledge can be useful for providing less superficial feedback.

Although obtaining this type of understanding could be beneficial in many ways, it is still an issue developers struggle with. The reasons are most likely multifaceted, but the one of interest to this paper is the problem of lacking motivation. Put simply, developers may be able to obtain a deeper understanding of a codebase, but they may simply choose not to because they see it as a waste of time or because there is nothing that motivates them to do so. This paper will introduce a technological intervention to tackle this problem named CodeQuizzer. This solution allows the most knowledgeable members of a software development team (for example tech leads) to set up quizzes about the team's codebases. Through doing these quizzes, developers are hopefully exposed to and made to think about more of the codebase's parts, thus, providing better quality feedback in their code reviews.

The lack of motivation problem in particular will be addressed through the usage of gamification elements in CodeQuizzer. Design elements such as progress bars, badges, levels, positive feedback and leaderboards, as well as the interactive nature of the quizzes themselves will hopefully motivate developers to actually do them. This then would

motivate them to also increase their understanding and lead to more efficient and better quality code reviews in the future.

The particular objectives of this thesis are several. Firstly, a prototype of CodeQuizzer is developed that illustrates its functionality, as well as the concept of using gamification for the purposes described above. Then, an evaluation study is done to test its design and implementation, but more importantly to find an answer to the following research questions:

- *Can gamification be used to motivate developers to gain a better understanding in the context of code review?*
- *How can gamification be used to motivate developers to gain a better understanding in the context of code review?*

If the prototype is well received in regards to boosting understanding and motivation to obtain it, then this shows gamification can in fact be used in this way and for this purpose. It also indicates that the way CodeQuizzer uses it in particular (via gamified quizzes) is a viable way to tackle the challenge. If the opposite holds true, a definitive answer is obtained for the second question (CodeQuizzer's particular approach is most likely not the way to go), but the first question remains more open ended. Nevertheless, the evaluation results still can be used to indicate if such a solution holds some ground for further research that attempts a different implementation.

Naturally, the end goal is to provide answers to both research questions presented. Some of the work done to gain these answers was briefly outlined in this section already, but is described in much greater detail throughout this report. The report begins by presenting the background research, which focuses on providing a workable definition of the code review process, as well as outlining its goals, expected outcomes and most importantly - challenges. Next, the design and implementation of both CodeQuizzer's prototype as well as its evaluation study are described and all decisions made are justified. Afterwards, the results of the study are presented and analyzed, upon which the conclusion of the thesis is presented with an answer to both research questions. Some discussion follows, together with limitations and recommendations for future work. Finally, the appendix and references are outlined as well.

Background research

In this chapter, the background research done for this paper is summarized and its relevant findings are outlined. This research takes the form of a literature review which has three parts - examining available literature on code review, examining available literature on gamification and conducting a set of expert interviews with software developers. The first aims to provide needed context for the code review process, its motivations, expected outcomes and challenges. The second gives some insight into how gamification has been and could be used in a software development context to solve similar challenges. The last aims to provide additional findings that literature may have missed, but also in particular to confirm or refute the identified biggest challenge that developers face while doing code reviews.

Literature examination on code review

As aforementioned, this section provides an overview and synthesis of the findings obtained via examining available literature on code review. The primary goals are to establish:

- What the code review process used to look like and what it looks like currently with the necessary tasks involved in it.
- The different motivations and expected outcomes of the code review process.
- The various challenges that developers face during the code review process.

These are all important bits of context and knowledge that will inform the rest of the work to be done. The particular reasons why they were chosen as the focus and why they are important are provided next, coupled with the findings obtained.

The code review process

In order to introduce a technological intervention into the code review process, it is first necessary to establish what the code review process actually is and what tasks it usually consists of in the modern day. This is to ensure that the intervention does not conflict with what is already there, but also in order to establish a workable definition of the process for the rest of the paper. Moreover, it is important to understand what the process looked like in the past so it can be established how and why it changed. This highlights the contemporary needs of developers in regards to this process which are at the core of the development of CodeQuizzer.

Early formalizations of code review

One of the earliest and most often cited formalizations of the code review process found while examining available literature is the one done by Fagan in 1976 [1]. Thus, this formalization will serve as a groundwork for how the code review process looked like in the past. Fagan formalized the code review process introducing the concept of *Inspections* at various points during the development of a piece of software. He claims that the software inspection process should be “formal, efficient, and economical” [1]. The way these qualities are to be achieved is through strictly defining all the aspects of code review. Inspections are to be done at set points in the process (mainly after specification, design and implementation are completed), the people involved are to be moderators, coders and/or designers and testers, and the process itself is to consist of strictly defined steps, performed in a strictly defined way and order [1]. The steps he proposes are *Overview*, *Preparation*, *Inspection*, *Rework* and *Follow-up*, with a particular focus on the team aspects involved in the *Overview* and *Inspection*, which should be performed together in a scheduled meeting with all the people involved [1]. Additionally, Fagan even provides a formalization of the questions to be asked and answered when inspecting at the different stages mentioned before [1]. Moreover, this process is a cyclical one, meaning that the Follow-up stage can easily lead to a full repeat of the entire process if enough new code has been introduced or if a reinspection is requested [1]. A simplified high level diagram of the process can be seen in Figure 1.

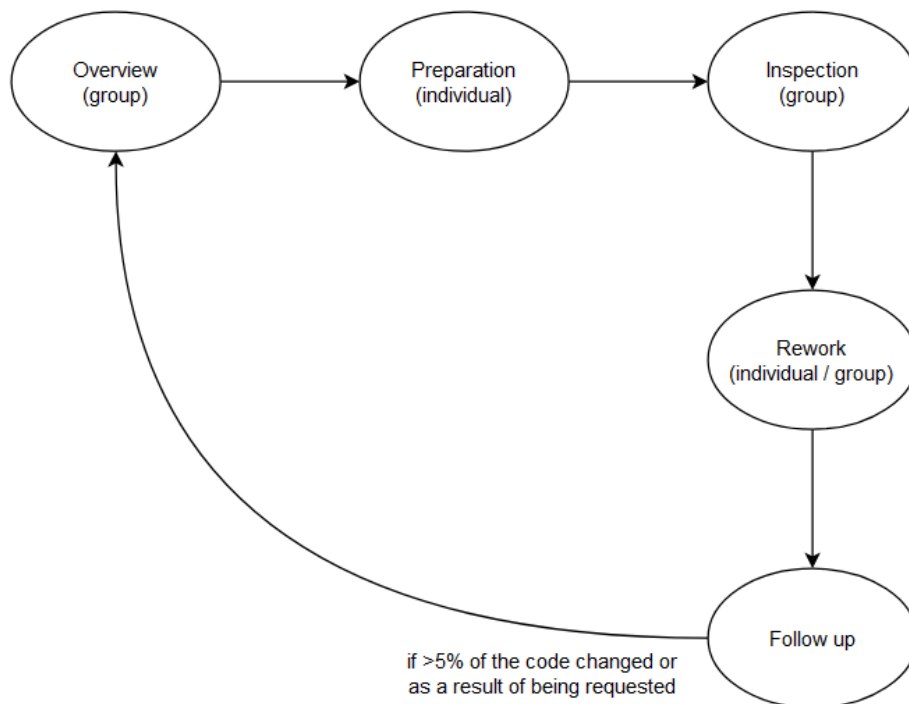


Figure 1. A high-level overview of the first formalization of the code review process (Fagan's inspection process).

Without diving much deeper into Fagan's work, the takeaway relevant for this paper is that code review started as a very formalized, sequential and strictly defined process. This is to ensure that errors are caught at an early enough stage so they do not have to be resolved at later stages, where this can be much more costly both in terms of time and effort [1]. This also shows that in the beginning the primary goal of code review was *error prevention*.

Modern code review

To find out how modern code review compares to this early formalization, some sort of definition must be given for this "modern" process. This paper looks at the one adopted by Microsoft, Google, AMD and other big OSS projects [2][3]. This is because these companies are leaders in the software industry and have been working in it for quite a few years to great success. So, it is only logical they can provide a solid workable definition for modern code review.

This definition stands in stark contrast to what Fagan proposed. It aims to be informal, lightweight, asynchronous and tool-based. Aside from error prevention as with Fagan, a

big emphasis here is on being flexible. No longer are there very strict definitions for the tasks involved in the process, nor their timing. There is still an outline behind how the process should generally be conducted, albeit more loosely defined than in the past. This outline generally consists of three steps, as given below and also in Figure 2.

- Creating a change for review before the new code is pushed to the main project repository.
- Reviewing the change (done by other developers), with alterations, fixes and better implementations being suggested.
- Merging the change into the main repository of the project once feedback is addressed, with the possibility for it to be rejected also being open.

Note: This process is once again cyclical, so after feedback has been addressed, the changes will be reviewed again (and possibly even again) until the code is deemed good enough to be merged in.

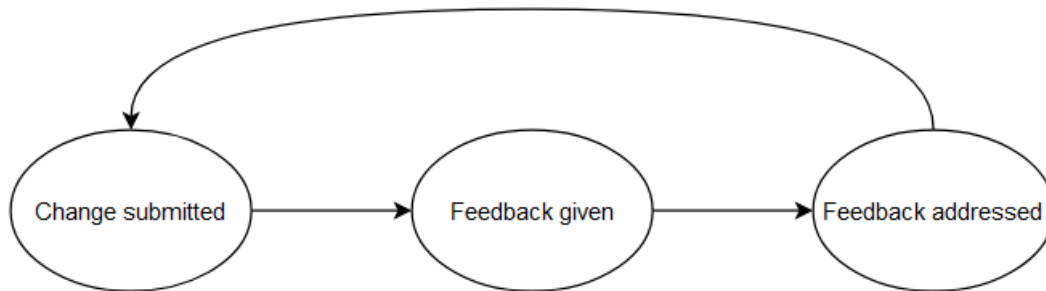


Figure 2. A high-level overview of the modern code review process as it is currently being done at most companies.

As mentioned already, this process is adopted by large software companies like Microsoft, Google, AMD and various big OSS projects [2][3]. However, it is also used by smaller size companies, as outlined by [4]. This study focused on interviewing employees at 19 different companies about how code review is done. The range of companies was relatively broad with the focus being on smaller companies in comparison to the large ones mentioned previously. Through the interviews, it was found that how code review is done on a small and medium scale is very similar to how it is done at a larger scale, with a lot of the literature's findings about such companies also being confirmed here [4]. There are of course several differences as well, but they will be discussed as they become relevant. The main takeaway is that the definition proposed is not exclusive to software

giants, but also applicable on a smaller scale, which makes it more universal and a better starting point for further research.

Regardless of company size, it seems that the idea behind modernizing code review is to have a more lightweight, flexible and less formal process. This process is often change based, meaning that it happens only when new changes are introduced. This is the way it is done at Microsoft, Google and other big companies [2][3], but also how it is done in smaller companies as well [4]. The key difference seems to be that some smaller companies do code reviews much less frequently with it being described as “irregular” [4]. Regardless of frequency though, it seems to be majority change based, rather than requiring explicit management and periodic scheduling. This once again contrasts to Fagan’s formalization.

Furthermore, the modern process described is asynchronous and is conducted via asynchronous channels of communication, such as mailing lists, rather than physical sit-downs to look over code as Fagan proposes [1]. This way of working happens almost all the time at bigger companies [2][3]. The reason for this is probably because in organizations of such size it is difficult to schedule, arrange and manage in-person meetings. This intuition seems to be confirmed as at smaller scale companies, sometimes physical sit downs still happen [4], probably due to the smaller size allowing them to happen easier. Sometimes it will also happen with bigger projects as well, such as at Sony Mobile [5], but generally communication in the context of the modern code review process aims to be asynchronous, regardless of company size.

A side effect of this asynchronicity is that code review can become difficult to track, if all communication is through mailing lists. Therefore, most companies use specialized and sometimes proprietary tools to do code review. This allows them not only to track the process, but also tailor it to the company specific requirements. For example, Microsoft uses CodeFlow [2], Google uses Critique [3] and AMD uses CodeCollaborator [2]. Smaller companies use various tools, with Gerrit, Crucible, Stash, GitHub pull requests, Upsource, Collaborator and ReviewClipse being mentioned [4].

Regardless of the specific tools used, it seems that as software development grew in both size and importance, so did the code review process. Moreover, it also evolved into something rather different than what was outlined by Fagan. The reasons for this change are likely to be multifaceted and complex, but it would be reasonable to assume that as the size and complexity of codebases increased, it became quite time consuming to do a

very formal and line-by-line style of review, which is also synchronous and requiring all parties involved to sit down together at various points of the process. Research also seems to support these reasons for the shift in how code review is done currently [6][7].

Furthermore, looking at Table 1, which highlights the primary differences between what was outlined by Fagan and what we define as modern code review in this paper, it can be seen that the changes all fall in line with the idea of a lightweight, asynchronous and less formal process. Therefore, this outline will also be the one taken as a starting point in further research and the idea of asynchronicity and flexibility will be respected in the design and implementation of CodeQuizzer.

	Traditional code review	Modern code review
Process outline	Formalized, rigid, time consuming	Informal, flexible, lightweight
Process timing	Synchronous, scheduled	Asynchronous, not scheduled
Communication	Direct, in-person, synchronous	Less direct, mostly textual, asynchronous
Tasks	Strictly defined and outlined	Loosely defined and outlined (sometimes not even outlined formally)
Reviewers	A set list of people in a set of roles	Flexible with assignment often left to the change author

Table 1. A comparison between Fagan’s inspection process and the modern code review process in regards to various aspects.

Motivations and outcomes of code review

Having established a workable definition of the code review process, next it is important to identify the motivations behind it and its expected outcomes. This is important for similar reasons as the previous findings - CodeQuizzer’s design and implementation should respect the wants and needs of developers in regards to code reviews. However, findings on this topic can also highlight areas that need improvement, as well as directions that can be taken to explore such areas further.

Motivations for doing code review

There are various ways to go about this, but in this paper this will be done by examining some case studies on code review in big software development companies - namely Microsoft and Google. The reason for this is because these companies are very successful and it is reasonable to assume they have put a lot of time and effort into optimizing all aspects of software development, including code reviews. Therefore, a lot of insight can

be gained by examining their motivations and expected outcomes. Additionally, as mentioned previously, medium and small companies seem to mirror what is done on a larger scale [4], so universally applicable insights could be obtained to some degree as well. Ultimately, this decision was also made with respect to the scope of this paper, which can only cover so much of the literature available.

At first glance, the assumed primary (or maybe even only) motivation for code review is what this paper will refer to as *defect prevention*. A *defect* in this context is some unintentional behavior caused by code quirks or logical fallacies in the implementation, resulting in some risk or damage to the system. A simple example would be some bug in the code. This assumption about the motivation of code review is indeed proven correct, as both at Microsoft [8] and Google [3], defect prevention is listed as one of the motivating factors. Defect prevention is of course desired before changes go public, which is relatively easy to ensure at large companies that do change based reviews. However, at smaller scale companies where code review is done infrequently or without a schedule, this desire to prevent defects before going live can become a more explicit motivator [4].

Furthermore, while at Microsoft preventing defects is still the primary motivation and goal of the process [8], at Google things are different. As mentioned in the case-study [3], the idea is to make changes to the codebase that other developers can understand, rather than prevent bugs. This shows that Google has adopted *maintainability* and *understanding* as its primary motivators for doing code review. So much so that defect prevention seems to be viewed almost as a side effect of the pursuit of maintainable and understandable code. The argument is that by aiming to write such code developers also produce consistent, secure and bug-free code [3]. This idea is also present at Microsoft, although less explicitly and more as a side effect of doing inspections [8]. In a sense, at Google maintainable and understandable code produces bug-free code, whereas at Microsoft the inverse holds - bug-free code is maintainable and understandable.

Closely relating to maintainability and understandability, another motivator at both companies is *knowledge transfer* [3][8]. The basic idea is that by doing a code review developers are exposed to code they otherwise would have not been, thus, increasing their understanding of the codebase as a whole. For example, if a feature is to be implemented by a developer who, through code review of similar features, has been exposed to the relevant parts of the codebase, tasks can be accomplished more efficiently due to the developer's better understanding. This was even precisely noted by one of the developers at Microsoft as a motivator [8]. Additionally, knowledge transfer may be more

explicit, with discussions about changes sometimes including external sources of information that may educate developers further on the relevant topics [8].

Outcomes of doing code review

Next, as far as outcomes are concerned, they mostly follow as expected from the motivations behind doing code review. At Google, developers expect that through this process they will ensure the company-wide norms are followed, accidents are prevented and there is some gatekeeping to the code being committed [3]. These expected outcomes all relate to the idea of defect prevention. Gatekeeping especially seems to be enforced, as code reviewers are required to not only own the parts of the codebase that changes are being made to, but also to have passed a readability certification, which ensures they are aware of the best practices for the language of the project [3]. This relates to the way things are done at Google, as they have fewer code reviewers assigned than Microsoft [3][8], but it seems they have a higher barrier of entry for who is allowed to review. While Microsoft may differ slightly in their implementation of the inspection process, defect prevention, code improvements and finding alternative solutions to problems are still desired outcomes [8].

In regards to knowledge transfer, this is again expected at both companies to some degree. At Google developers expect to be educated about the codebase through doing code reviews [3] and so do developers at Microsoft [8]. Although not a primary motivator, at Microsoft it is also expected that doing code reviews will lead to higher team awareness and transparency, particularly in regards to what changes are made and when they are made. Furthermore, a sense of shared code ownership and responsibility is created via code reviews, as exposing your code to the rest of the team and collaborative effort on it makes developers less possessive and protective of their work [8]. Interestingly enough, research seems to suggest that this might be one of the advantages of in-person meeting-based code review, similarly to what was proposed by Fagan. An experiment conducted in [7] found that developers had a better sense of code ownership and a higher confidence in the defects identified when doing more traditional in-person style code review.

To conclude this section and summarize the findings so far, a comparison between Microsoft and Google's motivations and outcomes is given below in Table 2. It can be observed that both companies do code reviews for similar reasons and expect similar results, however, they have a slightly different approach and primary focus. Therefore, these motivations and outcomes can serve as a groundwork for the design and

implementation of CodeQuizzer in regards to what developers want to achieve by reviewing code.

	Google	Microsoft
Primary motivator	Maintainability, understandability	Defect prevention
Expected outcomes and secondary motivators	<ul style="list-style-type: none"> - Education about the change - Ensuring norms - Gatekeeping code committed - Defect prevention 	<ul style="list-style-type: none"> - Increased knowledge of the codebase - Knowledge transfer - Higher quality code written - Alternative solutions proposed - Shared team ownership of code - Increased transparency and awareness

Table 2. A comparison between Microsoft and Google’s primary motivations and expected outcomes when doing code review.

Challenges in code review

Having established the motivations and expected outcomes of the code review process, this gives a solid direction of what developers want and need, but also what they may need help with. However, a more direct way to establish this would be to simply compile some challenges currently faced by those involved in doing code review. The same case studies will be examined as before, with the same rationale as to why behind this choice. However, additional sources will be used as supporting and guiding material to expand the insights gained. This is done in order to have as thorough an understanding as possible of the challenges faced in code review as they are most relevant to the goal of the research.

From the sources examined, the biggest challenge identified when it comes to code review is *Understanding*. This is a multifaceted issue that affects various different sub-issues faced at both Microsoft and Google. It is also identified as a problem at another large company - Sony Mobile, as well as in the other supporting literature examined.

From the literature examined, understanding takes two forms - understanding the changes presented for review and understanding the changes of the surrounding context of the codebase these changes are meant for. These are two different types of understanding both with two different sets of challenges developers face when trying to obtain them.

Understanding the changes presented for review

In regards to understanding the changes presented for review, this is necessary in order to provide a code review at all. Put simply, in order to provide a review it is necessary to grasp what the change code does or is supposed to do, before one can comment on if and how well it does said thing. Despite this fundamental need for this type of understanding, obtaining it is often identified as a struggle. The study done at Microsoft identifies this challenge explicitly, with developers mentioning it in the interviews done there [8]. This is supported by other findings as well, as sometimes the purpose or function of the change presented for review may not be clear immediately. In some cases obtaining this understanding may even require understanding what the person writing the code was thinking as they wrote it, rather than examining the code itself [9].

Moreover, what can further add to this issue of change comprehension is the nature of the changes and the way they are presented to the reviewers. If changes are large and contain a lot of code, it can often hinder understanding, but also motivation and consideration while doing code review [9]. The research done at Sony Mobile expands on this particular issue, suggesting that it can be solved by implementing a priority system for changes, so that hotfixes and high priority bits of code are reviewed first and given most time, regardless of change size [5]. This is an approach likely to be taken in some bigger projects, but less adopted by OSS projects or other companies which may be less mature in their approach to code review [5].

In addition to change size, the order in which changes are presented for review can also have an impact on understanding them [9]. Sometimes reviewers must first understand a part of a change before they can understand another part and if presented in an order that does not facilitate this, changes can become less comprehensible. This issue could be quite prominent as research seems to suggest that most reviewers simply observe changes in the order they are presented, without any systematic approach that could boost their understanding of them [9].

Similar issues of understanding change code are present at Google, despite their commitment to writing understandable code as aforementioned [3]. Moreover, medium and small scale companies seem to face the same issue. As outlined in the previous sections of this chapter, at such companies communication is more often than not asynchronous and written, however, it can occasionally be in-person and direct [4]. The relevance of this in regards to the issue of understanding is the reasoning behind such communication being opted for. The data gathered by the study suggests that in-person

meetings are needed mostly when code is not understandable without further clarification [4]. What's more, developers will also sometimes resort to looking at change descriptions or commit messages in order to gain the knowledge needed [9]. These sources of information are used at Microsoft as well, together with proactively approaching the people presenting the changes in order to obtain the necessary understanding [8]. All of these findings indicate that regardless of company size, obtaining the understanding of the changes presented for code review is a big challenge.

Understanding the change context

Understanding the context in which a change is going to fit into is equally as important as the fundamental understanding of the change itself. What is meant by *context* here is the surrounding context of the existing codebase. Research shows that a high level of contextual understanding leads to higher quality reviews and much deeper and more meaningful feedback [8]. Moreover, developers claim that their code reviews are not only of a higher quality when they have a good contextual understanding, but they are also done faster and more efficiently [8]. This is to be expected, as understanding the existing codebase can lead to a developer being able to orient themselves quicker when reviewing changes for said codebase. Interestingly enough, obtaining this level of understanding is not only seen as beneficial to the quality and efficiency of a code review, but also as a necessary task in the process itself. Developers at Microsoft claim that often a review will require reading much more code than just what is presented in the change [8]. Furthermore, obtaining this understanding will sometimes be needed in order to identify if some behavior is intentional or a result of human error [4].

Nonetheless, despite the many benefits that understanding the surrounding codebase context provides for a code review and the clear need for this type of understanding, obtaining it is still a big challenge. At Microsoft developers claim that understanding code takes them the most time when doing code reviews [8]. At Sony Mobile similar challenges are faced, but even more so in the direction of obtaining a good level of context comprehension. In comparison to the case studies done at Microsoft, Google, AMD and other big OSS projects, the study done at Sony Mobile examines a project with a much bigger emphasis on integration tasks [5]. This is because part of it is developed externally, whereas another part of it is developed in-house. What the study found is that whether something was developed in-house or not had an impact on the outcomes of code reviews. Developers interviewed pointed out that when something is external it is a lot more difficult to understand what the context for it is, making work more cumbersome [5]. This finding supports the idea that Google, Microsoft, AMD and other OSS projects are

not outliers when it comes to large companies dealing with the problem of contextual codebase understanding.

Furthermore, another big challenge in obtaining knowledge of the surrounding context can be lack of motivation to do so. The case study done at Microsoft suggests that employees may actively avoid taking the time to understand on a deeper level, choosing to instead focus on more superficial feedback, centered almost exclusively on defect prevention, as logical or stylistic issues are usually more easily identifiable as opposed to deep design or implementation problems [8]. This active avoidance of providing deeper feedback is most likely due to lack of the deeper contextual understanding needed in order to provide such feedback.

In regards to tackling this challenge of lack of knowledge of the surrounding context, often extra meetings are necessary for clarification. This is an accepted practice at Sony Mobile [5], but also happens at Microsoft [8] and other smaller companies as well [4]. As a side effect of such meetings alternative solutions can also be proposed, as mentioned already in the larger scale studies [3][8].

Moreover, another way to tackle the challenge of obtaining contextual understanding is by altering the process implementation itself. This can for example be done by choosing the most appropriate reviewers for a certain review based on their familiarity with the parts of the codebase that are relevant. The idea is that by doing so, the need to obtain contextual knowledge is eliminated as the reviewers already have it. This type of functionality is already implemented in certain code review tools which suggest reviewers in such a way [9]. Even proprietary tools like Google's Critique will suggest reviewers not only based on the aforementioned ownership and readability requirements but also based on how recently they worked on similar changes [3].

Other research also seems to support this line of thinking. For example, [10] found that in several big codebases a decent chunk of code review requests suffer from problems in regards to assigning an appropriate reviewer. Moreover, the bigger the codebase, the bigger this problem becomes. The study's proposed solution is a new algorithm for recommending the appropriate reviewer for the task based on them having history reviewing files adjacent to the ones proposed in the changes [10]. This algorithm was found rather effective in shortening the time it takes to review a change with the suggestion that this is because the appropriate reviewer has a larger understanding of

relevant code chunks, increasing their efficiency and once more illustrating the importance of contextual codebase knowledge [10].

Another study developed a tool that suggests reviewers based on code familiarity with the additional aid of automating the review process itself using static code analysis [11]. This was also found to be very effective in picking the right reviewers for the job [11]. The studies mentioned support the findings at Microsoft and Google as well, and while they propose good solutions to this particular problem, they can lead to only certain people looking over certain parts of the code as time passes, thus, limiting understanding on a more global level [9]. In such situations it was found that direct in-person meetings can be quite beneficial for a shared sense of understanding, common ground and ownership of code. This seems to apply to both large scale companies like Sony Mobile [5], as well as smaller companies such as the ones in [7].

All of these studies seem to support the idea that understanding the surrounding context of a change presented for review not only results in deeper and more high quality feedback, but also in making the overall process more efficient. Nonetheless, there seem to still be many struggles obtaining such understanding and while some solutions are in place already there is room for additional research into tackling this challenge.

Additional challenges

Although obtaining understanding was the biggest challenge identified by the literature, several additional challenges were identified. While these are not the focus of the research it is still worth highlighting them briefly in this section. The first set of such issues are related to the code review process implementation itself. A case study done with six Finnish software companies highlighted that companies do not always have a formalization of any kind for their code reviews and when they do - developers are not always aware of this [12]. If they are, then the practice at these companies seemed to suggest that these formalizations are not applied. The primary reason for this seemed to be lack of motivation to go through changes in a systematic and thorough way or simply lack of knowledge and awareness about the process or changes themselves. The paper recommends more training to increase motivation and spread knowledge.

Moreover, while understanding was identified as a big problem already it manifests itself in some other ways than just the two already identified. For example, understanding plays a role in various other challenges at Google, mostly in regards to the organizational struggles at the company. The case study [3] found that *Distance*, *Review subject* and *Context*

are other areas where understanding plays a role. Distance refers to both physical distance, but in the context of understanding - organizational distance. A tech lead and a junior developer may be far apart enough from each other in terms of experience or organizational positions that this distance leads to misunderstandings and delays [3]. *Review subjects* may also be unclear, meaning that there is a misunderstanding about what the purpose of a code review is. One party involved may see it as simply a final inspection step, whereas another may view it as a part of the implementation process itself. This lack of common understanding of the subject of a review can also be problematic [3]. Finally, the *context* of the review is sometimes misunderstood as well. If a change is a nice-to-have, but treated as urgent or vice versa, this may cause delays or friction between developers [3].

Expert interviews on code review

While understanding has been identified as the biggest problem by the literature examined in this paper, it is important to support these findings with additional research which takes into account the developers' grounded perspectives. Therefore, semi-structured expert interviews were conducted with the aim of gaining insight into the exact same topic - defining the code review process, its motivations, expected outcomes and challenges. This is so these findings can then be compared to the ones in literature to see if they support, refute or expand them. In this way a more solid groundwork for CodeQuizzer can be established.

The questions asked during the interviews can be found in Appendix A. As can be seen there, they are divided in sections, with each aiming to gain some information about a different topic. Firstly, there are some questions about the interviewee themselves in order to gather context about them that could inform or correlate with the reasoning behind their answers. The remaining sections are self-explanatory, with each one focusing on one of the three aspects already established in the literature examination section.

The interviews were conducted with 9 participants working at 2 companies in the Netherlands - one medium sized and the other large. In total, the participants have worked at 6 different companies over their careers. Some also have experience in running their own business and working at startups in the software industry. Their ages vary from 21 up to 28 years old, with an average age of about 24. Their experience level also varies from about 4 months to 8 years in the industry, with a mean of about 5 years. Positions

also naturally vary based on this level of experience from junior to senior, but also in responsibilities. Both frontend and backend developers were interviewed, but also tech leads, product owners and testers who have a background in development or are currently working as developers. The variety of perspectives is purposeful, as the goal is to get as broad of a pool of participants as possible given the limitations of this paper. This is also why the interviews are semi-structured, as to allow participants freedom to express as many thoughts as they have without strictly conforming to a structure that could discard valuable answers. The remainder of this section summarizes the findings of the interviews, divided into the same areas of interest seen previously.

The code review process

All participants have experience doing code reviews. Some recall the first time or times they did it to not be change based, but rather given on-demand when asked by a colleague. Others cite their first experiences being a conversation in person, rather than the asynchronous modern process. These initial experiences seem to align with how things are done at smaller companies [4] or take the in-person elements from the original definition proposed by Fagan [1]. Regardless, their most recent experiences and general perception of code review aligns entirely with the definition of modern code review outlined in this paper.

The tools used for conducting code reviews are also unanimous with every participant using Git services to conduct the process. Either GitHub or GitLab's change view or merge request view is used to examine proposed changes. Some participants also mention using an integrated development environment to get a better understanding of the changes, which already points to this sometimes being a struggle. This is somewhat similar to the findings in [9].

Moreover, seven out of nine participants begin the code review process by looking at the ticket that introduced the need for these changes, citing the reasoning being the need for understanding context, using statements like "Once I understand what's [...] to be achieved, then I dive into the changes." and explicitly mentioning the need to "have an understanding of what needs to be done". This already points to understanding as an issue even at this early stage of the interviews. It is interesting that the participants who do not mention this as a part of their process have managerial positions and one even explicitly mentions that they skip this because they most likely wrote the ticket and already have the necessary context.

After gaining the needed comprehension, the tasks listed as part of the process include looking over the code for defects, looking at the high level logic of the code, checking semantics and styling and leaving feedback. These tasks are sometimes done in a slightly different order, but the majority of participants name all of them as part of the process. This aligns with the findings of the literature as well, described in the previous section. The tasks themselves also are most likely a result of the motivations and expected outcomes which are described next.

Motivations for code review

As far as the motivations behind doing code review are concerned, they also mostly follow what literature has established in regards to this. Firstly, about 40% of participants cite defect prevention as the primary motivation for engaging in the process. Interestingly enough, this applies not only to preventing defects that would immediately be introduced by the changes, but also ones that could be introduced in the future. A participant mentions that code reviews should make the system more “expandable” and ensure that future bugs and pitfalls are avoided. Regardless, these interviewees’ opinions seem to align with Microsoft’s primary motivation for the process, which is defect prevention [8].

Another 40% of participants mention a different primary goal, relating in some way to code quality. “Quality and reliability of code” are listed and participants believe that these can be achieved by making code “readable and easy to understand” through code review. Additionally, code styling is also cited in relation to this, with reviews sometimes seen as a way to do “code cleanup”. It is even highlighted that defect prevention should not be a focus at all, as it is deemed impossible by a participant who claims “it's pretty hard to catch bugs in code reviews, just because they are edge cases”. Instead, code smells should be avoided and in the process and by doing so defects are prevented as well. This seems to be very similar to the stance found at Google [3].

Finally, the participants in managerial roles once again have a slightly different view, providing a primary motivator in the form of achieving feature complete code. In this sense, code reviews are seen as a way to ensure all functionality of the proposed feature is present. This seems logical, as product owners and tech leads may care more about these aspects due to their job’s unique responsibilities. It also seems to align with what was found previously in literature that suggests that distance between organizational roles could inform a different main focus and cause misunderstandings [3].

Expected outcomes of code review

In regards to expected outcomes, a big point mentioned is also code quality. Essentially, the participants who did not list this as a primary motivation, list it as an expected outcome. This is very similar to the difference at Microsoft and Google, with one company thinking bug-free code is clean code [8] and the other that clean code is bug-free [3]. Regardless, code reviews ensure readable and understandable code and that is an expected outcome. This outcome also carries an additional positive of making the codebase better in the long run, with a participant saying that “people who started newly in a project can understand a project much better”, if code is written well.

Another major outcome expected from the process is what was defined in the previous section as knowledge transfer [3][8], although here it is referred to as “knowledge sharing” by an interviewee. Despite different semantics, the concept is basically identical. Essentially, participants see code review as a way to familiarize themselves with the codebase, with one saying it can “help you understand other parts of the code” and “you can also see different parts of the code that you never worked on”. Another says that this is especially useful due to the process setup which allows this familiarization to happen “bit by bit”. These views seem to indicate that increased understanding through this knowledge sharing is an expected and valued outcome of the process. Moreover, some participants mention that they can get ideas from the way something is implemented in a proposed change, using statements like “You get to learn how others think and problem solve which impacts your future work“ and describing code review as “a very good way to [...] get some ideas from other people, understand what is going on, understand the project itself”. In this way this outcome is nearly identical to what was found in [3] and [8].

Functional testing is also cited by participants as an expected outcome. What is meant is that through testing the logic for defects, sometimes code review can also test if the feature functions as intended. This is subtly different from defect prevention, as it tests the whole change in a manner akin to end-to-end testing. It is important to note that this opinion is not universal, as some participants view this task as completely separate and dedicated specifically to testers. As a result, they don’t focus on this proactively. A participant highlights this difference by comparing his current company to his previous company, where testing was not a differentiated task from code review. This led them to “try to be more precise in the reviews, because there was no tester there”. Some participants are even completely satisfied with the automated tools that provide testing

and don't do anything in regards to this when code reviewing, saying "looks fine to me, as long as the tests passed".

Finally, some smaller outcomes are also listed, mainly optimization and gatekeeping. The former is especially important for mobile phone applications where performance is key, while the latter is seen as a way to provide a sense of safety about one's work. In this way code review serves as a barrier of entry for those who are "very scared of breaking something" with their changes. This is again similar to the findings in [3].

Challenges

The interviews highlighted various challenges in the code review process. These were all of different nature, but could somewhat be categorized. Below each of these findings is outlined and described in-depth.

Understanding

As indicated by the literature review in the previous section, understanding is the biggest challenge in the context of code review. The outcome of the conducted interviews also support this conclusion, as the majority of participants indicated either explicitly or implicitly that understanding is something they struggle with during the process. There was so much data in regards to this that to make it more comprehensible, it can be divided into roughly three areas of understanding - *understanding the code in the change*, *understanding the context in regards to the codebase* and *understanding the context of the change ticket*. This is similar to the findings in literature.

Understanding the change code

Firstly, participants claim to struggle with understanding the code in the change, similar to what literature already identified. This seems to take the largest amount of time for some, with participants listing "Understanding what is happening there [...] the new logic introduced", "understanding [...] how the person thought [...] to implement the idea" and "interpreting" the logic as their biggest time sinks. Some participants point to this more explicitly, saying "understanding different perspectives, different views" is the biggest challenge or straight up seeing it as a prerequisite for conducting a review, saying "if you cannot understand [...] what is being implemented, then [...] you cannot review the code".

Another participant warns of the dangers in regards to lack of this type of understanding, stating "you might think you understand the code and miss one specific edge case" and elaborating that "it's quite quick or quite easy to think that you understand the code

within a few seconds but it's not always that". This implies that this lack of understanding can have negative consequences and not investing the proper time into it can be a common pitfall. The importance of avoiding this pitfall is highlighted by another interviewee, who aims to be "very sure about every detail of the pull request, like I've written it myself".

However, despite this importance, it is sometimes the case that even the person who wrote the change does not completely understand their own work. One participant cites an example of a negative experience doing code review, when one of their colleagues got lost in their own logic, so they had to spend "a couple of hours explaining again" what it does and what is wrong with it. This outlines lack of understanding the change code as a multifaceted problem for most of the participants. This was an aspect of understanding also mentioned in literature with much the same struggles identified as well.

Understanding the codebase context

Understanding can be problematic in the context of the larger codebase surrounding the proposed changes as well. A vast majority of participants mention this is an issue or is something that they spend a large amount of time on, with some even saying "understanding the context" takes them the most time. This also "depends on the project and how well [they] know the code base" and if a change is "obscure". Moreover, it can become especially bothersome for more complex changes that require checking multiple files. A participant says this takes the most time, which might be related to lack of understanding of the relationships between components. Sometimes these components can even be coming from other unfamiliar teams, with one participant saying "you need to spend a lot of time just to understand whether their changes are fine and compatible with your team's changes". This seems to be similar to the struggles at Sony Mobile [5].

What's more, there might also be a lack of motivation to gain an understanding to begin with, as checking everything and gaining context is described as a "hassle". This lack of motivation can also be further impacted by a feeling of intimidation, as some say they were "a bit afraid of doing code reviews" because they were not "familiar with the codebase", especially at the beginning of their careers. This can be described as a "steep learning curve" because "it's a lot to take in initially [...] to understand a lot of code that you have never seen in your life". Even participants who do not experience this feeling of intimidation are empathetic towards it, with one claiming they do not personally get intimidated, but they understand why others might be "hesitant" to provide feedback for projects they don't know a lot about.

This hesitation might be caused by an increase in difficulty. This seems to be indicated by participants as a vast majority of them find code review for a project they are less familiar with more difficult, even if they do not find it more intimidating. According to one interviewee, this increase in difficulty is because “you don’t know the context [...] which generally tends to be quite important”. Another states that “you're not 100% sure if there could be some potential issues or it might interfere with other [...] parts of the code”.

Moreover, the quality of feedback also seems to decrease when lacking context knowledge, which in one participant’s words is because “if you don't know the project [...] very often you also don't know the best practices”, implying that the quality of the review goes down due to this. Other participants share the same sentiment, making the claims they “have less of value doing the code review” or even refusing to do the review outright, saying “I'm not doing the code review if I don't know the project”. This indicates context understanding has a very large impact on the quality of feedback. This is elaborated upon further with the reasons cited being mostly related to depth of feedback, with statements like “I can flag some stuff that looks weird on the surface, but there's less in-depth knowledge from my side” and “I’m not going to have a deep understanding [...] I'm just going to review it from experience”. These indicate that when having lower context understanding, one’s feedback tends to become superficial. This is further supported by claims like “there's not really that much to code review apart from general language best practices that are independent of the project” and also the fact that even though some participants claim to look for deeper issues with the code, it is often the case they rely solely on intuition and experience to do so. This implies that with better understanding of the context, the feedback also becomes more in-depth.

All in all, there seems to be almost total alignment between this issue as presented in the literature examined and the issue as identified by the expert interviews. All the same benefits of having contextual understanding are acknowledged by the participants and all the same struggles are faced with obtaining knowledge about the surrounding codebase of a change presented for code review.

Understanding the ticket context

An aspect of understanding that is also sometimes a struggle but was not identified by the literature examined is understanding the tickets behind the proposed changes themselves. Some participants claim that “reading and understanding the problem” or “understanding what is needed” takes a significant amount of time. Another states that

knowing “the reasoning behind changes that were made” is important. These are more tangential remarks, however, as the primary focus of understanding remains the actual code and the context it lives in. Even statements relating to understanding the tickets have an underlying implication that this is important in order to understand “whether the intention of the ticket is aligned with the implementation”.

Additional challenges

Aside from the most commonly cited struggles of understanding, there seem to be some other problems identified as well. Below each is touched upon briefly, as the primary focus of this paper is still on the issue of understanding.

Feedback

Feedback is cited as a point of struggle by some interviewees. Mostly these challenges seem to stem from two things - one is the wording of the feedback and the other is determining which feedback to actually give. The first mostly relates to the tone and level of directness used. This is referred to as “level of feedback” by one participant, who struggles with how direct and concise to be and how much “fluff” to add to their comments. Another states that they try to word things in a “positive constructive way” but also to explain and justify “why [...] I provide this feedback to you”.

The second point mostly relates to the struggle of what feedback is considered a “nitpick”, with some participants omitting comments they have in an attempt to avoid this. One person even calls this “overachieving” and sometimes prefers to not provide all their feedback as they want to use code review to give people “a confidence boost”.

Another factor impacting this is the time sensitivity of issues, with some participants saying they have approved code without leaving feedback even though they have it. The reason for this is precisely this time sensitivity, as for tickets that are high priority and need to be deployed quickly, it is preferable to deploy them now and fix any minor issues with the code later. This lack of feedback is also sometimes justified by a mindset of “no code is perfect” or offloading the task of a more detailed code review to the other assigned reviewer. This once again might stem from a lack of motivation.

Change size and presentation

Another challenge is relating to change size and change presentation, similar to what was found in [9]. Participants cite “too many lines of code, too many files in one [...] merge request” as a problem or even “scary for the reviewer”. This also impacts the review itself,

as it is stated that finding edge cases is “very hard” especially “when there are like 300 changes in the code review”. This can lead to a less in-depth review, but also to the review being considered a negative experience by interviewees.

Tool related challenges

Some participants cite that the Git tools they employ are sometimes cumbersome to use and navigate code with. This is why they use an integrated development environment or additional tools to navigate the code better. This can also impact the level of feedback, as some state that it is more difficult to leave feedback than it should be due to a bad user interface and experience.

Process related challenges

Finally, some participants seem to have some challenges relating to the process itself. One such challenge is a large amount of back-and-forth. This deals with the situation of a change having to go through a lot of cycles, which can be considered “annoying” for both the reviewer and the review receiver.

Another challenge in the same vein is the large time between requesting a review and receiving a review. This is even cited as the biggest potential for decreasing the time costs, as it can sometimes take weeks until feedback is received. This could be because of another issue, referred to as “context switching”, meaning transitioning from development work to code review work. This is cited as a reason for putting off code reviews by some and could be another area where motivation can play a role.

Finally, the people involved in the process can be a challenge in and of themselves. One participant claims they struggle with giving or receiving feedback to and from people they dislike. Finding a professional tone while keeping your personal feelings to the side can be an issue in this sense, somewhat similar to the organizational distance issues highlighted previously [3], although far more personal.

Conclusions on code review

To conclude the background research in regards to code review, the primary findings of the available literature as well as the expert interviews are outlined. The idea here is to compare the two and create a combination of both that hopefully provides the fullest picture of the process itself, the motivations, expected outcomes and challenges. These can then be used as not only a foundation for the development of CodeQuizzer, but also as

a starting point for looking into gamification aspects that could integrate well into the process while also helping with the issues found in it.

Examined literature establishes that the code review process began as a very formal, rigid, synchronous and time-consuming process in the late 70's, with interview participants having experience with some elements of this formalization as well. As time went on, code review slowly evolved into the modern code review process due to many factors, including a larger code and team sizes. This new process is widely adopted by companies, both large and small due to its informal, flexible, asynchronous and lightweight nature. This modern process definition is shared by both literature and the interview findings, as participants essentially described this definition when asked about what code review is according to them. Therefore, as this definition is shared between the two it will also be the definition used in this paper and respected by CodeQuizzer's design and implementation.

The motivations and outcomes of code review are various, but to sum up they are mostly related to defect prevention, code quality, gatekeeping and knowledge sharing. Different companies and people give these different priorities, but in general the idea is always to ensure a working, well implemented solution as efficiently as possible. This is the sentiment in the examined literature, but all of these aspects were also mentioned and elaborated upon by the interview participants. This once again shows alignment between the two and indicates this is a solid set of motivations and outcomes to consider for this paper and when developing CodeQuizzer.

Finally, the biggest challenge identified by both the literature and the expert interviews is that of obtaining the necessary level of understanding. The literature highlighted two types of understanding - understanding the code presented for review and the surrounding context (codebase) it fits into. The interview participants confirmed that they struggle the most with understanding as well and mentioned exactly the same types of understanding. Additionally, they highlighted understanding the tickets introducing the changes themselves as another understanding-related challenge.

Moreover, while the literature examined pinpointed some smaller tool related issues or issues that are less abstract and more dealing with practical problems, the interview results expanded this considerably, giving four more areas of issues that aligned with what has been seen so far. They also provided a more grounded and in-depth look at the problems themselves from a developer perspective. While these findings are also

important, they are outside the scope of this particular research, but can provide interesting insights and direction for other papers.

In conclusion, the literature and expert interviews align almost entirely on the three points of interest. The literature covers them more abstractly, while the interviews provide a more grounded look. Both together provide a solid definition for the code review process and its expected outcomes and motivations. Most importantly, both the literature examination and interview results point to the challenge of understanding being the largest one faced by developers doing code reviews. With all this in mind, next it is necessary to examine gamification as a concept and see how it could potentially be applied to tackle this challenge.

Literature examination on gamification

Gamification - a workable definition

In order to understand how gamification could potentially be applied in the context of obtaining understanding in code review, it is necessary to establish a groundwork about what gamification itself is similar to how this was done for the code review process. The definition used throughout this paper will be the one proposed by [13], which defines the concept verbatim as “the use of game design elements in non-game contexts”. The primary focus here is on *game* elements, rather than elements of play. Gamification is more about applying concepts from games as artifacts, rather than of the actions that one usually performs while interacting with them (i.e. playing) [13]. Sometimes these two may overlap, but the starting point are games, rather than play and they will also be the starting point for the solution proposed in CodeQuizzer.

Furthermore, [13] makes a distinction between gamifying an activity and transforming the activity itself into a game. The former is achieved through the aforementioned elements being borrowed, which implies that only parts of games are being applied to an activity that is not in a game context. If this activity is in and of itself transformed into a game, then this would no longer be gamification, but rather something more akin to a serious game (i.e. a game in a serious context) [13]. While this is perfectly acceptable, it does not align with the definition proposed for gamification, which only uses specific elements of games. Furthermore, this paper will only apply game design principles and features partially, rather than attempt to make a serious game out of the code review

process. This is in order to have the research applicable in a professional context, which is not the best suited for gaming activities.

Applying gamification to the problem of obtaining understanding

Having established a workable definition for gamification and what type of elements will be used in CodeQuizzer, it is next important to examine how these can be applied to the challenge of obtaining understanding for code review. As already outlined, getting the necessary understanding can be a challenging task in several different ways. Out of these identified ways, it seems that the most suited for a solution that implements gamification is obtaining *understanding of the surrounding codebase context*. Firstly this is because gamification is usually done in order to boost users' enjoyment, engagement or motivation when doing an activity [13] and as aforementioned in the previous section of this chapter, a big problem for developers can often be finding exactly the motivation needed to familiarize themselves with a codebase. Perhaps making this process more engaging and enjoyable via gamification can provide the motivation needed.

Secondly, this is because the other identified areas of understanding do not seem the most suitable to apply gamification to. When it comes to understanding change code, this is not often an issue of lack of motivation, but rather a lack of well-written understandable code. While submitting understandable code can certainly be gamified as well, it seems like a less natural fit to do so, although this approach is a good direction for another research paper. On the other hand, when it comes to understanding challenges that have to do with organizational distance or ticket content, these seem closer to communication issues than to issues of lack of engagement and motivation. So for similar reasons as the previous, this aspect of understanding is better delegated for future research.

Finally, while there exist various studies that apply gamification to code review a big chunk of them seem to focus on gamifying the process of actually reviewing code. This was identified by a literature review conducted in [14] which found that the part of the code review process that has to do with actually reviewing code is the most commonly reflected in studies aiming to gamify it. This could be in the same vein as the first direction for future research suggested previously (i.e. gamifying understanding the changes presented for review). This makes the argument of exploring other directions even stronger, as the gap in research there seems more present based on the literature examined.

Which gamification elements to apply?

With the idea of applying gamification to aid in obtaining understanding of a codebase established, it is next necessary to identify which specific game elements to apply. This can be quite difficult to pinpoint, as it can be troublesome to even establish which concepts are specific only to games. [13] argues that the *characteristic* elements should be the ones that are applied, but research is yet to be unanimous about which elements actually can be considered as such. Due to this lack of agreement, this paper will intuitively apply elements drawn from the author's own experience with games and gamification, but also inspired by other research done and solutions developed that use it for similar purposes.

In regards to CodeQuizzer specifically, there are several elements that could be applied. [13] splits these by level, but regardless of what categorization is used, it seems like leaderboards, badges and levels [15] are elements that could be applied in the context of CodeQuizzer to attempt to boost motivation for obtaining understanding. Additionally, various other elements like feedback, providing challenges and rewards, as well as experience and progress systems are all often used in solutions employing gamification as found in a study done in [16]. These can all also be applied in CodeQuizzer's design and implementation in order to achieve a similar goal. Which elements and how exactly they are applied is discussed shortly in the next section.

In addition to game elements, game design principles could be used when implementing the proposed solution in order to receive some benefits as well. Put simply, if the prototype solution uses some ideas from game design, it is possible that users will have a better time interacting with it and will in turn come out of the code review process having done it better and experienced fewer challenges than otherwise. There do not seem to be many studies done using this approach, or at least ones that fall within the scope of the papers examined here. While this is a gap in research, it falls outside of the scope of this research, so these types of elements will not be formalized and will be drawn from the researcher's background knowledge of game design.

Existing applications of gamification

To conclude the gamification section of this chapter, it is a good idea to examine existing implementations of it that operate in a similar domain to CodeQuizzer as they can provide inspiration or at least an overview of what is out there. In general, there seems to be a relative lack of research in the area of gamifying code review at all, at least from

what was gathered using this paper's methodology. Nevertheless, this type of application seems to provide some positive effects. For example, a study done comparing gamified and non-gamified tools for code review found through survey responses that gamified tools were slightly preferable in the experiment that was conducted [17]. Furthermore, only a small fraction of participants found the gamification elements unnecessary, indicating that there is something to be gained in applying this concept to code review. Although productivity and comment usefulness did not go up because of the application of game elements, the paper lists many limitations that may explain this, so there is hardly a conclusive understanding of how gamification can be applied to tackle code review challenges [17]. What's more, the study suggests that there is also some dependency on the aspects that are to be improved. For example, improving productivity would be something different than improving the spread of knowledge or understanding, such as with the challenge of this paper. Therefore, it might be a good idea to conduct research in this domain to find out more.

Another study indicates that gamifying code review in an educational context yields quite a few benefits [14]. For example, the quantity, thus, the participation in code review seems to have a significant increase when introducing game elements such as leaderboards, points and badges [14]. Furthermore, there was some indication that using game design elements and particularly time-based mechanics could be used to encourage participants to spread out their code reviews over time rather than do them all "last minute" [14]. Moreover, although the participants did not perceive gamification to have improved the quality of feedback given to them during code review, it did result in longer and more in-depth comments, likely due to its motivational and engagement-boosting nature [14]. This nature could similarly be applied in CodeQuizzer to boost motivation and engagement when getting to know a codebase as well.

Additionally, when expanding the scope of previous research beyond just code review, there seems to be some other uses of gamification in the software industry. For instance, in an Agile software development setup in the form of *Planning Poker* as described in [18]. The idea here is that the process of estimating how long certain tasks will take is gamified. The game elements are borrowed from the traditional card game of poker and the goal is to lower the time making estimations takes and to involve and engage the whole team in the task of estimating [18]. Aside from estimations, there have also been attempts to gamify prioritization of tasks in an Agile context, such as in [19], which indicated positive effects on engagement when applying gamification in this context. On a more abstract level, there have even been frameworks for gamifying education that

utilize Agile, such as the one proposed in [20]. All of this shows that combining Agile and gamification is not something out of the ordinary, which shows the software industry is no stranger to gamification. This should hopefully mean that a system like CodeQuizzer which uses it extensively is not so novel that it is not comprehensible for a professional in this industry.

In the same more general line of thinking, there are also studies that focus on improving software development itself through the use of gamification. For example, [21] focused on improving software quality through tackling the issue of developers ignoring warnings their tools provide about the code they have written. The study attempts to gamify the removal of warnings, concluding that gamification can be effectively applied to improve the motivation and engagement of developers taking part in this process, thus, producing better code with less warnings. Another study focused on better code quality through gamification is [22]. Gamification was introduced here to help enforce code conventions and produce more maintainable and readable code. The intervention introduced was mostly successful in both increasing compliance with code conventions, but also in producing more readable code. This confirms the suspicions about potential directions and gaps in research presented in the previous section.

A different approach to improve software quality was taken by [23], which attempted to gamify the aspects of the development process that have to do with documentation of the code written, with the argument that better documentation leads to better, more maintainable code. Gamification was not found to have a significant impact on the quality of the documentation, however. The authors suggest that this is because due to various aspects in their study that were not foreseen. Either way, it shows another attempt at gamifying software development and again confirms the aforementioned suspicions.

Conclusions on gamification

To conclude this section on gamification, it seems that there are some studies that find grounds to make the claim that gamification could be effectively used to help tackle some challenges in the code review process, mainly relating to motivation and engagement. Furthermore, gamification is applied elsewhere in a more broad context to tackle similar challenges to the ones faced by developers taking part in code review. However, there seems to be a relative gap in research on this topic. While some systems gamify general software development tasks or even understanding code presented for review, few attempt to gamify obtaining understanding of the surrounding codebases. Additionally, it also seems that while there exists a good amount of literature that focuses on education

(as outlined systematically in [17], for example), there seems to be less literature looking at the professional domain. Therefore, CodeQuizzer will focus on applying appropriate gamification elements to tackle the primary challenge of understanding the surrounding codebase of a change presented for code review in a professional setting. This higher level of understanding in turn should increase the quality and efficiency of code reviews as outlined in various other works' findings.

Prototype and evaluation

Having established a solid foundation of contextual knowledge and found a direction to go in, next a prototype solution was designed and implemented. This solution was then evaluated with the purpose of answering the research questions of this paper. In this section, the solution design and implementation are outlined, followed by an overview of the evaluation method used alongside it.

Prototype Design

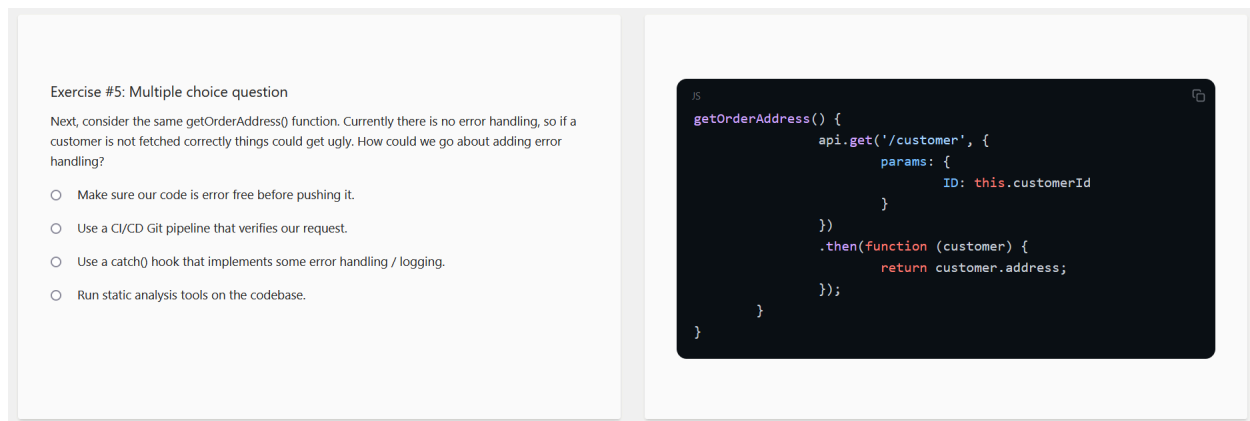
As aforementioned, the prototype solution is a quiz taking tool named *CodeQuizzer*. Its primary purpose is to aid developers in understanding their team's codebases via quizzes. The high level idea is that a senior member on a team (for example a tech lead) can set up quizzes in the tool that focus on relevant parts of a codebase. Each quiz contains a set of exercises that present a snippet of code that is important or relevant to understand, together with a task the quiz taker must complete. Each quiz has a *comprehension level* attached to it with the idea being that the higher the level, the more complex and in-depth the tasks and code presented are. Upon completion of a quiz, the developer earns the corresponding comprehension level, which then indicates how well they know the codebase.

The reasons for choosing this solution approach are various. However, the largest reason is that as outlined in the preliminary research, the solution to understanding a codebase can sometimes be as simple as taking the time to observe and interact with it. The problem in that case is moreso the fact that developers may not be motivated or engaged enough to actually do so. Keeping this in mind, it seems logical to use an interactive quiz tool in combination with gamification elements to make the process of familiarizing oneself with codebases more engaging and rewarding, which in turn improves the motivation to understand. In addition, this approach allows for relatively easy and logical integration of gamification elements such as progress bars, positive feedback, levels,

badges, etc. because they have already been tested in similar tools, but in a different context, as outlined in the preliminary research.

Quizzes

As mentioned already each quiz in CodeQuizzer contains a number of exercises that the quiz taker must do to complete it. The layout of an exercise can be seen in Figure 3, which in this case presents a multiple choice question exercise. As can be observed, there is a chunk of relevant code on the right side and the provided task on the left. This layout is in order to give visual access to the code at all times, without the user having to scroll up and down a page to observe it (as they would have to do if they were positioned one on top of the other).



Exercise #5: Multiple choice question

Next, consider the same `getOrderAddress()` function. Currently there is no error handling, so if a customer is not fetched correctly things could get ugly. How could we go about adding error handling?

- Make sure our code is error free before pushing it.
- Use a CI/CD Git pipeline that verifies our request.
- Use a `catch()` hook that implements some error handling / logging.
- Run static analysis tools on the codebase.

```
JS
getOrderAddress() {
  api.get('/customer', {
    params: {
      ID: this.customerId
    }
  })
  .then(function (customer) {
    return customer.address;
  });
}
```

Figure 3. An example of a multiple choice exercise which also illustrates the quiz layout.

Upon completion of the task (for example in Figure 3 - answering the question), the quiz taker can progress in the quiz. Upon doing so, they are shown their provided answer on the left and the correct answer displayed on the right. Alongside it, there is an explanation as to why the correct answer holds true. This layout can be seen in Figure 4.

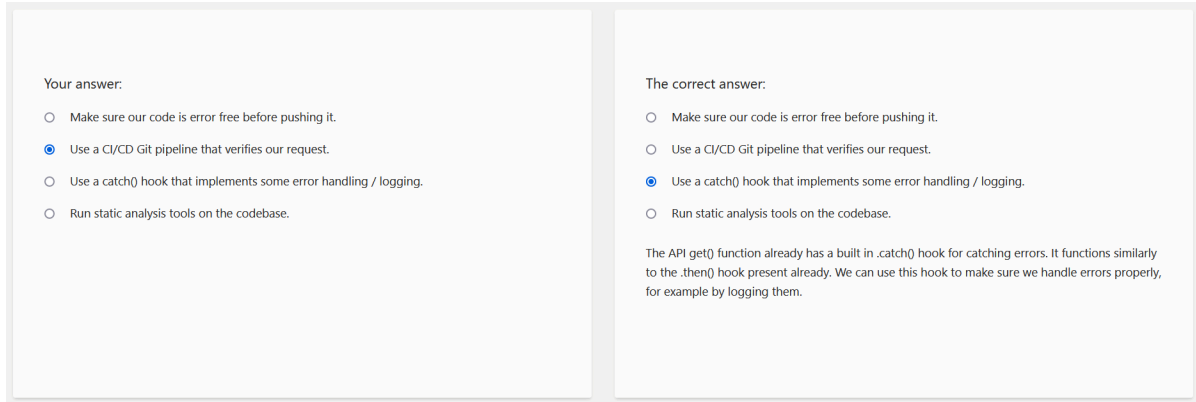


Figure 4. An example of the correct solution provided after answering a multiple choice exercise.

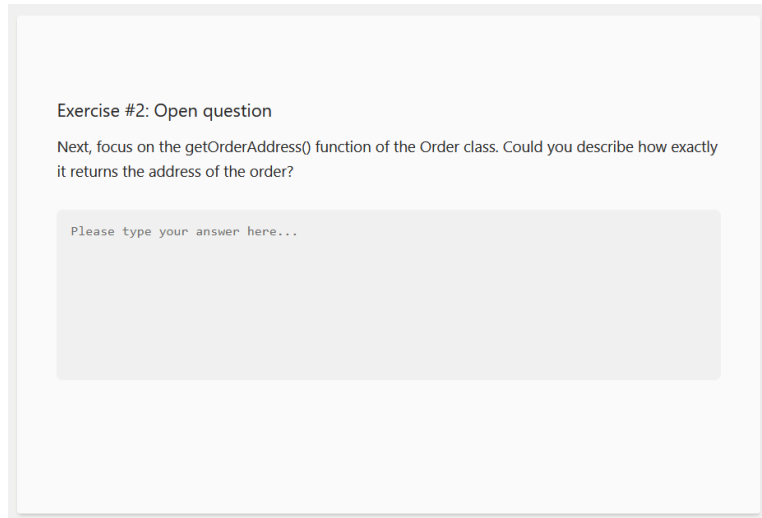
While the quizzes may resemble exams, the intention behind them is not as such. There are no grades or scoring systems, nor is there any pressure on providing the correct answers. This is because the main goal of doing a quiz is not to achieve a good grade or get penalized for wrong answers, but rather to observe the code provided and increase one's understanding of it via doing the exercises. This is also the primary reasoning behind the design choice of providing the correct solutions immediately after an answer is given. In this way quiz takers can reflect on their answers and see what they may have done wrong, without receiving negative feedback for their incorrect answers.

Ultimately, once all exercises have been completed, the users are presented with a completion message and their comprehension level is increased to the one corresponding to the quiz they just finished. Both this completion message and the comprehension levels are part of the Gamification elements used in CodeQuizzer, which are outlined later in this section.

Exercises

Each exercise in CodeQuizzer presents a different kind of task focused on the code snippet presented. Based on the task given, exercises come in 6 types: *Multiple choice question*, *Open question*, *Function flow*, *Component diagram*, *Functionality altering* and *Variable role* exercises. In this section an overview of each one is given, together with some relevant design decisions made about them.

Multiple choice and open question exercises



Exercise #2: Open question

Next, focus on the `getOrderAddress()` function of the `Order` class. Could you describe how exactly it returns the address of the order?

Please type your answer here...

Figure 5. An example of an open question exercise.

An example of a multiple choice question was already given in Figure 3 and an example of an open question is given in Figure 5 above. As can be seen from both, these types of exercises are fairly standard and follow what one might expect from an exam or other quiz giving platform. While as mentioned already, the intention of CodeQuizzer is not to give exams, these types of questions can have merit outside of that context as well.

The design rationale for including them is primarily flexibility. While not as interactive as other types of exercises, they allow much more freedom in the types of questions asked, which can be useful in a variety of cases where a more specific exercise type would not fit.

Function flow exercises

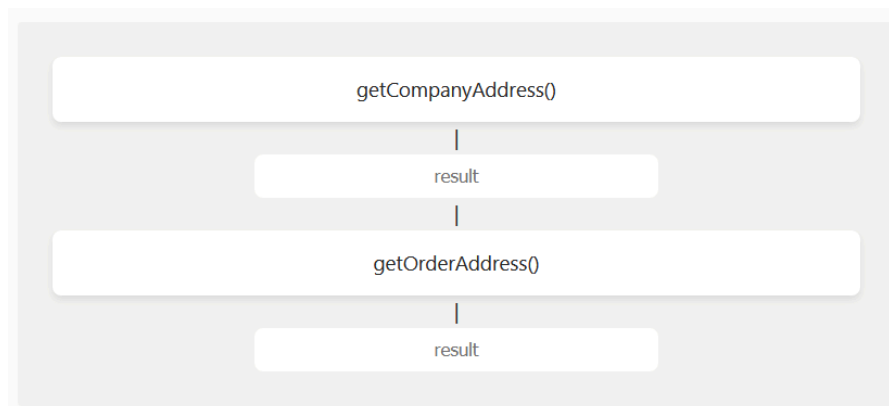


Figure 6. An example of a function flow exercise.

An example of a function flow exercise can be seen in Figure 6. The idea behind it is that the user is presented with a code snippet that contains various data properties as well as methods that access or manipulate them. In this case some functions for accessing orders on an order processing application. The quiz taker is then asked to retrieve a certain bit of information (for example: the address of an order). They can then put the function calls in the correct order by dragging them and also indicate the intermediate results of each of them.

The design rationale for including function flow exercises is that a variety of complex systems have various components that communicate with each other via some function calls. The order and intermediate results of these calls may be a source of confusion or necessary for understanding the larger context of a codebase, so such an exercise seems like a logical inclusion. Furthermore, it is relatively flexible, as it could focus on data flow between multiple components, data flow within one component or even data flow over the internet via an HTTP based API. In this way it is useful to increase understanding in a wide variety of contexts.

Component diagram exercises

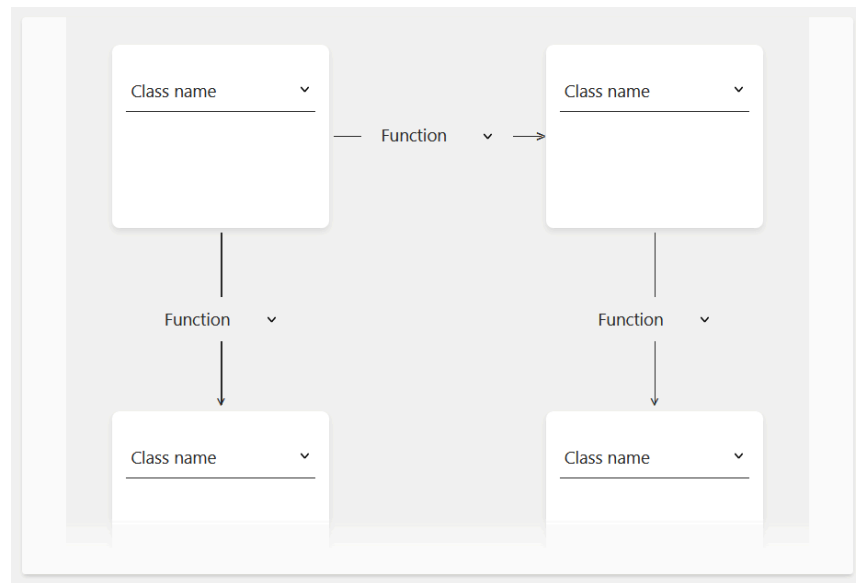


Figure 7. An example of a component diagram exercise.

As can be observed in Figure 7, which gives an example of a component diagram exercise, such an exercise provides a code snippet of various components of the codebase, together with some functions that are used for communication between them. The quiz taker is

then asked to select the appropriate components in the diagram and the appropriate functions that allow them to communicate with each other.

As component diagrams are a relatively familiar and proven system for designing software architecture, evoking the concept here seems like a good idea for improving one's understanding of a codebase's architecture, components and connections. The design of the exercise emphasizes a high level understanding, because other types of exercises are intended to serve as a deeper dive into specifics. This is also the reason why this diagram is simplistic, rather than an accurate UML (or other modeling language) diagram. The design intention is to use these exercises as the starting point of a quiz, so the quiz takers can build a simple and pragmatic mental model of the relevant code for the rest of the exercises.

Functionality altering exercises

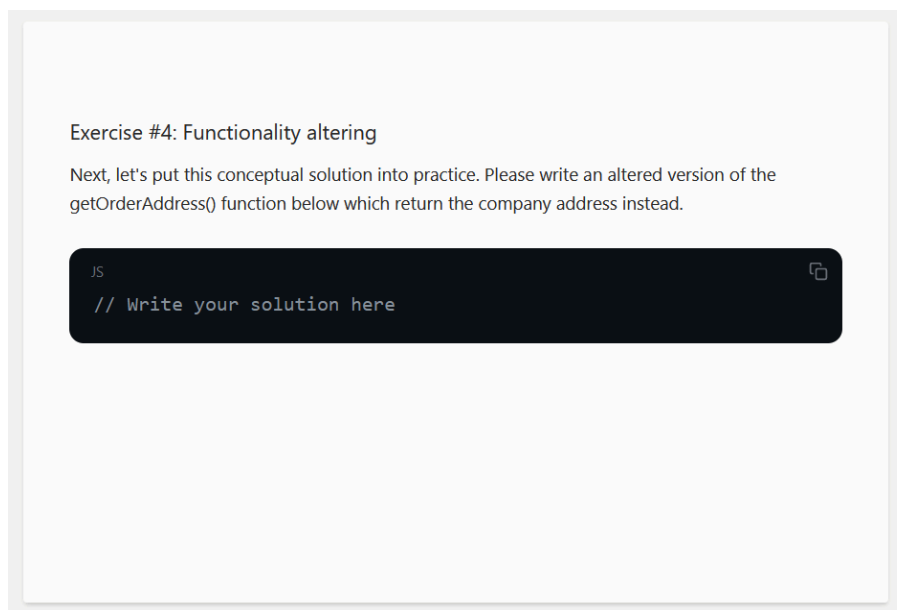


Figure 8. An example of a functionality altering exercise.

An example of a functionality altering exercise can be seen in Figure 8. This exercise provides a code snippet as expected and it asks the user to alter its functionality in some way. For example, to make a function return a different value, or take a different approach to a problem solved in the original code.

The rest of the exercise types provide a certain level of abstraction of the tasks, either by asking questions, or providing some kind of interactive elements. However, this type is

intentionally designed to provide an experience most alike a developer's day-to-day tasks - i.e. writing and altering code. Hopefully, through doing this exercise quiz takers think deeper about the snippets presented, but also do so from a more grounded and familiar perspective than what is provided in other exercise types, which are more abstract.

Variable role exercises

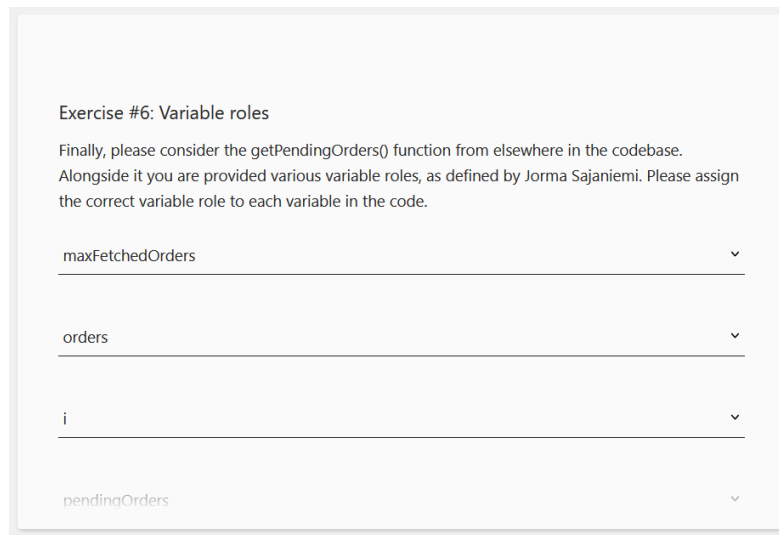


Figure 9. An example of a variable role exercise.

The final type of exercise is the variable role exercise seen in Figure 9. As with all others, a code snippet is presented, but this time the focus is on the variables contained within it. For each relevant variable, the quiz takers must select its appropriate role within the snippet.

The design of this exercise is primarily based on a pen and paper exercise provided in *The Programmer's Brain* by Felienne Hermans [24]. There it is suggested that doing this can help with the comprehension of large bits of complicated code, so it seems like a natural fit for CodeQuizzer, which aims to help with the same struggle. The roles themselves are based on some of the work of Jorma Sajaniemi, for example in [25]. What CodeQuizzer adds to this sort of exercise is intractability and a technology based implementation. Moreover, what this exercise provides is a way to address variables in code, as the rest of the exercise types do not provide an explicit focus on this.

Gamification Elements

In addition to improving understanding via quizzes, as mentioned before CodeQuizzer employs various gamification elements to motivate developers. This motivation is twofold

- on one hand gamification can be used in the quizzes themselves to motivate users to complete them, but it can also be used within the larger context of the CodeQuizzer system in order to motivate users to do even more quizzes. In this section all gamification elements are described, together with some reasoning for their usage and design.

Quiz progress bar



Figure 10. Quiz progress bar also used for navigation.

The first element is a progress bar for the quiz, which can be seen in Figure 10. Progress bars are a common gamification element, as discussed in the preliminary research. They seem like a good fit here both as a way to indicate to the user how far along the quiz they are, but also to motivate them to complete the quiz by seeing their progress increase. Additionally, in a fully functional implementation, this sort of bar could be displayed next to unfinished quizzes as well, potentially motivating users to return to them and complete them.

Quiz completion message

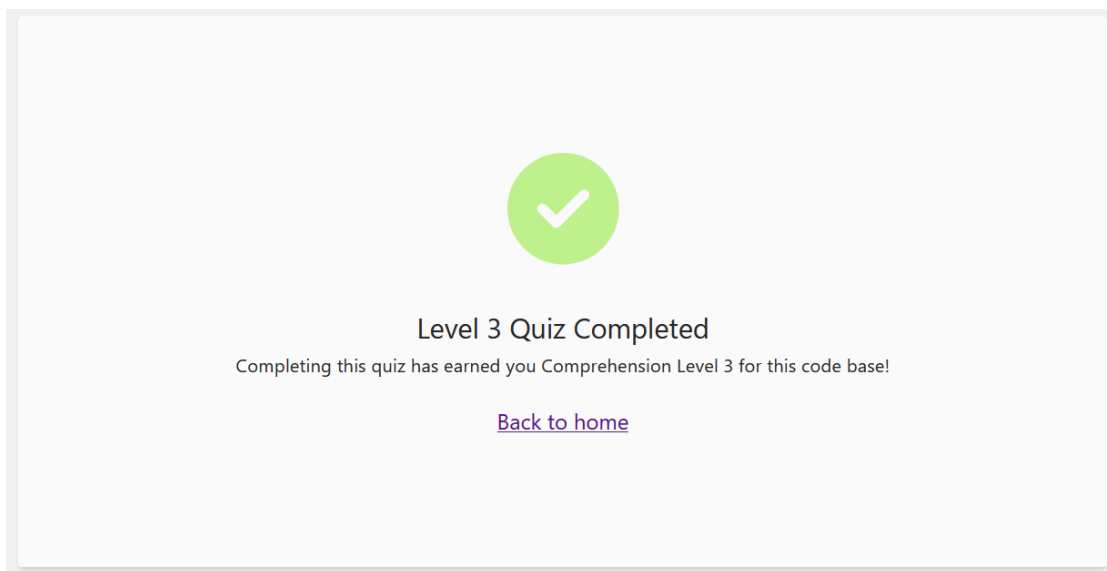


Figure 11. Quiz completion message presented once a quiz is finished.

Positive feedback is another gamification element that is used for motivation in the context of CodeQuizzer. In particular in the message that quiz takers receive after completing a quiz as seen in Figure 11. This message is not only to inform them the quiz is

complete, but also to congratulate them on their new level of comprehension of the code base. Furthermore, it has a visual element that is universally accepted as positive - the checkmark. In this way hopefully this message fosters a feeling of satisfaction in the users, which then motivates them to complete other quizzes in the future as well.

Profile badges

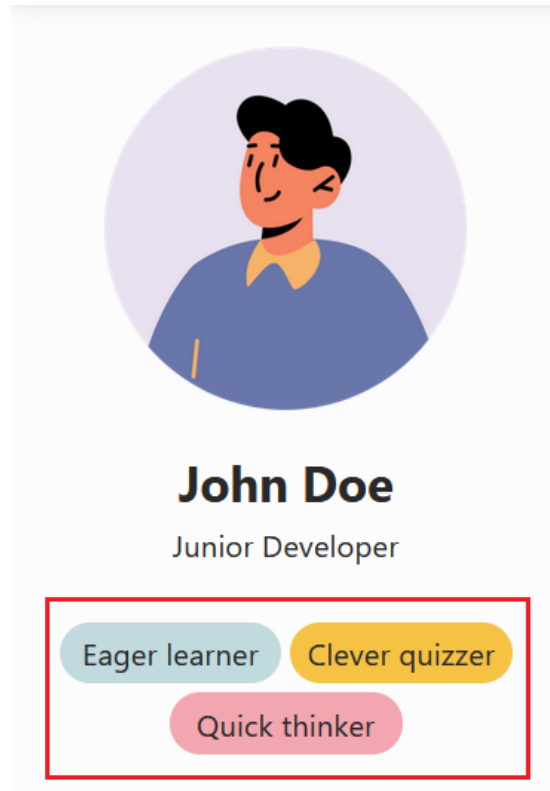


Figure 12. Badges displayed on a user's profile.

Alongside the quizzes themselves, the CodeQuizzer prototype features a home screen that contains the rest of the gamification elements used. The first among them are the profile badges contained in a user's profile section on the home screen as seen in Figure 12. Here they are only illustrative, but integrated in a fully functional solution these badges would be obtained by achieving certain things such as doing many quizzes, providing a lot of correct answers or doing quizzes quickly. The idea is that these badges would serve as intrinsic motivation for users who want to obtain them, but also as extrinsic motivation for users that may see them on another person's profile and wish to obtain them themselves. All this in turn would result in more quizzes being completed by users. The badges are textual with the intent of the playful names sparking interest in the users, but

also providing some hint as to what they could do to obtain them so they feel motivated to interact more with CodeQuizzer.

Comprehension levels

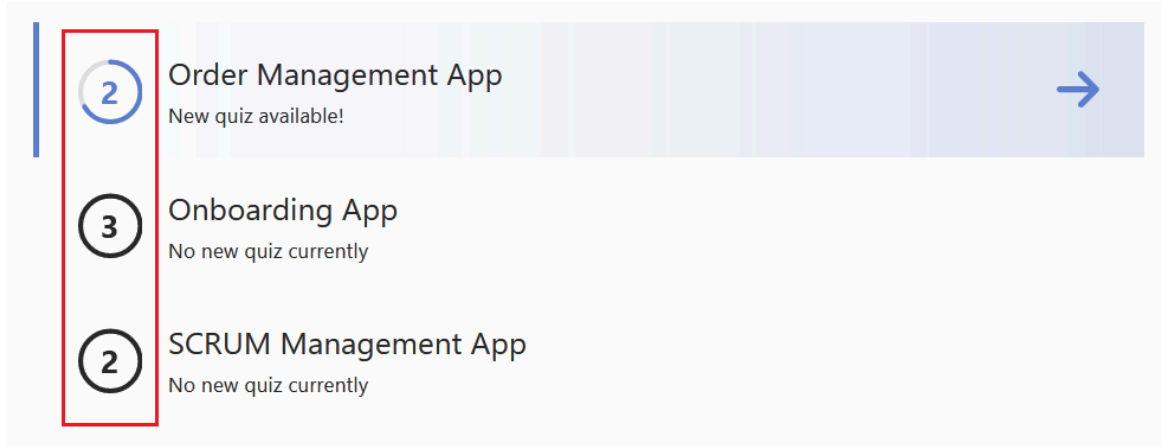


Figure 13. A list of a user's codebases together with their achieved comprehension levels for each.

Still on the home screen, there is a list of a user's codebases displayed as seen in Figure 13. As mentioned earlier, each user has a certain comprehension level achieved for each codebase they work on. Here they are displayed next to the codebase names and if a new quiz is available for a given codebase, this is indicated via highlighting but also via the circular progress bar on the left.

Levels are another common gamification element as described in the preliminary research. In CodeQuizzer, the comprehension levels are included as a primary motivator for doing more quizzes. The idea is simple - a user would want the highest level possible, so they would do more quizzes to achieve it. Moreover, in a fully functional solution, these comprehension levels could be interfaced with third-party platforms such as Git. For example, they could be displayed next to user's names to motivate developers to improve their levels by seeing others' levels or more interestingly, they could be used to select the most appropriate reviewers for a given code review, either manually or via some algorithm. In this way the primary gamification element in code quizzer can be integrated into the code review process directly as well.

Leaderboard

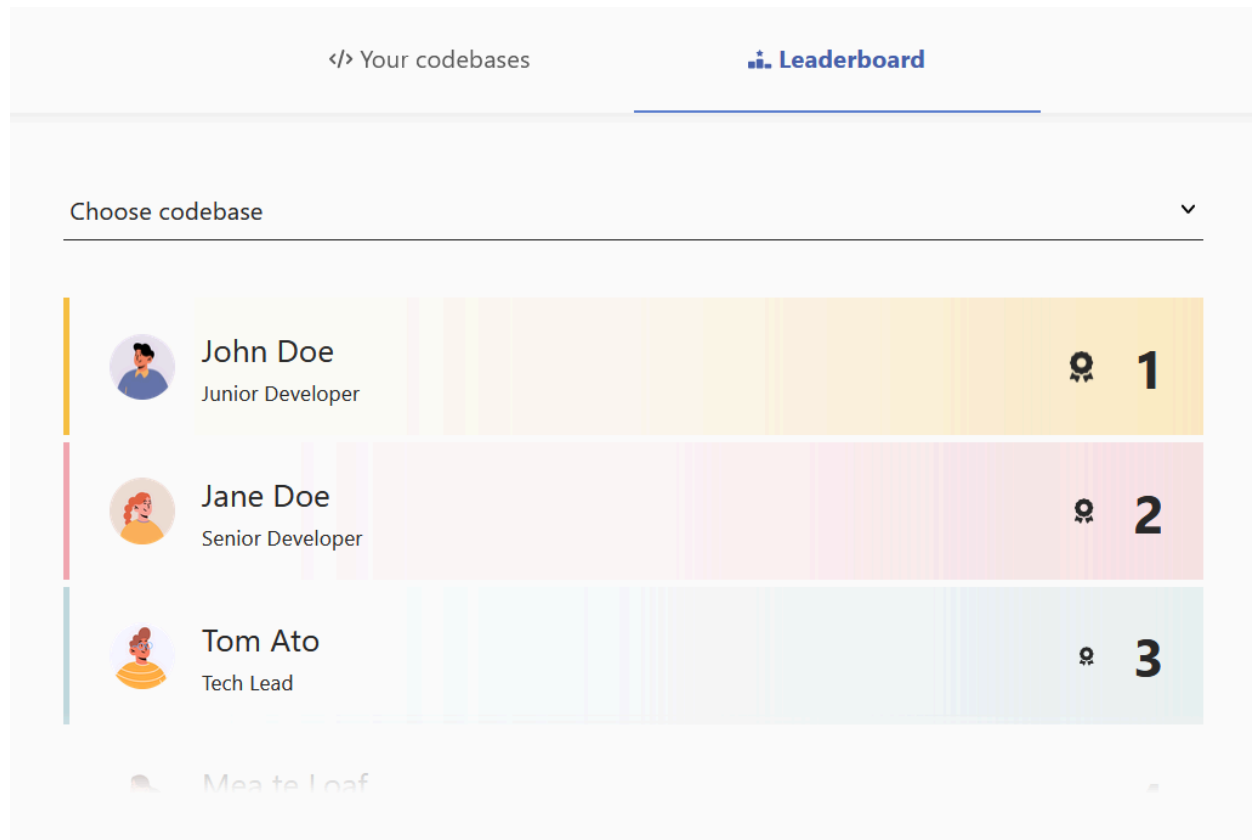


Figure 14. CodeQuizzer's leaderboard section.

The final gamification element used by CodeQuizzer is a leaderboard as seen in Figure 14. This is a self-explanatory concept - users can see a leaderboard of their team members. The more quizzes and code reviews done, the higher one's placement is on the leaderboard. Leaderboards can also be filtered by codebase, allowing users to observe which projects they may be ignoring in their pursuit of understanding.

The idea here is once again quite simple in that this public leaderboard can hopefully motivate developers to do more quizzes and code reviews. Aside from the general motivation of wanting to place at a high position, it is possible that some users may even have more personal motivations, such as rivalry with a particular colleague which might motivate them even further. Moreover, in a fully functional solution this leaderboard could be displayed in a third-party application as well, if emphasis on it is found to be particularly useful for increasing motivation.

UI / UX Design

While UI / UX design is by no means the focus of CodeQuizzer's prototype, it was still taken into consideration when designing it. This is to some degree unavoidable, as anything that users interact with will need some thought placed into UI and UX. In this case this is mostly based on the researcher's intuition and prior experience with these fields. However, there are still some decisions under this umbrella that may be relevant and interesting to discuss.

Firstly, the visual design was kept simplistic and clean, using primarily a neutral color (white) and an accent color (light blue). This keeps elements readable and visual strain to a minimum. Furthermore, using a unified accent color keeps the visuals consistent and allows important elements such as buttons to be easily identifiable at a glance. Other colors are only used in places that need to stand out further, such as the profile badges and the leaderboard. Regardless of the color values themselves, saturation is kept low to avoid color clashing and visually overwhelming the users.

Secondly, from a UX perspective, as mentioned already mostly intuition and prior experience was used to design the navigation layout and individual components. However, as a supporting reference and guide to commonly accepted web application rules, Google's Material Design 3 Guide was used [26]. This is to ensure a smooth and intuitive user experience across all aspects of CodeQuizzer.

Finally, it is worth mentioning that the layout and visual design was made with desktop exclusively in mind. The reasoning behind this is that this tool is meant to be used by developers in a professional setting and it is a safe assumption that those people would use desktops rather than mobile devices to conduct their day-to-day tasks. Therefore, as the target audience's primary devices have desktop resolutions, it seems logical to focus on those in spite of the commonly adopted mobile-first approach to web application layouts one may expect.

Evaluation Design

As aforementioned, CodeQuizzer's prototype is used to obtain an answer to the already established research questions. The way this is achieved is via conducting an evaluation study of the prototype. This section describes the high level approach of this study, together with highlighting the interesting and relevant design choices and their reasoning.

High Level Approach

The evaluation study uses a survey hosted online. The idea is that participants interact with CodeQuizzer's prototype in different ways and then provide their feedback by answering a variety of questions. The questions are grouped in 5 sections which first collect data about the participant's professional development experience, then about their impressions of the example quiz provided in the prototype, then about their impressions of each type of exercise, then about the gamification elements in CodeQuizzer and finally some data about the UI / UX experience.

Each of these sections focuses on answering the research questions or providing additional context for the answers given. The ultimate goal is to gather a general consensus about CodeQuizzer's ability to increase understanding through gamification as well as its effectiveness at it. In this way an answer is obtained to the two research questions regarding gamification's usage to increase understanding for code reviews. While the question of how this can be done is incredibly broad due to its nature (meaning that there may be infinite ways to use gamification for this purpose), evaluating how well CodeQuizzer achieves this task gives indication if this way is effective or not, based on participants' feedback.

The survey approach itself was taken for a variety of reasons. Firstly, it allows a wider range of participants to be included, as a more personal 1-on-1 evaluation (for example: an interview) can be more difficult to schedule and take more time and energy for participants to do. Therefore, choosing an evaluation method with a lower barrier of entry ensures a bigger pool of respondents. Furthermore, unlike the interviews conducted in the preliminary research, here the goal is not to gather the personal experiences and challenges of an individual developer, but more so their collective opinions about the prototype. This means that being able to use qualitative data is more useful for aggregation and it is also more convenient to work with for the evaluation study's purpose. Therefore, a survey format which allows for this type of data to be collected easier (i.e. with Likert scale questions) is better suited than a more free-form interview or focus group format in this case. Nonetheless, the space for qualitative data is also given with various open questions included in the survey as well.

Survey Design

Having covered the high level approach, various design decisions were made in regards to the specifics. These range from deciding what sections to include, what questions to

include in each section, as well as the format of the questions themselves. These decisions are relevant for the answers provided, so in this section they will be outlined per survey section, together with the reasoning behind them. A full list of the questions used in the survey can be found in Appendix B together with the type of responses participants can give.

Participant Context

The first section of the survey focuses on gathering information about the participants themselves. This information is mostly about their professional background in the software development industry. Information is gathered about how long they have worked in it, what their current job position is, how large their team is, etc. Gathering this type of data is important, because it could provide some additional context or justification for their answers. For example, the answers of a junior developer with a few years of experience may differ wildly from those of a tech lead with a decade in the industry. In particular, the questions about team size and the presence of a tech lead in it are essential, because if a participant only works in a team of 5 people where no one has a senior role this might bias their opinions about a system such as CodeQuizzer which is targeted at larger teams and requires a tech lead to set up the quizzes.

Example Quiz

As aforementioned, the prototype includes one example quiz. This quiz consists of 6 exercises in total - one for each type. While a longer quiz may be more illustrative of the actual quiz experience one might have, this length was chosen with regards to a different set of priorities. The most pragmatic concern informing this decision is the time a participant spends on the evaluation. If a long quiz is presented to them this may make their evaluation experience more negative, thus biasing their answers. What's more, the evaluation itself is focused primarily on the conceptual viability of CodeQuizzer. This means that the motivation behind it is not to provide a 100% accurate representation of how the tool will be, but rather to evaluate if the concept behind it (i.e. using gamification to increase understanding) has merit and could be used in the way that it aims to be. This goal can be achieved with a shorter quiz, while avoiding the drawback mentioned previously. Furthermore, the rest of the survey focuses on the exercise types and their effectiveness and for that an entirely representative experience is also not necessary with just one example of an exercise type sufficing to gather the needed data.

With all this in mind, the example quiz still focuses on one single mock-up codebase that the user can conceptualize in their mind (a web application for placing and managing

orders). This is a common use case that hopefully developers are at least somewhat familiar with or can at least grasp relatively quickly, while still providing enough intricacy to judge if CodeQuizzer helps them understand code better. The same motivation sits behind the choice to use Javascript for the code snippets, as it is pretty verbose, but free of types and a lot of language-specific restrictions, as those may complicate the process and bias the results obtained.

The questions asked about the quiz in general are mostly centered around gauging how well this type of quiz would help improve understanding of an unfamiliar code base. Furthermore, questions are asked about the general layout, what the opinion is of being shown the correct answers and also how much the quiz feels like an exam. These are not only asked to understand the general experience of the participants, but also to find if the goal of avoiding the exam feel is achieved or not, which is important for reasons already mentioned in the prototype design section.

Exercise Types

The next section of the evaluation survey gathers data about each individual exercise type, with the goal being primarily to gauge how useful these exercises are for boosting understanding and also how engaging and enjoyable doing them is. The former is important to know as understanding is the primary goal of doing exercises, whereas the latter is important to know in order to see if exercises are engaging enough to motivate users to do quizzes, as gamification should not be the sole motivator.

What's more, here it is worth mentioning that most questions in this but also in other sections ask the participants to rate their agreement with various statements. This Likert scale approach is fairly standard for a survey, but a purposeful design choice was made to provide scales that are even numbers (i.e. from 1 to 4 or from 1 to 6). In this way participants must always provide some level of agreement or disagreement with the statements provided, giving a stronger indication of what works and what does not. While this may bias the data to some degree, given the limited time for carrying out the study this is necessary to ensure there is a solid conclusion in the end.

Gamification

The next section of the survey focuses on the gamification elements present in CodeQuizzer. This is important to include because the research questions are both centered around gamification's effect on increasing understanding. Each element that was described in the prototype design is present in the implementation and the survey

contains questions about each one. The questions primarily focus on how motivational each of the elements is in order to answer the research questions.

Moreover, there are various element-specific questions that aim to gather data about certain aspects that the researcher suspects might be important to know. For example, for the quiz completion message the survey inquires about participants' opinions on the amount of visual elements present and the tone of the message itself. This is to understand how its motivational aspects can be improved. Furthermore, sometimes questions will present hypothetical scenarios not present in the current prototype implementation, such as integrating comprehension levels into third-party platforms like Git as described in the Prototype design section. This is so opinions can be gathered about potential future directions CodeQuizzer could expand in given a fully functional solution.

What's more, here it is worth mentioning that open questions are given to allow participants the opportunity to provide more detailed feedback that may have not been covered by the closed questions. This approach is present in other sections as well, but it is particularly important here, as motivation is a complex concept that might need more clarification and nuance. Furthermore, extra attention is paid to the verbiage of the questions, with them using present simple or conditional tense, steering the focus away from the prototype specifics and towards a more conceptual view of the system. This is also the approach taken in the rest of the sections, with past tenses used only when asking about specifics of the current implementation.

Usability

The final section of the evaluation aims to gather opinions on how usable the system is. The way this is achieved is via the common System Usability Score (S.U.S) test [27]. In brief, it contains 10 statements with answers that are on a scale of agreement ranging from 1 to 5 [27]. Based on these answers, a usability score on a 100 point scale is calculated that can serve as an indicator of how usable a system is [27]. While other types of data collection may lead to more nuanced and accurate evaluations of usability, the research questions both do not focus on this at all, so knowing this is only useful to explain biases in results, such as for example if the system is very unusable which could make the answers more negative, or to provide practical directions for research.

What's more, in the same line of reasoning, there is only one question that focuses on visual design in addition to the 10 S.U.S test questions. This is an open question and is

only present to gather extreme opinions about the visual design that could have biased the answers in the same way lack of usability may have.

Evaluation results

In this part of the paper the results of the evaluation described in the previous chapter will be summarized, outlined and discussed in-depth. For organizational purposes, these results will be focused on by section, with the sections being the same ones already used in the evaluation design description. Based on these results answers will be provided for both of the research questions in the conclusions of this paper.

Participant context

The survey was sent out to approximately 50 people of which 17 provided a response. Every participant responding has worked in or is currently working in the software development industry with 47% having worked for 1 to 3 years, only 3 people having worked for less than a year and the rest being an equal split in the remaining ranges of 3 to 10 years. Participants were aged between 21 and 31 years old with a median age of about 25. Their job positions also varied, with two tech leads, one service delivery engineer, one devops engineer, one C++ / Python developer and the rest being an equal split between backend, frontend and fullstack developers. The median team size is about 9 people with only one person being a solo developer. All participants but 2 have a tech lead or analogous role in their team.

As with the expert interviews, this variety in participants is purposeful. CodeQuizzer is a system that will affect an entire team's code review process, so it is important that the opinions of a variety of people is surveyed. Furthermore, the job descriptions present in the responses cover most of who would usually be on a software development team doing proactive development, whereas the variety of experience levels and ages cover junior, medior and senior developers. This hopefully creates as representative a picture as possible with respect to the time and scope limitations of this paper.

When it comes to the team sizes, they vary a bit less, but that is to be expected as plenty of research exists about what optimal team sizes are. Moreover, most respondents belonging to medium or large teams is to some extent a good thing, as the focus of the background research which informed CodeQuizzer was exactly those types of teams. So, they are, albeit not explicitly by design, the primary target audience of the solution as well.

Nonetheless, there is a solo developer outlier as mentioned previously. Their response will not be discarded, as it is still valuable, but the bias of their different context will be kept in mind when analyzing the responses further.

Aside from team sizes, the presence of a tech lead or similar role is also important, as CodeQuizzer relies on them to function and may not be understood properly if the participants did not have such a role on their team. Fortunately, as mentioned already all but two of the respondents have such a person on their team and the only outliers are familiar with the idea of such a role, therefore, their bias is hopefully minimal.

To conclude, the responses obtained cover a wide range of appropriate people who also have the necessary team sizes and roles in order to have the contextual knowledge needed to interact with CodeQuizzer's prototype. The variety hopefully ensures as representative a perspective as possible, while the contextual knowledge hopefully minimizes the skewing of the results due to lack of said knowledge. Next, the actual responses will be examined per section of the survey.

Example quiz

When it comes to the example quiz, the experience with it was well received by respondents. The most common rating it obtained was a 5 out of 6, with 89% of participants scoring it above a 4 and only 2 people giving it a 2. The format of the quiz itself was also the subject of positive feedback as every participant but one gave it above a 4 and around 35% gave it the maximum grade of a 6. The exact percentage distribution of the quiz experience and quiz format scores can be seen in Figure 15 below.

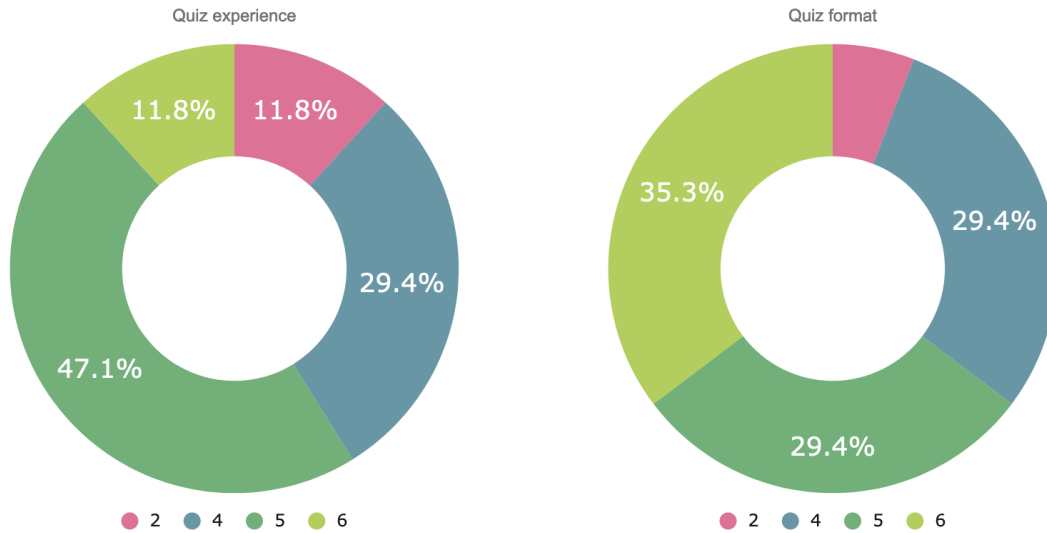


Figure 15. Percentage distribution of numerical scores for the quiz experience (left) and the quiz format (right).

Aside from the high numerical score, the experience was also described positively in participants' comments. The quiz was called "very user friendly", "coherent" and with a "nice design". This indicates a positive reception of the layout and format. On the other hand, the experience was described as "fun" and its questions as "engaging". This reception may be due to the gamification elements present or due to the user friendliness (or perhaps due to both combined). Nonetheless, the opinions on the quiz experience were positive, which indicates that the system itself is a success in this regard.

Unfortunately, despite best efforts, the opinion on whether the quiz feels exam-like is more split as seen in Figure 16 showcased below. Almost 70% of participants think that it resembles an exam to some degree, with almost 20% feeling strongly about their agreement and only one participant strongly disagreeing. This could be for a variety of reasons, but an educated guess is that this is because the format of asking a question with the expectation of a correct answer may inherently resemble an examination to those familiar with the concept. More research is necessary to pinpoint the exact reasoning behind this feedback.

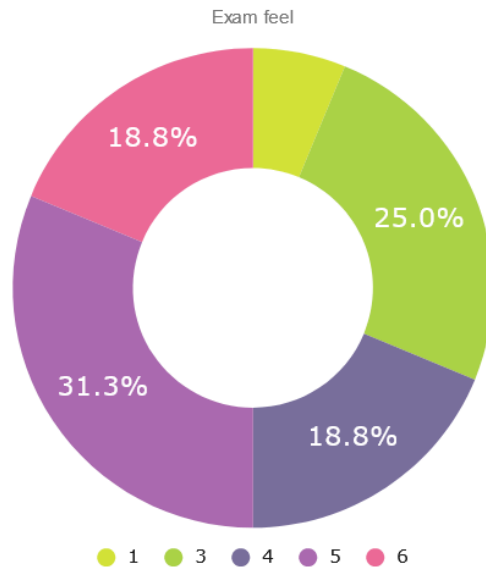


Figure 16. Percentage distribution of the participants' opinions on how much the quiz feels like an exam (1 being "Not at all", 6 being "Very much").

Nonetheless, the comments of respondents that did not find the quiz experience positive also seem to mention this exam-like feeling. For example, one pondered why there is no consequence for "if you fail" the quiz and made the statement that "[they] shouldn't be the one checking if [an exercise] is correct". This indicates that the quiz felt like an exam to them, as they expect some consequence for failure and also expect a higher up to "check" their performance. This is more akin to receiving a grade rather than a self-evaluation with the focus of increasing their understanding through it. Another participant makes this point more explicit by saying that the quiz "does not teach [them] that much about the code itself" and that it feels more like a "code exam". Moreover, the variable roles exercise in particular was also called "theoretical" by another respondent and it was indicated that this is not an exercise that aids in practical understanding. All of this feedback, coupled with the numerical data shows that, albeit for a minority, the quiz did not get across its purpose of increasing understanding, instead feeling like a performance evaluation tool. This together with the rest of the points for improvement that will be discussed shortly could have impacted the overall quiz experience for these respondents.

When it comes to the goal of increasing understanding of a codebase, opinions were also majority positive as seen in Figure 17. Around 65% of respondents indicated some level of agreement with the idea that such a quiz would be helpful to aid in their understanding with most of them giving firm or strong agreement. Moreover, in participants' comments, the quiz was called "helpful" and its explanations "useful". The latter statement was also

supported by numerical data as almost 90% of the participants agree that the correct solutions after questions are useful for their understanding.

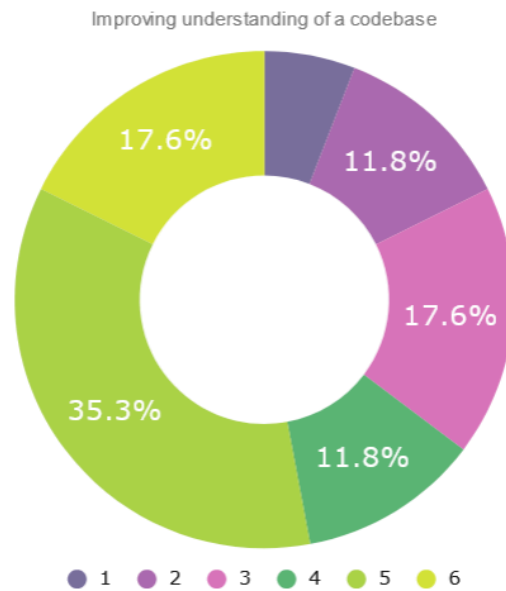


Figure 17. Percentage distribution of the participants' opinions on how much such a quiz would help improve their understanding of a codebase (1 being "Not at all", 6 being "Very much").

However, despite CodeQuizzer having a positive impact on the majority of user's understanding, the rest of them seem to find it less useful when it comes to this as also seen in Figure 17. More than 30% disagree to some extent that this type of quiz would help with increasing their understanding, with one even strongly disagreeing. Focusing on the respondents' comments provides the reasoning for this.

Firstly, there were some complaints about the content of the quiz and its presentation. For example, the wording of some questions being unclear and some elements being hidden behind scrolling such as the variable roles table as seen in Figure 18. For one user it was not clear that they could scroll to see more, while another explicitly said that the code "looks like an image" and that a scroll bar is necessary as indicator of scrollability. These are both valid points for UI/UX improvement going forward.

```
JS
getPendingOrders(maxFetchedOrders) {
  api.get('/orders')
  .then(function (orders) {
    let pendingOrders = [];
    for (let i = 0; i < orders.length; i++) {
      if (orders[i].status == 'Pending') {
        if (pendingOrders.length < maxFetch
            pendingOrders.push(orders[i
        ]
      }
    }
  }
  return pendingOrders;
});
}
```

Figure 18. Variable roles table being hidden behind scrolling, which was a point of feedback from participants comments.

Secondly, the specific programming language used (JavaScript) had an impact too, with one participant pointing this out as something that hindered their understanding. This is to be expected, as programming languages vary and if one does not have familiarity, this can be detrimental. Another participant added to this saying that the quiz feels like “following a paid online course for learning how to code”, which could also indicate lack of familiarity with the language itself. Perhaps if they knew more about JavaScript, this feeling would be less prevalent. Nonetheless, this is a bias that is present and will be kept in mind moving forward with the response analysis.

Finally, various respondents suggested that more personalized feedback could be beneficial in regards to the answers given. Several participants suggested that there should be some form of highlighting of their correct or incorrect answers to help them identify mistakes quicker. Another more explicit suggestion was that a participant would have liked to see “why [they were] wrong” in more detail by for example getting referred to the line of code they did not write down correctly. Introducing such more personalized and detailed feedback is a good direction to take for both increasing understanding by providing more detailed feedback on wrong solutions, but also as a way to lessen the exam feel by shifting the focus from aiming to achieve a high score increasing understanding.

To conclude the findings in regards to the example quiz presented, most participants find it to be a nice experience with big contributors to this being the UI / UX which was praised and the fact that the quiz was found engaging and fun. A majority also find the quiz concept useful for increasing their understanding with a particular contribution to this being the explanations after questions, which were universally praised. From a grounded perspective there were several UI/UX improvements that could be implemented. From a conceptual point of view, the main area for improvement is lessening how much the quiz feels like an exam, as this conflicts with the goal CodeQuizzer has. This could be done via introducing more detailed and personalized feedback when a provided solution is incorrect, shifting the focus from obtaining a high score to obtaining an understanding of code. This could in turn both lessen the exam feel, but also increase the understanding overall.

Exercise types

In order to deepen the analysis of the quiz and how well it aids in obtaining understanding of a codebase, it is necessary to examine the individual exercises that make up the quiz itself and in particular, the opinions of respondents about them. This section will do exactly this and in the interest of structure, this will be done in sub-sections, each focused on one type of exercise. At the end general conclusions on all the findings in regards to the exercises will also be presented.

Multiple choice exercise

When it comes to the multiple choice exercise presented in the prototype, more than 80% of participants agree that this exercise helps improve their understanding of the code presented as seen in Figure 19. Only three people disagree and no one strongly disagrees. This indicates that this type of exercise can be used to increase the understanding of unfamiliar code, so in this regard it is a success.

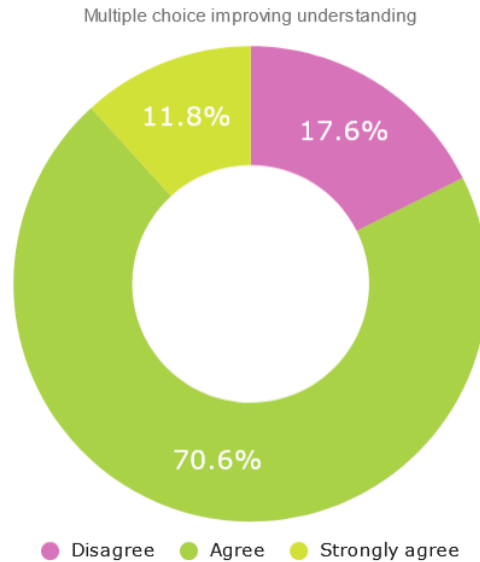


Figure 19. Percentage distribution of the participants' agreement with the statement that the multiple choice exercise helps improve their understanding of the code presented.

However, despite the responses indicating that this exercise helps achieve CodeQuizzer's goal, not a single participant indicated that this type of exercise helped the most in this regard. This could be for a variety of reasons, but a good assumption is that this is simply because the other exercises helped more. This assumption is supported by the overall positive results in regards to this point outlined previously, which show that participants did not choose a different exercise as the most helpful because the multiple choice one was not helpful at all, but rather because other exercises were more so.

Next, turning the focus point towards the experience respondents had with the exercise, everyone agreed that it was intuitive to do with more than 75% of the participants strongly agreeing with this statement. Aside from the positive opinion obtained, there is also another beneficial aspect of this finding. The exercise being intuitive for everyone eliminates the bias that a confusing exercise could have on its ability to aid in understanding, which adds further validity to the previous positive findings in regards to this.

Moreover, aside from being intuitive the exercise was also found interesting by the majority of respondents with about 70% agreeing that the exercise was interesting to do. Still, a minority disagrees with one participant even calling it "boring" in comparison to others. An educated guess as to why some might hold this opinion is that multiple choice questions are a familiar type of exercise to almost anyone who has done an exam,

therefore, they are not as fascinating as the others. Nonetheless, no one indicated strong disagreement so the exercise is still not totally disinteresting to the point of impacting engagement with the quiz.

Furthermore, despite not being selected by anyone as the most helpful exercise, the multiple choice exercise was selected by about 30% of participants as the one they most enjoyed doing as seen in Figure 25. The reasons found in respondents' comments are several, but one participant indicated that they enjoyed it most because it was the "easiest". Another one pointed out that it was "simple" and they enjoyed the fact they got "quick feedback". Yet another one sharing this opinion had much the same to say about it in their comment, also calling it quick and easy, but in addition indicating that they enjoyed that it was a "very common type of exercise". This seems to support the aforementioned correlation between intuitiveness and previous familiarity.

To conclude, the multiple choice exercise was received well. It was found helpful in aiding the understanding of unfamiliar code, albeit not the most helpful. Participants found it intuitive and interesting to do in general, with several indicating it was their favorite to do, owing much to its familiar nature which makes it quick and easy to do while still receiving useful feedback. Therefore, such an exercise should continue to be used in CodeQuizzer moving forward as it is overall a net positive for the quiz experience and also for increasing understanding.

Open question exercise

In regards to how well the open question exercise helps with understanding the presented code, participants indicated a positive opinion on this as well, albeit not as unanimously as with the multiple choice exercise, as can be observed in Figure 20. Around 70% agreed that it helped their understanding of the code, however, the remainder disagreed with one person even strongly disagreeing. Nonetheless, the numerical feedback is overall positive in regards to this, so this type of exercise can also be used for improving understanding.

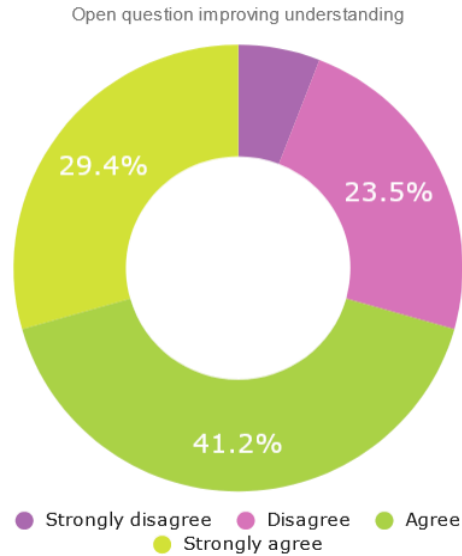


Figure 20. Percentage distribution of the participants' agreement with the statement that the open question exercise helps improve their understanding of the code presented.

Furthermore, one participant even indicated that this exercise was the most useful in regards to this. They elaborate further in their comment by explaining the reasoning for this being that this exercise made them think about the “entire sequence of commands”, rather than one specific component. This is interesting, as several other questions have the same focus, but it seems that for this person this was the most effective of them all. This could have to do with the question itself more so than the format and if that is the case this highlights the importance of well asked questions for a system such as CodeQuizzer.

Having proved slightly more polarizing but still useful for increasing understanding, this type of exercise also proved to be intuitive to do. More than 88% strongly agree on this point, with only two people indicating disagreement. This intuitiveness is probably for the same reasons of familiarity that were highlighted in the previous exercise type, but here there are no comments about this that can help support it. Nonetheless, the same points about bias being minimized due to the exercise being intuitive stand and they provide much the same benefit for the data. When it comes to the disagreeing respondents, unfortunately they did not add anything to provide reasoning in their comments, however, another participant indicated that they would have liked “more context”, so this could perhaps be what was missing to make the exercise more intuitive. This would align with the aforementioned idea that the content of the questions is just as important as the format.

In regards to being interesting, the feedback about this exercise is unfortunately a lot more negative. Less than half of the participants found it interesting, with the rest disagreeing and three even strongly disagreeing. Diving into the comments respondents left, there seems to be some reasoning provided. One participant explicitly states that if the idea is to maximize engagement (what they refer to as “fun”) the quiz “would be better without open questions”, as they find them “boring and time consuming”. Furthermore, the issue of the exam-like feel already identified appears here as well, with the same participant saying that open questions make the quiz “feel the most like a test”. Another agrees, adding that it is difficult to determine an answer’s correctness for this question, necessitating an external person to “judge” the answer, thus boosting the exam-like feel. These findings support the idea that familiar exam-like questions are less engaging, in addition to the idea that the exam-like feeling is a prevalent issue that gets in the way of CodeQuizzer’s goals and engagement levels.

Overall, this type of exercise proved to be successful in aiding understanding and also being generally intuitive and enjoyable to do for the majority. However, it was only the most interesting for one person, with the remainder’s negative opinion probably being caused by the exercise’s familiar nature. This nature also boosted the already identified issue of the quiz feeling like an exam, which is something to be avoided in the future. Nonetheless, it is still an exercise that should be kept moving forward based on majority response, but perhaps used in moderation and with extra attention on the content so that it is not confusing and misleading.

Function flow exercise

The function flow exercise also proved useful in terms of increasing understanding as can be seen in Figure 21. Around 76% of participants indicated that it helped them with this, with two even highlighting this as the most useful exercise in this regard. One answer as to why points out the interactive nature, which fits a person who likes to “learn by doing”, while another claims that the exercise makes them “think about how a function [works]” which in turn provides them with a better overview of the system. Yet another even says that this exercise “taught [them] the right thing”, indicating that doing it was also practically useful for their understanding.

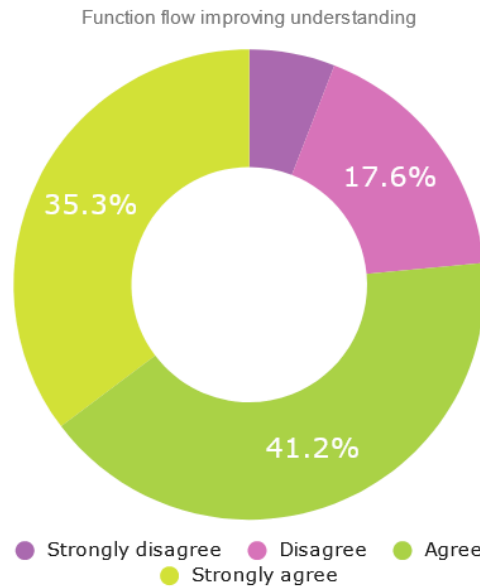


Figure 21. Percentage distribution of the participants' agreement with the statement that the function flow exercise helps improve their understanding of the code presented.

However, despite the positive feedback in general, this time more than 20% disagreed that this exercise was useful for their understanding. Fortunately, a participant left in-depth feedback in regards to this in a comment. Their response has to do with the specifics of the question itself, as their complaints primarily center around the fact that this exercise is not a good representation of nested function calls. This makes the exact return value and type of the functions unclear to them. Similarly it was also unclear where exactly different functions are called from which negatively impacted their understanding further. These are all valid points but they reflect more so on the content of the question rather than the format, similar to some of the feedback the open question received. This once again reaffirms the importance of exercise content for a system like CodeQuizzer.

In addition to the content-related feedback, there were also some more points about this exercise from another participant that did not receive it very well. They claim they felt like they were learning more general coding principles and knowledge with this exercise rather than specifics about the code presented. This response is an outlier, but still something to consider when implementing such questions. Perhaps they need to be presented in a more grounded perspective in order to achieve their true purpose of aiding the understanding of particular codebases.

Turning towards the intuitiveness of this exercise, more than 70% of the participants found it intuitive to do with one person even finding it the most intuitive. While the rest disagreed, no one strongly disagreed indicating this is not an incomprehensible exercise. In regards to why there was disagreement, the previously examined comment puts it best - the content made the question unintuitive and not understandable for some. This is not great, but it does at least support the idea that lack of intuitive understanding of an exercise's content biases the opinion and gets in the way of CodeQuizzer's goals.

In regards to the interest levels here things are slightly more positive, with 75% of the respondents indicating they found the function flow exercise interesting to do. Two participants even highlighted it as the most enjoyable to do. One reason had to do with the design, which was called "very nice", while another claimed that it was "the right amount of challenging", "clear" and helped with the common problem of confusing functions with vague names and interactions. This positive feedback shows that despite being somewhat less intuitive, this exercise was still interesting for most.

Overall, this exercise was received well in terms of interest and aid in understanding. However, it was found not intuitive by some due to its content and particular implementation. Furthermore, it felt too generic for some, although the vast majority liked its interactive nature and positive impact on a common programming problem. Based on this majority positive feedback that is relevant to the overall goal of obtaining understanding this exercise format should continue being used in future iterations, albeit with more carefully curated content and more specificity.

Component diagram exercise

The component diagram exercise was an overall success in terms of increasing understanding of the code presented. As seen in Figure 22, about 88% of participants indicated agreement or even strong agreement in regards to this point, with more than 40% selecting it as the most useful. Looking at the comments left, this good score could be due to the fact that it gives an "overview of the codebase" and helps with understanding "how... different entities can/do interact with each other". This overview aspect was echoed by another respondent, who also positively highlights the focus on "interconnectivity". Moreover, yet another participant says that this is a "good starting point for understanding the codebase", together with another who says this understanding is the most important for "someone starting in a new company or codebase". These statements mirror what the rest said, but also show that the order of

exercises are important. If this were to not be the first exercise, this type of positive feedback may have a lower chance of being received.

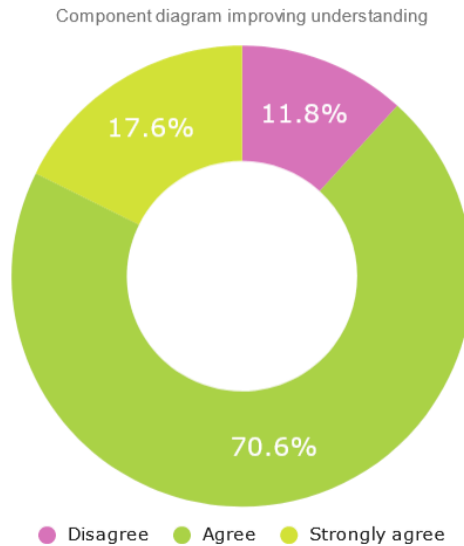


Figure 22. Percentage distribution of the participants' agreement with the statement that the component diagram exercise helps improve their understanding of the code presented.

As far as the two people who disagree are concerned, despite being a minority they provide some useful information in their comments as to why they disagree. Their reasoning is not content based, but rather has to do with the format itself. A participant was “confused”, due to the diagram linking the models themselves rather than the models returned by the function. Adding to this, they claim the diagram could be misinterpreted as linking classes and the return classes of the functions within them, rather than linking the classes themselves. Finally, it was also mentioned that the diagram is not quite as detailed as it needs to be because API calls may abstract connections with other components not present. All of these points are valid and indicate that the exercise needs to be more detailed and clear, perhaps following an already established modeling language such as UML.

In terms of intuitiveness, around 82% of participants also agree that the exercise is intuitive, with the remainder probably having similar reasons as the ones already outlined. Nonetheless, the majority opinion is positive in this regard, so the biases caused by this should be contained to the responses already examined previously which thankfully also provided detailed reasoning and points for improvement.

Most participants also found the exercise interesting to do, with only 2 disagreeing. About 23% even pointed out that this was the most enjoyable exercise for them as seen in Figure 25. Some reasons for this were already highlighted in the first paragraph, but a participant also said it was the “most useful” for them. Another aspect that could play a role here is fatigue, as the first exercise may be most enjoyable and interesting for some, as it is their first interaction with the system and they may have less energy and engagement with every subsequent exercise. This is mostly speculation though and more research needs to be done to confirm or deny this.

To conclude, this exercise was also a success across the understanding, intuitiveness and enjoyment fronts. Participants found it useful for increasing their understanding, but also engaging and intuitive. This was supported by its placement in the quiz, which shows that this is also a factor to consider when using CodeQuizzer. For the respondents with less positive opinions, this seemed to be due to a lack of detailed consideration in the format, which needs to become more robust and in-depth in future iterations of CodeQuizzer. Perhaps a direction to take is to implement an already widespread modeling language such as UML to avoid this sort of confusion.

Functionality altering exercise

The functionality altering exercise was the most well received in terms of helping to understand the code presented as observed in Figure 23. About 94% of respondents indicated that it was useful for them in this regard with more than 40% selecting it as the most useful. The reasoning behind this can once again be found in the comments given. One participant puts it bluntly by saying that “to alter the functionality you have to understand the code the best”, showing why this exercise was effective for them. Furthermore, it was highlighted that this exercise “forces you to work with the code” which is more practical and grounded in comparison to the more abstract and theoretical exercises, leading to better effectiveness. Finally, the comparison aspect of the exercise received praise, with several participants highlighting that they found having to consider a different approach to what is already there useful for their understanding. Furthermore, they mention having to first “think about what the implementation does” and then altering it to be different leading to a “deeper understanding”. Only one person did not find the exercise useful, but they unfortunately did not provide reasoning for this.

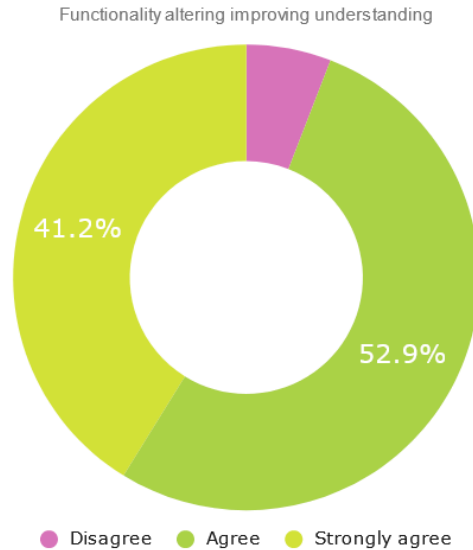


Figure 23. Percentage distribution of the participants' agreement with the statement that the functionality altering exercise helps improve their understanding of the code presented.

When it comes to intuitiveness, the exercise was found intuitive by a vast majority of people as well, which is to be expected as the exercise involves programming - a task most professional software developers should be familiar with. Nonetheless, four respondents disagreed with this, perhaps due to unfamiliarity with the programming language or general lack of experience with programming. It could also be that the task itself was just not as intuitive to some. Regardless, a majority still found it intuitive to do and no one strongly disagreed with this.

Moreover, 82% of participants agree that this exercise type is interesting and enjoyable as well, with about 23% selecting it as the most enjoyable to do as seen in Figure 25. The reasons highlighted in the comments are already covered in the first paragraph and have to do with the consideration of a different perspective. Furthermore, it could be that because this exercise is most similar to the traditional problem solving a developer does in their day-to-day, it is natural that they would enjoy it most.

To conclude, this is the most well-received exercise with overwhelmingly positive scores on its ability to help improve understanding, its intuitiveness and engagement. This is probably because it is most like the usual tasks a developer would encounter in their work, coupled with the interactive nature of the exercise which forces people to think (differently). The only negative feedback could most likely be explained by lack of contextual knowledge about the language or programming in general creating biases

which are hopefully not present in an actual implementation where everyone has a similar level of familiarity and understanding of the technology used in their teams. Overall, functionality altering should definitely be brought into the next iteration of CodeQuizzer with minimal changes.

Variable role exercise

In regards to increasing understanding, the variable roles exercise is unfortunately the least successful. As seen in Figure 24, Less than half of the participants found it useful for this purpose with 23% strongly disagreeing that it was helpful. Despite this, there was one participant who received it well, choosing it as the most useful in terms of increasing understanding (and also as their most enjoyable exercise). Their reasoning provided was that they had never heard of this variable role concept and it made them think about the code in a new way.

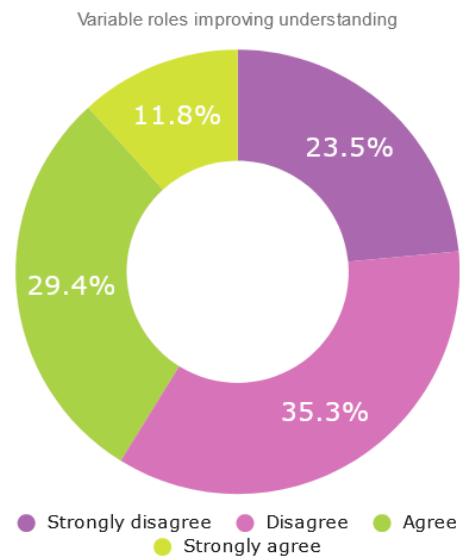


Figure 24. Percentage distribution of the participants' agreement with the statement that the variable roles exercise helps improve their understanding of the code presented.

Nonetheless, for the majority of other participants the exercise failed in its purpose overall. This was already somewhat touched upon in the previous section on the general impressions of the quiz with the visibility of the roles table being a big problem. The comments here expand upon this with one participant explaining that they had no idea the table was there and started selecting roles intuitively rather than based on their definitions. This suggests a big problem with the UI formatting which impacts the reception of the exercise. Aside from these issues, another hindrance was the theoretical

nature of the exercise as indicated in several comments. Some participants simply may not enjoy this more abstract and academic-like task for much the same reasons they may prefer more practical tasks such as the ones in the functionality altering exercise. This of course would impact their opinion for the worse as well.

Moreover, this line of reasoning could also explain why most participants did not find the exercise intuitive to do, with more than half finding it unintuitive to some degree. This could be due to the UI problems and the theoretical nature, but also due to the unfamiliarity with the concept of variable roles. On the other hand, most people found it interesting to do with one even finding it the most enjoyable to do, perhaps owing to the same exact reasons that make it less intuitive for some (new unfamiliar concepts).

To conclude, this type of exercise definitely needs a big rework if it were to be included in future iterations of CodeQuizzer. It needs to be made more interactive and less exam-like, in addition to introducing several UI changes that can make it more intuitive. Even so, it may not be enough for the people who simply do not benefit from theoretical tasks in terms of increasing their understanding. Therefore, it should be used in moderation and only in contexts that are a very good fit for its usage.

Conclusions on exercise types

To conclude this section of the evaluation results analysis, overall every exercise had something to offer for participants. Majority of them were well received in terms of their ability to help with understanding. The opinions that disagreed with this were impacted by a variety of biases that were not foreseen while developing the prototype. These biases are valid and would also be encountered in an actual implementation, therefore, they should be considered moving forward with any new iterations of CodeQuizzer.

The best received exercise type overall was the *functionality altering*, as this one is the most practical and also the one that forces developers to think the most about the existing code, while also interacting with it. The least well received exercise type was the *variable roles* due to its unfamiliar and theoretical nature, as well as due to being hindered by confusing UI. Nonetheless, a vast majority of exercises showed good potential for being interesting, intuitive and helping with understanding. Aside from the data already analyzed per exercise, this is also reflected in the variety of participants' choices for their favorite exercises in these regards, as seen in Figure 25. Moreover, the findings provided plenty of pointers for improvement as already discussed in detail.

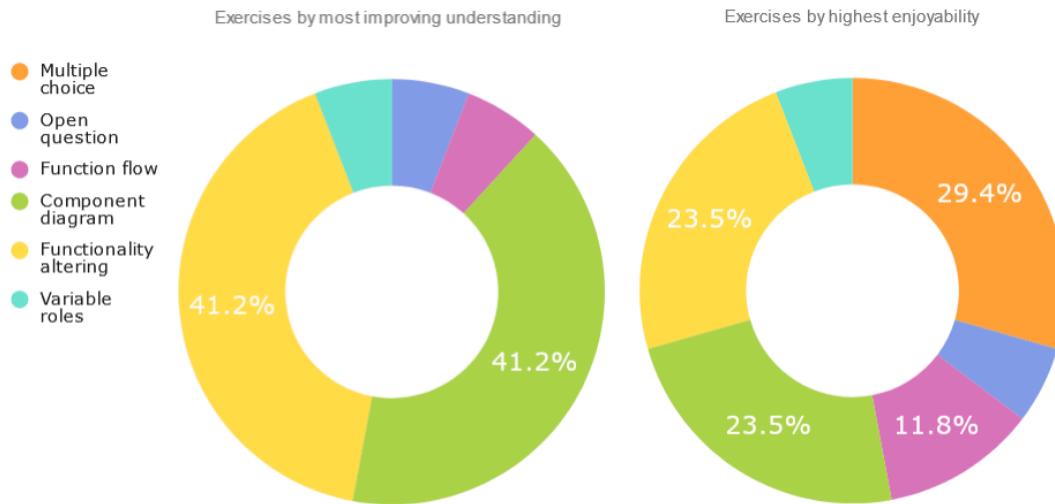


Figure 25. Percentage distribution of participants' choices for exercises that most helped improve their understanding (left) and were most enjoyable to do (right).

Gamification

Having examined the quiz related results in-depth, next it is necessary to focus on the gamification elements in CodeQuizzer and how well they accomplish the tasks they aim to accomplish. This will be done element by element as with the previous sections and then a conclusion on the overall picture will also be presented.

Quiz progress bar

The quiz progress bar was found motivational for completing the quiz by all but one participant. Around 70% of them indicated a firm or strong level of agreement with this statement as seen in Figure 26. One person even mentioned in a comment that while they did not enjoy the last exercise at all, they still completed the quiz because the progress bar showed them that there were no more left, so they felt motivated to finish.

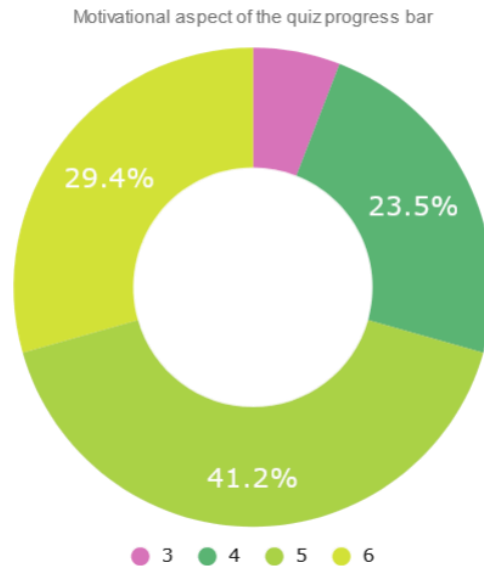


Figure 26. Percentage distribution of the participants' opinions on how much the quiz progress bar motivates them to complete a quiz (1 being "Not at all", 6 being "Very much").

Furthermore, a little under 75% of respondents suggested that if such a progress bar would be displayed next to an unfinished quiz, this would motivate them to go back and complete it, with more than half of those being firm or strong in their agreement. The ones who disagreed did not indicate why, so perhaps they are just not as motivated by such elements.

Aside from the motivational aspect, there were some other positive words left in participants' comments. Some found it "intuitive" and enjoyed the fact they could know how far in the quiz they currently are, how many total exercises there are and how much is left of the quiz. This indicates that aside from being motivating, the bar is also practically useful.

The points of improvement indicated were mostly related to the UI implementation. Some participants wanted to be able to navigate to a particular exercise by clicking on it in the progress bar, while others were slightly annoyed by the animation the bar uses. Furthermore, one participant suggested that a question that was answered incorrectly should have an "X" mark on the bar to indicate this, although implementing this could place the focus more on success rather than obtaining understanding.

Overall, despite the small UI/UX improvements, the progress bar is successful in achieving the goal of motivating users to complete the quiz they started and could even

be motivational to make them go back to a quiz if it were to be displayed next to it in a future iteration of CodeQuizzer. Therefore, this element is successful and should be kept around in the future.

Quiz completion message

The quiz completion message users receive at the end was also received positively. More than 90% of respondents were satisfied with the positive feedback and 35% indicated the maximum level of satisfaction possible. Only 1 person somewhat disagreed on this point, saying that because they had no way to fail the screen was not as satisfying as it could have been.

In regards to how motivational the screen is for doing another quiz, the opinion was a bit more divisive, but still no one found it totally demotivational and almost 60% indicated that it would be motivational for them to do another quiz as can be observed in Figure 27. It was suggested in the comments that this could be increased by collecting points along the way or by drawing inspiration from other learning systems such as Duolingo. Expanding the latter further, it was suggested that a streak-like system for doing consecutive quizzes could be implemented as a message on this screen to further boost motivation. Moreover, in general there is a need for more statistics on this screen, as one person found the “Level 3 comprehension achieved” message meaningless without the additional context such statistics could provide on “how well [they’re] doing”.

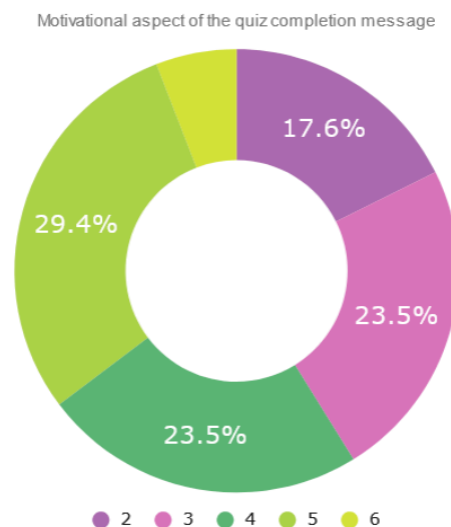


Figure 27. Percentage distribution of the participants' opinions on how much the quiz completion message would motivate them to do another quiz (1 being “Not at all”, 6 being “Very much”).

In regards to the visual elements present on the screen itself, no participant wanted fewer elements with around 70% being completely happy with the amount present leaving feedback such as “It’s good and is placed just the right position”. The rest wished there were more elements with confetti being suggested as an example of what else to include in addition to the aforementioned elements already covered. One person was a bit lost in the layout, saying it took them a while to find the back to home button, so this could be emphasized more as well.

Overall, the quiz completion message did its job in regards to providing the right amount of positive feedback for most respondents. It was also motivational for most, but could be improved even more by drawing more inspiration from existing learning tools. In essence, the gamification elements in regards to positive feedback were effective, but participants want even more of them to be present in the system and the screen itself.

Profile badges

When it comes to motivation, the profile badges also had a positive impact, with almost 70% of participants finding them motivational to do more quizzes. As seen in Figure 28 however, 35% of the total only indicated a slight level of motivation, which leaves room for improvement. For example, the satisfaction level in regards to the names used was only a 4 out of 6, with some participants citing the fact they have no idea what the names mean as an issue, in particular about the “Clever Quizzer” badge. Others said they would like to see why and when they were received and more than 70% wanted to see icons in some form, with about half wanting them instead of the names. Furthermore, another interesting suggestion to improve motivation was to more proactively “nudge” people to obtain these badges, similar to Duolingo.

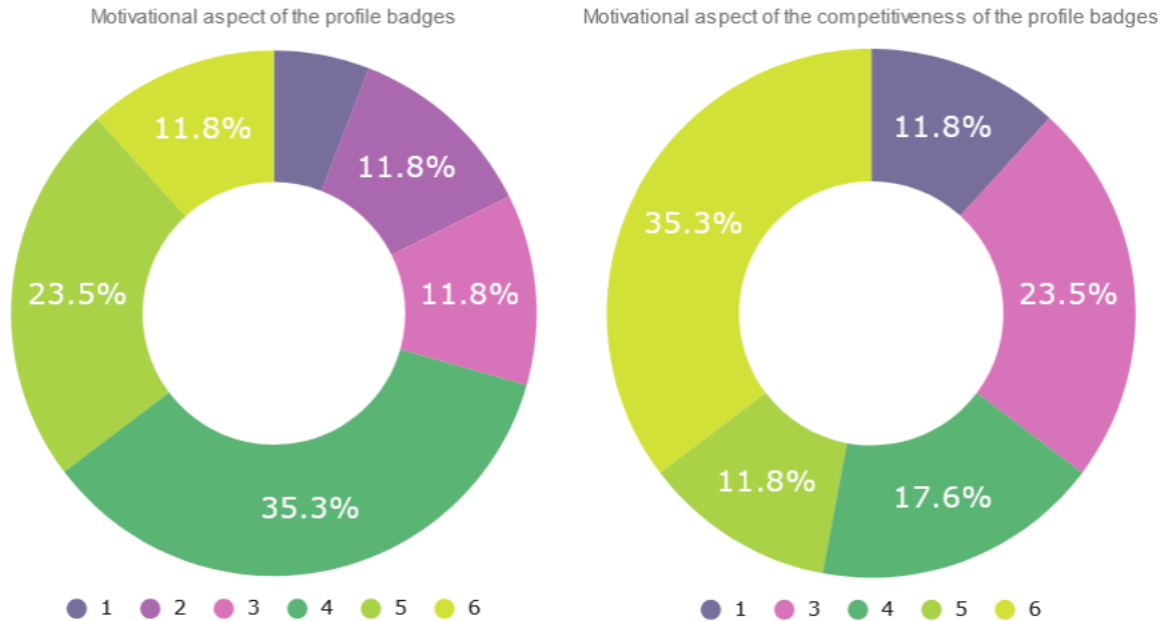


Figure 28. Percentage distribution of the participants' opinions on how motivational obtaining profile badges would be to do more quizzes (left) and how much seeing badges they do not have on other users' profiles (right) would be (1 being "Not at all", 6 being "Very much").

Next, when it comes to the competitive aspect of the badges, most people found that seeing badges on others' profiles would motivate them to do more quizzes to obtain them as also observed in Figure 28. Around 35% did not, however, and this is most likely because some are more motivated by competitiveness than others. This is supported by the comments left, as one said that it added a sense of "achievement and competitiveness", whereas another indicated that seeing badges may even make them "sad" rather than motivated. Moreover, another point made in the comments was that obtaining badges could become a goal in and of itself, overshadowing the idea of increasing understanding, because developers may do the quizzes just in pursuit of badges. Regardless of these points, the motivated respondents indicated a strong level of motivation and only 2 people strongly disagreed with the rest being more mild in their takes on this. This indicates that the badges have a more positive than negative effect on motivation in general.

In conclusion, while the competitive nature of the badges was somewhat divisive, the badges overall prove to be motivational, but could be expanded upon in the next iteration. Users need to feel like they have more value than they currently do and this could be achieved passively by adding more detail to them via icons and additional information or actively by urging users to obtain badges. Furthermore, there should be

extra care when bringing them into the next iteration of CodeQuizzer so they do not overshadow the bigger goal of increasing understanding for developers.

Comprehension levels

When it comes to the comprehension levels, they were found rather motivational by respondents as seen in Figure 29. Only about 11% indicated that they would not be motivated to do quizzes to increase their level and even so it was only a slight indication. The remainder agree that these levels are at least somewhat if not very motivational. This good feedback to receive as these comprehension levels are the backbone of the gamification in CodeQuizzer, so if they function as intended, then most of the system also does in terms of motivating users to increase their understanding via doing quizzes.

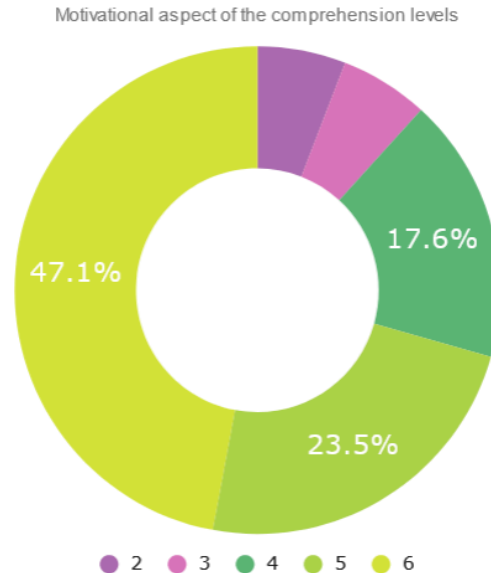


Figure 29. Percentage distribution of the participants' opinions on how much obtaining a higher comprehension level would motivate them to do quizzes (1 being "Not at all", 6 being "Very much").

What's more, comprehension levels are the primary way to link the CodeQuizzer environment to external systems where code review is being done. When it comes to this, participants are almost unanimously in agreement that comprehension levels would be useful when assigning reviewers to a change submitted. Approximately 87% agree that a higher comprehension level of the codebase would make them more likely to assign a developer for a review. Furthermore, about 88% agree or strongly agree that the idea of

assigning a low and high comprehension level is good for knowledge sharing within the team.

Additionally to the numerical data, the comments also indicate positive feedback in regards to this. The idea of assigning reviewers based on comprehension levels “has merit” according to one, whereas the idea of assigning reviewers of different levels is deemed beneficial for learning by another. Furthermore, the levels are seen as useful in another way - mainly to get a quick indication of how “new” to a project someone is, so that communication can go smoother and expectations can be set as to what level of understanding one has.

However, despite this external integration being received overwhelmingly positively by respondents, some of them have various concerns in regards to feasibility. As the CodeQuizzer prototype does not illustrate exactly how it would be integrated with external systems or how representative the comprehension level would actually be of a developer’s understanding, this naturally raises questions in participants’ feedback. One outright says that as implemented now the comprehension level has “no value”, showing that they are uncertain how well it could actually represent the understanding one has. This is supported by further comments that say that measuring such a level would be “hard”, especially in regards to team members that already have this understanding and “are very deep into the code bases but don't want to do the quizzes as it would be ‘a waste of time’”. More tangentially, another point of feedback regarding feasibility is how much time and effort tech leads would be willing to invest into this tool. These are all valid points and great starting questions for further research, however, they fall outside the scope of this paper, so answering them will be left for the future.

Overall, the comprehension levels are a success in terms of being motivational for respondents. Obtaining a higher level is deemed desirable, thus, it could serve as an effective motivator for doing quizzes, which in turn increase understanding. What’s more, there seems to be a lot of approval and excitement for using these levels in a different context with external systems as well, despite the many uncertainties. Said uncertainties are valid and definitely worth exploring further, but in the scope of this research, the answers obtained are satisfactory.

Leaderboard

In regards to motivation obtained via the leaderboard, around 76% of users found it motivational to some degree with about 30% finding it very motivational as observed in

Figure 30. However, around 20% did not and that is most likely due to the fact that not everyone is motivated by competitiveness, similar to the findings regarding badges. The comments left seem to support this theory as some claim that they “don’t like the competitiveness” while others compare it to Duolingo’s leaderboard which they claim they “never look at”. On the other hand some seem to think the idea “has merit” and they “like [it]” and that this is motivational in the same way as the badges but in a more “multiplayer / social context”. Regardless, most still find the leaderboards motivational for doing quizzes.

The same can be said about the competitive aspect of overtaking a colleague on the leaderboard with more than 80% finding this motivational as also seen in Figure 30, albeit less strongly than the general leaderboard concept. Some still have reservations about this though, with one comment claiming that “some people might feel reluctant to overtake their seniors in the leaderboard”.

In regards to the idea of dropping in the leaderboard due to inactivity, respondents seem to find it motivational as well, with 70% indicating that they would do quizzes in order to not drop in ranking as seen in Figure 30. However, one suggests this drop should not happen too quickly so as to not discourage developers from doing quizzes. This is a valid point and a future implementation should consider this problem of balancing as well.

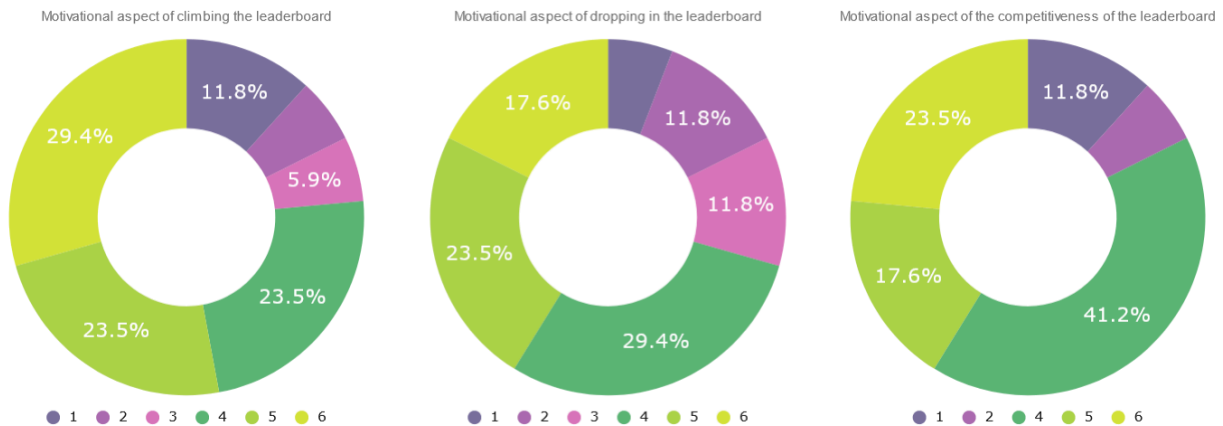


Figure 30. Percentage distribution of the participants’ opinions on how much climbing (left), dropping (middle) or competing (right) in the leaderboard would motivate them to do quizzes (1 being “Not at all”, 6 being “Very much”).

Overall, the leaderboard was divisive in the same way the other competitive gamification elements were, with it resonating with some and not so much with others. Despite this, it

was overall motivational for the group surveyed and the only points of improvement indicated have to do with this polarizing aspect of competitiveness.

Conclusions on gamification

To conclude, it seems like the gamification elements present in CodeQuizzer are a success in terms of motivating users to complete quizzes, but also to do more quizzes. There were some points of feedback and biases present, but nonetheless the numerical data indicates as such. Moreover, in terms of the gamification concept as used here, it seems there is strong indication that it can indeed be used in the way CodeQuizzer intends. In particular, the backbone of the gamification used (i.e. the comprehension levels) received unanimous approval and was seen as a good way to integrate the system into external tools, despite some feasibility concerns that are outside the scope of this research. What's more, the points of feedback that were not UI related all seem to indicate that participants simply want more gamification, such as more value for badges, leaderboard positions and comprehension levels, more positive feedback elements and more concepts such as streaks, bringing CodeQuizzer more in line with other gamification based systems.

Usability

As already outlined, as a final numerical measurement of CodeQuizzer's usability a System Usability Score was calculated. To do this 10 statements about the system's usability were given to participants and they were asked to indicate their level of agreement on a 1 to 5 scale [27]. Using this numerical data, a final score is calculated as a number from 0 to 100. Based on this number the usability of the system can be estimated [27].

As the survey involves several participants, it is necessary to somehow aggregate their responses into one collective response. The approach taken is to calculate the average numerical value per response across all the participants and use those values as a basis for the final score. The way that is calculated is already defined by research and involves subtracting one from the average score of each odd number question and subtracting each even numbered question's value from 5 [28]. This gives a set of final 10 values that are then summed to form a number from 0 to 100 [28].

Having established the way data was aggregated and how the system usability score was calculated, it is time to analyze the results of CodeQuizzer. Using the methods above the system achieved a score of 85 (rounded). This is a great score, as the average for systems

is around 68 [29]. This places CodeQuizzer well above average in terms of usability. Furthermore, taking a deeper look at the score obtained, not only is it above average, but it places firmly in the excellent category of usability, as defined in [29] and seen in Figure 31.



Figure 31. The S.U.S scale shows the level of usability of a system [29].

This high score indicates that CodeQuizzer is a highly usable and well designed system according to participants. This was already touched upon somewhat in the respondent's comments and grades of their experience with the quiz. What's more, participants also mirrored these thoughts in the comments here, saying that “not that any come to mind right now” in regards to additional remarks about usability and saying that “[they] didn't have to think hard about any of the elements once”. All of this adds to the conclusion that CodeQuizzer is a success when it comes to usability, which is also good for adding further validity to the other results obtained, as there should be minimal bias caused by the system's usability.

To conclude, there were still some pointers for improvement mentioned, with participants putting emphasis on the exercise content once again, saying tech leads may need assistance with formulating exercises correctly, especially in regards to the open questions. Furthermore, for participants who did not give the highest possible marks, the variable roles exercise was highlighted as the reasoning. This has already been identified as an issue, but it seems it is large enough to affect the system's overall usability score as well. Nonetheless, these points of feedback are all relatively minor and touch upon already identified issues. Overall, CodeQuizzer proved to be a highly usable system.

Conclusion

This research paper aimed to obtain an answer to two research questions via developing and evaluating the prototype for CodeQuizzer. Before presenting the findings and answers, first a reminder of both research questions is in order. They are as follows:

- *Can gamification be used to motivate developers to gain a better understanding in the context of code review?*
- *How can gamification be used to motivate developers to gain a better understanding in the context of code review?*

In regards to question one, CodeQuizzer was positively received in terms of its ability to aid in obtaining understanding of the code snippets it presents. Furthermore, all gamification elements used (i.e. progress bars, positive feedback, levels, leaderboards and badges) were found motivational by the majority of users in terms of their ability to motivate them to complete the given quiz, but also to do more quizzes in the future. Therefore, due to this positive feedback it can be concluded that gamification can indeed be used to motivate developers to gain a better understanding of their team's codebases. This understanding can then hopefully have a positive impact on the efficiency and quality of their code reviews.

In regards to question two, because CodeQuizzer and its gamification elements were found to be successful in increasing understanding and motivating developers to increase their understanding, it can be concluded that its approach is a viable one in terms of applying gamification as a solution to the challenge presented. In particular, the approach taken is to provide a quiz taking platform where quizzes allow developers to be exposed to, think about and interact with their team's codebases in ways they otherwise may not have. Through this, they obtain a better understanding of the codebases and in turn provide more efficient and higher quality code reviews. The gamification elements included all were able to motivate users to finish the provided quiz, but also to do more quizzes in the future, thus increasing their understanding even further. Based on its positive evaluation, CodeQuizzer itself is an answer to the question of how gamification can be used to motivate developers to gain a better understanding in the context of code review.

In addition to the answers obtained for both research questions, there were also several other interesting and relevant findings. Firstly, it seems like there is approval and

enthusiasm in regards to using the gamified levels of understanding in combination with external systems. In particular, this is in regards to choosing the appropriate reviewers for a given code review. The research shows that developers think it is generally a good idea to apply these levels in their reviewer choosing process either by selecting high level developers to ensure good quality and efficient reviews or by selecting a mix of both high and low level reviewers to facilitate knowledge sharing among the team.

Furthermore, it was found that while CodeQuizzer was generally well received both in terms of increasing understanding and in terms of using gamification to motivate developers to do so, the content of the exercises themselves plays a big role in its effectiveness. This means that not only the system itself, but also the way it is to be used is something worth considering if one were to maximize its ability to aid in obtaining understanding.

Finally, another key takeaway from the results obtained is that a system like CodeQuizzer could benefit even more from emphasizing and expanding its gamification elements and moving even further away from an exam-like structure in order to achieve its goals better. The former would make the motivational elements even stronger and could be achieved by leaning even further into the concepts of gamification and by introducing more of them that are not yet present. The latter would shift the focus towards the idea of obtaining understanding, rather than a “good grade” and could be achieved by introducing more personalized feedback for exercise solutions developers give.

Discussion

Having concluded the research and obtained an answer to both of the research questions that were established, it is also worth discussing this paper’s relevance and knowledge contributions to the domain. Furthermore, it is necessary to provide some reflection on how the research work was executed and what its results imply. This section of the report will do exactly this.

Firstly, when it comes to relevance, as already established code review is one of the essential tasks software developers perform in a professional environment. The various scientific works as well as the expert interviews show that developers expect code review to be something that ensures good quality, understandable, maintainable and scalable code. Aside from these gatekeeping aspects, they also expect it to be a place where knowledge and code ownership can be spread around the team. These are undoubtedly

positives for any software company, so ensuring that the code review process is something that is done in the best way possible is essential. However, many challenges are still faced by developers as shown in the literature and expert interviews. Of these, understanding is the biggest one that is still largely unsolved. Naturally, a solution like CodeQuizzer that attempts to tackle this big challenge while making the essential process of code review more efficient and of higher quality is something that is relevant to the professional domain, but also to the domain of research with a real world impact.

Moreover, gamification elements have been used in education and also in the software industry as established in the background research. They have even been used in regards to improving understanding for code review, but not for specifically gaining an understanding of the codebases surrounding a change presented for review. There seems to be a gap in research regarding this, despite the fact that this particular type of understanding seems well suited to apply gamification to due to similarities with learning tools already using gamification and due to there being several relevant studies that seem to imply that developers lack the motivation to improve their understanding - something that gamification can help improve. This gap in research, combined with the good fit of applying game elements to this particular challenge make the research done here not only a good contribution to the domain, but also a good application of appropriate concepts to a real world challenge.

What's more, in terms of knowledge contribution, as mentioned already there is a gap in research which the work done here can hopefully start to fill to some degree. This makes the knowledge contributed valuable as the positive reception of CodeQuizzer shows that applying gamification to the challenge at hand is a good direction to go in. This can hopefully inspire future researchers to continue trying to apply it in such a way to perhaps even better results. In this way, the knowledge obtained here can serve as a foundation to build up from in the pursuit of making even better systems that apply gamification in even more effective ways and achieve even more positive results.

Finally, on a more personal level this bit of research was fascinating to do as it is a relatively unexplored area and because it involved a plethora of interesting concepts that had not been combined in such a way before. It made it so the results obtained could never be expected and so that many design decisions had to be made and carefully considered in order to ensure the most effective system in CodeQuizzer. Overall, it was a great challenge that produced positive and more importantly - relevant results.

Limitations and future research

No research is unaffected by limitations and the work done here is no exception. Despite the positive results obtained, various limiting factors were present which are important to discuss. Furthermore, it is necessary to establish some recommendations in regards to what can be done next with the conclusions reached in this paper so that the gap in research can continue to be filled to even better effect. This final section of the paper will focus on these two aspects.

Limitations

In terms of limitations, first and foremost is the limiting factor of time. This research project was conducted over the course of about nine months which may seem like a big chunk of time but it is in fact relatively little for research like the one presented. This is because it involves a plethora of tasks and a combination of domains and areas that are not usually combined in such a way. This introduced a lot of uncertainties along the way especially in regards to narrowing down a specific direction of research. Due to the width and depth of the scope of research, having had more time more could have been accomplished which would make the research more sound as well.

Another limiting factor is the researcher's access to data. A master student's connections in the software development industry are relatively limited so naturally the access to participants was as well. Everything that could have been done was done to ensure not only a big number of participants, but also a variety of them in regards to perspectives, but nonetheless there could have been more achieved here. Starting with the number of participants, enough took part for a satisfactory result to be obtained, but if one were to take a statistical approach perhaps it was still not enough to claim statistical significance in this context. Furthermore, most of the participants came from one company where the researcher had the most connections and this could have biased their responses to some degree. Overall, given more time and a better network, higher quantities of varied data could have been gathered, obtaining a better and more significant answer to the research questions.

Finally, still on the topic of participants, while a number of things were done to minimize bias, this can never be completely eliminated. Starting from broad uncontrollable things like the participants' moods while responding to much more specific minutia like what type of wording is most effective for certain people, there were a plethora of design

decisions that could have introduced bias in the responses. This is of course unavoidable and the researcher did the best they could to minimize it and account for it, but bias is nonetheless still present. A future research with more control over environmental factors could help make this less of a limiting factor.

Future research

Based on the limitations described and the results obtained, there are a variety of directions research could go from here. Firstly, in the most broad sense as gamification proved to be effective to tackle the challenge of understanding the codebases surrounding a given change presented for code review, more research can be done in this direction but perhaps with a different approach that may or may not prove to be more effective than CodeQuizzer. Put simply, while CodeQuizzer is *a* way to apply gamification to this challenge it is by far not the only way to do so. Other approaches are also worth exploring in future research.

Secondly, if one were to continue the specific work presented here either with a second iteration of CodeQuizzer or with a spiritual successor system that uses the same concept, the results obtained provide a variety of directions for improvement. The most essential one is the exam feel of the platform. CodeQuizzer very much still feels like an exam for most participants surveyed which is not an inherent problem, but there is enough indication that this comes in the way of the primary goal of improving understanding. It is not yet clear how this could be best addressed but from participants' responses, there is suggestion that having more personalized elements and feedback could shift the focus from obtaining good results to understanding the code at hand. Future research could explore if such an approach is effective and if so - to what degree it is.

Moreover, another way to make CodeQuizzer better at what it tries to do would be to improve the motivational aspects to the gamification included so that users are more likely to do quizzes and improve their understanding. Fortunately, the results obtained provide a direction here too. As established, adding more value to the gamification elements present (in particular the levels, leaderboards and badges) can increase developers' desire to obtain them. Furthermore, additional elements could be introduced, once again per the participants' suggestions such as streaks or more proactive feedback. Introducing these changes to a future iteration and seeing what impact if any they have on user's motivation to do quizzes is another valid direction to investigate in future research.

Finally, while CodeQuizzer proved successful from a developer point of view, in reality developers are not the only ones involved in such a system. The tech leads of a team also play a big part in the system's functionality and maintenance, but their input has been put outside the scope of this paper. Therefore, another direction to explore is what tech leads would like in such a system and how it could provide the best user experience when it comes to setting up quizzes which are effective for boosting understanding. Furthermore, this could also touch upon and explore the aspect of feasibility and integration with third party systems, which was already mentioned as a concern in the previous sections. Overall, there is a plethora of work to be done before this system can be put in practice and it all provides valid and interesting directions for future research.

Appendix

Appendix A. Interview questions used during the expert interviews done as part of this paper's preliminary research phase

Section 1: Context

1. What is your position in your company?
2. How long have you worked there?
3. How long have you worked in this industry?
4. Could you summarize your work experience so far?
5. Can you shortly describe your work experience so far?
6. How old are you?

Section 2: Code review process

7. When was the first time you did code review?
8. When was the last time you did code review?
9. What tools have you used to conduct code reviews?
10. Can you describe (preferably step-by-step) your code review process?
11. How does a code review usually look for you? Please provide a step-by-step description if possible.
12. When you submit a change how in depth is your description of it? Do you expect people to just figure it out by themselves or?

Section 3: Code review motivations and outcomes

13. What do you want to achieve when doing code review?
14. What is your main focus when doing code reviews?
15. Are you looking for bugs most of the time, or do you look for deeper issues as well?
16. How do you go about identifying deeper issues?

Section 4: Code review challenges

17. What is your best experience with doing a code review?
 - a. Why?
18. What is your worst experience with doing a code review?
 - a. Why?
19. How long do you usually spend on code review?
20. What takes you most time when reviewing code?
21. Have you ever put off doing a code review?
 - a. If so, why?
22. How much time do you usually spend on writing feedback for proposed changes?
23. What does your feedback look like usually?
24. Have you ever just approved a change without leaving feedback?
 - a. If so, why?
25. Do you find code review for a project you are less familiar with intimidating or more difficult?
 - a. If so, why?
 - b. If not, what makes it no different?
26. If you could change the way code review is done at your company, what would you change?
27. If you could change one thing about the tools used for code review at your company, what would you change?
28. What things do you find most challenging when doing code review?

Appendix B. Survey questions presented to participants in CodeQuizzer's evaluation survey.

Section 1: Participant context

1. What is your job position?
 - a. Open question.
2. How long have you worked in the software industry?
 - a. Options: Less than a year, 1-3 years, 3-5 years, 5-10 years, 10+ years.
3. How old are you?

4. How many people do you work with in your team (on the same code bases / projects)?
 - a. All open questions.
5. Does your team have a senior position who knows the ins-and-outs of the team's code bases (for example a tech lead)?
 - a. Options: Yes, No.

Section 2: Example quiz

1. How would you rate your experience with the quiz overall from 1 to 6?
2. How satisfied are you with the quiz format?
3. How much would such a quiz help in improving your understanding of a code base?
4. How useful did you find the explanations provided for the correct solutions?
5. How much did the quiz feel like an exam?
 - a. All likert scale questions from 1 to 6.
6. Any other remarks about the quiz in general?
 - a. Open question.

Section 3: Exercise types

1. This type of exercise helps me improve my understanding of the code presented.
2. This type of exercise is intuitive to do.
3. This type of exercise is interesting to do.
 - a. All likert scale statements with separate answers for each exercise type in the range of 1 to 4.
4. Which exercise type helps the most with improving your understanding?
 - a. Options: all the exercise types.
5. Why?
 - a. Open question.
6. Which one is the most enjoyable to do?
 - a. Options: all the exercise types.
7. Why?
8. Any other remarks about any of the types of exercises in particular?
 - a. All open questions.

Section 4: Gamification

1. The quiz has a progress bar at the bottom. How much does this motivate you to complete the quiz?

2. If this bar would be displayed next to an unfinished quiz in your list, how much would this make you want to go back and complete it?
 - a. All likert questions from 1 to 6.
3. Any additional remarks about the quiz progress bar?
 - a. Open question.
4. At the end of the quiz there is a completion message. How satisfied are you from receiving it?
5. How much does it make you want to do another quiz?
 - a. All likert questions from 1 to 6.
6. Would you like more or fewer visual elements on it?
 - a. Options: There are just enough elements, Fewer elements, More elements.
7. Would you like the message to be longer and more positive?
 - a. Options: It's just right, I would like it longer and more positive, I would like it shorter and less positive.
8. Any additional remarks about the quiz completion message?
 - a. Open question.
9. Next, returning to the main page, observe the various badges in your profile section on the left. These are earned through doing quizzes and displayed for others to see. How much would earning these badges motivate you to do quizzes?
10. How much would seeing badges you don't have on other users' profiles motivate you to do quizzes so you can obtain them?
11. How satisfied are you with the example badge names provided?
 - a. All likert questions from 1 to 6.
12. Would you like to see icons instead of / alongside the badge names?
 - a. Options: No, just the names is fine, Yes I would like badge icons instead of names, Yes I would like badge icons together with the names.
13. Any additional remarks about the badges?
 - a. Open question.
14. Next, please observe the code base list again. Next to each code base is your comprehension level. You increase this level by doing quizzes. How much would earning a higher comprehension level motivate you to do quizzes?
15. If the comprehension level was displayed next to a user's name when they are assigned as a code reviewer for a proposed change, how much more likely would you be to select the reviewer with a higher comprehension level?
16. How helpful do you think it would be to assign a low and high level person to a change presented for review in order to facilitate understanding code bases throughout the team?

- a. All likert scale questions from 1 to 6.
17. Any additional remarks about comprehension levels?
 - a. Open question.
18. Finally, please navigate to the leaderboard section. Here you can see the general leaderboard as well as a leaderboard per code base. Doing quizzes improves your place in the leaderboard. How much would getting a higher spot on the leaderboard motivate you to do quizzes?
19. How much would dropping in the leaderboard motivate you to do quizzes to keep your rank?
20. How much would wanting to overtake a colleague's rank in the leaderboard motivate you to do more quizzes?
 - a. All likert scale questions from 1 to 6.
21. Any additional remarks about the leaderboard?
 - a. Open question.

Section 5: Usability

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.
 - a. All likert scale statements from 1 to 5.
11. Do you have any additional remarks about the user experience / interface?
 - a. Open question.

References

- [1] M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal*, vol. 38, no. 2.3, pp. 258–287, 1999, doi: <https://doi.org/10.1147/sj.382.0258>.

- [2] P. C. Rigby and C. Bird, "Convergent contemporary software peer review practices," *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2013*, 2013, doi: <https://doi.org/10.1145/2491411.2491444>.
- [3] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, "Modern code review," *Proceedings of the 40th International Conference on Software Engineering Software Engineering in Practice - ICSE-SEIP '18*, 2018, doi: <https://doi.org/10.1145/3183519.3183525>.
- [4] T. Baum, O. Liskin, K. Niklas and K. Schneider, "A Faceted Classification Scheme for Change-Based Industrial Code Review Processes," *2016 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, Vienna, Austria, 2016, pp. 74-85, doi: 10.1109/QRS.2016.19.
- [5] J. Shimagaki, Y. Kamei, S. Mcintosh, A. E. Hassan and N. Ubayashi, "A Study of the Quality-Impacting Practices of Modern Code Review at Sony Mobile," *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, Austin, TX, USA, 2016, pp. 212-221.
- [6] F. Shull and C. Seaman, "Inspecting the History of Inspections: An Example of Evidence-Based Technology Diffusion," in *IEEE Software*, vol. 25, no. 1, pp. 88-90, Jan.-Feb. 2008, doi: 10.1109/MS.2008.7.
- [7] P.M. Johnson, D. Tjahjono, "Does Every Inspection Really Need a Meeting?" *Empirical Software Engineering*, vol. 3, pp. 9-35, Mar 1998, doi: <https://doi.org/10.1023/A:1009787822215>.
- [8] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in *2013 35th International Conference on Software Engineering (ICSE)*, San Francisco, CA, USA, 2013, pp. 712-721, doi: 10.1109/ICSE.2013.6606617.
- [9] T. Baum, K. Schneider. "On the Need for a New Generation of Code Review Tools," in *Product-Focused Software Process Improvement*, Trondheim, Norway, 2016. Pp. 301-308, doi: https://doi.org/10.1007/978-3-319-49094-6_19.
- [10] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida and K. -i. Matsumoto, "Who should review my code? A file location-based code-reviewer recommendation approach for Modern Code Review," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Montreal, QC, Canada, 2015, pp. 141-150, doi: 10.1109/SANER.2015.7081824.
- [11] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation," in *2013 35th International Conference on Software Engineering (ICSE)*, San Francisco, CA, USA, 2013, pp. 931-940, doi: 10.1109/ICSE.2013.6606642.

- [12] S. Kollanus, J. Koskinen, "Software Inspections in Practice: Six Case Studies," in *Product-Focused Software Process Improvement*, Amsterdam, The Netherlands, 2006, pp. 377-382, doi: https://doi.org/10.1007/11767718_31.
- [13] S. Deterding, R. Khaled, L. Nacke, D. Dixon, "Gamification: Toward a definition," in *CHI 2011 Gamification Workshop Proceedings*, Vancouver, Canada, 2011, pp. 12-15.
- [14] S. Khandelwal, S.K. Sripada, R. Reddy, "Impact of Gamification on Code review process: An Experimental Study," in *10th Innovations in Software Engineering Conference*, Jaipur, India, 2017, pp. 122-126, doi: 10.1145/3021460.3021474.
- [15] C. Crumlish, E. Malone, *Designing Social Interfaces: Principles, Patterns, and Practices for Improving the User Experience*. Sebastopol, CA: O'Reilly, 2009.
- [16] M.H.A. Rahman, I.Y. Panessai, N.A.Z.M Noor, N.S.M Salleh, "GAMIFICATION ELEMENTS AND THEIR IMPACTS ON TEACHING AND LEARNING - A REVIEW," *The International journal of Multimedia & Its Applications*, vol. 10, No. 6, pp. 37-46, Dec 2018.
- [17] M.R.A. Souza, L. Veado, R.T. Moreira, E. Figueiredo, H. Costa, "A systematic mapping study on game-related methods for software engineering education," *Information and Software Technology*, vol. 95, pp. 201-218, Mar 2018, doi: <https://doi.org/10.1016/j.infsof.2017.09.014>.
- [18] J. Grenning, "Planning Poker Planning Poker or How to avoid analysis paralysis while release planning," *Hawthorn Woods: Renaissance Software Consulting*, vol. 3, 2002.
- [19] D. Silva, M. Lencastre, J. Pimentel, J. Castro, L. Lira, "Applying Gamification to Prioritize Requirements in Agile Projects," in *SAC '23: Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, New York, NY, USA, 2023, pp. 1498-1507, doi: <https://doi.org/10.1145/3555776.3577708>.
- [20] A. Mora, P. Zaharias, C. González, J. Arnedo-Moreno, "FRAGGLE: A Framework for Agile Gamification of Learning Experiences," in *Games and Learning Alliance (GALA 2015)*, Rome, Italy, 2015, pp. 530-539, doi: https://doi.org/10.1007/978-3-319-40216-1_57.
- [21] S. Arai, K. Sakamoto, H. Washizaki, Y. Fukazawa, "A Gamified Tool for Motivating Developers to Remove Warnings of Bug Pattern Tools," in *6th International Workshop on Empirical Software Engineering in Practice (IWESEP 2014)*, Osaka, Japan, 2014, pp. 37-43, doi: 10.1109/IWESEP.2014.17.
- [22] C. Prause and M. Jarke, "Gamification for enforcing coding conventions," in *10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Bergamo, Italy, 2015, pp. 649-660, doi: 10.1145/2786805.2786806.
- [23] C. Prause, J. Nonnen, M. Vinkovits, "A Field Experiment on Gamification of Code Quality in Agile Development," in *Annual Workshop of the Psychology of Programming Interest Group*, London, United Kingdom, 2012.

- [24] F. Hermans, *The Programmer's Brain*. Shelter Island, NY: Simon and Schuster, 2021.
- [25] J. Sajaniemi and M. Kuittinen, "An Experiment on Using Roles of Variables in Teaching Introductory Programming," *Computer Science Education*, vol. 15, no. 1, pp. 59–82, Mar 2005, doi: <https://doi.org/10.1080/08993400500056563>.
- [26] Google. (2023). *Material Design 3* [Online]. Available: <https://m3.material.io/>.
- [27] Usability.gov. (2019). *System Usability Scale (SUS)* | Usability.gov [Online]. Available: <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>.
- [28] J. Sauro. (2011). *Measuring Usability with the System Usability Scale (SUS)* [Online]. Available: <https://measuringu.com/sus/>.
- [29] M. Aubagna. (2021, Feb. 24). *How to Use the System Usability Scale (SUS) to Measure User Experience* [Online]. Available: <https://skeepers.io/en/blog/system-usability-scale-sus-user-experience/>.