Let`s Make it Fun: Gamifying and Formalizing Code Review

Naomi Unkelos-Shpigel and Irit Hadar
Department of Information Systems, University of Haifa, Haifa, Israel

Keywords: Code Review, Formal Method, Collaboration, Gamification.

Abstract: Code review is a highly important task in the software development lifecycle. However, some of the characteristics of code review hinder practitioners' performance of this task. Code review is considered to be tedious and uninteresting, and includes challenging human aspects, such as collaboration among stakeholders. Despite the many concerns that need to be taken into consideration when performing code review, a comprehensive, formal definition thereof is yet to be determined. In a previous research, a set of formal guidelines for code review was presented, in the context of performing this task in a gamified environment. In this ongoing research, we explore whether the field of software engineering provides a formal definition for code review, and whether a formal definition is needed. The preliminary findings of this research indicate that while the field does provide several definitions for code review, in all that concerns the human aspect of this task, a formal definition is in order. As a response for this need, we present a framework of the task of code review toward its formalization, embedding gamification for motivation enhancement.

1 INTRODUCTION

Code review has been long known as highly important for ensuring software quality (Fagan, 1967). Performing code review is typically perceived as a tedious, undesired task, which presents several challenges to the required collaboration and knowledge transfer between reviewers and programmers. As such, this task has the potential to benefit from motivation enhancement strategies. In recent years, gamification has been used in various tasks in order to motivate participants to take part in the task, and to enhance the quality of the process and products (Minelli et al., 2015). Lately, several attempts have been made for implementing this approach in the context of software engineering (Marshburn and Henry, 2013). Gamification is defined as "the integration of game mechanics in non-game environments to increase audience engagement, loyalty and fun" (Deterding et al., 2011). In order to use game elements correctly to enhance a process, a deep understanding of this process and motivation factors of participants is in order.

The aim of this ongoing research is to develop a formal framework for the code review process, with gamification elements embedded in the process, in order to motivate practitioners to participate in, and significantly contribute to peer code review.

2 LITERATURE REVIEW

2.1 Code Review

Defined by Fagan (1967), code review includes all manual line-to-line inspections, also called code inspection or code scrutiny. Research on code review aims to achieve a shorter and more effective review process, including developing tools for monitoring code review, its risks and challenges (Porter et al., 1995). These risks and challenges include insufficient collaboration between programmers and code reviewers, and gaps in the required shared understanding of the purpose of the code review (Bacchelli and Bird, 2013). Relevant strategies were offered for supporting code review, e.g., automating the code review process [ibid], and a tool enabling programmers to track significant code changes during code reviews (Zhang et al., 2014). However, these solutions refer only to the artefacts of code review; they do not encourage reviewers or programmers to participate in the code review or distribute lessons learned to other programmers and reviewers in the firm.

Several attempts took place in order to focus code review on deeper and more continuous inspection of the artifact (Farchi and Ur, 2008). The notion of having a homeworkless process, where the reviewer can focus on the quality of the outcomes rather than on searching for micro-defects in the code, represents

Unkelos-Shpigel, N. and Hadar, I.
Let's Make it Fun: Gamifying and Formalizing Code Review.
In *Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering (ENASE 2016)*, pages 391-395
ISBN: 978-989-758-189-2

a modern approach in which the role of code reviewers is grasped as quality assurance managers rather than bug detectors (ibid). Additional reinforcement to this approach can be found in recent research works, for example the research conducted by (Bacchelli and Bird, 2013), where practitioners were asked on how they perceived the code review process. The main findings suggest that practitioners see code review as a way to enhance the quality of code and transfer knowledge, rather than just fixing minor code faults.

Another important finding is that according to practitioners, performing code review promotes team awareness to code quality, and transparency of the coding process. Most importantly, code review helps all the involved parties to feel shared ownership of the code; workers and managers feel that the code is being examined by an expert on a regular basis, which helps them to be less protective about their code (ibid). Practitioners are encouraged to use tools that perform automated code review in which the minor bugs and faults are found, for static code review, and to perform peer code review for better understanding and shared ownership of the code. However, there is no standard or conventional guideline for performing this type of code review.

Our research is aimed to understand how code review can be formalized and performed via shared tools, for encouraging practitioners to take part in collaborative code review and ensuring that all parties have an accurate and complete understanding of their role in this process.

2.2 Gamification

Gamification is aimed at increasing enjoyment of tasks by integrating game mechanics in non-game environments. This has been proven to increase engagement, loyalty and fun (Deterding et al., 2011). Gamification of computer-supported applications addresses the use of techniques taken from games in order to encourage users' active participation and contribution. In recent years, various gamification elements have been embedded in different information systems and applications in general, and in some cases, in applications intended for the use of software engineers in particular.

Gamification was used, for example, to encourage students into doing software testing, in a system called "Secret Ninja Testing," (Bell et al. 2011), where students were presented with quests using characters from various action movies, and were asked to act as these characters while solving testing problems. They reported that the system helped the students to be exposed to the complete lifecycle of software development, and encouraged students to choose software engineering as a major in their studies. An effort to encourage students to use version control was also made using gamification, where a social software application was used, mainly using the notification feature (Singer and Schneider, 2012). The researchers reported that using the social features was helpful for many students in achieving an overall understanding of their project.

Research was also conducted in the context of using gamification starting at early stages of software development. Dubois and Tamburrelli (2013) identified three types of activities needed to be performed when engaging gamification into software engineering: analysis, integration, and evaluation, and found that students performing these activities had better results in software engineering. Another research showed that using gamification in virtual teams during requirement elicitation assisted the teams to locate experts and share their knowledge (Marshburn and Henry, 2013).

Recently, gamification has also been used in practice, for example, to praise software developers when the code they wrote was productive (Minelli et al., 2015), and to encourage practitioners to practice white box testing (Xie et al., 2015). In agile development, gamification was used to encourage the use of code conventions (Prause and Jarke, 2015).

3 AN EXPLORATORY INQUIRY ON CODE REVIEW PRACTICES

As a preliminary evaluation of the motivation for the research, we posted a set of two questions in several professional LinkedIn groups, in order to understand how practitioners are guided to perform code review in industry. The questions were: "Do you have any defined procedures or instructions on your code review process? Is it the same in all teams? Please elaborate."

Twenty-two software practitioners responded to these questions, providing interesting insights on the code review process. Their answers reflected several perceptions about code review, presented here with some examples from their original quotes:

- Code review involves using static (automatic) code analysis, for detecting simple bugs and faults:

"Stash [an automatic tool] gives us an audit trail of code reviews. This also means that we can

guarantee that all the code that makes it to production has been reviewed. However, I'm not convinced that our code review process necessarily makes our code 'better'."

"The usage of the automatic code-review [static code analysis] tools are very helpful for performing tedious code-reviewing like code styling, coding rules or even checking known best/anti practices."

- Code review should include experts performing code review:

"Not everyone can be a reviewer, it's a team decision."

"The single most important aspect is who runs the review, and how they do it. It's a learnable skill."

"Our [code review] process also means delayed integration because code sits on a branch waiting to be reviewed. This causes a whole bunch of other issues."

The latter quote demonstrates situations in which these experts are not immediately available to perform the code review.

- The expected contribution of peer code review:

"[Using automatic tools for code review as part of the process] frees up the [peer] code review to be more about what you are trying to achieve rather than are you using camel case or not, or whatever other rules you have set up."

"Your reviewers are more interested in ‘is this the right way and place to provide the solution’ rather than have you coded correctly."

- Various types of reviews:

*"The procedure or process on how to do the code review **vary a lot**, from ad hoc reviews to very formal and heavy process."*

"Our group pair programs, so we don't do a lot of formal reviews. We do play a lot of code ping pong while pairing."

To conclude, the respondents indicated the importance of code review, including both static and peer review. However, in all their answers, they indicated they do not have a formal procedure for performing peer code review, but rather only informal work instructions. In cases of pair programming, the code review is not considered as such, but a certain type of peer review does in fact take place. In some other cases, the participants indicated that code that needs to be reviewed, sometimes gets stuck, waiting for review.

The preliminary study was insightful, as it indicated that indeed there is a need for a formal definition of the process, in all the aspects that involve the interaction between the programmer of the code (who asks for a review), the reviewer, and the additional practitioners in the firm, who could benefit from receiving information and lessons learned from the review.

4 A FORMAL DEFINITION OF CODE REVIEW

Building on the basic definition for gamifying code review presented in (Unkelos-Shpigel and Hadar, 2015), the following gamification includes the steps – Create, Ask for review, Review, Extend knowledge (CARE). All the participants can create code to be reviewed, send it for review, receive the review, and finally, choose whether they want to contribute the information from the review and lessons learned to others in the firm.

We differ between novices and experts in the gamified process, as experts are less motivated to participate in the code review process - as they contribute knowledge rather than consuming it.The game follows these sequential rules, according to the rules of flow and group flow (ibid):

1. Each novice programmer is assigned with an initial score of zero. The reviewer – an expert programmer – is assigned with a higher initial score.
2. In addition to the individual scores, there is also a team score managed, which is updated according to the individually rewarded tasks.
3. When the code is ready, the programmers ask for a review, and are immediately rewarded with points.
4. The reviewer reviews the relevant segment of the code. If the reviewer approves the code, she is granted with points as well. Additional score is given for writing a review, which helps the programmer to improve the code. For bug detection, the reviewer will be rewarded extra points for each bug found.
5. The reviewers can also choose to share their review comments with members of other teams, raising both individual and team score. An additional mechanism is needed to evaluate the quality of the shared information, and its contribution to other stakeholders in the project.
6. The programmer can share tips and lessons learned from the review with other programmers as well, raising both individual and team score.

7. The programmers are also given badges according to their individual scores. The badge indicates their level in the game, labelled kilo, mega, or giga, etc., according to the number of points they earned.
8. Each team has its own profile, where all members of the team can view information about the team score and their relative ranking among all teams. The teams are rewarded each month according to their scores. The reward can be in the form of monetary incentive or other rewards (e.g., breakfast with a high management representative or coupons for fun activities).
9. If other programmers or reviewers use the know-knowledge and tips shared, the individual who wrote and/or shared this knowledge gets additional points.

The main actions in the code review process are illustrated in Figure 1.We modeled the process in BPMN, since we address code review as a business process.

5 SUMMARY AND FUTURE RESEARCH STEPS

In this ongoing research, we develop a collaborative gamified framework for performing code review. We discovered in our preliminary exploration that peer review is indeed performed in practice, but has no formally defined or even agreed upon process. We used game mechanisms and embedded them in the process so to create a collaborative framework and enhanced individual and team motivation, where code is written and substantially reviewed, later enabling to distribute to others the knowledge created in this process.

In the next research steps we intend to perform interviews and distribute questionnaires among developers. During the research and the evolution of the gamified framework and environment, we will approach additional developers, including from virtual social networks such as designated groups in LinkedIn. We plan to elicit their perceptions about our prototype, and their opinions about its potential effect on their performance, user satisfaction, and additional measures. Finally, we will implement our proposed solution in a case study in order to measure the actual behavioural change resulting from working with the defined code review process and the gamified environment.

REFERENCES

Bacchelli A., and Bird C. 2013. Expectations, outcomes, and challenges of modern code review, In *Proceedings of the 2013 International Conference on Software Engineering, pp. 712-721. IEEE Press*.

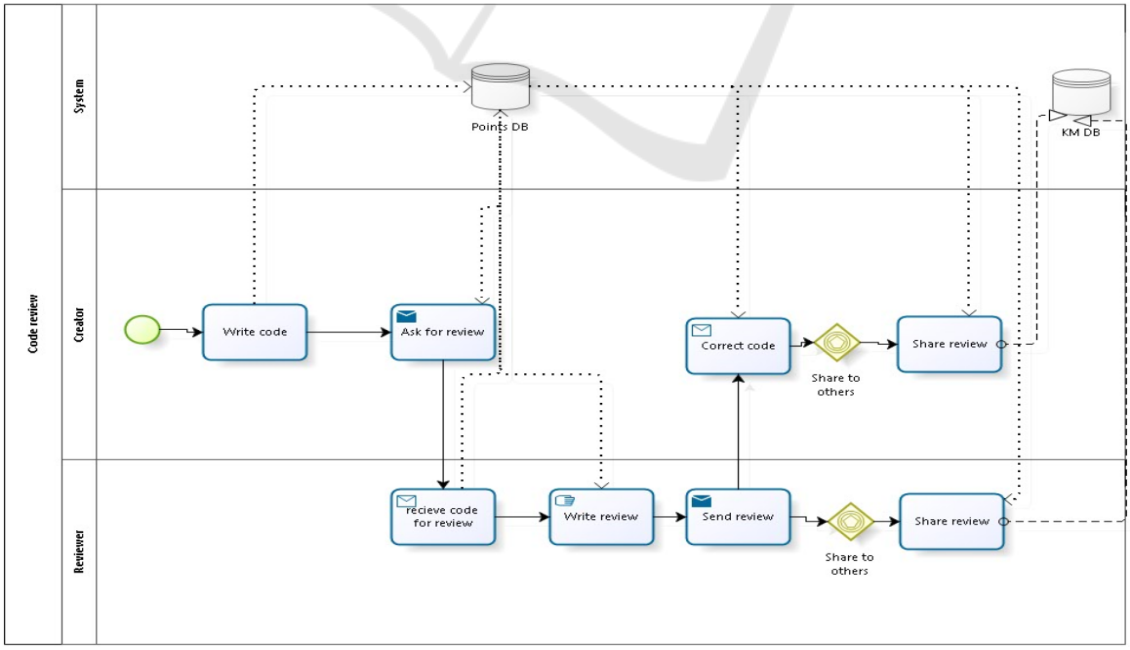

Figure 1: BPMN specification of the code review process.

Bell, J., Sheth, S., and Kaiser, G. 2011. Secret ninja testing with HALO software engineering. In *Proceedings of the 4th international workshop on Social software engineering, ACM*, pp. 43-47.

Deterding, S., Khaled, R., Nacke, L., and Dixon, D. 2011. Gamification: Toward a Definition. In *CHI 2011 gamification Workshop Proceedings*, pp.12-15.

Dubois, D. J., and Tamburrelli, G. 2013. Understanding Gamification Mechanisms for Software Development. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, ACM*, pp. 659-662.

Fagan, M. 1967. Design and code inspections to reduce errors in *program development, IBM Systems Journal*, 15(3), pp.182–211.

Farchi, E., and Ur, S. 2008. Selective Homeworkless Reviews. In *Software Testing, Verification, and Validation, 2008 1st International Conference on*, pp. 404-413. *IEEE*.

Hadar, I. 2013. When Intuition and Logic Clash: The Case of the Object Oriented Paradigm, Science of Computer Programming, 78, pp. 1407-1426.

Marshburn, D. G., and Henry, R. M. 2013. Improving Knowledge Coordination in Early Stages Of Software Development Using Gamification. In *Proceedings of The Southern Association For Information Systems Conference*. Savannah, Ga, USA.

Minelli, R., Mocci, A. and Lanza, M., 2015, May. Free hugs: praising developers for their actions. *In Proceedings of the 37th International Conference on Software Engineering,* 2, pp.555-558. IEEE press.

Porter, A. A., Votta Jr, L. G., and Basili, V. R. 1995. Comparing detection methods for software requirements inspections: A replicated experiment. Software Engineering. In *IEEE Transactions on*, 21(6), pp. 563-575.

Prause, C. R., and Jarke, M. 2015, August. Gamification for enforcing coding conventions. *In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. pp. 649-660. ACM.

Singer, L. and Schneider, K., 2012, June. It was a bit of a race: Gamification of version control. *In Games and Software Engineering (GAS), 2012 2nd International Workshop on*, pp. 5-8. IEEE.

Unkelos-Shpigel N. and Hadar, I. 2015. Gamifying Software Development Environments Using Cognitive Principles: The Case of Code Review. *8th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE 2015).*

Xie, T., Bishop, J., Horspool, R. N., Tillmann, N., and De Halleux, J.2015. CrowdSourcing code and process via code hunt. *In CrowdSourcing in Software Engineering (CSI-SE), 2015 IEEE/ACM 2nd International Workshop on* (pp. 15-16). IEEE.

Zhang, T. Song M. and Kim, M. 2014. Critics: an interactive code review tool for searching and inspecting systematic changes. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp. 755-758, *ACM*.