CodeArena: Inspecting and Improving Code Quality Metrics in Java using Minecraft

Simon Baars
University of Amsterdam
Amsterdam, Netherlands
simon.mailadres@gmail.com

Sander Meester
University of Amsterdam
Amsterdam, Netherlands
sander.meester@student.uva.nl

I. Introduction

Technical debt causes the cost of maintaining a piece a code to rise with it, resulting in more time spend on implementing changes later. Studies show that over 80% of the software development process is spent on software maintenance [6].

There are several factors that influence the maintainability of a codebase, such as code duplication and high code complexity. Various tools and techniques are available that help getting insight into and/or tackling these factors [9, 5, 7]. The functionality of such tools varies from merely informing about the current state of the code, to tools that assist developers in refactoring the code. However, most of these tools do not assist the developer in understanding the causes of factors that influence maintainability. The field of software maintenance and code quality is therefore difficult to make intuitive for developers.

We propose a solution that lets the developer interact with code that does not conform to quality guidelines [4] to gain awareness of what good quality is and how to minimise technical debt.

II. Proposed solution

To make developers aware of harmful coding practises and how they can improve their code, we created CodeArena [1] [2]. CodeArena is an extension to the popular 3D sandbox game called Minecraft. It allows developers to experience the quality of their code and gain progressive insight in the causes of hard-to-maintain code. This tool translates features of a codebase that are considered harmful to monsters in Minecraft, which can then be ”fought” to improve the codebase. Fighting the monsters will trace the user back to the source code. If the developer succeeds in solving the issue, the monster will die and the developer will be rewarded in-game. This way, the developer can gradually improve the quality of the code, while learning about code quality in an engaging way.

The images below show the in-game view of the tool. In figure 1 we see the player in the arena containing the spiders that represent code clones. Striking one of those spiders opens up the editor window as shown in figure 2, which shows the clone group the spiders represents. This clone can then immediately be refactored, resulting in the code quality being improved.

The developer can traverse different components of the codebase and gain experience points for each metric that has been improved. Points will be subtracted if the metrics are worsened after the developer changes the code. The metrics to be improved consist of code duplication, unit complexity, unit size and unit interface size [4]. Each of these metrics correspond to a different type of monster.

The Java project to be improved can be chosen from disk, or a GitHub URL can be specified to download a project from. After the project has been chosen, the tool will create a monster for each code issue found. The user will have to fix code issues in order to collect experience points.

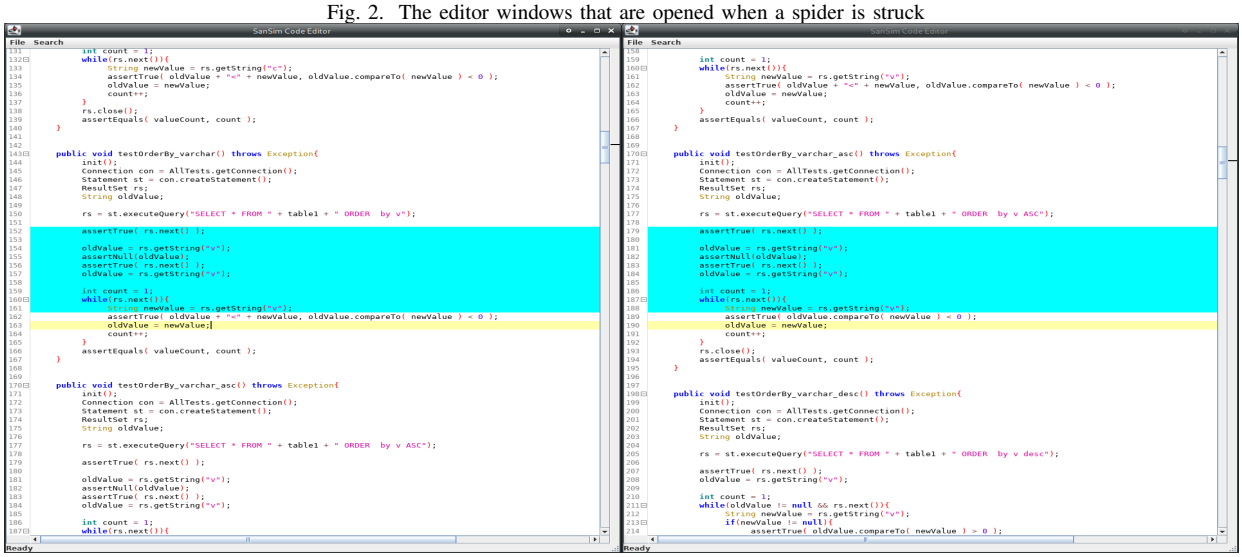
Fig. 1. Player’s view of clones represented as spiders

A. Technical architecture

Our tool is made using Minecraft Forge [2], which provides an API to extend Minecraft using Java code. In the Java code we scan a folder on the filesystem for Java projects, in which the user can drop the projects they want to improve. If the user decides to choose a project from GitHub, the project will automatically be downloaded to this folder. After the user chooses a project, an Abstract Syntax Tree (AST) will be built from the source code, and calculations will be performed on

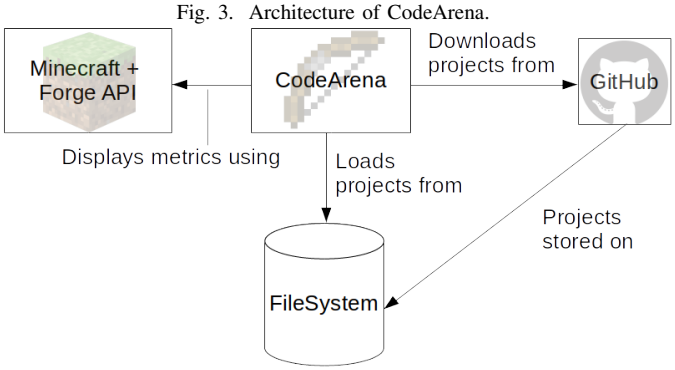
[1] Source on Github: https://github.com/SimonBaars/CodeArena
[2] Demo Video: https://www.youtube.com/watch?v=faQ3NKwQTL4

Fig. 2. The editor windows that are opened when a spider is struck



the basis of this AST. The overall architecture can be seen in figure 3.

Fig. 3. Architecture of CodeArena.



We have chosen to use a separate thread for our algorithm to calculate the different metrics. This way the user can keep using the game, and we can give the user direct feedback while the files are being scanned. Hence, the user does not have to wait for the algorithm to be completed, and can directly start fixing the metrics. When the user submits a metric as improved, we re-scan the changed files. Each metric improved in these files will account for points (or points are subtracted for worsening the quality), which are added to a scoreboard in the users' head-up display (HUD).

III. Discussion

We believe that this tool is an example of how we could educate novice developers using gamification. It is a novel way to get familiar to the field of technical debt and related concepts, and increases understanding by immersing the developer in the problems found in the code.

A. Threats to validity

The effectiveness of our approach is not tested. We do however mainly want to show the possibilities of using a 3D game to educate developers and do not claim to have fully tackled the stated problem. Our approach is however a step in a direction that shows promise [8, 1, 3, 10].

Since we make use of an AST to analyse the code, it will not work when code is not valid Java code. For instance, a missing bracket would invalidate the AST for that file. We therefore advice to first make sure the code is compilable before putting it through our tool.

B. Future work

The most important step now is to validate the effectiveness of our approach and find out if and how much developers benefit from using our tool. An approach would be to use it in an introductory programming course. We could then assess whether students that have been using the tool write high quality code and are aware what the characteristics of high-quality code are. This could be compared to other years of students to get a baseline.

Another step forward is, in our opinion, investigating the different approaches that can be taken when using Minecraft to visualise the code and problems that are in it. These different approaches have to be checked for effectiveness to see what works and what does not. Minecraft allows for almost limitless creativity, but not every approach will be viable.

IV. Conclusion

We presented an approach to educate novice developers to the field of technical debt and what the they can do to improve their code. Using our approach, developers can use Minecraft to interact with their code while gaining insight in the quality of the code, and immediately improve it while learning. Our tool is not validated for effectiveness, but is an example of an approach that can be taken to educate new developers, and could lead to new methods and approaches.

ACKNOWLEDGMENTS

We would like to thank R.A. van Rozen and dr. A.M. Oprescu from the University of Amsterdam for their feedback and enthusiasm.

REFERENCES

[1] Leonard Elezi, Sara Sali, Serge Demeyer, Alessandro Murgia, and Javier Pérez. A game of refactoring: Studying the impact of gamification in software refactoring. In *Proceedings of the Scientific Workshop Proceedings of XP2016*, page 23. ACM, 2016.

[2] Arun Gupta and Aditya Gupta. *Minecraft Modding with Forge: A Family-Friendly Guide to Building Fun Mods in Java.* " O'Reilly Media, Inc.", 2015.

[3] Thorsten Haendler and Gustaf Neumann. Serious refactoring games. In *Proceedings of the 52nd Hawaii International Conference on System Sciences*, 2019.

[4] Ilja Heitlager, Tobias Kuipers, and Joost Visser. A practical model for measuring maintainability. In *6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)*, pages 30–39. IEEE, 2007.

[5] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. Ccfinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, 28(7):654–670, 2002.

[6] Bennet P Lientz, E. Burton Swanson, and Gail E Tompkins. Characteristics of application software maintenance. *Communications of the ACM*, 21(6):466–471, 1978.

[7] Rüdiger Lincke, Jonas Lundberg, and Welf Löwe. Comparing software metrics tools. In *Proceedings of the 2008 international symposium on Software testing and analysis*, pages 131–142. ACM, 2008.

[8] Fiona Fui-Hoon Nah, Qing Zeng, Venkata Rajasekhar Telaprolu, Abhishek Padmanabhuni Ayyappa, and Brenda Eschenbrenner. Gamification of education: a review of literature. In *International conference on hci in business*, pages 401–409. Springer, 2014.

[9] Chris Parnin, Carsten Görg, and Ogechi Nnadi. A catalogue of lightweight visualizations to support code smell inspection. In *Proceedings of the 4th ACM symposium on Software visualization*, pages 77–86. ACM, 2008.

[10] Felix Raab. Codesmellexplorer: Tangible exploration of code smells and refactorings. In *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on*, pages 261–262. IEEE, 2012.