Is planning and refactoring as important as programming and production?

jc220866

February 2019

1 Introduction

When developers plan to work on a video game project for more than a Game Jam's worth of time, maintaining the cleanliness of the codebase is of utmost importance. Any code that goes through substantial changes with little to no thought behind structure, either planned in advance or refactored in arrears, will quickly become an untameable beast.[1][2]

Companies and development teams who attempt to manage these dense forests of messy, smelly code will only waste resources either allocating more hours or hiring more people to help program the game in a vicious, downwards spiral of unproductivity.[3] For this reason, maintaining the readability of a project's codebase is arguably more important than the content itself and adequate time must be given to maintenance rather than merely producing.[4][5][6]

However, how much time is appropriate? How long should programmers be expected to spend on planning and refactoring as opposed to actually producing the product? Should time and timeslots be planned and dedicated towards codebase maintenance and planning? Is refactoring and maintenance important enough to be considered worth an hourly salary? This essay will endeavour to answer all of these questions.

2 Pressure in the Gaming Industry

Within game development teams, a paradigm shift is necessary to appreciate the importance of proper planning and sufficient maintenance and refactoring of a codebase. Pressure constantly looms over game developers to push and release video game products far earlier than they are actually ready.[7][8] Programmers are allocated inadequate time to achieve unfeasible goals, placing pressure on them to produce instead of maintaining.

For the overall quality of video game software to improve, game development companies and publishers need to reduce the urgency of releasing new products. The scramble to capitalize on an existing intellectual property by release a new

title every year needs to be abolished also, as this is likely the catalyst to this overarching stress and pressure placed on developers.[7]

3 Maintaining a Codebase as a Team

[9][10]

4 Conclusion

Planning and refactoring do not directly contribute towards producing content meeting deadlines, but instead have a delayed positive effect on the maintainability, lifespan and ease of extension on of a piece of software. Were the aforementioned paradigm shift to occur in the gaming industry and more focus be placed on maintainability as opposed to meeting deadlines, the care given to the development of video games will increase and the quality will show for that. Until that happens, at this current time, the industry has come to accept unfinished releases, non-maintainable code and 'early access' as standard.[7]

References

[1] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. Di Penta, A. De Lucia, and D. Poshyvanyk, "When and why your code starts to smell bad," vol. 1, pp. 403–414, May 2015.

[2] A. N. Abdel-Hamid, "Refactoring as a lifeline: Lessons learned from refactoring," pp. 129–136, Aug 2013.

[3] R. C. Martin, "Clean code: A handbook of agile software craftsmanship.(2008)," *Citado na*, p. 19, 2008.

[4] M. Fowler, *Refactoring: improving the design of existing code.* Addison-Wesley Professional, 2018.

[5] J. Pérez, "Refactoring planning for design smell correction: Summary, opportunities and lessons learned," pp. 572–577, Sep. 2013.

[6] Y. Khrishe and M. Alshayeb, "An empirical study on the effect of the order of applying software refactoring," pp. 1–4, July 2016.

[7] A. Grossman, *Postmortems from Game Developer: Insights from the Developers of Unreal Tournament, Black & White, Age of Empire, and Other Top-Selling Games.* Focal Press, 2013.

[8] C. Keith, *Agile Game Development with Scrum (Adobe Reader).* Pearson Education, 2010.

[9] A. Begel and N. Nagappan, "Pair programming: What's in it for me?" pp. 120–128, 10 2008.

[10] R. Oliveira, "When more heads are better than one? understanding and improving collaborative identification of code smells," pp. 879–882, May 2016.