Understanding Gamification Mechanisms for Software Development [*]

Daniel J. Dubois
Massachusetts Institute of Technology
MIT Media Lab
Cambridge, MA, USA
ddubois@mit.edu

Giordano Tamburrelli
University of Lugano
Faculty of Informatics
Lugano, Switzerland
giordano.tamburrelli@usi.ch

ABSTRACT

In this paper we outline the idea to adopt gamification techniques to engage, train, monitor, and motivate all the players involved in the development of complex software artifacts, from the inception to the deployment and maintenance. The paper introduces the concept of gamification and proposes a research approach to understand how its principles may be successfully applied to the process of software development. Applying gamification to software engineering is not as straightforward as it may appear since it has to be casted to the peculiarities of this domain. Existing literature in the area has already recognized the possible use of such technology in the context of software development, however *how to design and use gamification* in this context is still an open question. This leads to several research challenges which are organized in a fascinating research agenda that is part of the contribution of this paper. Finally, to support the proposed ideas we present a preliminary experiment that shows the effect of gamification on the performance of students involved in a software engineering project.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management

General Terms

Human factors, management

Keywords

Software development, gamification, serious games, reward

1. INTRODUCTION AND MOTIVATION

Effective and efficient software development presents several technical challenges that relate for example to correctness, reliability, security, performance and privacy. Advances in research and technology have produced many methodologies and techniques to overcome these difficulties. However, despite these improvements, too often software projects produce artifacts with an unsatisfactory quality, or exceed their budget in terms of time and cost. This occurs not only because software engineering technologies and methodologies still require further investigations, but also because of *human factors*. Indeed software design and development is intrinsically a *human-centric* and *brain-intensive* activity in which the experience, motivation and discipline of developers represent crucial ingredients. However, stimulating and maximizing these elements is still an open challenge.

The idea we propose is a method to use *gamification* techniques to engage, train, monitor, and motivate all the players involved in the development of complex software artifacts. The final purpose is twofold: from one hand we want to improve software-engineering education/training, from the other hand we want to improve the quality of development activities for already experienced teams.

The term gamification is very recent and one of the first attempts to define it has been made in [13] with the following statement: *"gamification is defined as the use of game design elements in non-game contexts"*. The most elementary gamification element consists of a *rewarding mechanism* that awards people in response of the accomplishment of certain activities (also known as *challenges*) that need to be encouraged. The initial adoption of gamification was as a marketing strategy to increase customer engagements and, because of its effectiveness, rapidly spread to other domains such as employee or project management. Gamification, if applied to software development, may provide several advantages. First, because of its rewarding mechanisms, it may motivate developers to learn new technologies and increase their productivity (e.g., the Visual Studio Achievements [5]). Second, it may improve the quality of their work if adopted to encourage best practices (e.g., testing, code conventions, etc.). From a management perspective, it may be used as input to give economic incentives and to support the evaluation of employees as well as of teams. Deriving a method for applying gamification to software development is not as straightforward as it may appear since gamification principles and mechanisms need to be casted to the peculiarities of this domain. This leads to several research challenges, which constitute the goal of this work that may be summarized as the investigation and design of ad-hoc gamification mechanisms for software development, which depend on the behavior to be incentivized. Once a method has been found and experimentally validated on a given context, it can be easily reused in similar contexts with a limited cost.

In this paper we propose: (*i*) a research approach for adopting gamification in the different phases of the software-engineering lifecycle (both for educational and productivity purposes), (*ii*) a discussion on how such approach can be developed to address the identified issues, and (*iii*) preliminary results in the context of software engineering education on how gamification affects undergraduate students behaviors and performance.

[*]This work has been partially funded by the Fondazione Fratelli Rocca (Progetto Roberto Rocca fellowship – MIT-Italy Program) and by the European Commission (Marie Curie Intra-European Fellowship FP7-302648 RunMore).

ESEC/FSE'13, August 18–26, 2013, Saint Petersburg, Russia
ACM 978-1-4503-2237-9/13/08
http://dx.doi.org/10.1145/2491411.2494589

2. BACKGROUND AND RELATED WORK

2.1 Background on Gamification

Gamification has been initially adopted by marketers and website product managers as a tool to maximize customer engagement. For example StackOverflow [3], a popular question-and-answer site for developers, in which users receive points and/or badges for performing a variety of actions, including spreading links to questions and answers via Facebook and Twitter. Because of its effectiveness, gamification rapidly became popular and spread to other domains such as employee performance management and training (e.g., Work [6]) or project management (e.g., RedCritter [1]).

As a consequence of these initial promising applications in industry, academic research started to investigate gamification from many viewpoints. Indeed, even if the research on gamification is quite recent, there is already an established literature on this topic. For example in [12] and [21] the authors propose some guidelines for the design and development of games. In these guidelines they consider many key aspects to build successful games that include the necessity of challenges, interactions, the inclusion of creativity in the gaming experience, and finally a view of games as contexts for social play. This preliminary literature led to the adoption of the same features that make a game successful to the context of user applications to have similar engaging effects [19, 31], thus introducing gamification patterns into non-game contexts. In addition, existing works have identified the idea of collaboration [15] and competition [16, 27] as individual factors for developing gamified applications. In these works the authors observe for example that elements of collaboration/competition games can stimulate hobbyists to develop unpaid tasks. An example is the common creation of the so-called *mods*, which are modified versions of existing games developed by volunteers [22]. Moreover, other works analyze how the power of competition may be a huge source of motivation for players to achieve the so-called pro status (i.e., professional players) [28, 29]. In our previous work [14] we have shown how to engineer gamification at user experience level in mobile applications using not only collaboration and competition, but also exploiting context-aware capabilities of mobile devices.

In this paper we propose a research approach for understanding and achieving all the aforementioned effects to improve the quality of a software engineering process.

2.2 Gamification and Software Engineering

In a survey on social software engineering [7] the authors point out the need to: (*i*) integrate results from social and psychological sciences in the software lifecycle, (*ii*) engineer social networking services and collaborative tools. With respect to this our idea has the objective to exploit gamification to advance the state of the art in social software engineering in both directions.

Preliminary works on the topic of gamification have been recently presented, however they just tend to show some abstract gamification ideas applied to very particular cases. Unfortunately, little effort has been spent so far to *understand* how gamification can be applied on different situations. For example Passos *et al.* [20] propose an idea that, at the best of our knowledge, is the closest to the one we are proposing: they suggest to gamify the entire software lifecycle by dividing the whole process into tasks and assigning points and achievements based on task completion. The main limitation of such work is that the authors do not explain *how* to actually use gamification (e.g., type of gamification strategy, pros/cons, etc.), which is still an open question and the main problem addressed by our paper. Xie *et al.* [30] discuss the importance of gamification in the software engineering education area using PEX4FUN, a .NET multi-language platform for education for providing learning games based on *coding duels*. Sheth *et al.* propose a methodology for highly addictive, socially optimized software engineering [23] and show their experience in project gamification of an introductory computer science class [24]. Singer and Schneider [25] show their experience in the gamification of a version control system based on the number of commits. Bacchelli *et al.* propose Seahawk [8], a tool for integrating the gamified question-and-answer engine of StackOverflow [3] to a software development IDE, in which users receive points and/or badges for answering questions. Finally, game-inspired concepts have also been widely used in agile software development methodologies such as eXtreme Programming [11], in which the planning activity is modeled as a planning game to maximize the scores earned during a development iteration. Worth to mention are also the work by Snipes *et al.* [26] that investigates a game-like system to motivate developers to use more efficient development techniques, and the work by Bacon *et al.* [9, 10] that investigates the adoption of scoring systems in the context of software development.

With respect to the state of the art described above our idea is to propose a methodology that can be repeatable for the different phases of a software engineering process and be used to understand and assess the effects of the gamification approaches already identified, as well as possible new ones. The benefit of this research will be first in the educational sector, since the learning process of software engineering has widely shown to benefit from gamification strategies; second in the industry sector, since employees may be continuously motivated to improve the performance and quality of the produced software artifacts with very small investment in time and money from the company.

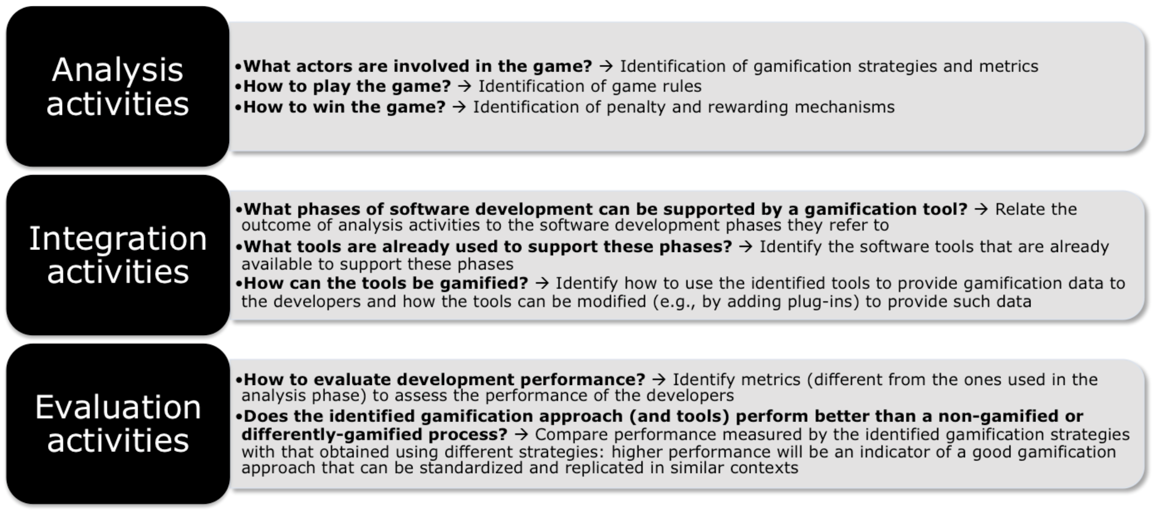
3. RESEARCH APPROACH

3.1 General Approach

We propose a research strategy that is based on three different sets of complementary activities. The first is the set of *analysis activities*, which analyze different gamification approaches and identify the most appropriate mechanisms to be applied to the different phases of the software development process. The second is the set of *integration activities*, which integrate the identified mechanisms into the existing software development toolchain through ad-hoc modules/plugins. Finally, the third is the set of *evaluation activities*, which evaluate the identified solutions. An abstract view of the activities is reported in Figure 1 and discussed as follows.

The *analysis* activities consider a set of gamification strategies, which aim at establishing a collaboration game between developers and the stakeholders, and a competition game between people in the same role. We also propose to investigate the definition of game rules, the use of reward and penalty mechanisms and metrics, achievement systems, reputation mechanisms, and cheating/overfitting prevention. We envision that the knowledge base produced by analysis activities will be the conceptual foundation of the gamification mechanisms for the design, implementation, testing, and maintenance phases of a software engineering process.

The *integration* activities focus on the development of software modules/plugins for introducing gamification elements into the software development process. These tools are intended to be used to perform evaluations with students and organizations willing to experiment these novel techniques. For these activities we propose to target development environments such as Visual Studio, automatic code analysis and reporting tools (e.g., Hudson/Sonar [2]), and collaborative development frameworks (e.g., Application Lifecycle Management [4]). Finally, the *evaluation* activities consist in the application of the methodologies and the software modules developed during analysis and integration activities in real usage scenarios. Possible evaluation settings can be the *educational scenario* and the *industrial scenario*. The final goal is to analyze the benefits and the risks of a software development process that uses gamification mechanisms against an equivalent software develop-


Analysis activities

- **What actors are involved in the game?** → Identification of gamification strategies and metrics
- **How to play the game?** → Identification of game rules
- **How to win the game?** → Identification of penalty and rewarding mechanisms

Integration activities

- **What phases of software development can be supported by a gamification tool?** → Relate the outcome of analysis activities to the software development phases they refer to
- **What tools are already used to support these phases?** → Identify the software tools that are already available to support these phases
- **How can the tools be gamified?** → Identify how to use the identified tools to provide gamification data to the developers and how the tools can be modified (e.g., by adding plug-ins) to provide such data

Evaluation activities

- **How to evaluate development performance?** → Identify metrics (different from the ones used in the analysis phase) to assess the performance of the developers
- **Does the identified gamification approach (and tools) perform better than a non-gamified or differently-gamified process?** → Compare performance measured by the identified gamification strategies with that obtained using different strategies: higher performance will be an indicator of a good gamification approach that can be standardized and replicated in similar contexts

Figure 1: Research activities for assessing gamification in a software development context.

ment process that does not rely on them. Obtained results are compared and evaluated using metrics already established in the software engineering community such as the time needed to complete the project, design quality, test coverage, number of bugs, etc.

3.2 Approach Development in an Education Scenario

Since the general research approach proposed above is meant to be used in different scenarios (e.g., basic education, advanced education, professional training, general professional productivity, and so on), in this subsection we discuss a possible way to develop it in the context of basic education of object-oriented software engineering for undergraduate students in computer science that have to demonstrate their software engineering skills through the development of a project. We assume that the goal of the project is to acquire the ability to design software, implement it, document it, and finally test it. At the end of the project a final mark is assessed. Typically, during classes students are told some best practices. For example that copy/paste should be avoided, that all the exposed interfaces of their code should be properly documented, that their test cases should cover – ideally – all their code (lines and branches), to commit working code only, and so on. Since several of these best practices can be expressed in terms of metrics that can be measured from the code, in the analysis phase a good decision would be to adopt the optimizations of such metrics as simple game rules. For example, in the case of test coverage metric 0% would be the most negative reward, while 100% would be the most positive reward. When analyzing and deciding metrics it is important to also identify and discourage possible *cheating* behaviors. A possible cheating behavior is when some students try to maliciously maximize their rewards by adding test cases without assertions or with obviously true assertions to the code base. Cheating phenomena such as the one described above can be partially prevented by letting students know that their code will get random manual/automatic checks (without disclosing how) and that in case of confirmed cheating they will likely *lose the game* (e.g., fail the course). In the integration phase of our approach a possible tool that could analyze the code and report the metrics to the students is needed. As said in the previous subsection the Hudson/Sonar [2] tool can be used for this purpose: by using this software students would receive a report every time they submit some code modifications. The report contains information about their metrics (the default settings of Sonar include the compliance against 1000+ predefined rules, the lines of code, test coverage, duplications, and JavaDoc API documentation). This way students would be given the challenge to improve all those numbers (reward) and at the same time fulfill the functional requirements of the project. To evaluate the results of the chosen game rules and integration tools we need to define an evaluation phase. In this phase the work of the students must be evaluated using an evaluation metric that is different from the ones used as game rules and, at the same time, unknown to the students: this is important to prevent and detect cheating behaviors. Finally the evaluation metric should be assessed for every student. A good metric that has the above characteristics is the final non-normalized mark of the project derived from a manual code review. That metric should also be evaluated in another group of students whose learning process has been gamified in a different way, or not gamified at all. If after the evaluation we obtain a correlation between high marks and the proposed gamification strategy that is higher than a non-gamified or differently gamified approach, then the proposed gamification strategy is likely to be useful and can be reused in similar contexts.

4. PRELIMINARY RESULTS

In our software engineering courses taught at Politecnico di Milano we have seen a significant increase in the quality of software artifacts produced by the students with the gamification approach described in the previous section with respect to the artifacts produced by students without it. The reason is that students during their coding activities get addicted at removing code smells as soon as Sonar reports them to preserve their *rules compliance score*. In this section we want to give a practical example of gamification assessment to understand the effect of *competition* in the same scenario. Our experiment involved two sections of information engineering students that had to develop a software as a final activity to obtain their bachelor degree. Each section was composed of 32 groups, each one composed of 2 to 3 students. The project consisted in the implementation of a popular board-game in Java either using Swing or Android SDK. The gamification strategy was the same one proposed in Section 3.2 with the following variation: groups of the first section (section A) could only see their own Sonar metrics in Sonar reports, while groups of the second section (section B) could see the metrics of all the other groups of the same section, thus stimulating competition. The main considered Sonar metrics were the lines of code (LOC), the percentage of Sonar rules compliance, the percentage of branch coverage of test

Table 1: Project results for section A (no competition)

Group	LOC	Rules	Test Cov.	Dupl.	JavaDoc	Mark
A1	1,451	74.3%	22.1%	8.9%	85.3%	20
A2	1,907	90.1%	0.0%	14.0%	57.6%	18
A3	3,215	66.5%	28.9%	36.9%	74.9%	27
A4	3,856	95.0%	10.8%	3.2%	70.9%	24
...	...	...	...	...	...	...
A29	4,155	92.9%	38.0%	7.2%	74.3%	30
A30	2,947	90.7%	45.8%	0.8%	95.7%	25
A31	2,859	99.2%	42.9%	0.3%	81.0%	28
A32	4,979	99.8%	38.4%	0.9%	100.0%	31
Average	**3,678**	**90.0%**	**38.8%**	**4.8%**	**79.5%**	**25.13**

Table 2: Project results for section B (with competition)

Group	LOC	Rules	Test Cov.	Dupl.	JavaDoc	Mark
B1	2,596	90.6%	67.4%	0.5%	93.4%	21
B2	4,865	80.8%	47.5%	0.7%	77.7%	25
B3	2,772	82.0%	48.5%	1.8%	68.9%	24
B4	1,935	94.1%	74.8%	2.0%	97.8%	20
...	...	...	...	...	...	...
B29	5,243	92.0%	65.6%	3.6%	100.0%	26
B30	5,838	95.2%	42.4%	0.4%	97.9%	31
B31	1,554	91.2%	76.3%	1.4%	99.4%	23
B32	3,830	82.9%	61.7%	6.9%	76.4%	28
Average	**3,247**	**90.6%**	**56.9%**	**4.3%**	**88.9%**	**25.53**

cases, the percentage of code duplication, and the percentage of JavaDoc documentation. The results of the experiments, partially reported in Tables 1 and 2, show that although the overall results are not very different, some metrics are better in Section B. The reason for this is that the students use the metrics of other students as a benchmark for their behaviors (students were told that having too many lines of code may be a smell of bad design, therefore a lower value may be considered better in many groups). We can also observe that in Section A the students still try to maximize their metrics (e.g., trying to reach 100% in metrics measured as percentages), but most effort is spent in metrics that were considered more important by the teaching assistants (i.e., rules compliance), while that was not true for other metrics such as test coverage and API documentation. Although these preliminary results do not provide enough evidence to draw a final conclusion about the role of competition, we have a concrete hint that gamification with competition may be better than without competition, therefore, as a future work, it may be worth to perform further investigations with a larger sample of students.

5. CONCLUSIONS

This paper is part of a long running research stream on gamification [14] and education [17, 18]. In particular in this paper we proposed a research approach for understanding the use of gamification for improving the software development process in different contexts. We experimented our approach in the context of software engineering education. Our preliminary experiments indicate that, while integrating gamification in a software development process is a relatively easy task, developing a gamification method and predicting its effect is much more difficult. Indeed, increasing software quality is an emergent property of gamification that is difficult to derive without a proper experimental evidence. Future work includes the application of our approach in a non-educational context and the analysis of causes and effects of specific aspects of gamification (e.g., distress levels, cheating behaviors, etc.).

6. REFERENCES

[1] RedCritter. `http://www.redcritter.com`.

[2] Sonar. `http://www.sonarsource.org`.

[3] StackOverflow. `http://stackoverflow.com`.

[4] Visual Studio – Application Lifecycle Management. `http://www.microsoft.com/visualstudio/eng/alm`.

[5] Visual Studio Achievements. `http://channel9.msdn.com/achievements/visualstudio`.

[6] Work. `http://work.com`.

[7] N. Ahmadi, M. Jazayeri, F. Lelli, and S. Nesic. A survey of social software engineering. In *ASE*, 2008.

[8] A. Bacchelli, L. Ponzanelli, and M. Lanza. Harnessing stack overflow for the IDE. In *RSSE '12*, pages 26–30. IEEE, 2012.

[9] D. F. Bacon, Y. Chen, D. Parkes, and M. Rao. A market-based approach to software evolution. In *ACM SIGPLAN OOPSLA '09 conference companion*, pages 973–980, New York, NY, USA, 2009. ACM.

[10] D. F. Bacon, D. C. Parkes, Y. Chen, M. Rao, I. Kash, and M. Sridharan. Predicting your own effort. In *AAMAS '12*, pages 695–702, Richland, SC, 2012. International Foundation for Autonomous Agents and Multiagent Systems.

[11] K. Beck and C. Andres. *Extreme programming explained: embrace change*. Addison-Wesley, 2004.

[12] C. Crawford. *Chris Crawford on game design*. New Riders Pub, 2003.

[13] S. Deterding and R. Khaled. Gamification: Toward a Definition. In *CHI '11 Gamification Workshop*. ACM, 2011.

[14] D. J. Dubois. Toward Adopting Self-organizing Models for the Gamification of Context-aware User Applications. In *GAS '12 – ICSE Workshop*, pages 9–15. IEEE, 2012.

[15] J. McGonigal. *Reality is broken: Why games make us better and how they can change the world*. Penguin Press HC, 2011.

[16] D. Nieborg. Am I Mod or Not?–An Analysis of First Person Shooter Modification Culture. In *Creative Gamers Seminar–Exploring Participatory Culture in Gaming. Hypermedia Laboratory (University of Tampere)*, 2005.

[17] M. Nordio, C. Ghezzi, B. Meyer, E. Di Nitto, G. Tamburrelli, J. Tschannen, N. Aguirre, and V. Kulkarni. Teaching software engineering using globally distributed projects: the dose course. CTGDSD '11. ACM, 2011.

[18] M. Nordio, R. Mitin, B. Meyer, C. Ghezzi, E. D. Nitto, and G. Tamburrelli. The role of contracts in distributed development. In O. Gotel, M. Joseph, and B. Meyer, editors, *SEAFOOD*, volume 35 of *Lecture Notes in Business Information Processing*, pages 117–129. Springer, 2009.

[19] M. Oja and J. Riekki. Ubiquitous framework for creating and evaluating persuasive applications and games. In *Grid and Pervasive Comp. Workshops*, pages 133–140. Springer, 2012.

[20] E. Passos, D. Medeiros, P. Neto, and E. Clua. Turning Real-World Software Development into a Game. In *SBGAMES '11*.

[21] K. Salen and E. Zimmerman. *Rules of play: Game design fundamentals*. MIT press, 2003.

[22] W. Scacchi. Modding as an open source approach to extending computer game systems. *Open Source Systems: Grounding Research*, pages 62–74, 2011.

[23] S. Sheth, J. Bell, and G. Kaiser. Halo (highly addictive, socially optimized) software engineering. In *GAS '11 – ICSE Workshop*, pages 29–32. ACM, 2011.

[24] S. Sheth, J. Bell, and G. Kaiser. A Competitive-Collaborative Approach for Introducing Software Engineering in a CS2 Class. Technical Report CUCS-017-12, Michigan State University, 2012.

[25] L. Singer and K. Schneider. It was a bit of a race: Gamification of version control. In *GAS '12*. IEEE.

[26] W. Snipes, V. Augustine, A. R. Nair, and E. Murphy-Hill. Towards recognizing and rewarding efficient developer work patterns. In *ICSE '13*, pages 1277–1280, Piscataway, NJ, USA, 2013. IEEE Press.

[27] O. Sotamaa. Have Fun Working with Our Product! *Critical Perspectives on Computer Game Mod Competitions*, 2005.

[28] T. Taylor. *Raising the Stakes: E-sports and the professionalization of computer gaming*. MIT Press, 2012.

[29] T. Taylor and E. Witkowski. This is how we play it: what a mega-LAN can teach us about games. In *FDG '10*. ACM.

[30] T. Xie, N. Tillmann, and J. de Halleux. Educational software engineering: Where software engineering, education, and gaming meet. In *GAS '13 – ICSE Workshop*. ACM, 2013.

[31] G. Zichermann and C. Cunningham. *Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps*. O'Reilly Media, 2011.