Impact of Gamification on Code review process - An Experimental Study

by

SHIVAM KHADELWAL, sai krishna sripada sripada, Y.Raghu Babu Reddy

in

Innovations in Software Engineering Conference, ISEC

Report No: IIIT/TR/2017/-1

Centre for Software Engineering Research Lab
International Institute of Information Technology
Hyderabad - 500 032, INDIA
February 2017

Impact of Gamification on Code review process - An Experimental Study

Shivam Khandelwal[*]
Software Engineering
Reseach Center
IIIT Hyderabad, India

Sai Krishna Sripada[†]
Software Engineering
Reseach Center
IIIT Hyderabad, India

Y. Raghu Reddy[‡]
Software Engineering
Reseach Center
IIIT Hyderabad, India

ABSTRACT

Researchers have supported the idea of gamification to enhance students' interest in activities like code reviews, change management, knowledge management, issue tracking, etc. which might otherwise be repetitive and monotonous. We performed an experimental study consisting of nearly 180+ participants to measure the impact of gamification on code review process using 5 different code review tools, including one gamified code review instance from our extensible architectural framework. We assess the impact of gamification based on the code smells and bugs identified in a gamified and non-gamified environment as per code inspection report. Further, measurement and comparison of the quantity and usefulness of code review comments was done using machine learning techniques.

Keywords

Architectural Framework, Classification, Code Reviews, Evaluation, Gamification, Text Analysis

1. INTRODUCTION

In Software Engineering domain, processes followed while performing bug hunting, code reviews, requirement elicitation, change management, knowledge management, issue tracking, etc are considered repetitive and time consuming [3]. The repetition and monotony can have a negative effect on the engagement levels of the stakeholder. Gamification has been suggested as a mechanism for improving the usage of software engineering processes/tools related to such processes [9]. Gamification is accomplished by using game design elements in non-gaming contexts to improve developers' engagement, motivation and performance. For example, badges, points, levels, etc. are a few game design elements.

[*]shivam.khandelwal@research.iiit.ac.in
[†]saikrishna.sripada@research.iiit.ac.in
[‡]raghu.reddy@iiit.ac.in

ISEC '17, February 05-07, 2017, Jaipur, India

DOI: http://dx.doi.org/10.1145/3021460.3021474

A framework that facilitates creation of gamified instances of these repetitive tasks, using subset of available game design elements can potentially enhance developer's interest in carrying out the tasks. In our previous work, we proposed such an extensible architectural framework [11]. We created four game design elements as services which were badges, leaderboard, timeline and avatar. However there are not enough studies to show the impact of gamification on efficiency, developer interest, productivity, etc. of the tasks. In this paper, we study the impact of the gamification by comparing five tools in gamified and non-gamified environments for performing code review activity. We compare, two open access gamified tools and two open access non-gamified tools, along with one gamified instance of code review process generated from our framework. We compare the number of bugs and code smells identified, followed by usefulness of code review comments extracted from both gamified and non-gamified environments and infer the impact of gamification. The subjects of the study were 183 Sophomore undergraduate students attending Structure Software Analysis and Design (SSAD) course during monsoon 2015 (August - December) at IIIT Hyderabad[1].

Increase in learning and engagement can be correlated to achieving the goal of the activity with a decrease in time taken. While "goal" can both be qualitative and quantitative in nature, to measure quantitatively, one must consider the semantic aspect (usefulness) of the activity being done. A quantitative study [2] at Microsoft measures the usefulness of code review data. We extrapolate their study and measure the impact of gamification by assessing the usefulness of comments written during the code review process in both gamified as well as non-gamified environments. Usefulness of code review comments is calculated using a machine learning model created to perform data-driven predictions.

This paper contributes to the academic/research community by quantitatively measuring the impact of gamification in performing code review activity. We correlate gamification with:

1. Improvement in usefulness of code review comments
2. Identification of bugs and code smells in code written by developers

The rest of the paper is structured as follows: Related work is discussed in Section 2. Section 3 details the experimental study conducted, Section 4 discusses the machine

[1]http://www.iiit.ac.in

learning model created to predict usefulness of code review data. The results of the experimental study are shared in 5 along with feedback from the subjects. Threats to validity and future plans are presented in Section 6 and Section 7 respectively.

2. RELATED WORK

Gamification is the use of game design elements, game mechanics and game thinking in non-gaming contexts to improve the user's engagement, motivation, and performance. Pedreira et al. [7] found out that *"points" "Badges"* are the most commonly used game design elements. Tillman et al. [12] showed gamification of software engineering activities have helped developers to perform activity better.

Peer code review process acts as an effective strategy to catch bugs during early stages of product development [1]. The process can also help assess coding standards [5] and improve software quality [6]. While Text classification methods to automatically measure usefulness of code review comments are being studied by some researchers [2], there aren't enough studies on impact of gamification. Some studies on application of text classification algorithms in various activities like fault localization [8], bug prediction[4] can be used further to facilitate creation of new gamification instances.

3. EXPERIMENT PLANNING

3.1 Experiment Definition

The objective of the experiment is to determine the differences between gamified and non-gamified environments in following contexts:

Code Review Comments: In a normal code review setting, reviewers have to enter their review comments either using some inbuilt tools or some external tools. The quantity of these comments along with their usefulness is a measure of subject's interest in the activity.

Code Inspection Report: The subjects had to fill code inspection report along with their review comments. The inspection report utlized a specific template that was used for providing a summary of bugs and code smells found during the process. Subject's interest is measured by numbers and types of bugs and code smells detected by subjects.

3.2 Context Selection

The experiment was performed on 183 sophomore undergraduate students enrolled in SSAD course at IIIT Hyderabad. All the students registered in course were selected without any screening process, as all of them possessed adequate knowledge required for the experiment. They were aware of the main purpose,of experiment i.e. code review process with and without gamification, but did not know the exact hypothesis tested.

The students were well versed in programming and had previously developed software applications using object oriented programming languages. As the part of same course, each student had programmed Donkey Kong game (using *Python*) in earlier assignment using OOPs concepts. The code base for the assignment was used for peer review process during this experiment because that contained adequate amount of complexity and all the students were familiar with its requirements. On an average, each student's code base was about 500 lines. The students were also taught about peer code review process, code refactoring and software quality prior to the experiment. They were introduced to code smells, anti-patterns, and refactorings. In addition, there were some activities carried out using the linting and refactoring tools.

Table 1: Comparison of selected code review tools

Tool	Gamified (Y/N)	Likes	Activity Stream	Points	Leader board	Badges
Bitbucket	No		Yes			
Phabricator	No		Yes			
Github CC	Yes			Yes	Yes	
Codebrag	Yes	Yes				
GamifiedSD	Yes	Yes	Yes	Yes	Yes	Yes

A brief study was conducted to shortlist gamified and non-gamified tools that support code reviews based on usage and popularity in developer community. We shortlisted only those tools which work with git repositories and had inline commenting functionality. We picked two gamified (Github CC[2], Codebrag[3]) and two non-gamified tools (Bitbucket[4], Phabricator[5]) from that list. Parallely, we shortlisted a few game design elements that are most commonly used for gamification of various tasks.

In our previous work, we had proposed an extensible architectural framework for gamifying Software Engineering tasks. We had detailed the process of creating a new game instance using game design elements. Along with the 4 tools mentioned earlier, we used one more code review tool i.e. **GamifiedSD**[6], instantiated from our architectural framework [11]. The game design elements available in all the tools are shown in Table 1. In GamifiedSD, the game design elements were developed and kept as Ember components[7]. The module "diff utility" (extracted from open source diffcommenter[8]) was used for implementing inline commenting in our code review instance.

3.3 Hypothesis Formulation

For the experiment, we formulated our null hypothesis and alternate hypothesis as given below and evaluated the results against the hypothesis:

- **Null hypothesis,** H_0: There is no impact of gamification on Code Review process and subject's productivity.
  H_0: $Productivity(Gamified) = Productivity(Non-Gamified)$

- **Alternative hypothesis,** H_1: There is positive impact of gamification on Code Review process and subject's productivity.
  H_1: $Productivity(Gamified) \neq Productivity(Non-Gamified)$

[2]https://github.com/tzachz/github-comment-counter
[3]http://codebrag.com/
[4]https://bitbucket.org/
[5]https://www.phacility.com/
[6]https://github.com/skbly7/gamifiedSD
[7]http://emberjs.com/api/classes/Ember.Component.html
[8]https://github.com/Babazka/diffcommenter

The independent variable for the study was the type of code review tool used (T) by the students, since there were five different tools (gamified and non-gamified) used.

The dependent variables i.e. measures needed to evaluate the hypothesis are 1. Number of code review comments (N) 2. Usefulness of code review comments (U) 3. Number of bugs identified (NB) 4. Number of code smells identified (NC) 5. Time spent on code review process (TS).

3.4 Experiment Design

3.4.1 Between-group strategy

We created a total of 5 groups. Each group was assigned one of the code review tools described in Table 1 and between-group strategy was followed for our experiment. The reason behind the decision was that we wished to measure the effect of gamification w.r.t. type of tool used (T). By dividing the subjects into 5 groups and using multiple non-gamified and gamified tools instead of just one large group, we ensured that differences in systems' complexity would not bias the results.

The 183 subjects were then assigned to one of above groups randomly. The random allocation was done to ensure that the results are not spurious or deceptive via confounding. As stated earlier, we made sure that all the subjects were aware of OOPs concepts and code review process by giving them prior assignments, activities and tutorial.

3.4.2 Experimental task

The code bases of all the students (from previous assignment) were uploaded to code review tools. It was ensured that students didn't get their own code base. Also, the authorship or any indicator in the code that could potentially disclose the author was scrubbed.

Each subject carried out code reviews for 5 peer-written code bases using code review tool assigned to them in a 15 day time period. The process included logging the reviews as inline comments, and reporting of bug type and code smells on code inspection form available as Google form. To assist students in their work, a pre-defined set of most common code smells and bug types were given to them [10].

4. USEFULNESS OF CODE REVIEW COMMENTS

Usefulness of code review comments was measured by implementing a machine learning model which was implemented using python's sklearn[9] library. Various feature extraction and supervised learning algorithms were analyzed before finalizing on an algorithm for generating the results, as per the classification and prediction process shown in Figure 1.

4.1 Train and Test data generation

A total of 2500 comments were randomly selected (500 per code review tool). These were manually classified on a likert scale of 1-5 for usefulness (with 1 being the lowest level of usefulness and 5 being highest) by two researchers with prior experience in classification techniques. The labelled data had distribution of 59-605-484-1111-241 on likert scale and the dataset from study is available on Github[10]. This

[9]http://scikit-learn.org/

[10]https://github.com/skbly7/usefulness

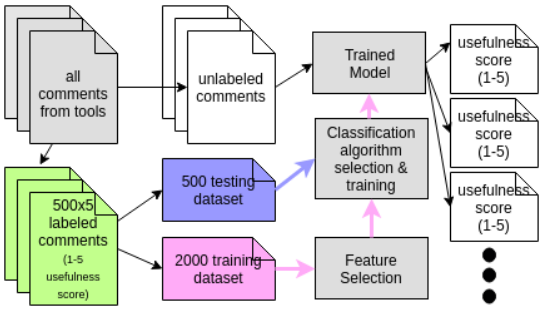

Figure 1: Text classification and prediction process

labelled data was then randomly divided into set of 2000 training and 500 testing dataset.

4.2 Text pre-processing

All the code review comments were pre-processed in following steps: 1. Converting strings (containing html) to Beautiful Soup[11] object. 2. Removing the <*code*> and <*pre*> tags. 3. Converting Beautiful Soup object to plain text (without HTML) 4. Converting all words to lower case. 5. Converting the special character "-" into single white space. (example "use-less" to "use less") 6. Tokenizing words with in sentences. 7. Removing the set of stop words inherited from an open sourced GitHub repository[12]. 8. Stemming of words using the "pystemmer" module. (removing the end words/prefixes/suffixes of words) 9. Removing special characters, punctuation and numbers from the data.

Table 2: Predictive model generation

	tf-idf				word2vec				character n-grams			
	precision	recall	f1-score	support	precision	recall	f1-score	support	precision	recall	f1-score	support
GausianNB												
1	0.40	0.55	0.46	11	0.40	0.55	0.46	11	0.03	0.82	0.07	11
2	0.55	0.78	0.64	109	0.45	0.80	0.57	109	0.48	0.14	0.21	109
3	0.48	0.52	0.50	107	0.41	0.34	0.37	107	0.21	0.06	0.09	107
4	0.62	0.24	0.35	226	0.61	0.23	0.33	226	0.57	0.35	0.43	226
5	0.24	0.64	0.35	47	0.26	0.66	0.37	47	0.33	0.32	0.32	47
total	0.53	0.46	0.45	500	0.50	0.42	0.40	500	0.44	0.25	0.29	500
DecisionTree												
1	0.36	0.82	0.5	11	0.33	0.73	0.46	11	0.58	0.64	0.61	11
2	0.8	0.84	0.82	109	0.8	0.79	0.8	109	0.61	0.72	0.66	109
3	0.75	0.79	0.77	107	0.76	0.76	0.76	107	0.58	0.53	0.56	107
4	0.81	0.71	0.76	226	0.78	0.77	0.78	226	0.7	0.67	0.68	226
5	0.4	0.4	0.4	47	0.32	0.26	0.28	47	0.35	0.34	0.34	47
total	0.75	0.73	0.74	500	0.73	0.72	0.72	500	0.62	0.62	0.62	500
RandomForest												
1	0.42	0.73	0.53	11	0.67	0.55	0.6	11	0.8	0.73	0.76	11
2	0.74	0.83	0.78	109	0.76	0.83	0.79	109	0.64	0.71	0.67	109
3	0.77	0.68	0.72	107	0.77	0.73	0.75	107	0.68	0.51	0.59	107
4	0.78	0.85	0.81	226	0.75	0.87	0.8	226	0.69	0.81	0.75	226
5	0.64	0.19	0.30	47	0.62	0.11	0.18	47	0.54	0.28	0.37	47
total	0.75	0.75	0.73	500	0.74	0.75	0.73	500	0.67	0.67	0.66	500
LinearSVM												
1	0.67	0.18	0.29	11	0.6	0.27	0.37	11	0.43	0.27	0.33	11
2	0.86	0.83	0.84	109	0.85	0.81	0.83	109	0.54	0.52	0.53	109
3	0.78	0.74	0.76	107	0.73	0.76	0.74	107	0.64	0.35	0.45	107
4	0.76	0.89	0.82	226	0.77	0.82	0.8	226	0.62	0.85	0.72	226
5	0.61	0.36	0.45	47	0.49	0.4	0.44	47	0.56	0.21	0.31	47
total	0.77	0.78	0.77	500	0.75	0.75	0.75	500	0.6	0.6	0.57	500

4.3 Classification Model Generation

The predictive model was based on three feature vector creation methods namely *tf-idf, word2vec, and character n-grams* and four classification methods namely: *Linear SVM, Random Forest, Decision Tree, and Gaussian Naive Bayes.* We took all 12 possible combinations for feature vector creation and classification methods and built the classification model. The classification models were trained on training

[11]http://www.crummy.com/software/BeautifulSoup/

[12]https://github.com/Alir3z4/stop-words

Table 3: Measurement of Dependent Variables

Tool	Type	#Students	#Comments (N)	#Bugs (NB)	#CodeSmells (NC)	#TimeSpent (TS)
GithubCC	Gamified	37	1510	568	428	3575
Phabricator	Non-Gamified	34	1979	682	756	2661
GamifiedSD	Gamified	38	2778	661	770	3577
BitBucket	Non-Gamified	36	2935	482	783	3441
CodeBrag	Gamified	37	2944	670	630	3308

dataset using default parameters provided by sklearn library (v0.18rc2). The models generated were now tested on testing dataset and the precision, recall and support results are shown in Table 2 for this step.

The main criteria for choosing a predictive model was **better overall accuracy** and **better recall value**. As shown in the Table 2, Linear SVM along with tf-idf approach gave the best results. Thus, we chose this classifier for predicting usefulness automatically for rest of comments.

5. RESULTS AND ANALYSIS

5.1 Dependent Variables

The values of dependent variables i.e. N, NB, NC and TS for the code review tool used (T) are shown in Table 3 . In the table, N is extracted directly from the tools using database access and javascript based crawlers. Values for NB, NC and TS have been extracted from code inspection document. The amount of time spent is comparable for all the tools however subjects using BitBucket were unable to find bugs as effectively (482 vs 568+) as compared to other tools. In contrast, subjects using Phabricator were able to detect more bugs in lesser time. The usefulness of code review comments (U) is calculated by classification model and is shared in Table 4. The usefulness scale can be interpreted as: (1) Completely irrelevant, (2) minor conformance violation to PEP8[13] standards. (3) Major coding style violation like missing docstring, naming conventions, indentation, etc., (4) Minor Bug or Code smell identification, and (5) Major Bug or Code smell identification (with reference to code).

5.2 Hypothesis Testing

The averages of all the dependent variables from 3 gamified and 2 non-gamified tools (T) are shown in Table 5 for testing the hypothesis. As per the results, hypothesis H_0 can't be rejected, because if H_1 holds true all the dependent variables should have shown an increase. There are various reasons which we still believe need to be investigated for a more accurate testing of the hypothesis. We share the same in later sections.

5.3 Usefulness of comments

The usefulness model generation and information on usefulness scale was discussed in Section 4.3 and 5.1. Although various models and tokenizers were explored prior to classification, the highest accuracy obtained over training data was less than 80%. This indicates that the model can be further improved, by using advanced techniques, and further tuning of existing classifiers. Also, as the classifier was comparatively young, the classification was verified manually to make sure it doesn't effect the experiment's results.

[13]https://www.python.org/dev/peps/pep-0008/

5.4 Survey Results

All the subjects who used gamified tools were asked to fill in a survey questionnaire after the code review activity was completed. The survey included the following questions: (1) Which game design elements for Gamification did you like the most?, (2) Did gamification help in peer code review activity?.

In spite of having no improvement on dependent variables, it was surprising to note that nearly 54% of the students were in strong favour of gamification in code review process. Only 9% students disliked it and felt it is not required. The results also provided us a prioritized list of game design elements like leaderboard, self score, likes, challengers, badges, rewards, etc. needed for gamification of code review process.

6. THREATS TO VALIDITY.

The experiment conducted was limited to coding practices and code reviews performed in an academic setup. While these activities are practiced in industry too, the results observed in this study may not extrapolate to the industry due to difference in maturity and experience. In fact, we believe that the impact of gamification would have been different on graduate students in academia. So, for a better understanding of the impact of gamification on code review process, a wider sample set is necessary.

Given that nearly 35+ students were working on the same assignment for each tool, we imposed a restriction on students that, a bug or code smell reported by a reviewer cannot be reported by another reviewer. We observed this had an adverse impact on the measurements. Instead, we proposed conducting these reviews privately and allowing students to perform code reviews without any restrictions for more meaningful reviews.

7. CONCLUSIONS AND FUTURE WORK

Our study results failed hypothesis H_1 and suggested that there is no impact of gamification on code review process. However, a secondary look into the topic is needed because of various extraneous variables which acted unknowingly and need to be controlled further in study design. The major variables which hampered the experiment are: (1) Public setup in which subjects could see each others' comments, (2) Unequal time spent by different subjects, (3) BitBucket had email notification on every comment by peer which could

Table 4: Usefulness of comments (U) by classification.

	Usefulness Scale						Useful	Useful
Tool	1	2	3	4	5	Total	(style = useful)	(style ≠ useful)
GithubCC	15	356	251	800	88	1510	75.43%	58.80%
Phabricator	38	493	298	969	181	1979	73.16%	58.11%
GamifiedSD	9	844	602	1224	99	2778	69.29%	47.62%
BitBucket	55	844	592	1283	161	2935	69.36%	49.19%
CodeBrag	27	647	563	1578	129	2944	77.10%	57.98%

Table 5: Comparison between Gamified and Non-gamified environment

Aspect	Gamified environment (total subjects: 112)	Non Gamified environment (70)
Time spent (TS)	93.39	87.17
Average (NB)	16.95	16.62
Average (NC)	16.32	21.98
Usefulness>=3 (U)	73.75%	70.89%
Usefulness>=4 (U)	54.17%	52.78%

have led to peer-pressure and higher group output, (4) Slow game design elements updation on GithubCC, (5) Different codebases to subjects in different group, (6) Different tools were used instead of variation of GDEs in single tool, and (7) Restriction on identification of a bug that has already been identified

If we revisit the hypothesis considering above effects, we can see huge increase in N (58.2 to 76.2) and TS (78.2 to 91.8) with no decrease in comments usefulness (73%). This is an encouraging observation for conducting similar study again. We also believe that the learning algorithms can further be modified to ensure higher accuracy in predicting the "usefulness" of code review comment. In future, there is a need to auto-validate and predict correctness of code inspection document for completely unbiased results.

The motivation for gamifying an application varies depending on the target subjects, activity being gamified and the context in which a particular activity is gamified. In academia the primary goal is to improve the quality of code, enhance learning, increase engagement, etc, where as in industry it can be more focused to "productivity". Therefore gamifying an application should be based on activity, context and the target audience. Also same game design elements are likely to yield different results in a different settings. Hence, additional studies done considering all these parameters are likely to give a more accurate measure of the impact of gamification.

8. REFERENCES

[1] A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. pages 712–721, 2013.

[2] A. Bosu, M. Greiler, and C. Bird. Characteristics of useful code reviews: An empirical study at microsoft. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, pages 146–156, 2015.

[3] S. Deterding, D. Dixon, R. Khaled, and L. Nacke. From game design elements to gamefulness: Defining "gamification". In *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, MindTrek '11, pages 9–15, 2011.

[4] L. Jiang and Z. Su. Context-aware statistical debugging: From bug predictors to faulty control flow paths. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, pages 184–193, 2007.

[5] X. Li. Using peer review to assess coding standards - a case study. *Frontiers in Education Conference, 36th Annual*, pages 9–14, 2006.

[6] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 192–201, 2014.

[7] O. Pedreira, F. Garcia, N. Brisaboa, and M. Piattini. Gamification in software engineering, a systematic mapping. *Information and Software Technology*, 57:157–168, January 2015.

[8] S. Roychowdhury and S. Khurshid. Software fault localization using feature selection. In *Proceedings of the International Workshop on Machine Learning Technologies in Software Engineering*, MALETS '11, pages 11–18, 2011.

[9] L. Singer and K. Schneider. Influencing the adoption of software engineering methods using social software. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 1325–1328, June 2012.

[10] S. Sripada and Y. Reddy. Code comprehension activities in undergraduate software engineering course - a case study. In *24rd Australian Software Engineering Conference (ASWEC), 2015*, May 2015.

[11] S. K. Sripada, Y. R. Reddy, and S. Khandelwal. Architecting an extensible framework for gamifying software engineering concepts. In *Proceedings of the 9th India Software Engineering Conference*, ISEC '16, pages 119–130, 2016.

[12] N. Tillmann, J. De Halleux, T. Xie, S. Gulwani, and J. Bishop. Teaching and learning programming and software engineering via interactive gaming. In *35th International Conference on Software Engineering*, pages 1117–1126, May 2013.