Code Review Participation: Game Theoretical Modeling of Reviewers in Gerrit Datasets

Norihito Kitagawa
Nara Institute of Science and Technology, Japan
kitagawa.norihito.ke7@is.naist.jp

Hideaki Hata
Nara Institute of Science and Technology, Japan
hata@is.naist.jp

Akinori Ihara
Nara Institute of Science and Technology, Japan
akinori-i@is.naist.jp

Kiminao Kogiso
The University of Electro-Communications, Japan
kogiso@uec.ac.jp

Kenichi Matsumoto
Nara Institute of Science and Technology, Japan
matumoto@is.naist.jp

ABSTRACT

Code review is a common practice for improving the quality of source code changes and expediting knowledge transfer in a development community. In modern code review, source code changes or patches are considered to be assessed and approved for integration by multiple reviews. However, from our empirical study, we found that some patches are reviewed by only one reviewer, and some reviewers did not continue the review discussion, which can have negative effects on software quality. To understand these reviewers' behaviors, we model the code review situation based on the snowdrift game, which is used to analyze social dilemmas. With this game-theoretical modeling, we found that it can explain reviewers' behaviors well.

Keywords

Code review; Game Theory; Empirical Study

1. INTRODUCTION

Modern code review is less formal, lightweight, and tool-assisted code examination process, and is well-established best practice in both open source and proprietary software projects. The general process can be summarized as follows [1]. First, a developer creates a change and submits it for review. Second, other developers discuss the change and suggest fixes. The change can be re-submitted multiple times to deal with the suggested changes. Finally, one or more reviewers approve the change to be integrated in the main source code repository, or reject the change. While finding defects is the main motivation for code review, reviewers expect additional benefits, such as knowledge transfer, increased team awareness, and creation of alternative solutions to problems [2].

Rigby and Storey revealed several characteristics of reviewers in open source projects, that is, objective analyzer, expert or fatherly adviser, and enthusiastic supporter or champion as positive personas; grumpy cynic and frustratedly resigned as negative personas [3]. Since review requests are broadcasted to a community, there is a concern of discussion deadlock because of too many stakeholders being involved. This study reported that bike shed effect, a long discussion that involves few core developers, does not appear to be a significant problem. This is because individuals with lower status in an open source software community generally receive little attention from the community. Rigby and Bird reported that the median number of reviewers is two in not only open source review, but Google-led projects (Android and Chromium OS) and Microsoft projects (Bing, Office, and MS SQL) [1]. Through a case study with Qt, VTK, and ITK projects, McIntosh et al. found that lack of reviewer participation has a negative impact on software quality, and reviews without discussion are associated with higher post release defect counts [4]. Bosu and Carver revealed that OSS developers' reputations can have impact on code review outcome [5]. They reported that core developers receive quicker first feedback on their review request, and complete review processes in shorter time. On the other hand, peripheral developers may have to wait 2 to 19 times longer than core developers for the review process.

We have investigated Gerrit[1] datasets in Qt and OpenStack to see reviewers' actual activities [6]. Although code review are expected to be processed by multiple reviewers, we found that some reviews were conducted solely, and some reviewers stopped discussion before reaching agreements. Understanding the mechanism of this phenomenon is important because lack of reviewer participation has a negative impact on software quality [4].

To understand reviewer participation, we build game theoretical models of code review situation based on the snowdrift game. The snowdrift game is two-person games with two strategies, to cooperate and to defect, and can model social dilemmas similar to the prisoner's dilemma. In code

CHASE'16, May 16 2016, Austin, TX, USA

DOI: http://dx.doi.org/10.1145/2897586.2897605

[1] https://www.gerritcodereview.com/index.md

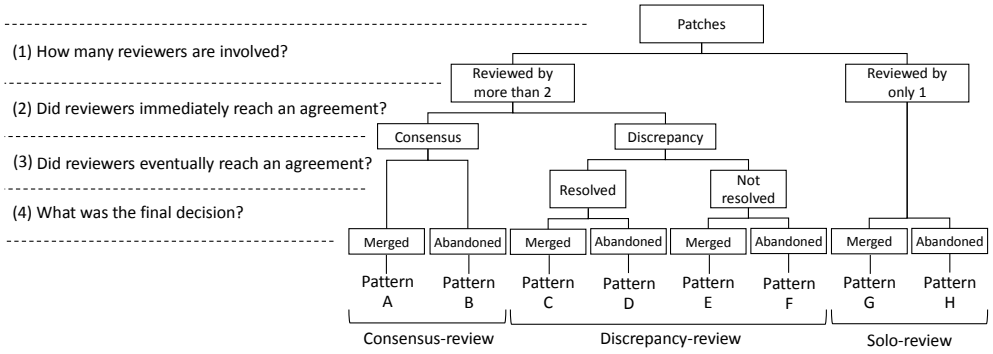

Figure 1: Code review participation and agreement patterns.

Table 1: Frequencies of review patterns.

	Consensus		Discrepancy (Resolved)		Discrepancy (Not resolved)		Solo	
Pattern	A	B	C	D	E	F	G	H
Qt	45,861 (67%)	1,046 (2%)	8,345 (12%)	0 (0%)	4,131 (6%)	7,911 (12%)	559 (1%)	259 (<1%)
OpenSrack	61,356 (71%)	106 (<1%)	6,042 (7%)	33 (0%)	2,109 (2%)	11,437 (13%)	1,409 (2%)	4,057 (5%)

review cases, cooperate means conducting review, and defect means not doing review. Based on theoretical analysis with this model, we revealed that (i) a reviewer has motivation to chose defect if the other reviewer choses cooperate. On the contrary, a reviewer will cooperate if the other will defect. (ii) a reviewer will cooperate when the benefit of review (cooperate) is higher than the cost. These explanation match the actual phenomenon. This paper extends the previous study presented in a national workshop [6], by formally describing models and deepening data analysis to validate models.

2. REVIEWER PARTICIPATION

Figure 1 illustrates possible patterns in code review participation and agreement. Patterns can be determined according to the following four questions:

(1) How many reviewers are involved? We divide reviews (patches) into two, that is reviewed by only one reviewer or by more than two reviewers. Two is the median number of reviewers in major open source and proprietary software projects [1].

(2) Did reviewers immediately reach an agreement? Sometimes multiple reviewers will reach an agreement to positive or negative immediately.

(3) Did reviewers eventually reach an agreement? Even if initial decisions do not reach consensuses, patch authors and reviewers can discuss and/or improve patches to reach an agreement.

(4) What was the final decision? Final decision will be made to accept (will be merged) or reject (abandoned).

With these questions, we can identify eight patterns of code review participation and agreement (pattern A to H).

We investigated the code review datasets of Qt and OpenStack projects used in the previous study [4]. The datasets contain 70,765 and 92,984 review reports respectively, which have patch IDs from source code repositories, votes of reviewers, and reviewer comments.

Table 1 presents the frequencies of patterns in the two projects. We can see that $69-71\%$ of reviews were processed in consensus review. In addition, $7-12\%$ of reviews were initially regarded discrepancy, but resolved after discussion. Hence, it revealed that $79-81\%$ of reviews are processed as expected. However, there are cases in which only one reviewer was involved. From manual inspection, we found that although some reviewers were asked to review patches, they did not review them in such solo reviews. The remaining pattern E and F are reviews that are initially discrepancy and not resolved before the final decisions. These patterns can be considered that some reviewers stopped the discussion instead of resolving discrepancy.

Although most of the reviews were processed with multi reviewers and discussed until reaching agreements, sometimes reviewers did not review at all or stopped discussion. Why does this happens? To discuss this phenomenon, we introduce game theory for modeling code review situations.

3. GAME THEORETICAL MODELING

3.1 Terms and Definitions

We begin with an (informal) introduction of several game-theoretic terms and definitions used in this section. We basically refer to the explanations Chapter 3 of [10].

A (perfect information) strategic form game is the most

Table 2: The utility matrix of code review based on the snowdrift game

		Reviewer B	
		Review (Cooperate, C)	Not review (Defect, D)
Reviewer A	Review (Cooperate, C)	$(b-\frac{c}{2}, b-\frac{c}{2})$	$(b-c, b)$
	Not review (Defect, D)	$(b, b-c)$	$(0, 0)$

familiar representation of strategic interactions in game theory. A natural way to represent games is via n-dimensional matrix. In general, each row corresponds to a possible action for player 1, each column corresponds to a possible action for player 2, and each cell corresponds to one possible outcome from each player's action. Each player's utility for an outcome is written in the cell corresponding to that outcome, with player 1's utility listed first. Finally, we assume that every player knows all the information, including how much utility an adversary receives at a cell. That is why we call this game perfect information.

A pure strategy is the strategy that players select a single action and play it. We call a choice of pure strategy for each player a pure-strategy profile.

A mixed strategy is the strategy that players select a single action based on the action selection probability of co-player.

The Nash equilibrium (NE) is a stable strategy profile: no player would want to change his strategy if he knew what strategy the other players were following. In a pure-strategic form game, players rely on NE to make decisions. In a mixed-strategic form game, players rely on expected utility to make decisions.

In this paper, we study both pure and mixed strategic form games in which there are only two players, namely a reviewer A and a reviewer B. The games consist of two-dimensional matrix (in other words, each player has binary actions).

3.2 Game Description

We model the code review situation based on the snowdrift game. The snowdrift game is known to be a good model to analyze social dilemmas [11]. The concept of snowdrift game can be described as follows. There are two drivers that are caught in a blizzard and trapped on either side of a snowdrift. They can either get out and start shoveling (cooperate) or remain in the car (defect). If both cooperate, they have the benefit b of getting home while sharing the labour c. Thus, the utility is $b - \frac{c}{2}$. If both defect, they do not get anywhere and utility is 0. If only one shovels, however, they both get home but the defector avoids the labour cost and gets the utility b, whereas the cooperator gets the utility $b-c$. There is an article that claims that the snowdrift game is a better model for funding of open source software compared to the well-known Prisoner's Dilemma [2].

Table 2 shows the utility matrix of the snowdrift game in code review. In Table 2, the first column corresponds to a possible action for the reviewer A, and the first row corresponds to a possible action for the reviewer B, and each cell corresponds to one possible outcome. Each reviewer chooses either cooperate (C) or defect (D). Each player's utility consists of benefit b and cost c. If costs are high ($2b > c > b > 0$), these utilities recover the Prisoner's Dilemma, that is, both player should defect. By contrast, if $b > c > 0$, the utilities generate the snowdrift game.

As reported in the previous study [2], knowledge transfer, increased team awareness, and creation of alternative solutions to problems can be regarded as benefits. The effort of reviewing code can be regarded as costs. The expected removed effort that will be needed to fix remain bugs if review fails may be regarded as a benefit of review too.

3.3 Pure Strategic Form Game Analysis

We calculate a NE of the pure strategic form game. We examine each action profile in turn to find a NE.

(C, C) Each player can increase his utility by choosing another action (D). Thus, this action profile is not a NE.

(C, D) Each player can not increase his utility by choosing another action. Thus, this action profile is a NE.

(D, C) Each player can not increase his utility by choosing another action. Thus, this action profile is a NE.

(D, D) Each player can increase his utility by choosing another action (C). Thus, this action profile is not a NE.

Thus, we conclude that the game has two NE, (C,D) and (D,C). From this result, it is revealed that each reviewer has motivation to not review (defect) if the other review (cooperate), but to review if the other does not review.

3.4 Mixed Strategic Form Game Analysis

We calculate an expected utility of the game. First, we calculate an expected utility of the reviewer A. We assume the probability that the reviewer B chooses C on q and chooses D on $1-q$. Then, the expected utility of the reviewer A when he chooses C is $(b-\frac{c}{2})q+(b-c)(1-q)$. The expected utility of the reviewer A when he/her chooses D is bq.

If the following equation hold, the reviewer A is considered to chose C.

$$(b-\frac{c}{2})q + (b-c)(1-q) > bq$$

This can be summarized as:

$$q < 1 - \frac{c}{2b-c} = 1 - \frac{1}{2n-1}$$

where $n = \frac{b}{c}$.

This means that the reviewer A can get more expected utility by choosing C when n (the ratio of b to c) is increased, that is, the ratio of benefit b to cost c is higher. Similarly, if the benefit is rather larger than cost, the reviewer B will choose C (cooperate, review). On the contrary, if the ratio of b to c is small, each reviewer seems to chose D (defect, not review).

3.5 Summary

From the NE analysis, we showed that each reviewer has a motive of choosing different action from co-reviewer. Actually, reviewers have discarded reviews in 15%-18% review

[2]Paul Chiusano, The failed economics of our software commons, and what you can about it right now, https://pchiusano.github.io/2014-12-08/failed-software-economics

categories, and only one reviewer has reviewed in 1%-7% review categories.

From the expected utility analysis, we showed that both reviewers have a motive of choosing C as b becomes larger than c in $b > c > 0$. Actually, reviewers have reviewed cooperatively in 78%-79% review categories. Why is the rate of cooperative reviewer such high? In other words, why is the benefit much higher than the cost for reviewers? McIntosh et al. found that poorly reviewed code has a negative impact on software quality [4], and reviewers seem to know that. Thus, we consider that the benefit is much higher than the cost for reviewers, and reviewers cooperatively review.

To explain the distributions of review patterns in Table 1, how much benefits can we expect for reviewers compared to costs? From the datasets, we found that $b = 8.3$ if $c = 1$. Although, the values of expected benefits should vary with reviewers in real environment, this observation is interesting since average reviewers seem to expect much higher benefits compared to the cost of reviews. Further empirical evaluation should be needed and interesting.

4. RELATED WORK

Game theory is a bag of analytical tools designed to help us understand the phenomena that we observe when decision-makers interact [9]. A mathematical formulation makes it easy to define concepts precisely, to verify the consistency of ideas, and to explore the implications of assumptions. Bacon et al. showed the vision of software economics, which intends to re-imagine the software development process in terms of decentralized, self-regulating economies [14]. We will proceed this kind of vision by integrating theoretical and empirical analysis.

There are some studies introducing game theoretical analysis into software engineering problems. Bacon et al. proposed a market-based mechanism of bidding for features and fixes [12]. Rao et al. designed a dynamic model of the software engineering ecosystem for incentivizing correcting the root cause of bugs instead of fixing them superficially [13]. Hata et al. studied the characteristics of sustainable OSS projects based on leader-follower game modeling and data mining of GitHub dataset [7]. Sun et al. studied the mechanism of indivisible objects exchange, which can be used for re-scheduling of tasks, such as bug fixing [8].

5. CONCLUSION

This paper studied the participations of reviewers in OSS code review by theoretical and empirical analysis. We modeled the reviewing situation based on the snowdrift game in which a reviewer cooperates and defects with each other. Based on game-theoretical models, we revealed two motivations of reviewers (i) a reviewer has a motive of choosing different action from other reviewer (ii) a reviewer cooperates with other reviewers when the benefit of review is higher than the cost.

In this paper, we revealed motivations of reviewers based on a strategic form game. However, we do not consider a situation that a reviewer knows the other reviewer's action. In a real code review scenario, it is natural a reviewer can see other reviewers' actions. To consider this situation, we will use a extensive form game in our future work. An extensive form game is represented as a tree, which is also known as a game tree. Each non-terminal node in a game tree corresponds to the moment of the choice of one player, and each edge from a node corresponds to an action possible for the player at the moment. Each terminal (or leaf) node represents a final outcome, under which each player receives a utility. With this model, we can analyze the motivations of reviewers at the situation that a reviewer knows the other reviewer's action.

6. ACKNOWLEDGEMENTS

This study has been supported by JSPS KAKENHI Grant Number 26540029, and has been conducted as a part of JSPS Program for Advancing Strategic International Networks to Accelerate the Circulation of Talented Researchers.

7. REFERENCES

[1] P. C. Rigby and C. Bird, "Convergent contemporary software peer review practices," in ESEC/FSE '13, 2013, pp. 202–212.

[2] A. Bacchelli and C. Bird, "Expectations, outcomes, and challenges of modern code review," in ICSE '13, 2013, pp. 712–721.

[3] P. C. Rigby and M.-A. Storey, "Understanding broadcast based peer review on open source software projects," in ICSE '11, 2011, pp. 541–550.

[4] 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 MSR '14, 2014, pp. 192–201.

[5] A. Bosu and J. C. Carver, "Impact of developer reputation on code review outcomes in oss projects: An empirical investigation," in ESEM '14, 2014, pp. 33:1–33:10.

[6] N. Kitagawa, H. Hata, A. Ihara, K. Kogiso, and K. Matsumoto, "Cooperation in code review: A theoretical and empirical study," in FOSE '15, 2015, pp. 203–212. (in Japanese)

[7] H. Hata, T. Todo, S. Onoue, and K. Matsumoto, "Characteristics of sustainable oss projects: A theoretical and empirical study," in CHASE '15, 2015, pp. 15–21.

[8] Z. Sun, H. Hata, T. Todo, and M. Yokoo, "Exchange of indivisible objects with asymmetry," in IJCAI '15, 2015, pp. 97–103.

[9] M. Osborne and A. Rubinstein, A Course in Game Theory. MIT Press, 1994.

[10] Y. Shoham and K. Leyton-Brown, Multiagent systems: Algorithmic, game-theoretic, and logical foundations. Cambridge University Press, 2008.

[11] C. Hauert and M. Doebeli, "Spatial structure often inhibits the evolution of cooperation in the snowdrift game," Nature, vol. 428, no. 6983, pp. 643–646, 2004.

[12] D. F. Bacon, Y. Chen, D. Parkes, and M. Rao, "A market-based approach to software evolution," in OOPSLA '09, 2009, pp. 973–980.

[13] M. Rao, D. C. Parkes, M. Seltzer, and D. F. Bacon, "A Framework for Incentivizing Deep Fixes," in WIT-EC '14, 2014.

[14] D. F. Bacon, E. Bokelberg, Y. Chen, I. A. Kash, D. C. Parkes, M. Rao, and M. Sridharan, "Software economies," in FoSER '10, 2010, pp. 7–12.