JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 27 No. 11, pp. 257-270, November 2022
https://doi.org/10.9708/jksci.2022.27.11.257

An Analysis on Employing Developer Profit Incentive to Expedite Open Source Software Development

Jung-woo Sohn*, Yohan Ko*, Younguk Yun*

*Assistant Professor, Dept. of Software, Yonsei University, Wonju, Korea
*Assistant Professor, Dept. of Software, Yonsei University, Wonju, Korea
*Assistant Professor, Dept. of Software, Yonsei University, Wonju, Korea

[Abstract]

This paper analyzes the effect of profit incentives within the setting of bounty open source project. A simple decision-making model based on classical utility maximization is presented for open source developers that includes income effects from the bounty prize. We then simulate the decisions of multiple developers to assess the effect from the bounty prize. Our result shows that learning costs can greatly reduce the software quality improvement benefit from bounty project. It also suggests that open source projects can benefit more when they have multiple small bounty projects than a single large bounty project since it reduces the learning cost and the opportunity cost for the open source developers.

▸ **Key words**: open source, developer incentive, simulation, bounty project, bounty bug, income effect

[요　약]

본 연구는 오픈소스 현상금 프로젝트에서 개발자 이윤 동기의 효과를 분석했다. 분석을 위해 현상금 프로젝트의 상금 취득에서 기인하는 소득 증대 효과를 미시경제학의 효용 최대화 이론에 근거한 오픈소스 개발자 의사결정 모형으로 제시한다. 그리고 이 모형을 바탕으로 현상금의 효과를 평가하기 위해 다수의 오픈소스 프로젝트 참여 개발자들의 행동을 시뮬레이션으로 구성했다. 그 결과는 현상금 프로젝트에서 기인하는 소프트웨어 품질 향상의 상당 부분이 개발자의 소스 코드 학습 비용으로 상쇄될 수 있다는 점을 보여 주고 있다. 또한 이 결과는 오픈소스 프로젝트의 경우 개발자의 소스 코드 학습 비용과 개발 작업 참가의 기회비용을 감소시킨다는 측면에서 하나의 대규모 현상금 프로젝트보다는 소규모로 작게 쪼개진 현상금 프로젝트를 여럿 운용하는 것이 더욱 향상된 프로젝트 결과물을 만들어 낼 가능성이 크다는 점을 시사하고 있다.

▸ **주제어**: 오픈소스, 개발자 유인, 시뮬레이션, 현상금 프로젝트, 버그 현상금, 소득 효과

• First Author: Jung-woo Sohn, Corresponding Author: Jung-woo Sohn
*Jung-woo Sohn (jwsohn00@yonsei.ac.kr), Dept. of Software, Yonsei University
*Yohan Ko (yohan.ko@yonsei.ac.kr), Dept. of Software, Yonsei University
*Younguk Yun (yu_yun@yonsei.ac.kr), Dept. of Software, Yonsei University
• Received: 2022. 10. 18, Revised: 2022. 11. 14, Accepted: 2022. 11. 14.

I. Introduction

Open source software has been a success in a variety of areas, such as the Linux operating system and the Mozilla Firefox web browsers. The free availability open source software is not only a valuable benefit to the computer users, but also an important resource for program developers since the source code is open to public.

From an economic perspective, an interesting characteristic of open source software development is that it does not directly rely on the profit incentives of individual developers [1,2]. Since the distribution of open source software is free, open source developers cannot obtain any profit by selling software copies. On the other hand, commercial software developers can earn revenue by selling software copies that prohibit making unlicensed copies of the software.

With this lack of direct profit incentives, there are a number of ongoing debates about the nature of the incentives for open source developers [2-6]. But the current popularity of open source software and its wide use is an evidence that open source can work without the need for direct profit incentives.

However, several aspects of open source development lag behind those of their commercial software counterparts. End-user applications such as office suites, with a focus on user interface development, are known to be less successful under the open source development model [7]. Certain feature implementations or bug fixes require long hours of effort from skilled programmers and can take a long time until they are implemented, to the detriment of the overall project. In other words, there are situations in open source software development where voluntary participation from the developers is not enough.

One alternative explored by the open source community to solve this problem is to provide open source developers with a profit incentive so that feature implementations or needed bug fixes are more quickly produced. Bounty Source [8,9] is one such example that actively tries to engage the profit incentives of developers for the enhancement of the project software by matching people who wants to pay for the feature implementations and the developers. Bounty projects are particularly popular for finding security vulnerabilities and there are a number of open source projects hosting “bug bounty” program including Drupal content management system [10], Apache web server [11], Internet Bug Bounty [12, 13], and such.

When necessary, a bounty project poster announces a “bounty project” that will award bounty prize money to the developer who successfully implements the bounty project requirements. The bounty process starts by posting a bounty project and the prize amount. Competing project proposals are submitted by developers interested in winning the bounty prize. The bounty poster then selects the best one and the bounty project starts. When the project is completed and the final code is submitted, the bounty developer is paid with the prize money [28].

A question that arises from using bounty projects is how effective the bounty project model can be. While the money spent on bounty projects will clearly expedite the needed open source software enhancements, there seem to be potential issues that arise from employing profit incentives in open source development.

One phenomenon to note is that bounty developers have little incentive to share their ideas or source code with other possibly competing developers because they might lose the chance to win the bounty prize. Of course, this can happen only while the bounty project is going on. Most open source bounty projects specify the release process of the bounty project source code to the public once the winning developers have secured the prize. Also, other voluntary open source developers might not want to make contributions to the bounty source code before the bounty project completion because that might complicate the division of any reward.

A problem arising from this is that other open source developers incur increased learning costs associated with the sudden influx of source code contributions from any completed bounty projects. For example, they may need to understand significant portions of new bounty project code contributions to write their own. Since the source code from a bounty project will be kept hidden from other voluntary developers before the project completion for the reason stated above, this learning cost can become higher than the cost for the usual voluntary open source projects where source code updates are open to public and traceable all the time.

Another problem can be associated with the potential opportunity cost of losing a developer to a bounty project. Since the participation of a proficient developer in any bounty project means the absence of his or her contribution in the voluntary open source project, the amount of the aggregate contributions going to the voluntary open source project development will experience a temporary decrease for the duration of the bounty project.

This study attempts to analyze these potential negative effects of profit incentives within the setting of open source project with bounty money prize. For the analysis purpose, we present a model of individual decision-makings for an open source developer based on the classical utility maximization theory in microeconomics. Then the decision making of rational individual agents was simulated to analyze the aggregate effect of individual developer contributions to a voluntary open source project. The bounty project model is introduced finally and compared with the pure voluntary open source project case.

Our decision-making model has the following characteristics. First, utility maximization has a firm foundation on the rationality framework from neo-classical economics that has long been found out to be effective for building decision-making models beyond of economics disciplines such as marriage [24] and can be applied to model open source development incentives. While behavioral economics presents alternative research methods for handling anomaly cases hardly explainable by rationality frameworks, its uses are considered to be still rooted in rationality, mapping subjectivism into objectivism of individual decision-makings and observing consistent preference revelation [25]. Second, our simulation model can generate dataset for analysis hardly collectible by empirical methods. Open source development is inherently collective from participation of multiple developers and software good is known to be under network externalities [26] or two-sided market effect [27] where the value of any software increases as the pool size of its users or developers increase. Therefore, observations and proper empirical data collections and scaling up for a large existing open source project can have limitations from the start and complex interactions between developers might be better modeled as simple interactions between agents in simulations.

II. Related research

1. Individual developer incentives

Most of the open source developers refer to non-monetary motivations such as “fun” or “intellectual challenges” as the driving forces that lead to the production of open source software. Raymond [4] early focused on this fun factor for the primary open source development incentives.

One interesting point is that Raymond’s argument starts from the individual incentives for open source developers. The “fun” or desire to solve an “intellectually challenging” problem are still effective given a situation that there are no audience at all to review his or her code. Raymond’s explanation thus views open source developers as free individuals who seek their own satisfaction, which can be directly translated into the economic analysis framework for the individual utility maximization problems [29].

2. Open source developer incentives

2.1. Signaling incentives based on group reputation

Instead of individual utility, Lerner and Tirole [14] centered on developer incentives related to the open source developer community, by focusing on *signaling incentives* for open source developers. An open source developer might want to signal his or her programming proficiency for building the future job market career (career concern gratification incentive).

Lerner and Tirole's approach sheds lights on another aspects of non-altruistic incentive for open source developers in that this signaling incentives become higher as the number of audience is increasing, unlike Raymond's broad definition that spans between individual and community level.

2.2 Free-riding behavior vs. making contributions

Johnson [16] proposed a game-theoretical model for open source development based on the concept of "private provisioning of public goods [18]." Johnson focuses on free-riding behavior of open source developers that an open source developer must decide whether to participate in the open source project or not before actually making code contributions. When the prospect of a missing feature desired by a developer is likely to be developed by other developers, he or she does not need to participate in the project and will be better off by free-riding. On the other hand, when the missing feature is unlikely to be developed by others, he or she will be better off by writing the code and making contribution.

3. Implications about open source developer incentives

Considering these arguments for the open source development incentive discussed so far, the "fun" or "intellectual challenges" writing source code are assumed to be main incentives that motivate open source developers in this study. In other words, it is assumed that open source developers actually enjoy putting part of their leisure hours available into open source development. Factors such as the developer pool size n or free-riding behavior from development prospect can play an important role in the analysis but their consideration is excluded in this research in favor of building a simple model to start with.

III. Research model

1. Modeling of pure voluntary open source development

1.1 Research question

As specified above, the research question for this paper can be summarized as:

- What kind of negative effects will be caused by introducing bounty projects with prizes into open source software development projects?
- What will be its degree of the negative effect in comparison with the gross software quality boost expected from the successful bounty project completion?

1.2 Individual decision making

Let L be the amount of programming labor hours put to open source software development work. Also assume that the developer is endowed with a total of $\overline{L}$ hours available. She will either use the available hours for (1) earning her income with the hourly wage rate w or for (2) writing open source software code for a leisure activity. Let L be the amount of hours used for writing open source software and l be the labor hours for income earning. For the purpose of building a simple model, consider the substitution relationship between L and l such that $L+l=\overline{L}$.

The amount of consumption for all other goods is denoted as c with the price of p, where p is normalized to one for simplicity. With the introduction of the classical Cobbs-Douglas utility [29] function such as $U(c,L)=c^{1-\alpha}L^{\alpha}$ for the developer preference (with $0 < \alpha < 1$), the developer will attempt to maximize her utility in such a way that [19]

$$\max_{c,L} U(c,L) = c^{1-\alpha} L^{\alpha} \text{ subject to } pc = wl + I$$

Rewriting this maximization problem with L instead of l leads to

$$\max_{c,L} U(c,L) = c^{1-\alpha} L^{\alpha}$$
$$\text{subject to } pc + wL = w\overline{L} + I \qquad (1)$$

where I is the income for this developer other than the earning from her labor l. Then the optimal choice for this developer will be

$$\begin{aligned} L^* &= \alpha \cdot \frac{w\overline{L}+I}{w} \\ c^* &= (1-\alpha) \cdot \frac{w\overline{L}+I}{p} \end{aligned} \qquad (2)$$

In the later experiment and the aggregation model with multiple developers, α will be drawn randomly, which will reflect the different preference structure from one developer to another. The higher α is, the more hours L^* the developer is willing to invest on open source development given the same level of income.

1.3 Incorporation of multiple developers and time factor

Now let N be the total number of developers interested in writing code for an open source project. Let t be defined as time variable over the natural numbers. The net contribution from a developer i for the open source software quality increase $\Delta q_{i,t}$ can be simply modeled as follows:

$$\Delta q_{i,t} = \Delta q_{i,t}(L^*_{i,q_{t-1}-q_{t-1-\Delta S_i}}, \Delta S_i)$$

where L^{*i} is the amount of hours that were chosen by the developer i for her open source development effort. ΔS_i denotes the time window of the developer i. It refers to the time span she considers back to the time $t - \Delta S_i$ when she participates in current development stage t.

Decomposition of the positive and the negative effects leads to the following equation

$$\Delta q_{i,t} = F_i(L_i^*) - C_i(q_{t-1} - q_{t-1-\Delta S_i}, \Delta S_i) \quad (3)$$

where F_i is the positive effect based on the production function for developer i, which has diminishing marginal returns with respect to the L^{*i}, and C_i is the negative effect to the project from the developer i. C_i here is analogous to a cost function for producing the open source software code. Note that the developer does not attempt to maximize her net contribution $\Delta q_{i,t}$ here. She tries to maximize her utility by choosing the optimal level of L^{*i} (and c^{*i}). This is to model the tendency of open source developers to use their free time for open source development work. For the open source developers, optimal allocation of the available programming hours are primary concerns. Thus, it is possible in this model that net contribution can be zero even when the developer spent L^{*i} hours. One such example can be that she might have spent all her hours reading source code but she did not write any new code.

The purpose of introducing the time window ΔS_i is to model the learning cost for open source developers. For example, suppose a developer i decides to participate in the open source project at time t. While she does not need to read all the source code to write her own contribution, she might need to spend time to read existing source code particularly relevant to her needs. A proficient developer might have to look at the source code developed for smaller sized window of ΔS_i but a novice developer has to read the source code developed for a longer period. Thus, the net contribution $\Delta q_{i,t}$ will be decreasing as the developer i has higher $\Delta q_{i,t}$ which is modeled into the C_i term on the equation (3).

In addition, the learning cost for reading source code developed by other developers increases more when larger amount of source contribution is created for a unit time span, which can be modeled as the overall improvement of software quality increase $q_{t-1} - q_{t-1-\Delta S_i}$. As $q_{t-1} - q_{t-1-\Delta S_i}$ for the software quality increases, for example, from such cases as the project becomes popular, the learning cost for the developer i to understand the source code changes and to keep up with the project development will increase and the developer will not be able to produce more source code contributions to the project.

For the simulation in this paper, typical quadratic benefit/cost functions were chosen for F_i and C_i such that

$$F_i = c_0 \cdot \beta_i \sqrt{L_i^*}$$
$$C_i = \gamma_i \cdot c_1 (q_{t-1} - q_{t-1-\Delta S_i})^2 + c_2 \cdot \Delta S_i^2$$

where c_0, c_1, and c_2 are constants. β_i and γ_i are introduced so that F_i and C_i can be different from one developer to another, reflecting the different individual characteristics in making open source contribution. Higher β_i means that the individual contribution will be greater given the same level of development hours L^{*i} . Higher level of γ_i leads to less contribution from the developer i given the same software quality increase $q_{t-1} - q_{t-1-\Delta S_i}$ and the time window size ΔS_i.

1.4 Aggregation of individual contributions

The individual contributions $\Delta q_{i,t}$ from a developer i will be aggregated into the project at time stage t as follows.

$$q_t = \sum_{i=1}^{N} (\Delta q_i + \varepsilon_i) \text{ for all } \Delta q_i \geq 0 \tag{4}$$

where Δq_i is the quality increase by the net source code contribution from the developer i and ε_i is its error factor which follows the normal distribution of $N(0, \sigma^2)$ with σ as a constant.

The net contributions below zero are discarded in order to model that the open source project code remain the same even when no contribution is made. One example of a developer with negative Δq_i in this model can be one who spent hours for open source development but was not successful producing any contribution at time t. The reason to introduce the error factor ε_i is to include the uncertainty factor that any source code patch contribution might be accepted or not into the project source tree by the project leader [20]. For the purpose of simplicity, σ^2 is set to zero for the simulation run.

2. Modeling of bounty open source project

2.1 Important variables and basic assumption

An open source project with bounty prize can be simply modeled with the following variables. First, Δq_p is the software quality improvement required from the bounty project specification. Second, ΔI is the bounty prize that will be awarded to the bounty developer once she successfully completes the bounty project. Third, Δt_p is the time duration for the bounty project.

As stated in the introduction part, bounty project developers will have less incentive to disclose the source code for the bounty project while the bounty project is going on. Two resulting negative effects on the project software quality will be that (1) sudden release of source code from the bounty projects will increase the learning cost for other voluntary open source developers working on writing derivative source code and that (2) opportunity cost to the voluntary open source project can happen from participation of the developers in the bounty project who otherwise could have kept making contributions to the voluntary open source project.

2.2 Individual decision making for bounty developers

From the maximization problem of equation (1) and its optimal solution of equation (2), the bounty prize money can be incorporated into the increase of income as $w\overline{L}+(I+\Delta I)$ rather than $w\overline{L}+I$. The income effect will result in the increased level of development hours. (with the assumption that L is normal good [21].) For a bounty developer i, the amount of increased labor hours $L^{*\Delta I}$ put into open source development will be as follows.

$$L^{*\Delta I}=\alpha\cdot\frac{w\overline{L}+(I+\Delta I)}{w}$$

2.3 Project participation decision and the modeling of bounty project cost

For a developer i, her optimal choice of open source development hours is defined to be L^{*i} given the income I and it will be increasing to L^{*i}, ΔI given the increased income of $I+\Delta I$ with the bounty prize. It is assumed that all developers can participate either in the voluntary open source project or the bounty project but not both at the same time.

Let t_0 be the time when the bounty project starts with the specified project requirement of achieving Δq_p. At the time t_0-1, each developer will estimate the time which will take for him to complete the bounty project with the current $\Delta q_{i,t_0-1}$information. The estimated time for the developer i to finish the bounty project will then be simply $\Delta q_p/\Delta q_{i,t_0-1}$. The developer j who can finish the bounty project requirement Δq_p within the shortest time will be chosen for the bounty project developer.

The increased learning cost and opportunity cost in the bounty project can be modeled as follows.

2.3.1 Learning cost for voluntary open source developers

From equation (3), it can be seen that the individual net contribution $\Delta q_{i,t}$ will decrease as the software quality Δq_t increases from time $t-1-\Delta S_i$ to $t-1$. Therefore, the learning cost model from the equation (3), with no bounty projects, can be used for the bounty project model as well.

The software quality increase from the bounty project completion can be modeled as follows. At the completion of the bounty project when $t=t_0+\Delta t_p$, the software quality will get a sudden shock or boost of Δq_p because of the bounty source code release to the public. In other words, the aggregation of individual net contributions at time $t=t_0+\Delta t_p$ will include Δq_p as well in the equation (4) such that

$$q_{t_0+\Delta t_p}=\Delta q_p+\sum_{i=1}^{N}(\Delta q_{i,t_0+\Delta t_p}+\varepsilon_i) \tag{5}$$

Suppose a voluntary open source developer i who experiences Δq_p boost within her time window, that is, $t-1-\Delta S_i\le t_0+\Delta t_p\le t-1$. Then she will experience the increased learning cost C_i caused from the software quality boost of Δq_p for the total of ΔS_i stages. This increased learning cost reflects the additional bounty project will include the increased learning cost from the bounty project that she must review.

2.3.2 Opportunity cost for the bounty project developer

During the bounty project period at t where $t_0\le t\le t_0+\Delta t_p$, the opportunity cost to the voluntary open source development for a bounty project developer j is $\Delta q_{j,t}$. That is, the individual contribution of the developer j to the open source project will be $\Delta q_{j,t}=0$ temporarily for $t_0\le t\le t_0+\Delta t_p$.

IV. Experiment results

1. Global parameter values

Based on the models proposed in the previous section, the quality improvement of open source software with respect to time is tested though the simulation. The total number of developer pool is set to $N=1000$ and the total running time is $t=1000$. α_i, β_i, and γ_i are randomly drawn from the uniform distribution between 0 and 1, creating diversity in the personal characteristics in open source developers.

The income and wage related variables are randomly chosen from uniform distributions. The hourly wage level w_i is defined to be $1 \le w_i \le 10$ for all $i \in N$. Additional income endowments I_i are in the range of $0 \le I_i \le 100$. $\overline{L_i}$, the maximum amount of labor available which can be put to either for open source code writing or for earning income, is from the uniform distribution $0 \le \overline{L_i} \le 50$. Note that it has been assumed to be that $l_i + L_i = \overline{L_i}$ for a developer i.

The developers are assumed to look back for the time duration between $t-1$ and $t-1-\Delta S_i$ while reading and learning the existing source code. When they have large amount of contributions during that period, they have to spend more time for learning. This time window ΔS_i is randomly drawn from the uniform distribution where $30 \le \Delta S_i \le 45$. That is, any developers who want to participate in the project will have to understand the source code improvements for at least past 30 stages to 45 stages in maximum.

The results from the experiment were very similar across runs, so the result from a typical representative run is shown in the following figures. In the future, a large-scale experiments will be run and the average results will be obtained for more robust findings.

2. Pure voluntary open source development

2.1 Experiment result for voluntary open source development

Given the parameter values specified in the section above, the individual optimal choices of labor hours L^* for open source development can be determined using the proposed model. One typical distribution of resulting L^* values are shown with a histogram in figure 1. When L^* is generated 100 times, the mean of L^* values is computed to be 15.7290 and the standard deviation is 1.3422. This shows consistency with the typical mean of 15.7944 from figure 1. It can be also seen that there are more number of developers who are willing to make small contributions than those who make large amount of contribution, which seems to reflect it reasonably that the number of proficient open source developers tend to be small.

Figure 2 and 3 show a typical open source project development progress based on the proposed model as time advances. It can be observed that the project software quality keeps improving linearly. Figure 3 shows in detail the graph in figure 2 for the time duration $1 \le t \le 100$. The software quality experiences a relatively steep increase at the start of the project and keeps increasing at the reducing rate. This reflects that an increased amount of software quality results in increased learning costs for the open source project developers.

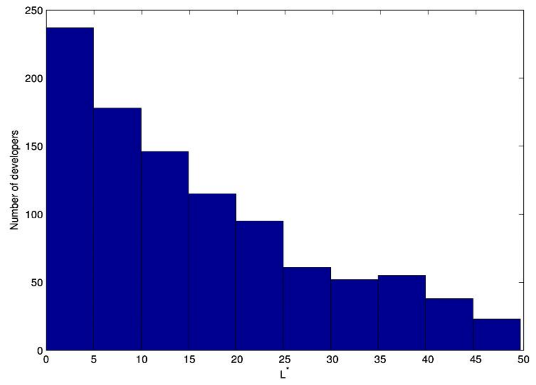

Fig. 1. One typical distribution of the optimal labor choices L^* for open source development. (Mean L^* = 15.7944, standard deviation σ_{L^*} = 12.7143)

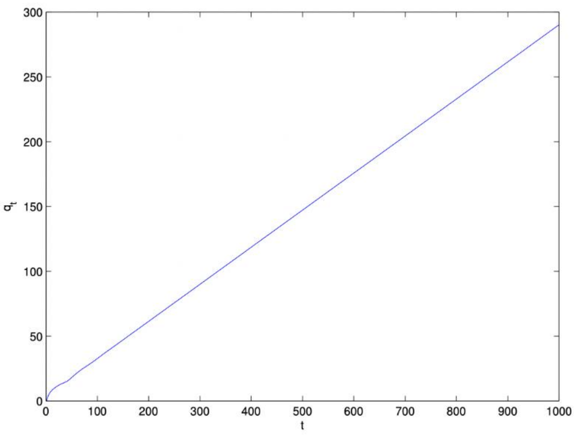

Fig. 2. Open source software quality q_t vs. time t

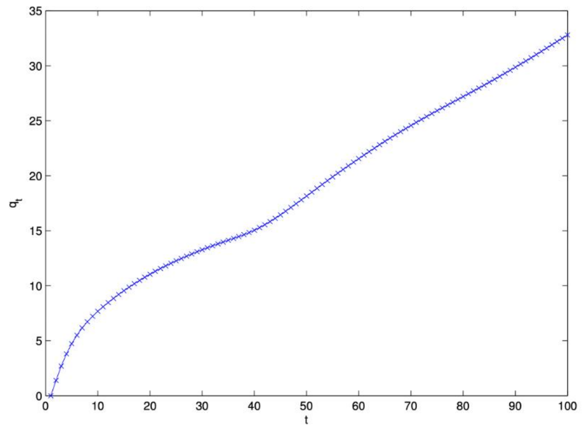

Fig. 3. Open source software quality q_t vs. time t with $1 \le t \le 100$

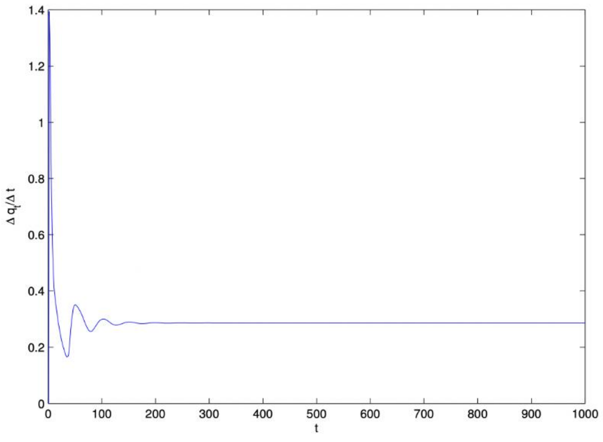

Fig. 4. Open source software quality change per unit time $\Delta q_t / \Delta t$ vs. time t

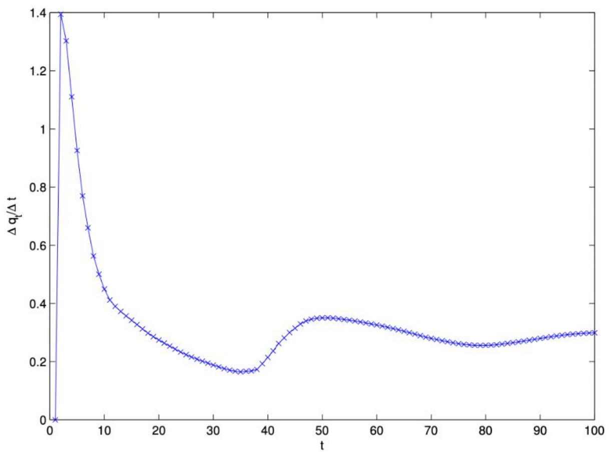

Fig. 5. Software quality change per unit time $\Delta q_t / \Delta t$ vs. time t with $1 \le t \le 100$

The effect of the developer learning cost C_i can be observed in more detail by checking the slopes of the figure 2 and 3. Figure 4 and 5 are the plots for $\Delta q_t / \Delta t$ versus t. A sudden increase in software quality can be seen at the start of the project as there are no learning costs. As more and more codes get written, we can see a drop in productivity of the participating developers. When the $\Delta q_t / \Delta t$ slope hits the bottom, it begins to increase gradually. After learning the new code influx, developers are ready to contribute to the project again and their $\Delta q_{i,t}$ increases. This cycling pattern keeps going on and converges to a value. (around 0.2859 from the graph) One interpretation of this cycle can be that the individual net contribution from the open source developers converges as well in the long run. When there are many contributions, they will spend most of their time to understand and to keep up with the contributed source code. When not, they will write more new source code contributions instead.

3. Open source development with bounty project

3.1 Parameter values for the bounty project

The bounty project start time t_0 is set to 100. A representative project leader is assumed to announce the project specification with the required quality improvement of $\Delta q_p = 3$ and the bounty money of $\Delta I = 50$ at the time stage t_0. With the announced bounty money, individual developers calculates their optimal choice of increased development labor $L^{*}{}_{i,\Delta I}$ and their individual contributions $\Delta q_{i,t} = t_0$ accordingly. Based on the $\Delta q_{i,t} = t_0$ values, each developer submits the expected time $\Delta q_p / \Delta q_{i,t} = t_0$ to finish the bounty project if she were selected to be a bounty project developer. The project leader then designates the developer who can finish the project within the shortest time period as the winner of the bounty project.

In the experiment described below, one developer with the individual contribution of 0.0273 was chosen as the best bounty developer. The time duration Δt_p for the bounty project is 110 and the software quality increase of $\Delta q_p = 3$ was added as a positive shock back to the entire project at the time $t_0 + \Delta t_p$. During the bounty project period, the $\Delta q_{j,t}$ for this bounty project developer j was set to zero for the purpose of modeling the opportunity cost.

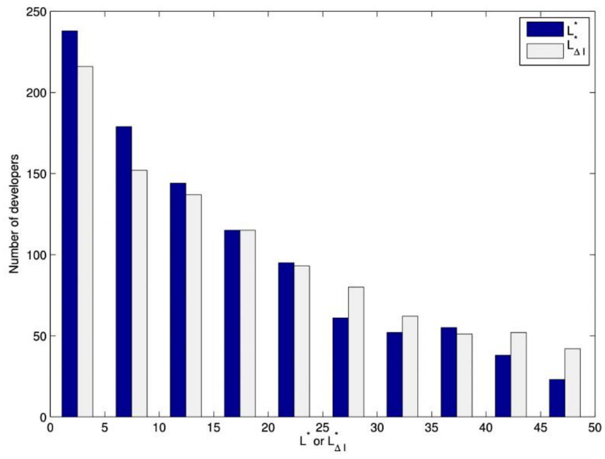

Fig. 6. The distribution of the optimal labor choices L^{*} and $L^{*}{}_{\Delta I}$. (Mean $\overline{L^{*}{}_{\Delta I}}$ = 17.6828, stdev $\sigma_{L^{*}{}_{\Delta I}}$ = 13.4759)

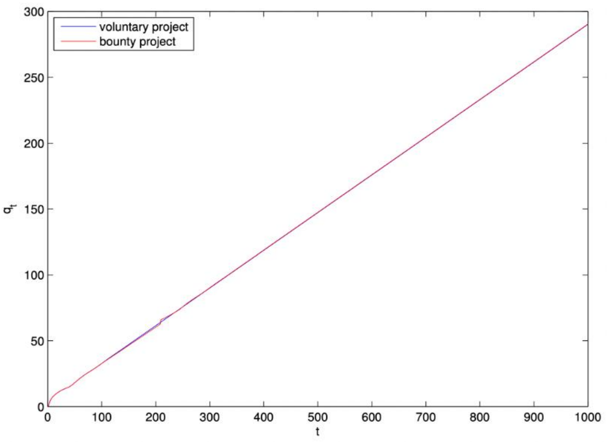

Fig. 7. Open source software quality q_t vs. time t for bounty project

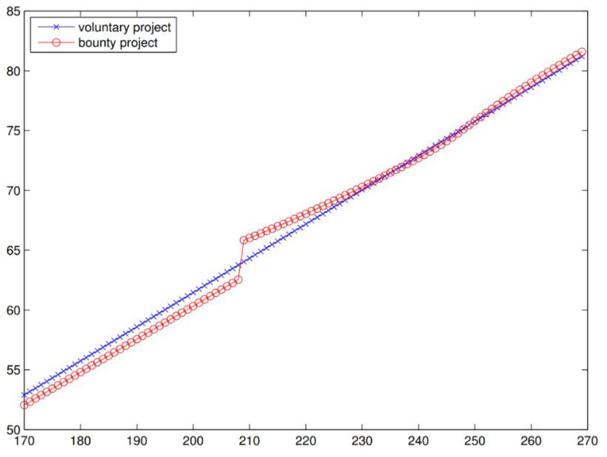

Fig. 8. Software quality q_t vs. time t for bounty project with $170 \leq t \leq 270$

3.2 Experiment result for the bounty project case

The increased optimal labor choice estimations resulting from the increased income level with the bounty prize are shown in figure 6. It can be easily identified from the histogram that the development hours for open source development are overall increased with the higher income level caused by the bounty prize money.

The mean of $L^{*}{}_{\Delta I}$ is increased from 15.7944 to 17.6828. When $L^{*}{}_{\Delta I}$ is generated 100 times for verification purpose, the mean and the standard deviation of $L^{*}{}_{\Delta I}$ values are computed to be 17.6179 and 1.3422 respectively. Note that the typical mean of 17.6828 is close to 17.6179 from the multiple data generation and that the standard deviation of 1.3422 remains the same as the previous L^* case.

As can be seen from the figure 7, and the detailed figure 8, the open source software quality q_t experiences a slight drop in its increase rate at the time t_0 and sudden boost of Δq_p at the time $t_0 + \Delta t_p$ when the bounty project is completed. After the shock of Δq_p release, the slope again becomes relatively flatter due to the increased learning cost from the quality boost. It converges to the slope of voluntary open source project graph in the long run with the cycling behavior observable in the pure voluntary open source case.

The slope information $\Delta q_t / \Delta t$ from the figure 9, 10, and 11 provides more details of what is happening inside the bounty project.

Figure 10 visualizes the opportunity cost for losing the voluntary open source developer j for the bounty project developer in more detail. When the developer j participates in the bounty project, she does not produce any contribution to the voluntary open source project. This results in the reduced $\Delta q_t / \Delta t$ for the project duration and can be identified from the drop in the graph. It begins at the bounty project start at $t = 100$ and continue to persist until the end of the bounty project at $t = 209$.

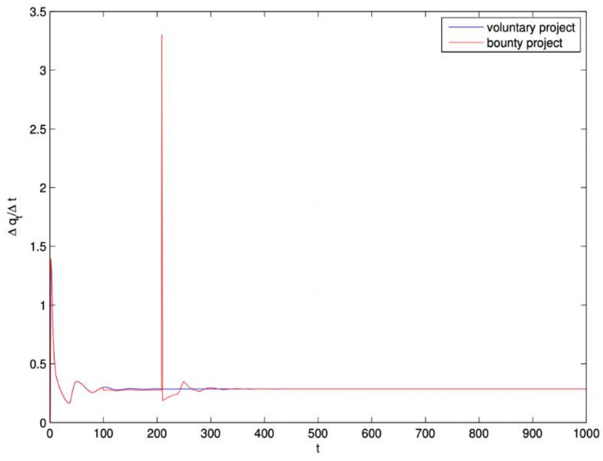

Fig. 9. Software quality change per unit time $\Delta q_t / \Delta t$ vs. time t for bounty project

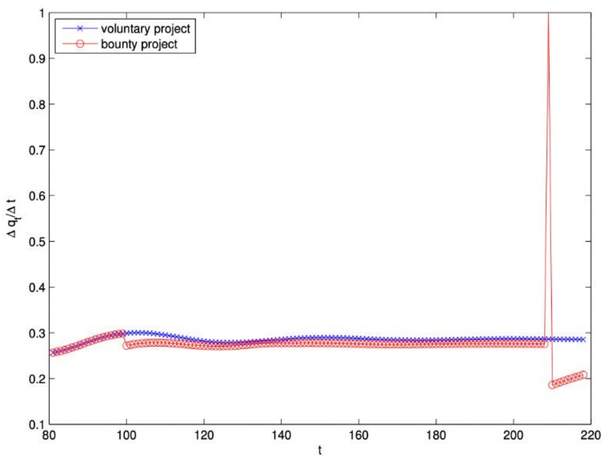

Fig. 10. Software quality change per unit time $\Delta q_t / \Delta t$ vs. time t for bounty project $80 \le t \le 220$

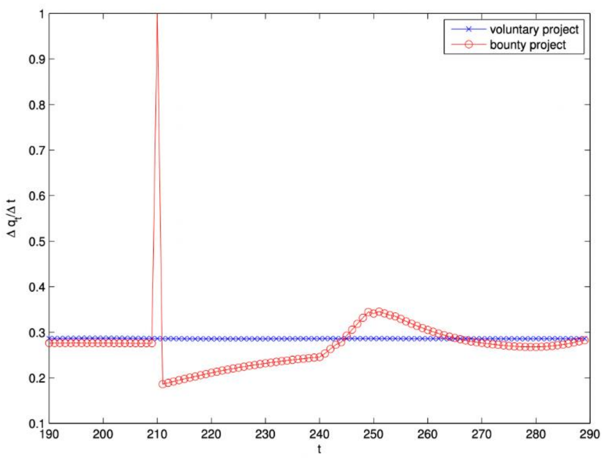

Fig. 11. Software quality change per unit time $\Delta q_t / \Delta t$ vs. time t for bounty project $190 \le t \le 290$

At the end of the bounty project, the software quality gets a sudden boost from the bounty project code release at $t = 210$. Figure 11 shows the effect of bounty project code release. When faced with the bounty project contribution shock, the other voluntary open source developers have to incur an increased amount of learning cost and the rate of contributions experiences a drop. Then the contributions begin to slowly catch up as time goes on because of the reduction in the learning cost.

Note that the time window ΔS_i plays a role in this catch-up process. In this experiment, ΔS_i is randomly drawn from the uniform distribution where $30 \le \Delta S_i \le 45$. It means that all the developers consider the software quality improvement at least for the past 30 stages to determine their learning cost. In the graph, after 30 stages where $t = 240$, the slope of increase of $\Delta q_t / \Delta t$ gets steeper. This can be explained that the quality improvement shock from the bounty project begins to move out of the scope ΔS_i for some developers with their ΔS_i values being around 30. As time advances, the number of developers with the quality improvement shock staying outside of their ΔS_i is bound to increase and the overall contribution from the developers will increase too. After this increase reaches a maximum, the learning cost again begins to take effect and $\Delta q_t / \Delta t$ begins decreasing. It then forms a cyclic behavior and converges to a point as was similar to what is observed in the pure voluntary open source case.

The resulting software quality values at the end of the experiment time period $(t = 1000)$ are as follows. $q_{t=1000}$ becomes 290.1886 for the voluntary open source development and $q_{t=1000}$ reaches 290.4262 for the bounty project. Considering that the bounty project requirement for the software quality improvement was $\Delta q_p = 3$, the actual increase in software quality from the bounty project is only 0.2376, which is about 7.92% of Δq_p.

V. Conclusions

5.1 Findings and implications

It can be identified from the experiment that a large portion of actual software quality improvement effect expected from bounty project is taken away by the increased learning cost of the other voluntary open source developers who try to catch up with the released bounty project improvements, and the opportunity cost of the bounty project developer who otherwise could have participated as a voluntary open source developer. As specified in a typical experiment result, only 7.92% (0.2376) of $\Delta q_p = 3$ boost from the bounty project completion was reflected to the entire project software quality in the long run.

Two variables surface as major factors that need further research effort to reduce the total cost. First, for learning cost, the quality improvement from the bounty project Δq_p is the major factor that reduces the learning cost C_i for voluntary open source developers. Since Δq_p can be arbitrarily specified by the open source project leaders, there is a possibility that having appropriate level of Δq_p will result in the minimized learning cost. One possible treatment for further experiment test is to compare the software quality improvement from one big shock of Δq_p with that from M small shocks of $\Delta q_p / M$. This can be interpreted in real world as comparing the case of having a big bounty project requirement with dividing the total bounty project requirement into many small sub-projects.

Second, for opportunity cost, the bounty project time duration of Δt_p is found to be the major factor. Since the individual contribution of $\Delta q_{j, t_0 - 1}$ for the bounty developer j will stay zero during the bounty project duration Δt_p, the total opportunity cost that the open source project has to incur will be $\sum_{t = t_0}^{t_0 + \Delta t_p} \Delta q_{j,t}$. The model presented in this paper assumes that all the bounty developers report at the shortest time period needed to finish the bounty project and they are implicitly competing against one another in terms of submitting the shortest project completion period estimation.

There are two ways for the project leaders to reduce Δt_p. It can be seen from the equation (3) that having lower quality improvement requirements for Δq_p will be one way for Δt_p reduction and increasing the bounty prize ΔI will be the other. In certain cases where the project requirement of Δq_p can be hard to change, giving more bounty prize money will be the practical way to reduce Δt_p, resulting the reduction in the opportunity cost as well. Thus, finding the optimal bounty prize ΔI with the consideration of minimizing the opportunity cost to the project can be important and needs to be explored as another future research question.

In addition, it will in the best interest for the open source project to select developers with high income effect for the bounty project. In other words, targeting developers who are not currently making a large contribution to the voluntary open source project, but willing to increase their contribution greatly when awarded the bounty prize will be the best for the project. This is because having current low participation in the voluntary project will lead to the small amount of opportunity cost from losing $L^{*\ j}$ when the developer j starts off with a bounty project. And the large increase in the bounty project contribution from $L^{*\ j,\Delta I}$ will result in the large amount of software quality boost which would have not been possible without the bounty prize. For example, assume a developer j is highly motivated by the income effect from the bounty prize. Then selecting her to provide the bounty project leads to a small opportunity cost (as she was not working on the open source project much as a voluntary developer) with a high software quality boost accruing to the project from her bounty project work. If the selected bounty developer is already making a large contribution to the bounty project, the opportunity cost of losing her voluntary contribution might be high and erode from any increased level from the bounty project quality boost.

The current open source bounty project specifications with practices of finding the most proficient bounty developers seems to not have any methods or mechanisms to target the developers with current low contribution but potential high contribution. In other words, there is a possibility that the competition might lead to a bounty prize winner with high proficiency but also with high opportunity cost to the project. One topic for future research is exploring how to implement possible identifying mechanisms to identify the developers with *high bounty income effect*.

5.2 Further model extension

One limitation of the current model is that the total number of open source developers are fixed to N for the purpose of model simplification. In real world, however, network effects [15] (and two-sided markets [23, 27]) are known to be an important characteristic of software goods where more number of users increase the value of the software, attracting even more number of users. Considering this, a possible extension of our model can be to target a startup project with low developer critical mass, and add a bounty project shock, and then analyze the parameters that help those projects get more participating developers and produce high quality open source software.

We anticipate that the results from this study can eventually provide guidelines to the open source community on the best use of their limited resource, both monetary and voluntary.

REFERENCES

[1] I. Hahn et al. “Economic incentives for participating in open source software projects.” ICIS 2002.

[2] A. Friebergerhouse. “Will work for free.” Forbes ASAP Supplement, pp. 77-80, February 2001.

[3] Y. Ye and K. Kishida, “Toward an understanding of the motivation of open source software developers,” ICSE'03: Proceedings of 25th International Conference on Software Engineering, pp. 419-429, 2003.

[4] E. Raymond. Homesteading the noosphere: http://www.catb.org/~esr/writings/cathedral-bazaar/homesteading

[5] A. Schiff. “The economics of open source software: A survey on the early literature.” Review of Network Economics, 1, March 2002.

[6] E. Haruvy, A. Prasad, and S. Sethi. “Harvesting altruism in open source development.” Journal of Optimization Theories and Applications, 118(2):381-416, August 2003.

[7] M. Andreasen, H. Nielsen, S. Schrøder, & J. Stage, J. “Usability in open source software development: opinions and practice.” Information technology and control, 35(3), 2006

[8] Bounty Source, https://bountysource.com
[9] J. Zhou, S. Wang, C. Bezemer, Y. Zou, and A. Hassan, "Bounties in Open Source Development on GitHub: A Case Study of Bountysource Bounties", arXiv:1904.02724, 2019.
[10] Drupal bounty program, https://www.drupal.org/drupalorg/docs/build/drupalorg-bug-bounty-program
[11] Internet Bug Bounty: High severity vulnerability in Apache HTTP Server could lead to RCE, https://portswigger.net/daily-swig/internet-bug-bounty-high-severity-vulnerability-in-apache-http-server-could-lead-to-rce
[12] The Internet Bug Bounty program, https://www.hackerone.com/internet-bug-bounty
[13] L. Breidenbach, P. Daian, F. Tramer and A. Juels, "The Hydra Framework for Principled, Automated Bug Bounties," in IEEE Security & Privacy, vol. 17, no. 4, pp. 53-61, July-Aug. 2019, doi: 10.1109/MSEC.2019.2914109
[14] J. Lerner and J. Tirole. "Some simple economics of open source." Journal of Industrial Economics, 52(2):197–234, June 2002.
[15] M. Katz, and C. Shapiro. "Systems Competition and Network Effects." Journal of Economic Perspectives, 8 (2): 93-115. 1994 DOI: 10.1257/jep.8.2.93
[16] J. Johnson. "Open source software: Private provision of a public good." Journal of Economics Management and Strategy, 11:637–662, Winter 2002
[17] A. Schiff. "The economics of open source software: A survey on the early literature." Review of Network Economics, 1, March 2002
[18] R. Thaler. "Anomalies: The ultimatum game." Journal of Economic Perspectives, 2(4):195–206, Fall 1988
[19] G. Becker. "A theory of the allocation of time." The Economic Journal, 75:493–517, 1965
[20] P. Rigby, D. German, and M.-A. Storey. "Open source software peer review practices: a case study of the apache server." In Proceedings of the 30th international conference on Software engineering (ICSE '08). Association for Computing Machinery, New York, NY, USA, 541–550. 2008 https://doi.org/10.1145/1368088.1368162
[21] Normal goods definition, https://www.wallstreetmojo.com/normal-goods/
[22] Income effects, https://corporatefinanceinstitute.com/resources/knowledge/economics/income-effect/
[23] M. Rysman. "The economics of two-sided markets." Journal of economic perspectives, 23(3), 125-43. 2009
[24] G. Becker, "A theory of marriage." Economics of the family: Marriage, children, and human capital. University of Chicago Press, 299-351. 1974
[25] M. J. Rizzo, and G. Whitman. "*Escaping paternalism: Rationality, behavioral economics, and public policy.*" Cambridge University Press, 2019.
[26] N. Economides, "Network externalities, complementarities, and invitations to enter." European Journal of Political Economy12.2 211-233. 1996
[27] O. Hinz, O. Thomas, and S. Bernd. "Estimating network effects in two-sided markets." Journal of Management Information Systems 37.1 (2020): 12-38. 2020
[28] A. I. Chittilappilly, L. Chen and S. Amer-Yahia, "A Survey of General-Purpose Crowdsourcing Techniques," in IEEE Transactions on Knowledge and Data Engineering, vol. 28, no. 9, pp. 2246-2266, 1 Sept. 2016, doi: 10.1109/TKDE.2016.2555805.
[29] H. R. Varian. "*Microeconomic Analysis.*" Third edition, Norton, New York, 2009.

Authors

Jung-woo Sohn received B.S. and Ph.D. degrees in Aerospace Engineering and Information Sciences and Technology from Seoul National University, Korea in 1998 and The Pennsylvania State University, USA in 2013, respectively. Dr. Sohn joined the faculty of the Department of Software at Yonsei University in 2021. He is currently an Assistant Professor in the Department of Software, Yonsei University. He is interested in market system design, trading agent design, and agent-based simulations.

Yohan Ko received B.S. and Ph.D. degrees in Computer Science from Yonsei University, Korea, in 2012 and 2018, respectively. Dr. Ko joined the SSD firmware engineer at SK hynix, Korea, in 2018. He is currently a Professor in the Department of Software, Yonsei University. He is interested in dependable computing, DNN accelerator, and embedded systems.

Younguk Yun received B.S., M.S. and Ph.D. degrees in Electronic Engineering from Kwangwoon University, Korea, in 2014, 2020, respectively. Dr. Yun joined the faculty of the Software at Yonsei University in 2021. He is currently an Assistant Professor at the Department of Software, Yonsei University. He is interested in Indoor positioning, Healthcare and Autonomous driving with AI signal processing of RADAR, camera, IMU and Various IoT sensors.