I Want to Invest in An Open Source Project: Shall I Hire Insiders or Outsiders?

Vahid Etemadi[1], Jesús M. González-Barahona[2] and Gregorio Robles[2]

[1]Shiraz University of Technology, Shiraz, Iran
[2]GSyC/LibreSoft, Universidad Rey Juan Carlos, Madrid, Spain

Abstract

Many companies and institutions want to contribute to Open Source projects in order to ensure its maintenance and evolution. One way of doing it is to contract developers to work on the project. In this situation, a frequent question is whether it is better to hire developers who have already collaborated with the project (i.e., *insiders*, usually volunteers that are active in the project) or if it is better to hire external developers (*outsiders*, with no previous relation to the project). An analogy to the logic behind the Insider/Outsider separation can be found in the well-known onion model, where they refer to insiders as core developers. The goal of this paper is to simulate and compare the performance (in terms of the time and the cost) of a company hiring a number of developers in two different scenarios, where the hired developers are a) insiders, and b) outsiders. We will therefore use a method used in the research literature before to simulate the behavior of software projects and determine the best strategy. Developers are therefore assigned bugs to work on in every round, being a round a given timespan (e.g., three months) or a given number of bugs are received (e.g., 50 bugs). During each round, a Non-dominated Sorting Genetic Algorithm II (NSGA-II) was used to evaluate the candidate assignments in terms of the *time* and the *cost*. With the current settings, the results show that there is no significant difference between the two scenarios in the terms of the two performance metrics considered. Upon the results so far, it seems we cannot favor one of the scenarios over the other one, based on the time and the cost. We believe further statistical analysis on the obtained data could power up our research in the future.

Keywords

OSS team, Core developers, contributors, digital twins

1. Introduction

Free/Open Source Software (FOSS) has lived a major transformation since its early days. It started as a movement of communities of volunteers, but since 20 years ago it has been gaining attention from the software industry [1]. Nowadays, it is not infrequent to see professional developers hired by companies and institutions interested in driving the FOSS project forward.

Companies and institutions have different ways to collaborate with and contribute to FOSS projects. One of them is to hire developers that devote time to the project as part of their

BENEVOL'21: The 20th Belgium-Netherlands Software Evolution Workshop, December 07–08, 2021, 's-Hertogenbosch (virtual), NL

v.etemadi@sutech.ac.ir (V. Etemadi); jgb@gsyc.urjc.es (J. M. González-Barahona); grex@gsyc.urjc.es (G. Robles)
https://vahidetemadi.github.io/ (V. Etemadi); http://gsyc.es/~jgb (J. M. González-Barahona); http://gsyc.urjc.es/~grex (G. Robles)
0000-0003-1188-5708 (V. Etemadi); 0000-0001-9682-460X (J. M. González-Barahona); 0000-0002-1442-6761 (G. Robles)

professional tasks. This is especially interesting in those projects that are already mature and have difficulties attracting new collaborators. These projects have become frequently an essential part of an infrastructure (e.g., OpenSSL or Log4J), but are not seen as *attractive* by volunteers, probably because the tasks to be done are mainly of perfective or corrective nature and the technologies are seen as old.

A possibility for making the project sustainable is to hire volunteers already working on the project (*insiders*) and ensuring their contribution by making them professionals. This has the advantage that they are already familiar with the inner functioning, the process, the tools, the code, and the people. Another possibility is to hire external developers (*outsiders*) and assign them to work on the project. Although these developers have to get to know many aspects of the project, including its source code, it has the benefit that the community grows, at least in number of participants, if we assume that volunteer developers won't leave the project.

In this paper, we try to devise and evaluate these two scenarios, and find out if one outperforms the other. For doing so, we will take data from two FOSS projects that have been previously used in the research literature to compare bug assignment strategies. By using a simulation, we would like to see how the results are when a set of already existing developers devote more time to the project (i.e., they were volunteers and have been hired to work now full-time on the project) or when several new developers are included in the project (i.e., external developers are hired to work full-time on the project).

Our contribution is to perform an exploration study on the performance evaluation of different perspectives in creating bug fixing teams in terms of the time and the cost over several successive round. We basically rely on several research publications that formulate bug fixing tasks as a two-objective problem, and take advantage of NSGA-II to solve it [2, 3]. We use a real data-set which goes through a simulated context to assign the tasks to developers. The model is then applied to the software maintenance and evolution phase of the software projects, as this has been reported to be the phase where it is harder to attract volunteers to the project [4]. In particular, the model takes the change requests and supports the maintainers to narrow down their plannings for evolving the software.

This paper is structured as follows: Section 2 offers a short explanation related to the concepts targeted in this report. Next, we formulate the research questions and present a short description on their goal in Section 3. We offer some insight of the simulation process in Section 4. Section 5 iterates over the steps followed to operationalize our idea. We provide the result obtained so far in Section 6. Finally, Section 7 intends to discuss the results and the potential impact of our research.

2. Related Work

Although FOSS projects are self-organized (some authors even argue that they follow stigmergic principles as found for instance in ant colonies [5]), developers in FOSS projects can in general be categorized as peripheral and core developers following an onion structure [6]. Core developers are usually those who have a key role in the project in opposition to peripheral ones whose contributions are minor. These two groups of developers are supposed to be different in terms of the time they commit and devote to the project. In recent times, many core developers are

full-time paid employees from companies and other institutions, while peripheral contributors remain volunteers [7]. Even though volunteers do not devote much time to the project, their overall contribution is very valuable and projects strive to have a healthy community that attracts newcomers. In addition, it should be noted that this picture is not static, but evolves over time; it has been observed that there are generations of core developers that usually last three to five years [8]. It is also known that the time that it takes from the first contribution to become a core developer is of about 24 months in the mean for volunteers, but only 6 months for a paid developer [9].

A questionable, but interesting fact, given these models, is about the triplet events of joining, staying in, and abandoning projects (i.e., attraction, retention and turnover). Each of these events, that could take many forms, impacts the project success metrics [10]. This research work aims at focusing on the *developer attraction* that affects the composition of the team.

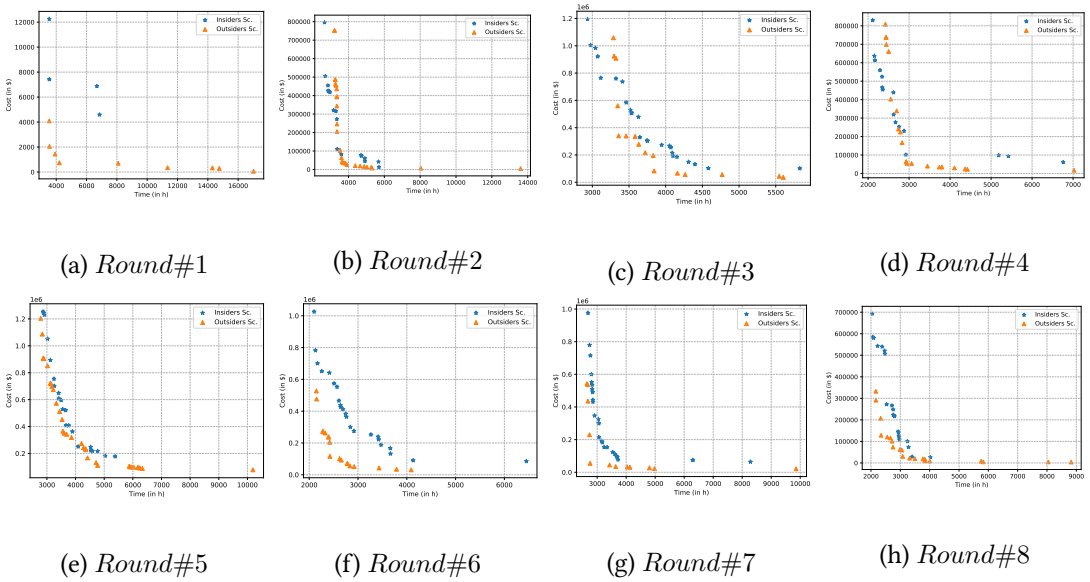
It could be hypothesized that following a particular type of attraction paradigm could come up with different consequences. These consequences could reveal themselves in form of the project performance metrics. Among all, time and cost are well-known problem-specific metrics which are assumed to play an important role in practitioner's opinion to switch to a particular team structure.

To measure these two metrics, we need to look at the activities that consume time and need resources. Software maintenance and evolution is known to be the most costly part of a software project, and bug-fixing tasks are a key part in maintaining software. Thus, we think that to be able to evaluate different strategies in team composition we need to evaluate the time and the cost of bug-fixing tasks.

3. Research Questions

In this Section, we present the two RQs that we would like to answer in this research. Basically, our aim is i) to compare the two scenarios in terms of their performance in every single round of maintenance, and ii) to offer a round-wise time and cost comparison for the two options. In detail:

- RQ1: In terms of the final Pareto-fronts, how does each scenario perform in comparison to the other one in terms of the objectives (time and cost)?
  The motivation behind this RQ is to allow readers to observe and compare the final solutions provided by simulating each scenario. Each Pareto-front includes non-dominated assignments in terms of time and cost. Non-dominated assignments have the feature of not strongly dominate other solutions in the set (e.g., being exactly lower and not equal). The goal then will be representing sample Pareto-fronts (coming from an iteration in running), in terms of the time and the cost, for each single round.
- RQ2: From an all-in-one round-wise representation, what are the best solutions offered by each scenario in terms of the minimum *Time*?
  For this RQ, a round-wise representation of minimum time over successive rounds is required. All-in-one means combining all rounds together and come up with a single comparison. This helps to avoid bias due to any particular condition of one or several rounds. That the main difference between RQ1 and RQ2 is that we only focus on the



(a) *Round*#1 (b) *Round*#2 (c) *Round*#3 (d) *Round*#4

(e) *Round*#5 (f) *Round*#6 (g) *Round*#7 (h) *Round*#8

Figure 1: Pareto-fronts over rounds for *Insiders* and *Outsiders* developer participation as the core. Each scatter plot represents the non-dominated solution in the objective space in terms of the time and the cost (JDT project)

solution with minimum time, while for the overall view we do it for both time and cost. In the future, we would like to include other criteria such as minimum cost.

4. The Assignment Model

So far, we have chosen time and cost as the two substantially relevant metrics for the evaluation. These are computed for every single candidate assignment. This means that we need a computational search viewpoint to find the optimized assignment(s). We use a search-based software engineering (SBSE) [11] approach for bug-fixing task assignments, as it has already been done in several previous works [3, 2]. In this model, the problem is defined as a two-objective problem whose fitness function for an assignment S is defined as:

$$Fitness(S) = F(Time, Cost) \tag{1}$$

In [3] and [2], the authors explain how these two objectives contradict each other. The formal model in Eq. 1 is applied on every candidate assignment. The time variable in this equation is computed based on i) the estimated effort required to have task done and ii) the productivity of the assigned developer. For further information, we refer to Section 3.2 in [3]. Then, since we assume that we have access to the hourly wage of a particular developer (we know that it could differ from culture to culture), we are able to simply multiply time to wage to obtain how much that particular developer should be paid. This process are supposed to take place for all the candidate assignments. This is the core of a SBSE approach, in this case applied to the

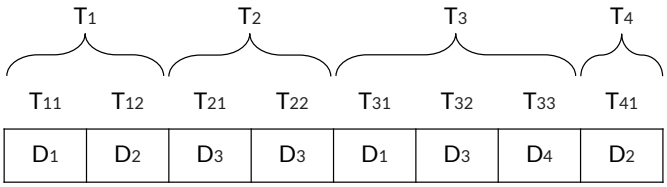


Figure 2: Representation of assigning tasks (bugs) to developers. Indices in the this vector are the sorted tasks by their prerequisite order taking code dependencies into consideration.

assignment of tasks. We believe an SBSE paradigm for bug fixing task assignment is a robust solution that offers advantages over the other approaches (e.g., avoids developers becoming overloaded with many bugs). For example, there are similar approaches with the mission of finding the best assignee that belong to the category of information retrieval and machine learning. These two types of approaches take advantage from popular techniques, however, usually are expected to underestimate if the project has to handle concurrent fixing tasks and only a few number of fixers are available. In our model, we specifically rely on NSGA-II [12] to explore the search space. NSGA-II comes from the family of Multi-Objective Evolutionary Algorithms (MOEA), and basically follows the principles of a Genetic Algorithm (GA). NSGA-II is well-suited for the problems with more than one objective (as in our case, with time and cost). Typically, during each iteration of the algorithm, following steps are taken:

1. Create a population of candidate assignments. When creating each assignment, core developers take a 100% chance to be nominated for fixing bugs compared to volunteer developers who only have a 10% chance. This tries to model core developers to be full-time developers (with a 40-hour week), while volunteers devote in the mean 10 times less (i.e., only 4 hours per week [13]).
2. Initiate the evaluation process to measure the objectives.
3. The generated solutions are ranked upon a method of choice, and a selection process is started.
4. Genetic operators are applied to the selected solutions to create next generation of candidate solutions.
5. If the maximum Number of Fitness Evaluation (NFE) is met, the algorithm stops and the non-dominated solution is extracted as one of the Pareto-optimal solutions.

Since we are using a genetic algorithm for evaluation, every assignment takes a shape as shown in Fig. 2. In this figure, the set of bugs $T = \{T_1, T_2, T_3, T_4\}$ is broken down to associated subtasks to be then assigned to a developer set $D = \{D_1, D_2, D_3, D_4\}$. Fig. 2 is only an example assignment with the flexibility of becoming bigger (or lower) in terms of number of tasks and available developers. To sum up, our model is able to take a fixed number of bugs and developers and offer a Pareto-optimal set of the best assignments as the fixing plan.

Table 1
Stats of the datasets, including two Eclipse components, JDT and Platform.

Dataset	# of Bugs Included	# of Packages	Time Interval
JDT	240	12	2004 - 2006
Platform	240	12	2002 - 2004

5. Operationalization

To answer the RQs, we performed several experiments with data obtained from a real context. We have therefore followed a simulation-based experimentation [14] for this purpose. Following a simulation-based study might introduce some limitations. All the experiments rely on a *real* data-set (including bugs features and developer properties), as provided by [2]. We offer the implementation of the scenarios in the replication package under the directory *DeveloperAttraction* directory [1].

Given our research questions, two scenarios were devised. We call these scenarios *Insiders*, where no external developers are added to the project, and *Outsiders*, where additional external developers are assumed to participate in fixing efforts. These two scenarios share similarities, as well as some differences. In the next subsections, we elaborate on the similarities and differences in the implementation.

5.1. Dataset

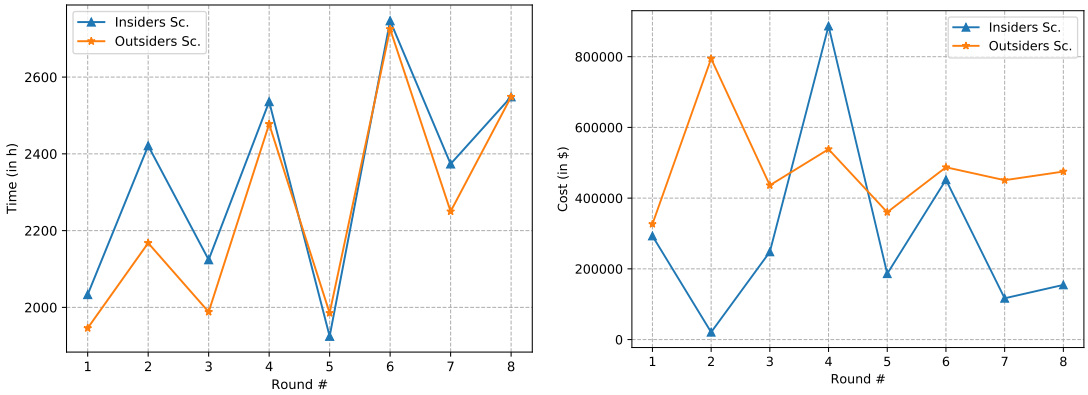
To fed our model, we lend a preprocessed dataset of bugs (including the packages that should be modified and the effort required to fix a bug) and developers from the former study by Karim et al. [2]. The dataset contains a bug-set from JDT and from Platform, two components of the Eclipse project. We consider a project that has a total number of 240 bugs; however, this is expected to be extend to more projects and more number of bugs in future work. Table 1 offers a short description of this dataset. The bugs, the relevant assignee, and other features are available in the Bugzilla issue tracking system as well.

Each scenario has access to a developer pool of core and peripheral developers. It is supposed that the projects have 20 developers before hiring new ones. Both scenarios have 5 core developers and engage 5 more (who are assumed to be a) already involved volunteers or b) external professionals, depending on the scenario). So, each scenario will have 10 core developer in total. However, the total number of developers will keep being 20 (in the case already involved volunteers are hired) or 25 (if, in addition to the 20 original developers, 5 new developers are hired).

5.2. Shared Features

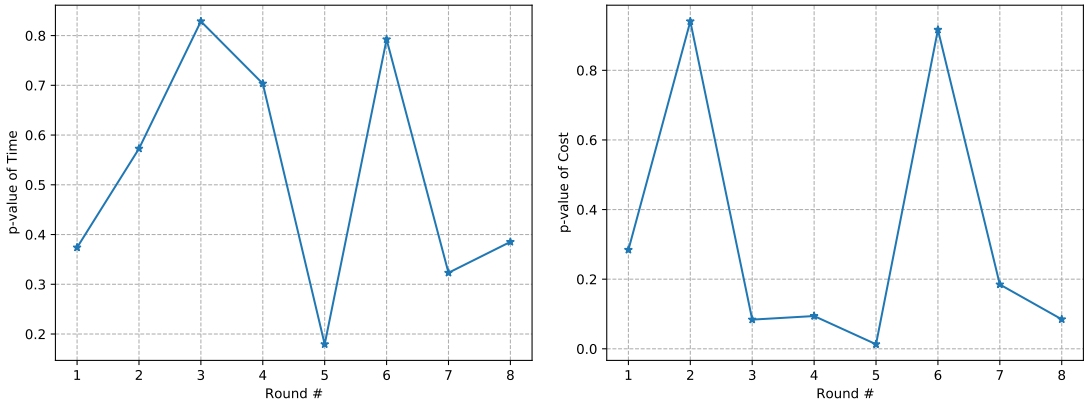
For each scenario, a round-wise bug fixing procedure is considered. During each round, a fixed number of bugs are assumed to be assigned to the members of the team. In terms of team composition, each scenario has the same number of core developers. We assume core

[1] Available at https://github.com/vahidetemadi/SCM_TA/



(a) (b)

Figure 3: (a) Solution with minimum time, and (b) Associated costs for the selected solutions of each round (JDT project)



(a) (b)

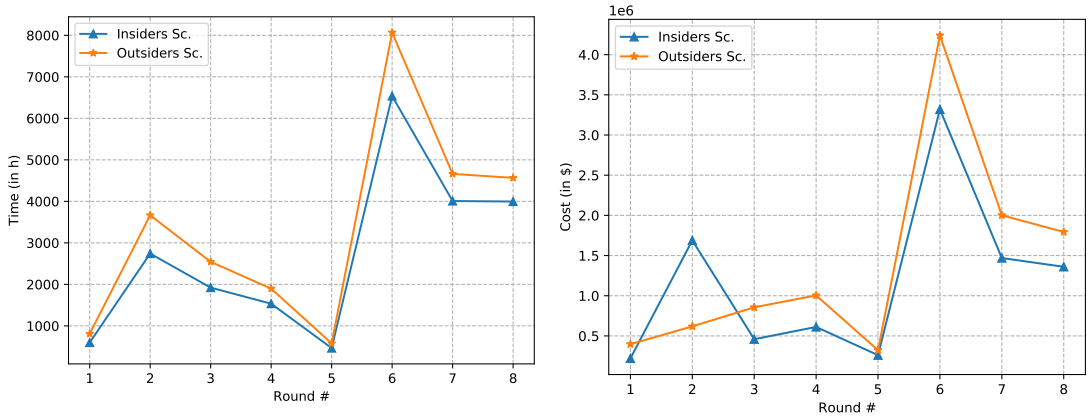
Figure 4: p-value for (a) Solution with minimum time, and (b) Associated costs for the selected solutions of each round (JDT project)

developers' expertise is the same in both scenarios. Peripheral (or volunteers) developers devote less time to the project, and are expected to contribute to it for free.

5.3. Differences

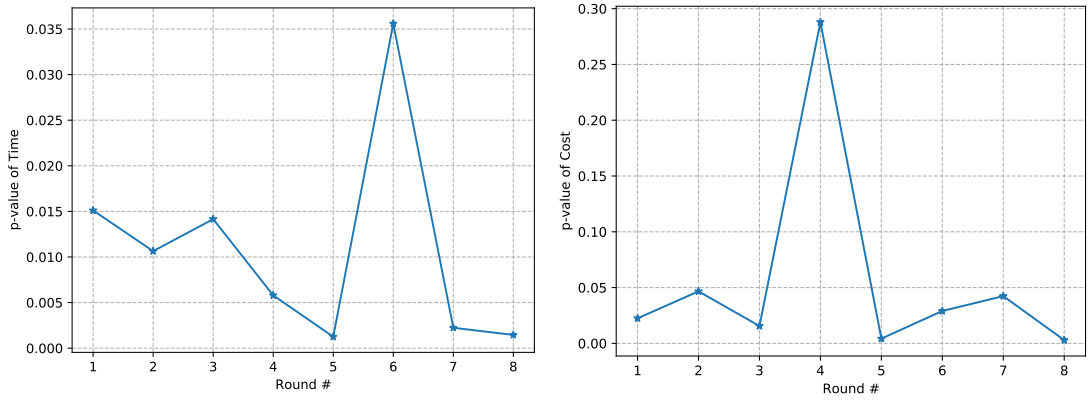
The difference among scenarios is in the number of volunteer developers, as in the *Insiders* scenario some of them have become core developers by means of being hired to work full-time on the project. Hence, the *Outsiders* scenario has a *higher number* of volunteers (free contributors).

With the current settings, the additional contributors are only 20% of the total number of the *Outsiders* scenario (5 developers).



(a) (b)

Figure 5: (a) Solution with minimum time, and (b) Associated costs for the selected solutions of each round (Platform project)



(a) (b)

Figure 6: p-value for (a) Solution with minimum time, and (b) Associated costs for the selected solutions of each round (Platform project)

6. Preliminary Results

In this section, we provide the answers to our RQs.

6.1. Answer to RQ1

The preliminary results for the eight rounds are represented in Fig. 1 (the # of rounds is a function of the # of bugs in our settings, meaning it is not considered as a limitation). The scatter plots visualize the final solutions' time and cost, extracted from the Pareto-fronts. These solutions are non-dominated, meaning none of them strongly dominates the other one in terms of time and cost. These results show a very close performance in terms of the objectives for

the two scenarios. It is interesting to notice that the minor differences in performance repeat themselves over all the rounds.

6.2. Answer to RQ2

As discussed earlier, the experiment is run for each round, resulting in a set of non-dominated solutions that are available to be selected. This non-dominated set includes the solutions that one of them needs to be selected by the owner. A possible criterion for selecting a solution is based on minimum time. So, let's select those solutions (for each round) with minimum time as the best one (for each scenario) and compare them.

Similar to what was presented as answer to RQ1, we provide a full-round-wise representation of the selected solutions. Full-round-wise takes the solution with minimum time at the end of each round and compares two scenarios. For those solutions with minimum time, the round-wise representation of the cost is offered too. Fig. 3 illustrates the solutions with minimum time and the associated costs for JDT project. Besides, Fig. 3 offers the results for project Platform.

In addition to the answers to the RQs, we would like to clarify how these results are aligned with the motivation of our work. Each sub-figure in Fig. 1 explains what a practitioner would expect from his investment on the *Insiders* and *Outsiders* scenario. Imagine a project received 30 bugs to be assigned, and the financial costs are to be taken into account. Round #1 in Fig. 1 (a) compares the best solutions obtained from our model in terms of time and cost. Clearly, if we compare the solutions in their objective space point-by-point, nothing that shows significant dominance is inferable. If we do the same exercise for the 8 rounds of assignment (captioned as Rounds #1 to 8), we find a similar situation. In fact, we believe a round-wise assignment view allows practitioners to generalize their conclusion while looking at the whole process. The purpose of Fig. 3 is exactly to offer this whole view. The difference we have found is marginal, both in terms of the cost and the time, regardless of which assignment is selected.

To avoid the effect of randomness in our results, we decided to report on the p-value and effect size of the obtained results too. Being able to statistically enrich the comparison gives practitioners the chance to better judge the results. Typically, we needed to state this hypothesis that "there is a not significant difference between two scenarios in terms of the time and the cost". Then, we computed the p-value of the time and the cost for these two scenarios, and ended up with the results shown in Fig. 4. As the results suggest, we have to accept our null-hypothesis that states that there is no significant difference between both projects, particularly in terms of the time and the cost. In other words, it means hiring Insiders or Outsiders would not make a big difference for the project.

7. Discussion and Impact

Based on our preliminary results, project stakeholders will be able to compare the performance of the two scenarios and understand that there is not a big deal in favoring one over the other, at least if we only considers time and cost. We are aware that such a conclusion requires a stronger and broader experiment. Other factors might have an impact in the scenarios, and have not been taken into account. For instance, we have not considered the problems that

newcomers have when joining a project [15], although previous research has shown that the joining process for professional developers is relatively fast [9].

Relying on a simulated research work always introduces some threats and limitations. Finding a perfect reflection of the context in this regard is a tough task. However, we followed some principles to bridge this gap. For instance, all the input data-sets are from real FOSS projects.

Re-running the simulated process also helps with facing randomness that might exist in the simulation, as we considered in our implementation. Another limitation is related to the generalization of the conclusion that we plan to mitigate by including other projects. More statistical test also should be used to offer better insights too, like we have done in a previous work [3].

Our recommendation to offer a simulated implementation of the real maintenance phase of FOSS projects could be considered as an application of *digital twins* for FOSS projects [16]. So far, we have focused on the attraction events and in the bug fixing process to be mirrored in their twins. However, it could be extended to other activities in maintenance and evolution as well. This twin could sit next to the real project and assist the community in taking decision regarding team arrangement and where to locate resources more efficiently. In our case, both scenarios are sort of a digital model that simplifies real process for hiring developers for FOSS projects.

8. Threats to Validity

In this Section, we intend to shortly list the threats to validity and what our strategies have been to mitigate them.

8.1. Construct validity

In this research, we planned to rely on the constructive theoretical and practical techniques that have already been proved to be reliable. For instance, there might be threats in terms of using an SBSE paradigm for modeling bug fixing task assignment. However, we are able to see that our idea is not new and has already been discussed in former studies [3, 17, 2].

8.2. External validity

Our research results might be questioned in terms of its applicability to other, similar projects. At this stage, we only focus on two specific projects, and this raises the concern of not being able to end up with a general conclusion. We accept this is a big concern, and we are planning to include more number of projects to alleviate it.

In this research, we only focused on the bug fixing tasks. That might question our findings for being only specific to a particular category of post-release issues (bug fixing tasks). However, we hypothesize that the results could be the same for all kinds of change requests, since task assignment engine treats all kind of task the same, except when they are labeled with a particular priority.

8.3. Internal validity

We presented two RQs which are aligned with the purpose of the study. To answer these questions, we followed the standard way of performing an empirical analysis. We relied on verified techniques (see [3, 2, 18]) to offer a digital model of the environment, rerun the assignment algorithm for several times, and tried to offer an statistical test in an effort to avoid potential biases.

8.4. Conclusion validity

We tried to be very careful about the final results which are dependent to the flow of activities in a planned analysis. The validity of the results allows to ensure a clear conclusion. We believe the flow of the current analysis, which is enriched with previous relevant studies, is likely to avoid potential threats.

A key concern regarding both scenarios is introduced when the project might be threaten by the possibility of developer turnover or heterogeneity. In our former study [17], we proposed a solution to alleviate the long-term effects of developer churn that happens due to many reasons.

9. Conclusion

In this research, we tried to compare two usual scenarios (from the perspective of a company or an institution wanting to invest in a FOSS project) to hire new developers tor expand the core developers team. To achieve this, we were assisted by a digital model that focuses on the time and the cost as two quantitative criteria for evaluating the outcomes of each scenario. The obtained results, obtained after a statistical analysis, show that there is no significant difference between hiring Insiders or Outsiders in terms of time and cost. However, as future work, we plan to include more projects and performing further analysis to compare the investment options. Moreover, something that might sound missing in the current study is not including *bus factor* as a criterion that software communities could rely on for making decision. Thus, in future work, we aim at including the bus factor and other qualitative criteria to offer a broader and clearer perspective of all possible solutions.

Acknowledgment

The second and third author acknowledge the support of the Government of Spain through project “BugBirth” (RTI2018-101963-B-100).

We also appreciate the insightful comments from the BENEVOL 2021 reviewers. Their comments have been very constructive to enhance our paper.

References

[1] G. Robles, I. Steinmacher, P. Adams, C. Treude, Twenty years of open source software: From skepticism to mainstream, IEEE Software 36 (2019) 12–15.

[2] M. R. Karim, G. Ruhe, M. M. Rahman, V. Garousi, T. Zimmermann, An empirical investigation of single-objective and multiobjective evolutionary algorithms for developer's assignment to bugs, Journal of Software: Evolution and Process 28 (2016) 1025–1060.
[3] V. Etemadi, O. Bushehrian, R. Akbari, G. Robles, A scheduling-driven approach to efficiently assign bug fixing tasks to developers, Journal of Systems and Software 178 (2021) 110967.
[4] D. M. German, The gnome project: a case study of open source, global software development, Software Process: Improvement and Practice 8 (2003) 201–215.
[5] G. Robles, J. J. Merelo, J. M. Gonzalez-Barahona, Self-organized development in libre software: a model based on the stigmergy concept, ProSim'05 16 (2005).
[6] K. Crowston, J. Howison, The social structure of free and open source software development, First Monday (2005).
[7] Y. Zhang, M. Zhou, A. Mockus, Z. Jin, Companies' participation in oss development-an empirical study of openstack, IEEE Transactions on Software Engineering (2019).
[8] G. Robles, J. M. Gonzalez-Barahona, I. Herraiz, Evolution of the core team of developers in libre software projects, in: 2009 6th IEEE international working conference on mining software repositories, IEEE, 2009, pp. 167–170.
[9] I. Herraiz, G. Robles, J. J. Amor, T. Romera, J. M. González Barahona, The processes of joining in global distributed software projects, in: Proceedings of the 2006 international workshop on Global software development for the practitioner, 2006, pp. 27–33.
[10] J. Tian, Software quality engineering: testing, quality assurance, and quantifiable improvement, John Wiley & Sons, 2005.
[11] M. Harman, B. F. Jones, Search-based software engineering, Information and software Technology 43 (2001) 833–839.
[12] K. Deb, S. Agrawal, A. Pratap, T. Meyarivan, A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii, in: International conference on parallel problem solving from nature, Springer, 2000, pp. 849–858.
[13] R. A. Ghosh, R. Glott, B. Krieger, G. Robles, Free/libre and open source software: Survey and study, 2002.
[14] S. Easterbrook, J. Singer, M.-A. Storey, D. Damian, Selecting empirical methods for software engineering research, in: Guide to advanced empirical software engineering, Springer, 2008, pp. 285–311.
[15] I. Steinmacher, I. S. Wiese, T. Conte, M. A. Gerosa, D. Redmiles, The hard life of open source software project newcomers, in: Proceedings of the 7th international workshop on cooperative and human aspects of software engineering, 2014, pp. 72–78.
[16] J. Ahlgren, K. Bojarczuk, S. Drossopoulou, I. Dvortsova, J. George, N. Gucevska, M. Harman, M. Lomeli, S. M. Lucas, E. Meijer, et al., Facebook's cyber–cyber and cyber–physical digital twins, in: Evaluation and Assessment in Software Engineering, 2021, pp. 1–9.
[17] V. Etemadi, O. Bushehrian, G. Robles, Task assignment to counter the effect of developer turnover in software maintenance: A knowledge diffusion model, Information and Software Technology (2021) 106786.
[18] F. Sarro, F. Ferrucci, M. Harman, A. Manna, J. Ren, Adaptive multi-objective evolutionary algorithms for overtime planning in software projects, IEEE Transactions on Software Engineering 43 (2017) 898–917.