Unveiling developers contributions behind code commits: An exploratory study

Unveiling Developers Contributions Behind Code Commits: An Exploratory Study

1st author
1st author's affiliation
1st line of address
2nd line of address
Telephone number, incl. country code
1st author's email address

ABSTRACT

The process of developing software is very dynamic and the activities involved on it are very diverse. For instance, codes have to be written, tested and revised, e-mails have to be sent, bugs have to be communicated, managed and fixed. In other words the contributions a developer can do when developing software are very diverse. In this context, this paper describes an empirical study whose goal was to assess and compare the developers' contributions through software repository mining. Two medium-sized projects – an open source and a commercial project – were analyzed. Overall, 17,490 commits and 10,308 bugs reports were analyzed. In the first part of our study, we have classified the developers based on their contribution to the software repository in three groups – core, active and peripheral developers. After that, we have collected a series of metrics – code contribution, buggy commits and resolution of priority bugs – for all the developers of the investigated projects. Finally, we have analyzed how the collected values for these metrics considering the different developer groups. Our study findings show significant differences in the contribution provided by the developers groups considering the open-source and the commercial project.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management – *productivity, programming teams.*

General Terms

Measurement, Experimentation.

Keywords

Mining software repositories, developer contribution.

1. INTRODUCTION

In a software development environment, a developer is expected to participate and contribute in many kinds of activities besides writing code. Participate on meetings, write e-mails, talk on telephone and commenting on bugs are few examples of those activities [1], not to mention other activities like criticizing specification documents, or even criticizing the software process. In this context, it would be crucial for software project managers to know which developers stands out from the others on specific contributions[1] for being more aware of the team they are coordinating. For instance, it could facilitate the motivation of the developers or it could help the identification of risks - like a training that is missing for those who lack some kind of contribution [2]. However, this variety of activities involved in the software development turns the measurement of the developers' contributions into a challenge since every kind of activities should be considered. Manually collecting the data for measuring the contributions of the developers, could be very time consuming and could elevate the project's costs . On the other hand, software repositories like: version control systems, change request systems, communication archives (eg. e-mails), databases, logs etc. [3] naturally record the actions of the developers. Those repositories are utilized to support the execution of the developers' everyday activities. Turning the static information recorded in software repositories into relevant information has been the focus of the mining software repository discipline over the last years [4] [5] [6] [7] [8]. Thus, mining software repositories can be an interesting strategy to investigate the developers' contributions at a low cost. Some effort has been done to evaluate the developers' contributions from mining software repositories [1] [9] [10].

In [1], a model for measuring the individual developer contribution is proposed. This model combines a list of actions of the developers – in the software repositories – with traditional metrics like lines of code (LOC) which produces a *contribution score* to help the assessment of productivity. Another work performed regarding developers' contributions is [10]. It is an investigation of the sent e-mails versus commits made, which was performed to verify if the developers are sending e-mails as much as they are commiting on open source software projects. Furthermore, in [9] a study that applied a author topic model approach to investigate which developers have contributed more for which topic (ex. module of a system) is presented. The study utilizes a subset of the eclipse 3.0 as its case study . Their results consists on a matrix indicating which developers have the higher probability of contributing to a specific topic and a graphic indicating which developers are similar based on the topics they have contributed with. As we can see, previous studies on this subject concentrate on the actions that developers perform on software repositories. In this paper, our main goal is to better understand which developers have the greater contributions and how those contributions can be related to his/her actions on software repositories. To do so, we analyzed two software projects: the ArgoUML[2] and a commercial system for managing the conformity of bank transactions. Both systems repositories were hosted in SVN[3] and the bug reports stored at Bugzilla[4] and ClearQuest[5] respectively. Overall, 17,490 commits and 10,308

[1] As in [1], we use the term "contribution" in this work to name all the activities a developer perform during the process of software development.

[2] http://argouml.tigris.org
[3] http://subversion.apache.org/
[4] http://www.bugzilla.org
[5] http://www-03.ibm.com/software/products/us/en/clearquest/

bugs were analyzed. Our general research question was: *"Which kind of developers has the most contributions on software projects?"*.

As the contributions of our work we can state that: (i) it proposes a new way of assessing the developers' contributions by grouping them and analyzing their contributions using repository mining techniques; (ii) new metrics were proposed: *code contributions* and *priority bugs* (section 2). Additionaly, the *buggy commit* [11] [12] metric was firstly used in this context. Moreover, some outcomes consistently detected through this study include:

- In the ArgoUML project, the *core* developers produced less buggy commits than *peripheral* and *active* developers, and they also have more contributions regarding the code metrics. In addition, although *core* developers presented a smaller proportion of high priority bugs resolution when compared to the *active* and *peripheral ones*, they solved more than a half of the total amount of high priority bugs of the project.

- On the other hand, for the commercial project, the *active* developers presented a smaller buggy commit proportion compared to the *core* developers. As for the high priority bugs, the active developers presented a higher proportion of resolution. Furthermore, the active and core developers did not present significant differences regarding the code metrics.

The remainder of this paper is organized as following: after the introduction, the methodology used in this work is presented (Section 2). Later, the results obtained in our study and the statistical analyses are described (Section 3). Then, some discussions about the insights and the experience we gained are exposed (Section 4) and the threats to validity of our work are also described (Section 5). Finally we conclude this paper listing our findings and stating our future work intentions (Section 6).

2. STUDY SETTINGS

The main aim of our empirical study is to investigate which kind of developers has a superior contribution in terms of metrics collected while mining information from existing software repositories. Such information can help software project managers to better understand the productivity and contribution of team members and guide them when taking decisions regarding the project. In this endeavor, our study was guided by the following general question: *"What developers have the most contributions on software projects?"*. We then segmented this general question into three specific questions described as following:

- RQ1. What kind of developer – *core*, *peripheral* or *active* – performs less buggy commits?

- RQ2. What kind of developers – *core*, *peripheral* or *active* – has more contributions regarding code?

- RQ3. What kind of developers – *core*, *peripheral* or *active* – fixes more bugs with high priority?

Each research question led to the mining of specific metrics which supported their answer. The *core*, *active* and *peripheral* developers are pre-defined groups which we classified the developers into. The heuristics used for the developers' classification as well as the mining of the metrics are detailed in the following sections.

2.1 Metrics For Quantifying Developers' Contribution

Buggy Commits (RQ1). The buggy commit metric applied in this work is based on [11] [12]. Figure 1 depicts how the buggy commit metric is collected.

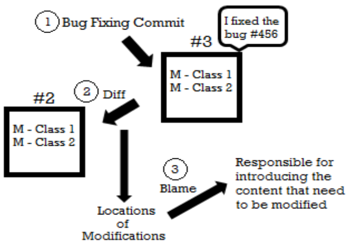


Figure 1. How Buggy Commit Metric Is Collected

First the miner searches for the *bug fixing commits*. The *bug fixing commits* are the commits that are known to be fixing some bug. The miner finds the *bug fixing commits* through the following heuristics: (i) the commit has the word "fix" (and its derivatives) on the message or/and (ii) the commit has an id pointing to a bug registered in the change request system (eg. Bugzilla). Once a *bug fixing commit* is found, the location of the modifications made to fix the bug are registered using the *diff* command of the version control system (VCS). For instance, if a comparison operator needed to be modified to correct a bug, the line of this modification is registered. Then, the miner finds the responsible for introducing the content that needed to be changed using the *blame* command of the VCS. The investigated projects adopted the Subversion as their VCS.

Code Contributions (RQ2). The code contribution is a set of metrics that captures the actions presented in Table 1. The metrics are calculated for each developer and are collected from the VCS. The collection of the metrics is made commit by commit, and the authors of the commits properly receive an increment on their respective counter for each of these metrics.

Table 1. Actions Gathered by the Code Contributions Miner

Action	Description
Code Addition	Code with plus signal "+" in VCS
Code Removal	Code with minus signal "-" in VCS
Method Addition	An entire method is added
Method Modification	A part of the a method is modified

Priority Bugs (RQ3). The priority bug metric is quite simple to be collected: a search for the bugs that were solved by a developer is made in the change request system (CRS). After that, the priorities of the bugs are verified. If the priority of a bug happens to be P1, P2 or P3, this bug is classified as a priority bug that has been solved, and the priority bug metric associated to the related developer is incremented. In order to be the responsible for the fix, the developer must have been the last one that resolved the bug with the 'FIXED' resolution on the bug activity history.

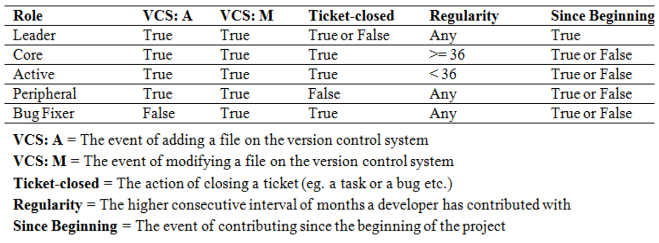
2.2 Target Systems

We have chosen an open source and a commercial system to be analyzed in our empirical study. The chosen open source project was the ArgoUML. It is a popular Java open-source project that represents an UML modeling tool which includes support for all standard UML 1.4 diagrams. The ArgoUML project has 372,056 LOC and has a considerably number of registered users (1,474 at the time of our study). In addition, it has been used in other mining software repository studies [13]. It also runs on any Java platform and is available in ten languages. On the other hand, the commercial project was chosen due to the proximity of the collaborators of the company with our research group. The project represents a large scale web information system of an on-line bank conformity management system. The system includes functionalities to avoid frauds and to ensure the conformity of the products sold in the bank's agencies.

2.3 Study Phases

The methodology of our study consisted on a set of four phases: (i) first we selected the projects we were interested to analyze, (ii) later, in order to address our research questions we investigated how we could segment and classify the developers into representative groups in such way that we could analyze the groups instead the individuals; (iii) after the classification, we applied the repository mining techniques for each project and collected the results; (iv) finally, we analyzed the results and tested the hypotheses related to the research questions (Section 3). Next we explain in more detail the *developer classification* step.

Classification of Developers. One of the fundamental phases was the developer classification. It was based on the development roles presented in [14] and the classification heuristics were based on the strategy defined in [15]. Figure 2 summarizes the heuristics by detailing how each developer is classified in a role. For instance, a developer is classified as a *core* developer, if he/she has: (i) added and modified a file on the VCS; closed a ticket (eg. task or bug etc.); and (iii) contributed at least 36 consecutive months of the project (*regularity*). However for the commercial project, as we analyzed a shorter period of time, we considered the regularity to be the total time we analyzed for the project, which was one year (Section 3.2)



Role	VCS: A	VCS: M	Ticket-closed	Regularity	Since Beginning
Leader	True	True	True or False	Any	True
Core	True	True	True	>= 36	True or False
Active	True	True	True	< 36	True or False
Peripheral	True	True	False	Any	True or False
Bug Fixer	False	True	True	Any	True or False

VCS: A = The event of adding a file on the version control system
VCS: M = The event of modifying a file on the version control system
Ticket-closed = The action of closing a ticket (eg. a task or a bug etc.)
Regularity = The higher consecutive interval of months a developer has contributed with
Since Beginning = The event of contributing since the beginning of the project

Figure 2. Heuristics Used In The Developers' Classification

We initially used FRASR[6] to collect the events for each developer on the different software repositories. After that, we used process mining techniques to apply the heuristics for the classification [15]. The resulting classification for the ArgoUML project was: 5 *core* developers, 7 *active* developers and 23 *peripheral* developers whilst for the commercial project the resulting classification was: 2 *core* developers and 5 *active* developers. There was no *peripheral* developer for the commercial project. This happened because the collaborators classified as *peripheral* on that project, were not actually developers. This is explained in more detail in the discussions' section (Section 4).

3. STUDY RESULTS

This section describes the results of our study for the investigated target systems.

[6] http://www.frasr.org

3.1 ArgoUML Results

This discusses the collected results for the buggy commits, code contributions and resolution of priority bugs for all developers of the ArgoUML project.

3.1.1 *Analysis of the Buggy Commit Metrics*

Table 3 summarizes the collected results for each developer group. Each column corresponds to a developer group, and the rows show the frequency of the *buggy commits* and *non-buggy commits* in each group. We performed a Chi-square test with the collected results to identify the existence of a relationship between the groups (columns) and the frequencies (rows). The test returned a p-value of < 0.0001. Thus, considering a 5% significance level, we were able to reject the null hypothesis, which is: *H_0 – there is no relation between the groups of developers and the proportions of buggy commits and non-buggy commits*.

The percentage of buggy commits over the total amount of commits is greater in *peripheral* developers whilst it is smaller in *core* developers (Table 3). In order to analyze the alternative hypotheses, we executed other three Chi-square tests to the following groups: (i) *core* versus *active*, (ii) *core* versus *peripheral* and (iii) *active* versus *peripheral*. As we carried out multiple comparisons, we performed the Bonferroni correction [16] to counteract the *type one error* [17]. It is important to note that for the *core* versus *active* developers comparison, we obtained a p-value of 0.06 using the Bonferroni correction. However, since Bonferroni correction is known to be too conservative as it augments the probability of getting the type two error [17], we also considered using another correction for dealing with multiple comparisons to verify the resulting p-value, which was Benjamini & Yekutieli (BY) [18]. We obtained a p-value of 0.036863 using the BY method. Since the Bonferroni correction resulted in a p-value next to 5% and the BY correction resulted in a p-value below 5%, we also considered our evidences to be statistically relevant for this comparison. Thus, Table 4 shows the conclusions of this analysis on the *A1*, *A2* and *A3* statements.

3.1.2 *Analysis of the Code Contribution Metrics*

In order to assess the code contributions collected metrics, we performed the One-way ANOVA test [19] to verify the statistical difference between the means of the developer groups. Table 3 shows the results for these metrics (number 2 to 5) which led us to reject the null hypothesis for the majority of the metrics (except for the *method addition* metric). The considered null hypothesis is: *$H_{0.}$ there is no difference between the groups' averages regarding code contributions metrics*. Later we performed several Tukey HSD pair-wise tests [20] for the contributions metrics which we were able to reject the null hypothesis. Our aim was to analyze the alternative hypotheses. As we can observe in Table 2, there is no statistical difference between the code contributions of *active* developers and *peripheral* developers whilst *core* developers distinguished for the most of the cases from the other two groups (except for *method additions* when compared to *active developers*). Hence, Table 4 shows the conclusions of this analysis on the *A4* and *A5* statements.

3.1.3 *Analysis of the Priority Bug Metrics*

Table 3 presents the mined results for the priority bug metric of the ArgoUML project. The columns contain each developer group, and the rows contain the frequency of high priority bugs and low priority bugs that were solved for each group.

Table 2. P-values Obtained For The Metrics From ArgoUML And Commercial Project

ArgoUML				4. Commercial Project
Metric	**Core x Active H_1'**	**Core x Peripheral H_1''**	**Active x Peripheral H_1'''**	**Core x Active H_1**
1 Buggy Commit	p = 0.060321/0.036863*	p < 0.0001	p = 0.000225	p < 0.0001
2 Code Addition	p < 0.05 (HSD)	p < 0.01 (HSD)	**Non Significant**	**Non Significant**
3 Code Removal	p < 0.01 (HSD)	p < 0.01 (HSD)	**Non Significant**	**Non Significant**
4 Method Addition	**Non Significant**	p < 0.05 (HSD)	**Non Significant**	**Non Significant**
5 Method Modification	p < 0.01 (HSD)	p < 0.01 (HSD)	**Non Significant**	**Non Significant**
6 Priority Bug	p = 0.007563	p < 0.0001	p < 0.000379	p < 0.0001

Table 3. Mined Results For The Buggy Commits and Priority Bugs Metrics Utilized In The Chi-Square Tests

	ArgoUML			Commercial Project	
	Core	**Active**	**Peripheral**	**Core**	**Active**
	Buggy Commit Metrics				
Buggy Commit	324	94	84	120	94
Non Buggy Commit	8786	1931	940	1353	2372
Totals	9110	2025	1024	1473	2466
Percentage	**3.6**	**4.8**	**8.9**	**8.4**	**3.8**
	Priority Bugs Metrics				
Priority Bug	502	233	125	33	219
Non Priority Bug	1505	526	171	76	17
Totals	2007	759	296	109	236
Percentage	**25**	**30.6**	**42.2**	**30.3**	**92.8**

Table 4. Conclusions For The ArgoUML And Commercial Project

A1	***H_1'***. *core* developers proportionally produce less buggy commit than *active developers*
A2	***H_1''***. *core* developers proportionally produce less *buggy commits* than *peripheral developers*
A3	***H_1'''***. *active* developers proportionally produce less *buggy commit* than *peripheral developers*
A4	*core* developers have more *code additions*, *code removals*, and *method modifications* when compared to *active* and *peripheral* developers.
A5	*core* developers have more *method additions* when compared to *peripheral* developers whilst the same cannot be said when compared to *active* developers
A6	***H_1'***. *core developers* proportionally solve less priority bugs than *active developers*
A7	***H_1''***. *core developers* proportionally solve less *priority bugs* than *peripheral developers*
A8	***H_1'''***. *active developers* proportionally solve less *priority bugs* than *peripheral developers*
C1	***H_1***. *active* developers have a minor proportion of buggy commits compared to *core* developers.
C2	***H_0***. There is no considerable difference between the *core* and *active* groups' averages regarding code contributions metric considering a 5% significance level
C3	***H_1***. active developers proportionally solve more priority bugs than core developers.

We performed the Chi-square test, which resulted in a p-value of < 0.0001, so we were able to reject the following null hypothesis: *H_0. there is no relation between the group of developer and the proportion of the priority bugs that were solved*. We found the resolution percentage of high priority bugs to be the greater for *peripheral* developers whilst to be the smaller for *core* developers. Hence, we performed other pair-wise Chi-square tests to analyze the alternative hypotheses. The resulting p-values (Table 2) show us a statistical relevance under a 5% significance level. Thus, the conclusions obtained of this analysis can be seen in Table 4 on the *A6*, *A7* and *A8* statements. Finally, although we found the resolution proportion of high priority bugs of *core* developers to be the smallest of the three groups, it is worth noting that they have resolved more than a half of the total of high priority bugs that were analyzed for the project.

3.2 Commercial Project Mining Results

This section presents the obtained results for the commercial project. We mined the buggy commit, code contribution and priority bugs metrics for the development period from January 2008 to January 2009 we also adapted the classification method for this period.

3.2.1 Analysis of the Buggy Commit Metrics

Table 3 exhibits the collected results for the buggy commit metric considering the commercial project. The columns represent the developer groups, and the rows the amount of buggy commits and non-buggy commits for the investigated software project. Table 3 also shows that the percentage of performing a buggy commit is greater for *core* developers than *active* ones. The study did not find any peripheral developers during the classification process (Section 4). We executed the Chi-square test, which resulted in a *p-value* of < 0.0001. Thus, we were able to reject the following null hypothesis: *H_0*: *there is no relation between the developer groups and the proportions of the buggy commits* (Table 4, C1 statement). We consulted the staff of the commercial project in order to find an explanation for this result. We observed that a senior developer of the company participated only in beginning of the project, and he gave a significant contribution at this period. Later, two junior developers have been enrolled for the company to work on this project with the responsibility of implementing the new functionalities while the senior developer moved to another project. Hence, the senior developer was classified as an *active* developer for the sake of the smaller period of work, while the two junior developers were classified as *core* developers due to the greater period of work in the project. Furthermore, we also observed that the senior developer's results strong contributed for the smaller proportion of the buggy commits for the *active* developers. Due to space restrictions we did not make available the results of each developer, but the interested reader may refer to http://goo.gl/UWcPBX.

3.2.2 Analysis of the Code Contributions Metrics

The results mined for the code contributions metrics can be seen in Table 2, which presents the T-tests [21] performed with the *core* and *active* developer groups. From the p-values obtained, we were not able to reject the null hypothesis, which is: H_0. *there is no difference between the groups' averages regarding code contributions metrics* (Table 4, C2 statement). The possible reason for this result is the same as discussed in the mining of buggy commit metric (Section 3.2.1). The senior developer classified as an *active* developer had a significant code contribution in the period that he/she works in the project. For instance, for the *method additions* metric, this senior developer presented the value of 315,689 whilst the two developers classified as *core* developers presented 10,681 and 15,887, respectively. When discussing with the project manager, we discovered that the senior developer started the development of the project which involves the implementation of the architecture as well as the first set of functionalities the system should provide. The mined result for each developer can be found in http://goo.gl/UWcPBX

3.2.3 *Analysis of the Priority Bugs Metrics*

Table 3 shows the obtained results for the priority bugs metric of the commercial project. From the results we can observe that the proportion of solved priority bugs for *core* developers is smaller than the proportion presented by the *active* developers. The Chi-square test was applied and resulted in a p-value < 0.0001, so we were able to reject the null hypothesis which is: H_0. *there is no relation between the group of developer and the proportion of the priority bugs that were solved* (Table 4, C3 statement). We contacted the staff of the project to better analyze the obtained results. From this interaction, we found that the project's process of bug assignment consists on assigning the same developer that implemented the use case (UC), which a bug is concerned of. For example, if a *developer01* implemented the *UC(01)*, the bugs that occur when executing the functionalities of *UC(01)* are automatically assigned to *developer01*. In this context, since the senior developer, which started the implementation of the project, had a significant contribution for the project, he/she also addressed a greater amount of bug resolution. Thus, he/she gave a significant contribution in this metric for the *active* developers group. For instance this senior developer has solved a total of 213 high priority bugs in the period of one year while only 33 high priority bugs were resolved by all core developers. The collected priority bugs metric of each developer can be verified at: http://goo.gl/UWcPBX

4. DISCUSSIONS

In our work, we tried to apply the same methodology for the ArgoUML and the commercial project. One of our aims was to observe the differences between them when mining the contributions of the developers. Next we highlight the main differences found in the empirical study.

The classification method was not adequate for the commercial project. In our study, we have used the developer classification method presented in [15], which was previously applied to open source projects. When applying this classification method for a commercial project in our study, we noticed that a very experienced developer was classified as an active developer because he/she has contributed only during part of total period of the project. On the other hand, this developer has the most significant contribution for the project, and should be classified as a *core* developer of the system since he/she has implemented the system architecture and the first main functionalities. In addition, he also presented better values for buggy commit and code contribution metrics compared to the other developers, including the core developers. This shows that the classification method proposed in [15] does not seem adequate to be used in commercial projects. Thus, new classification methods should be explored in the context of commercial projects.

Many peripheral developers were not developers. By applying the classification method to the commercial project, we found that many *peripheral* developers were not in fact real developers. They have additions, modifications and deletion actions on code repository as well as ticket-closed actions, but those actions were not related to commits of source-code. For instance, some additions were related to updates in the project plan artifact, and some of the task-closed were related to a requirement related task.

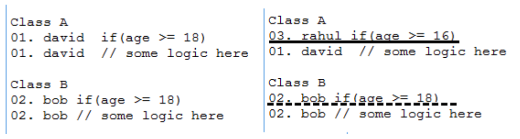
***Developer Contribution Mining for Project Management*.** We believe that mining and quantifying the contributions of developers, can help the activity of software project management. For example, the developers whose contributions are considerably under the average compared to other developers could be analyzed with further care. Several questions can be raised in this context, such as: Why specific developers have a reduced number of contributions during a specific period? What are the expected contributions for team leaders or architects in a project? What is the productivity of new members in a team? We are currently applying the metrics presented in this paper, to analyze the contributions of a software team in a series of commercial projects from the same company in order to help software project managers to take decisions.

5. THREATS TO VALIDITY

In this section we describe the threats to validity of the obtained results for our empirical study. We first describe the internal validity and then the external validity.

5.1 Internal Validity

***Buggy Commit Mining*.** The technique used for the *buggy commit mining* was based on [11] [12]. One possibly threat of this technique is finding an improper responsible for the buggy commit due to changes of requirements. For instance, let's consider the code of a system for buying movie tickets presented in Figure 3. `Class A` and `Class B` share the same logic that verifies the age of the user. In revision 3 (the number on the left of the name), *Rahul* modified the age checking of `Class A` due to a change of the requirement, but he forgot to also modify class B.



```
Class A
01. david  if(age >= 18)
01. david  // some logic here

Class B
02. bob if(age >= 18)
02. bob // some logic here
```

```
Class A
03. rahul if(age >= 16)
01. david  // some logic here

Class B
02. bob if(age >= 18)
02. bob // some logic here
```

Figure 3. Revision, Author and Line of Code for the Classes of a Movie Ticket E-Shopping System

In revision 3 (the number on the left of the name), *Rahul* modified the age checking of `Class A` due to a change of the requirement, but he forgot to also modify class B. Once a bug is registered and another commit is made to correct the bug, the miner for the buggy commit metric would find *bob* as the responsible for the bug on `Class B`. When bob introduced the code in `Class B`, the code was right, so is he the true responsible for the bug? Although the situation presented here is a threat for the mining of the *buggy commit metric*, we believe this situation is an exception and not the common context when mining the 17,490 commits of the analyzed projects.

The Developer Classification Method. In our study, we have used the same heuristics presented in [15] the process of developer classification, because of that we inherited the same threats. However, we believe that these heuristics were appropriate for our study, specially the analysis of the open-source system, as we could find relevant statistical differences between the developer groups. We plan to validate the heuristics for classifying developers in open-source and commercial projects by confirming with project managers and leaders that the roles played by the developers are in fact aligned with the classification.

5.2 External Validity

The study presented in this paper was conducted through the analysis of a popular open source project and a commercial project. Although the selected open source project is representative of the development processes used among large open source software projects, we cannot extend our results to similar open-source projects. Further evaluation is necessary through the conduction of new empirical studies that mining and analyzing other existing open source projects and commercial projects, in order to confirm, and possibly generalize, our findings. In fact, we are currently conducting a new empirical study involving several open source systems, and the collected results are up to now quite similar to the ones we have found in the ArgoUML project.

6. CONCLUSION

Our work analyzed the developers' contributions of two software projects: ArgoUML and a commercial project. The contributions were analyzed through three different metrics: buggy commits (RQ1), code contributions (RQ2) and priority bugs (RQ3). We grouped the developers into groups which reflected their actions in the software repositories. The groups were: *core*, *active* and *peripheral* developers inspired by the roles described in [14]. Our study revealed the following main conclusions: the developers groups denominated as *core*, *active* and *peripheral*, presented statistically significant differences concerned the contribution metrics. For instance, for the ArgoUML project, we found the core developers to be the group with the majority of contributions regarding code and they also presented the smaller proportion of buggy commits. In addition, we also found the active developers to be in the middle of core and peripheral developers. For example, they presented a smaller proportion of buggy commits when compared to the peripheral ones, but they also presented a higher proportion when compared to the core developers.

Other main finding of our work, is the fact that the developer groups differed significantly concerning the contributions metrics when we compare the two projects. For example, the active developers of the commercial project presented a higher proportion of high priority bugs resolution and a smaller proportion of buggy commits when compared to the core developers whilst the contrary happens on the ArgoUML project. These results led us to think about new strategies to deal with commercial project when analyzing the developers' contributions like, for instance, the investigation of new classification strategies and metrics assessment strategies. As future work, we intend to replicate this study to other open source and commercial software projects. We are also currently working in performing the mining techniques presented in this work in a private company environment to support the project management activities. We also intend to analyze other kinds of contributions like, for example, the patches submitted by developers on the open source projects.

7. REFERENCES

[1] E. Kalliamvakou, G. Gousios, D. Spinellis, and N. Pouloudi, "Measuring Developer Contribution from Software Repository Data," *Mediterranean Conference on Information Systems*, 2009.

[2] J. Jiang and G. Klein, "Software development risks to project effectiveness," *The Journal of Systems and Software*, pp. 3-10, 2000.

[3] A.E. Hassan, "The Road Ahead for Mining Software Repositories," *Frontiers of Software Maintenance*, pp. 48-57, September 2008.

[4] T. Zhang and B. Lee, "A Hybrid Bug Triage Algorithm for Developer Recommendation," *SAC '13 Proc. ACM-SAC*, 1088-1094 2013.

[5] S. Wang, D. Lo, and L. Jiang, "An empirical study on developer interactions in StackOverflow," *SAC '13 Proc. of ACM*, 1019-1024 2013.

[6] S. Wang, F. Khomh, and Y. Zou, "Improving bug localization using correlations in crash reports," *Proc. of MSR'13*, pp. 247-256, 2013.

[7] Debdoot, and Malika Garg Mukherjee, "Which work-item updates need your response?," *Proc. of MSR'13. IEEE Press*, 2013.

[8] Hoda, et al Naguib, "Bug report assignee recommendation using activity profiles.," *Proc. of MSR'13. IEEE Press*, 2013.

[9] P. Rigor, Sushil Bajracharya, C. Lopes, and P. Baldi, "Mining Eclipse Developer Contributions via Author-Topic Models," *ICSE Workshops on MSR '07*, May 2007.

[10] S. K. Sowe and A. Cerone, "Integrating Data from Multiple Repositories to Analyze Patterns of Contribution in FOSS Projects," *Electronic Communications of the EASST*, 2010.

[11] J. Eyolfson, L. Tan, and P. Lam, "Do Time of Day and Developer Experience Affect Commit Bugginess?," *Proc. of MSR*, pp. 153-162, 2011.

[12] J. Sliwerski, T. Zimmermann, and A. Zeller, "When Do Changes Induce Fixes?," *Proc. of MSR'05*, pp. 1-5, 2005.

[13] Vladimir, et al Rubin, "Process mining framework for software processes.," *Software Process Dynamics and Agility. Springer Berlin Heidelberg*, pp. 169-181, 2007.

[14] Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye K. Nakakoji, "Evolution patterns of open-source software systems and communities," *Workshop on Principles of Softw. Evolution. ACM*, pp. 76–85, 2002.

[15] W. Poncin, A. Serebrenik, and M. van den Brand, "Process mining for software repositories," *(CSMR)*, pp. 5-14, March 2011.

[16] O. J. Dunn, "Multiple comparisons among means," *ournal of the American Statistical Association*, pp. 52-64, 1961.

[17] M. Shermer, *The Skeptic Encyclopedia of Pseudoscience*, 2nd ed.: ABC-CLIO p. 455, 2002.

[18] Y. Benjamini and D. Yekutieli, "The control of the false discovery rate in multiple testing under dependency," *Annals of statistics*, pp. 1165-1188, 2001.

[19] V. Bewick, L. Cheek, and J. Ball, "Statistics review 9: one-way analysis of variance.," *CRITICAL CARE-LONDON*, pp. 130-136, 2004.

[20] R. Lowry. (2013, August) One Way ANOVA – Independent Samples. [Online]. http://vassarstats.net/textbook/ch14pt2.html

[21] J. J. O'Connor and E. F. Robertson, "Student's t-test," *MacTutor History of Mathematics archive*.

[22] R. Shokripour, A. John, Z. M. Kasirun, and S. Zamani, "Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation," *Proc. of MSR'11*, pp. 2-11, 2013.

[23] P. J. Adams, A. Capiluppi, and A. Groot, "Detecting Agility of Open Source Projects Through Developer Engagement," *IFIP*, pp. 333-341, 2008.