More Common Than You Think: An In-Depth Study of Casual Contributors

Gustavo Pinto
Informatics Center
Federal University of Pernambuco
Recife, PE, Brazil
ghlp@cin.ufpe.br

Igor Steinmacher
Department of Computing
Federal University of Technology – Paraná
Campo Mourão, PR, Brazil
igorfs@utfpr.edu.br

Marco Aurélio Gerosa
Institute of Mathematics and Statistics
University of São Paulo
São Paulo, SP, Brazil
gerosa@ime.usp.br

***Abstract*—Source code hosting websites (code forges) have recently changed to more social environments, and the contribution process evolved to the so-called pull-based development model. Due to the facilities brought by this evolution, Open Source Software (OSS) projects are now facing a high exposure, leading to an increasing number of contributors. However, not all these contributors want to have a long-term engagement with the project. In fact, popular projects are known to have a restricted set of core developers who drive the project, but now these projects count on a broad set of "not that involved" developers, which are responsible for a long tail of small contributions. In this paper, we shed the light on this important but overlooked set of developers: the casual contributors (also known as drive-by commits). First, we mined popular software repositories hosted on GitHub to investigate how common casual contributions are, and what are their characteristics. Second, we conducted two surveys with (1) the casual contributors and (2) the project maintainers aimed at understanding what motivates casual contributors and how they are perceived. Our results showed that although casual contributors are rather common (48.98% of the whole population of contributors in the projects analyzed), they are responsible for only 1.73% of the total number of commits. We also found that casual contributions are far from being trivial: even though a significant proportion of them are fixing typos and grammar issues (28.64%), we found several of them that have fixed bugs (30.20%), added new features (18.75%), and refactored code (8.85%). Still, we found that both casual contributors and project maintainers believe that casual contributions have more benefits than drawbacks. As a casual contributor said: "every bit helps".**

I. Introduction

The development of Open Source Software (OSS) is usually an intrinsically collaborative activity [68]. More recently, with the growth of OSS communities, a plethora of social coding environments were created. These environments changed the way developers contribute to OSS projects [63], in particular by providing a single process of contribution, which is called *pull-based model* [46]. The contribution process is streamlined: interested developers clone (or "fork") public projects, implement improvements, and then offer the modifications back to the original project [63]. As a result of these facilities, OSS projects are now facing a high exposure, leading to an increasing number of contributors [46], [47], [63].

This kind of environment, together with its contribution model, encourages newcomers to participate in the process. However, despite the facilities aforementioned, newcomers still need to get acquainted with the project specificities, which increases the learning curve and may prevent one to contribute. Consequently, a significant number of newcomers end up abandoning the project [73]. To mitigate this problem, many studies have been focusing on different aspects of newcomers joining process, including how to become a core member, motivation and retention [72], [75], [39]. These studies were concerned about the dynamics that drive newcomers to become long-term contributors [42], [71]. However, while some contributors want to have a key role on the project, some others do not share the same desire, although they still want to contribute. In fact, it is well-known that popular projects have a restricted set of core developers, who drive the project, but also a broad set of "not that involved" (or inactive) developers, which are responsible for a long tail of small contributions [45]. Although these developers do not want to become active members, they foster diversity and collaboration.

In this study, we shed light on what we call "casual contributors". This phenomenon is already known in the software engineering community, and is gaining even more attention lately [63], [55], [64], [46]. In particular, Pham *et al.* [63], [64] was the first to observe this behavior, referring to it as "drive-by commits." According to their study, drive-by commits are "*simple commits that leave their creators rather uninvolved with the project and that can be created with very little project-specific knowledge*". Interestingly, this kind of contribution is becoming more common. According to Gousios *et al.* [46], casual contributions account for 7% of the pull-requests made to GitHub projects in 2012. More interestingly, however, is the *belief* that these contributions are based on fixing documentations issues (*e.g.,* a spelling error or a missing translation), in which the contributor could quickly make a correction for it.

Despite the growing number of newcomers interested in contributing to OSS [56], little is known about this particular kind of contributor: the casual contributor. According to the literature, more research is needed to better understand the process, benefits [63] and implications [46], [76] of such contributions. Starting from this premise, this paper presents an empirical study aimed at illuminating the casual contributors. In particular, our study is unique in its focus on understanding (1) how common they are, (2) what are the characteristics of their contributions, and (3) how they are perceived. Answering these questions incurs in guidance for software developers, researchers, tool builders, and educators (Section IV).

In this paper we conducted a two-phase study aimed at providing answers to these questions. The first phase is based on a quantitative and qualitative analysis of data from GitHub. As one of the most popular code forges, with more than 11M users and 29M projects [36], GitHub is often used for software engineering studies [46], [47], [57], [66], [51]. We complemented the analysis from GitHub data with our second phase: two surveys conducted with 197 casual contributors and 65 project maintainers — that is, core developers responsible for inspecting changes and integrate them into the project [47]. The replication package is available online [65]. Our study produced a set of findings, many of which were unexpected. We discuss them in detail in Section III. In the following, we highlight three of them.

- **Casual contributors are rather common**. 48.98% of the overall contributors that we analyzed are actually casual contributors. However, these contributors are responsible for only 1.73% of the total contributions in our set of analyzed projects.
- **Casual contributions are far from being trivial**. After a manual inspection of a sample of casual contributions, we found that although 28.64% of them are related to grammar and typo fixes, 30.20% of them fix bugs, 18.75% propose new features, and 8.85% refactor code.
- **Casual contributions are well liked.** These contributions are perceived as a beneficial phenomenon from the perspective of the project maintainers and casual contributors. Personal needs was the most reported motivation for the casual contributors. As a shortcoming, projet maintainers reported an increasing number of reviews, which demands time from core developers.

II. Study Methodology

Here we describe the research questions (Section II-A), and how the study was conducted (Section II-B and Section II-C).

A. Research Questions

Our examination of the literature revealed that the phenomenon of casual contributors (or drive-by commits) is already known in the software engineering community [46], [55], [63]. However, these studies do not examine how common is it or how project maintainers perceive it. With the growth of popularity of OSS systems, and due to the simplicity of sending a contribution to these projects, it is important to expand our understanding about this phenomenon. The overall research goal of this paper is to gain an in-depth understanding of the casual contributors, as well as the benefits and problems behind it. As a first step towards our research goal, we designed the following research questions:

RQ1. How common are casual contributors in OSS projects?

This research question investigates how commonplace are casual contributors in our set of projects. As we shall see in Section II-B, we used GitHubArchive to find representative OSS projects. After identifying our target subjects, we semi-automatically study their commit logs and file contents.

RQ2. What are the characteristics of a casual contribution?

Here we dig into the details of a casual contribution. First, we studied the number of additions and deletions performed in a casual contribution. Second, we selected a representative sample of 384 casual contributions for a manual inspection. This sample size provides a confidence level of 95% with a ±5% confidence interval. This manual inspection is aimed at understanding the intention of a casual contribution.

RQ3. How do casual contributors and project maintainers perceive casual contributions?

Finally, we investigated what are the motivations of casual contributors, how project maintainers perceive them, and what are the benefits and problems behind it. In order to answer this question, we conducted two surveys with casual contributors and with the projects maintainers.

B. Study 1: Mining software repositories

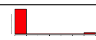
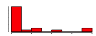
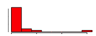
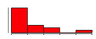
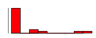
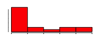
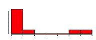
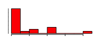
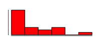
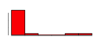
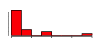
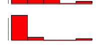
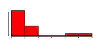
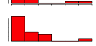
As mentioned in a recent study [51], a large proportion of the software repositories hosted on Github (1) have very few commits, (2) are inactive, or (3) are not software projects. To mitigate this risk, we used the same methodology of Ray *et al.* [66], which selects the most popular projects (in terms of the number of stars), of the most popular programming languages in Github. Therefore, we studied the top 20 most popular projects written in: C, C++, Clojure, CoffeeScript, Erlang, Go, Haskell, Java, JavaScript, Objective-C, Perl, PHP, Python, Ruby, Scala, and TypeScript. To select these projects, we ran a query on GitHubArchive [35], on Oct. 6, 2015. Our initial corpus comprises 320 mature, non-trivial, OSS projects. These projects had a total of 73,960 contributors who performed 2,039,376 contributions.

When manually analyzing these projects, we observed some projects that were wrongly sampled. For example:

1. ***Projects that were wrongly classified***. One example is project beautiful-web-type [26], which is classified as Ruby, but only 1.1% of its source code is Ruby code. We hypothesize that this misleading categorization is because some data on GitHubArchive is outdated. Thus, when GitHubArchive downloaded the project it is likely that the project was written mostly on Ruby, but due to the evolution of the project, the project started to use other programming languages. We then removed 15 projects.
2. ***Projects that are not software projects***. For instance, we found that some popular projects are textbook projects [27], or bookmarks projects [14]. We believe it is important to focus on software projects, because the particular characteristics of a programming language and contribution process might influence not only the coding process, but also the community engagement [66]. We removed 21 projects that matched with this criterion.
3. ***Projects that are not collaborative***. We removed one project that was created and maintained by a single developer [22]. Albeit this project is interesting and popular, it does not serve for the purposes of this study.

We ended up with a curated list of 275 popular, non-trivial OSS projects. A typical empirical software engineering paper studies under 10 projects [58]. Table I shows the descriptive characteristics of our projects in terms of line of code, organized by programming language.

TABLE I
LINES OF CODE PER PROGRAMMING LANGUAGE.

Language	Mean	Median	Standard Dev.	Histogram
C	991,500	114,800	3,052,684	
C++	711,051	115,200	1,192,624	
Clojure	15,932	5,100	35,042.08	
CoffeeScript	16,640	5,470	34,676.8	
Erlang	22,400	11,420	24,822.19	
Go	151,000	30,650	246,005.3	
Haskell	29,850	14,180	32,069	
Java	160,100	35,690	238,195.8	
JavaScript	81,030	38,960	108,604.1	
Objective-C	15,050	9,012	14,899.88	
PHP	71,150	8,465	146,096	
Perl	4,851	12,310	17,856.52	
Python	43,340	12,800	63,073.03	
Ruby	94,640	14,140	223,956	
Scala	56,520	33,740	69,785.2	
TypeScript	59,540	43,230	63,272.07	

Also, we found contributors who had contributed more than once using different full names and/or email addresses. To mitigate these cases, we used a disambiguation technique proposed by Bird *et al.* [38]. Although the technique works on both full names and email addresses, we applied it on the full names only, since git disambiguates contributors with the same full name but with different email addresses. The technique works as follows: For each full name, we (1) removed all punctuation, accentuation, suffixes (*e.g.,*"jr"), (2) turned all whitespace into a single space, and (3) split the name (using whitespace and commas as cues) into first name and last name. We consider names similar if the full names are similar, or if both first and last names are similar. We removed 1,281 duplicate contributors that matched this criterion. Table II groups projects according to the number of different contributors.

As we can see from this table, most of the analyzed projects have between 10 to 249 different contributors (71.98% of the total). Interestingly, 15 projects became popular with no more than 9 contributors. On the other hand, we found that projects that are able to attract a high number of contributors (*e.g.,* +1,000 different contributors), such as `django` or `rails` projects, are exceptions in our dataset. We found only 7 projects with this characteristic. Similarly, `Linux` is the only outlier found with more than 10,000 different contributors.

TABLE II
CONTRIBUTORS DISTRIBUTION.

# Contributors	#	%	Examples
1 — 9	15	5.45%	`openbay`, `misultin`, `ccv`
10 — 49	71	25.81%	`masscan`, `websocketd`, `ruby`
50 — 99	65	23.63%	`picasso`, `xctool`, `twemproxy`
100 — 249	62	22.54%	`memcached`, `sbt`, `yesod`
250 — 499	34	12.36%	`bitcoin`, `ipython`, `jquery`
500 — 999	21	7.36%	`jekyll`, `cakephp`, `php-src`
1,000 — 4,999	6	2.18%	`django`, `homebrew`, `rails`
5,000 — 9,999	0	0%	—
≥ 10,000	1	0.36%	`linux`

C. Study 2: Surveys with practitioners

To better understand the motivation, benefits, and drawbacks of this kind of contribution, we conducted two surveys with (1) the casual contributors and (2) the project maintainers. Both surveys were based on the recommendations of Kitchenham *et al.* [53], following the phases prescribed: planning, creating the questionnaire, defining the target audience, evaluating, conducting the survey, and analyzing the results.

The questionnaire with the casual contributors had 9 questions and was structured to limit responses to multiple-choice and open-ended questions. We asked questions about their contribution behavior and their motivation to make casual contributions. We selected a random sample of 865 casual contributors to send the questionnaire. However, 105 emails returned due to technical reasons (*e.g.,* domain name not found). A total of 760 emails were successfully sent. Over a period of 14 days, we obtained 197 responses, resulting in 25.92% of response rate. This response rate is 5 time greater than the ones found in software engineering surveys [52].

The questionnaire with the project maintainers had 5 questions and had only free-forms. It focused on better understanding their perspective about the casual contributors. Our target population is formed by software developers that had contributed the most to our analyzed projects. For each project, we manually analyzed the top-3 developers with the highest number of contributions. However, we found cases where the contributors found performed very few contributions (usually less than 10). This happens because some projects were mostly maintained by a single developer. We removed these developers. We sent the questionnaire to 621 project maintainers, but 13 emails were returned due to technical reasons. 608 emails were successfully sent. Over a period of 14 days, we obtained 64 responses, resulting in 10.52% of response rate.

We qualitatively analyzed the answers to the open-ended questions following open-coding and axial-coding procedures [74]. When analyzing the quantitative data from the casual contributors survey, we observed that 65.8% of our respondents contribute to OSS at least once per month, and 75.2% of them are used to making casual contributions. Most surprisingly, however, we received 50+ emails from the participants, congratulating us for conducting this study. Also, when asked whether the respondent is interested in receiving the results of the study, *all* respondents said yes. This shows that this is a subject that practitioners are interested in.

III. Study Results

Here we report the results in terms of each **RQ**.

A. RQ1. How common are casual contributors in OSS projects?

To start answering our first research question, Figure 1 presents an overall picture of the studied projects. Here each histogram groups the projects analyzed of each programming language. We removed outliers from the histograms that would otherwise skew the proportion of the figures. Several interesting observations can be derived from this figure. First, to some extent, the analyzed projects have a similar characteristic: most of the contributors perform very few contributions (contributions per contributor: median: 2, mean: 27.57, 3rd Quartile: 4, Std. deviation: 238.69). Based on this figure, it seems that the shape of the curve follow a power law distribution [40].

Interestingly, we found that a non-negligible number of contributors (48.98%) performed a single contribution. Based on this finding, we decided that the casual contributor is a contributor that performed at most one commit to a software project. Still, we observed that the more long-lived the projects are, the more likely they are to receive a high number of casual contributions. For instance, the `linux` project, which has about 20 years of coding history, received a total of 5,626 casual contributions (39.28% of the total). Similarly, the `rails` project, with 11 years old, and the `django` project, with 10 years old, received respectively 1,860 (54.21%) and 687 (61.57%) casual contributions.

This significant number of casual contributors might lead one to believe that an important proportion of the projects are intrinsically made by casual contributions. In reality, we found the opposite: these casual contributors are responsible for only 1.73% of the total number of contributions in our corpus of OSS projects (`linux`: 1.02%, `rails`: 3.46%, `django`: 3.19%). For a more detailed perspective, Figure 2 shows the percentage of the casual contributors (top) and contributions (bottom) for each programming language analyzed.

These figures show a couple of interesting information. First, it reinforces our first finding (and corroborates Gousios's findings [46]): a large group of contributors are responsible for a long tail of small contributions. We believe this happens because GitHub provides low-barrier mechanisms to one get involved with an OSS project. Thus, the barrier for performing simple contributions, such as fixing a typo, is negligible. Second, we can see that the programming language used matters. For instance, projects written in static typed programming languages (*e.g.,* C, TypeScript, and C++) seem to be less favorable to receive casual contributions than those using dynamic typed ones (*e.g.,* Ruby, Python and, JavaScript). There are some exceptions, though. PHP (a dynamic weakly typed programming language) have a similar percentage of casual contributors as C or C++ (two static strongly typed ones). One possible explanation is because scripting programming languages are more concise than procedural and object-oriented ones [60], and size really matters when it comes to casual contributors. Similarly, pure functional programming languages (*e.g.,* Haskell, Clojure, and Erlang) seem to be less favorable to receive casual contributions than scripting ones (*e.g.,* Ruby, Python, and JavaScript). This might be part because functional programming is still becoming popular.

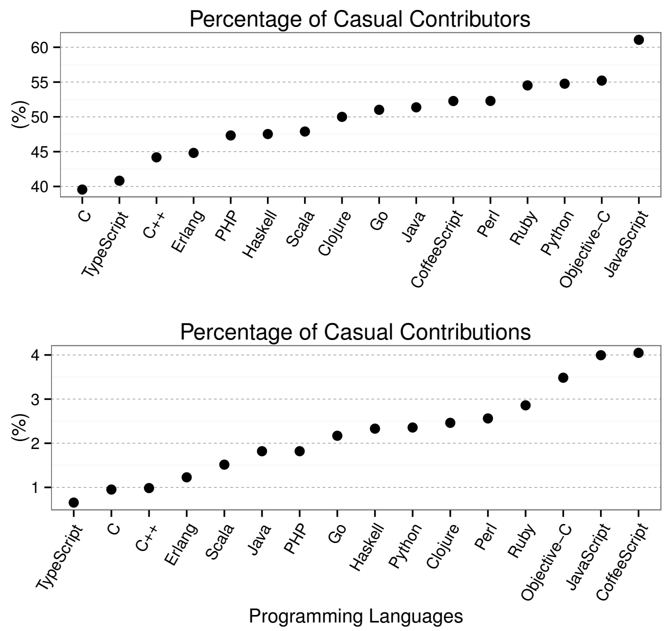


Fig. 2. Percentage of casual contributors and contributions per programming language.

Since we observed that dynamic languages are more likely to receive casual contributions, Figure 3 shows the percentage of casual contributors and contributions of projects written with the Ruby programming language.

As we can see, even though the `rails` project has a total of 1,860 casual contributors (54.19% of the total contributors), it is not the Ruby project the presents the highest proportion of casual contributors. In fact, 67.76% of contributors of the `capistrano` project are casual contributors. These casual contributors are responsible for 11.60% of the overall contributions performed in this project. Differently, we observed that the `ruby` project, which is the implementation of the Ruby language, was not capable of attracting casual contributors. Only 5.26% of its contributors are casual contributors, which are responsible for only 0.01% of the contributions performed in this project.

B. RQ2. What are the characteristics of a casual contribution?

In this RQ, we analyzed the characteristics of the casual contributions in both quantitative and qualitative terms. Figure 4 shows the median of additions, deletions, and files touched of each casual contribution performed on the Ruby projects analyzed.

As we can see in Figure 4, the number of additions, deletions and files touched of contributions do not vary signifi-

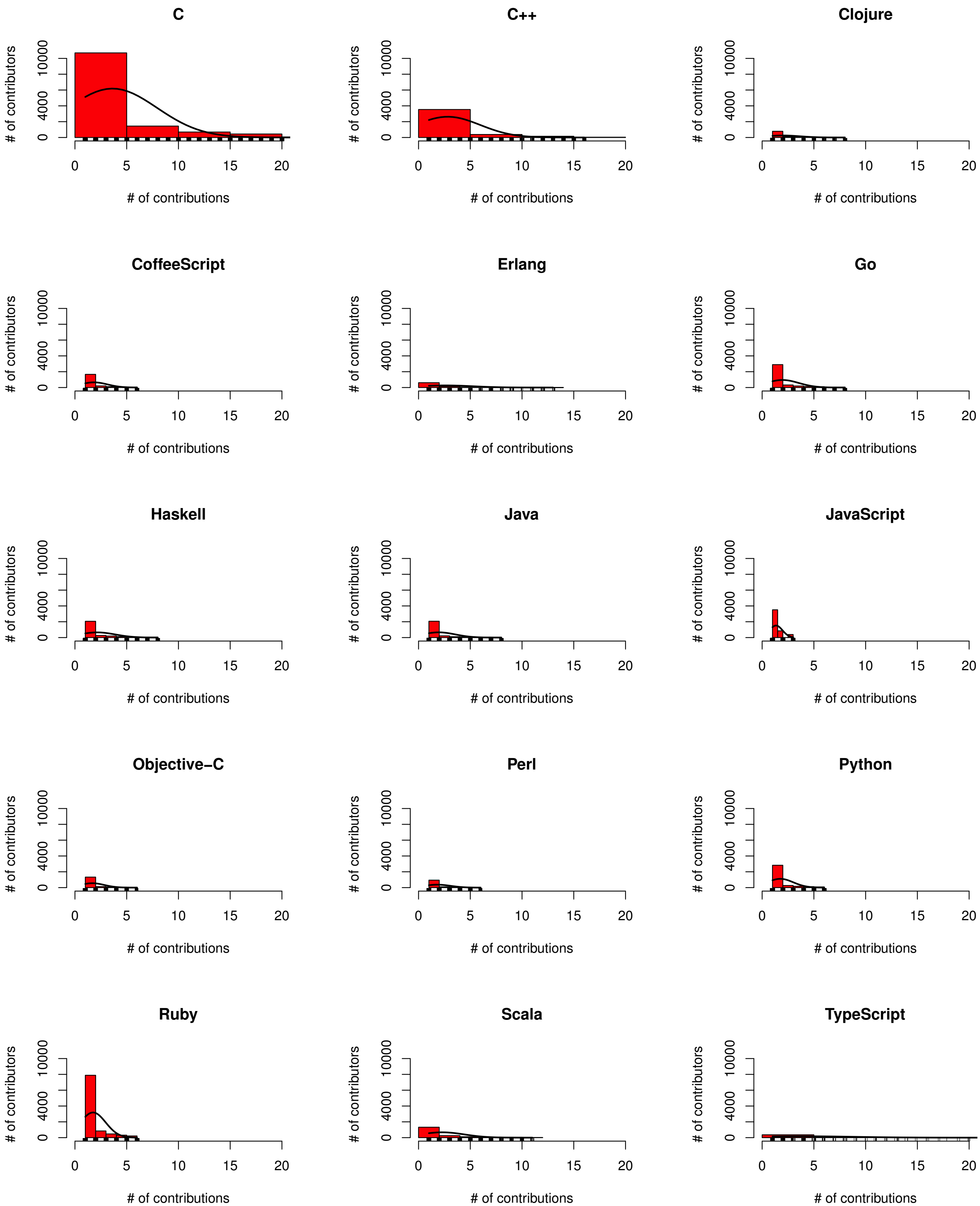


Fig. 1. Number of contributors and contributions per programming language (without outliers).

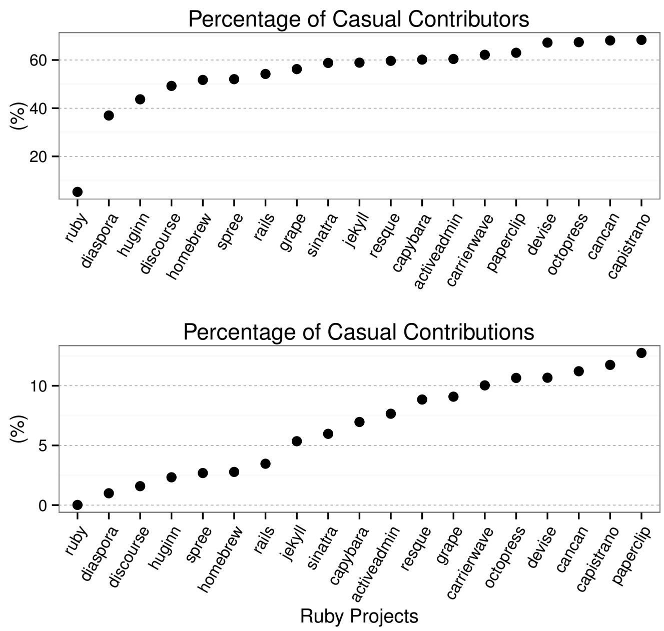


Fig. 3. Percentage of casual contributors and contributions in Ruby Projects.

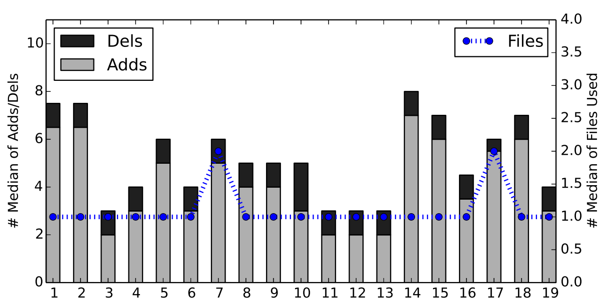


Fig. 4. The median number of additions, deletions, and files touched of each casual contribution performed on the Ruby projects analyzed. From left to right, projects are: 1. `activeadmin`, 2. `cancan`, 3. `capistrano`, 4. `capybara`, 5. `carrierwave`, 6. `devise`, 7. `diaspora`, 8. `discourse`, 9. `grape`, 10. `homebrew-cask`, 11. `huginn`, 12. `jekyll`, 13. `octopress`, 14. `paperclip`, 15. `rails`, 16. `resque`, 17. `ruby`, 18. `sinatra`, and 19. `spree`.

cantly among the analyzed projects. The project `paperclip` is the one with the highest number of additions and deletions among the Ruby projects (Adds mean, 3rd Quartile, and Std. deviation: 4.15, 5.75, and 1.72. Dels mean, 3rd Quartile, and Std. deviation: 1.02, 1.0, and 0.26). With more than 8 years old, 63% of its contributors are casual ones, who contributed to 12.74% of the project. Analyzing the contributions with the highest number of additions and deletions, we found that these casual contributions are far from being trivial. For instance, user `geemus` implemented an integration with `fog` [34], a library to manage cloud services. This contribution has 225 additions (98 lines of Ruby code, 107 lines of testing code, and 22 lines of configuration code) and 2 deletions (in a configuration file), performed in 6 different files. In this contribution, the author added an isolated module, which did not require him to deal with details of existing code. Conversely, the contribution `4b8dce4` [4], which has 184 additions and 63 deletions in 2 different files, added a support for blacklisting certain content_types. For this case, the contributor needed to understand the internal details of existing code in order to improve it. Yet, this contribution was also backed up with new 155 lines of testing code.

To provide a different perspective, we also analyzed the contributions with the lowest number of additions and deletions. In fact, 22.7% of the casual contributions performed on Ruby projects changed a single line of code (22.93%, when considering all analyzed projects). Even though the size of them are rather small, the intentions of the contributions vary greatly. For instance, some of them are aimed at (1) preventing a type from being null [21], (2) updating documentation files [29], (3) setting an option to a default value [15], or (4) improving performance [33]. In particular, this performance contribution is an interesting example of how these contributions can be challenging and, at the same time, concise. In this contribution, the author inlined two methods (thus decreasing the number of virtual function calls). Such one-line modification was done in a file with 595 lines of code, and required the author an in-depth knowledge of the application source code. These results suggest that the casual contributions cannot be seen as trivial.

To further investigate this matter, we selected a statistically significant sample of 384 casual contributions. This sample was randomly selected, and no distinction between the projects was made (*e.g.,* to select more samples from the largest and oldest projects). These contributions were performed in 138 different projects and have, on average, 107.9 additions, 29.88 deletions, performed in 2.02 different source code files. For each contribution, we studied the commit message and the code changes. Since some code changes require an in-depth knowledge of the application domain, we also searched in mailing lists, Q&A websites, and the issue associated with it, if available. We identified 8 categories of casual contributions, summarized at Table III. Discussions for each one of them are provided next.

TABLE III
THE CATEGORIZATION OF THE CASUAL CONTRIBUTIONS.

Category	**#**	**%**
Bug Fix	116	30.20%
Documentation	110	28.64%
Add New Feature	72	18.75%
Refactoring	34	8.85%
Update Version/Dependencies	25	6.51%
Improve Error/Help Messages	14	3.64%
Improve Resource Usage	8	2.08%
Add Test Cases	5	1.30%

Bug fix (116 occurrences). It is the most common kind of casual contribution found in our dataset. Also, the kind of bug fix, as well as the solution employed, are rather diverse. Some examples include: layout fix [17], fixing compilation problems [8], and fixing a broken URL [10]. Still, some bug fixes are far from being trivial, as the one that fixed a race condition at the linux operating system [24]. Not only difficult to identify (such bugs are non-deterministic), the solution employed was also scattered between C preprocessors, which difficulties the reasoning of the compiled program. Yet, we found that some of these bugs were discovered by the casual contributor, and were unknown from the core developers, as stated in a commit message: "*Ran into some issues with sending an unescaped apostrophe, but by adding it to the list of characters to escape, this problem is now fixed for me.*" [9]. Also, only 28 bug fixes were associated with a Github issue. We believe that if more bugs were reported with issues, more casual contributors would work on them.

Documentation (110 occurrences). This category includes fix for typos, grammar, translation, formatting, and documentation issues. We believe that these contributions are popular because programmers are still getting acquainted with the project. Thus, while reading the documentation or the source code, they might have found an issue and decided to fix it. Although these contributions do not require significant programming effort, we found contributions that have thoroughly rewritten the original material (*e.g.,* translations usually require dozens of additions [28]). Also, we found that 27 out of these 110 contributions were fixing typos on code examples. This finding reinforces the importance of complete and verified working code examples [69].

Add New Feature (72 occurrences). This category groups contributions that add new features, plugins, notification mechanisms, and/or drivers. Some of the examples include (1) adding a new option for a command line tool [6], (2) adding support for disabling an option [18], and (3) adding support for IPv6 remote hosts [5]. Interestingly, 24 out of the 72 contributions in this category were performed at the Linux operating system. Most of them were adding support for a new driver/device, which usually require few additions. For instance, a contribution that added support for a new USB device just needed to add two global variables, which holds the serial id of the USB device, and then add these two variables to the array of device ids [31], totalizing 8 additions.

Refactoring (34 occurrences). Refactorings are code-to-code transformations that change the internal structure of a program but not its behavior [44]. While some of the refactorings are straight-forward to be performed, *e.g.,* removing unused variables [30], some others require better technical skills, significant effort, and a good knowledge of the application source code, *e.g.,* composing several methods into a single one [25].

Update Version/Dependencies (25 occurrences). This category summarizes contributions that are aimed at upgrading the version of a software or its dependencies. This can happen on build files [3], configuration files [12], or on its dependencies [23]. More interestingly, however, is that although these modifications do not require one to be strongly familiar with the source code, it does require the casual contributor to be familiar with the project file cycle. This finding suggests that, although some casual contributors do not actively contribute to OSS projects, they closely *follow* their evolution.

Improve Error/Help Messages (14 occurrences). In this category, we grouped contributions that improved error/warning/help messages. Examples include: downgrading the logging level from error to warning [1], or providing more detailed help messages [2]. We hypothesize that the casual contributors have faced unclear/annoying error messages, and decided to improve them, as one casual contributor reported: "*The condition is harmless and no need to scare the user*".

Improve Resource Usage (8 occurrences). Here we grouped casual contributions that are aimed at improving resource usage, typically CPU [32], but we also found contributions that deal with network bandwidth [16], and disk space [13]. In particular, we found one interesting example in which the contributor updated the URI of an image, from a relative location, to an absolute one, thus avoiding HTTP roundtrips [19]. This contribution, albeit straight-forward to implement, requires one to understand details of the application source code. As another example of the non triviality of these contributions, we found a contribution aimed at improving the performance of system call at the linux operating system [20]. As the casual contributor described: "*the syscall personality (PER_LINUX32) has poor performance because it failed to find the Linux/x86 execution domain. [..] To resolve the issue, execution domain Linux/x86 is always registered in initialization time for IA-64 architecture*".

Add test cases (5 occurrences). Finally, we found casual contributions with the unique intention to improve the test suite. These contributions did not add new functional code, only testing code. Interestingly, one user submitted a failing test case to report a failure in the software [7]. Also, none of these contributions were associated with an issue, which suggests that the contributor felt the necessity of improving the test suite, or reporting an error. Still, we observed that the test cases are rather simple. On average they have 18 additions and 0.4 deletions. In all cases, the project that received the contribution had an existing test suite, and the contributor added new testing code using the existing testing infrastructure.

C. RQ3. How do casual contributors and project maintainers perceive casual contributions?

Here we discuss the answers of our two surveys.

Firstly, we explicitly asked casual contributors and project maintainers "what motivates casual contributors' behavior". The top cited motivation was ***scratch their own itch***, highly mentioned by both casual contributors (90 out of 197) and project maintainers (23 out of 64). Developers reported that when they are blocked by or bugged with some issue, they fix it and send it back to the community. In many cases, this is related to something that blocks or impacts developers own projects or their work. For example, casual contributor reported that *"urgent bug fix or feature requirement for my projects"* are motivators; and a project maintainer said that *"being blocked by the bug is a big motivator"*. This finding corroborates with the high number of bug fixes found in **RQ2**. Also, it indicates that the the claim made by Eric Raymond [67] that "every good work of software starts by scratching a developer's personal itch" is valid, and is in accordance to what is reported by Fogel [43]: *"software results when the programmer has a personal interest in seeing the problem solved"*. It seems that scratching itches is becoming easier and more common given the high frequency observed in this work.

Part of this high number of casual contributions can be explained by the pull-request model, which provided a ***clear and easy contribution process***. It was mentioned by 9 out of 64 project maintainers. Therefore, there are indications GitHub tools facilitates the scratching personal itches. This was clearly explained by a project maintainer: *"[Casual Contributors] have a minor itch to scratch, the barrier to contributing is low (GitHub makes it easy; source is pretty simple to modify; standard structure) so there's no artificial reason you wouldn't see a lot of small contributions"*.

Aligned with some studies on the motivation behind OSS contributors [77], [54], we found that ***give back to community*** fosters casual contributions, as said by one casual contributor *"As I use a lot of OSS projects, I like to give back to the community"*. Another motivation that is inline with the literature is ***gaining reputation and prestige*** [61], [48], [49].

Not among the top cited motivations, we found that four casual contributors reported that their motivation was ***improving the project***. The following quotes clearly illustrate such motivation: *"I want to improve the quality of the project"*, *"That the project is in better shape after my contribution"*. Although they do not explicitly mention what kind of improvement they refer, they can be related to the categories of contributions reported under **RQ2**, namely *refactoring*, *improve error/help messages*, *improve resource usage* and *add test cases*.

In addition to motivation, we investigated the reasons why casual contributors do not become full active contributors. ***Lack of time*** was far the most cited reason by the casual contributors (96 out of 197), like one mentioned *"I don't have time to devote to a more active role"*. Some of them justifies it because they already spend too much time in their job, or because there is ***no income from it*** (*e.g., "not paid to do so"*). It seems that OSS projects are receiving more contributions from developers that work on commercial closed-source industry, which software relies on OSS projects. From the perspective of the project maintainer, ***Lack of time*** was also the most mentioned reason why casual contributors do not become a long term contributor (17 out of 64 respondents). The following quote exemplify it: *"People often don't have the time or desire to be long term contributors"*.

Another recurrently mentioned reason to not become an active contributor was actually the top cited motivation to contribute. Both casual contributors and project maintaers reported that ***developers mainly want to scratch their own itch***, so, since the beginning, they are not willing to actively contribute. Some casual contributors mentioned that this is usual, and when they face some small issue or something blocking their work, they easily drop a contribution to the projects without any further commitment: *"I just need a bug fixed, I don't aim to improve the project substantially"*, reported a contributor. Aligned with the answers received from the casual contributors, project maintainers mentioned that, as soon as the contributors are done scratching their itches, they usually do not need or do not want to get back: *"They saw something simple that needs fixing and fixed it. They don't necessarily want to actively follow the project"*.

We also found people who reported that they do not contribute because of their ***limited skills or knowledge***. Some also mentioned that the effort and knowledge needed to become a full contributor was too high. In both cases, they prefer to work on small or peripheral issues, which do not need specific abilities and usually require low effort. Like one of them said: *"lack of skills (most of the low hanging fruit is gone)"*.

Project maintainers noticed this, and eight participants mentioned that ***code/project is hard to learn*** was a reason why casual contributors do not become more active. They reported that casual contributors send a fix to a simple bug or a small feature, but they do not have the skills or the capability to go deeper in the code, as illustrated by the following quote: *"Interest in particular projects wanes quickly (like any other sort of aspirational project), and actually going deeper to solve real problems or add bigger features is in another difficulty class from what most of our contributors are capable of as beginners"*. Interestingly, however, one casual contributors suggested the opposite, as one mentioned: *"While I only have one commit in the Linux kernel [..] I'm a Debian developer, where I maintain multiple packages, among them a patched Linux kernel. I've reported multiple bugs against the Linux kernel, helped testing fixes, follow multiple mailing lists etc"*. Therefore, although some of the casual contributors can be considered beginners, some of them are highly skilled.

We found other less recurrent reasons. For example, the respondents reported that ***casual contributors are usually active in other projects***, and, sometimes they need to implement a feature or fix a bug from a specific project. One project maintainer mentioned that *"...they're working on a higher-level problem and have been blocked by our lower-level library for some reason"*, and another pointed that *"Casual contributors on our project may also be active contributors on other projects in the ecosystem"*. This reason was confirmed by 13 casual contributors. They mentioned that they do not have enough time to contribute to other projects, because they maintain or actively contribute to other OSS projects. Another group (8 people) mentioned that they do not become active contributors because there are too many project they use and make contributions eventually. Thus, instead of being loyal with a few set of projects, they make sparse contributions.

An interesting reason reported by the one project maintainer was that: *"the project maintainer fail to encourage more contributions which ideally leads to active contributors. A welcoming and friendly project with professionalism that treats everyone with respect and gives the required credit is the first thing a project has to get right"*. This was also reported by two casual contributors (*"the project is not very welcoming for new contributors"*). Surprisingly, some casual contributors attributed the reason to their satisfaction with the product (*"the project is good enough on its own for my needs"*) and to their feeling that projects do not need more active contributors, like one of them mentioned *"Projects that I use daily [..] have enough contributors"*. In these cases, it would be important to make it clear for the casual contributors that, if the project is maintained by volunteers, it is necessary to have a continuous influx of long term contributors.

We also asked the participants their opinion about the main benefits and problems brought by the casual contributors phenomenon. The overall impression is that the benefits overcome the drawbacks brought by this phenomenon. One quote from a project maintainer shows: *"Every little piece helps everyone else. We stand on the shoulders of many small giants. Problems? None"*. Among the answers received from the project maintainer, the most reported perceived benefits were ***Small peripheral issues solved quickly*** (mentioned by 16 project maintainers), ***Bugs that would never appear are closed (a new set of eyes)*** (mentioned by 13 project maintainers) and the ***continuous code improvement*** (mentioned by 6 project maintainers). Regarding the last mentioned category, once again we could triangulate our findings with the categories of contributions found in **RQ2**. In the following quote we can see a report from a project maintainer that explicitly mentioned an improvement received from a casual contributor: *"Some [contributions] are more substantial. e.g. The other day a guy reached out to me cold and offered a simple numerical trick that speed up my automatic differentiation package by 2 orders of magnitude in one mode"*.

On the other side, the most reported problems were ***Time spent by the core members to review newcomers' code*** (reported by 12 people) and ***contributions may go unmaintained*** (reported by 5 people). One interesting thing called our attention in this question: 10 project maintianers mention they see ***no notable problems*** with casual contributions. One project maintainer was a little sarcastic while answering that: *"What are the problems? I think this is the wrong question. You're just going to go down a rabbit hole of people whining and complaining about sloppy code, and contributors not understanding various complications within the code structure"*.

IV. Implications

Casual Contributors. First, they can see that they are not alone and that this behavior is, in fact, rather common in OSS communities (**RQ1**). Second, 22.93% of the casual contributions changed a single line of code (**RQ2**). Thus, a developer does not need to be shy to contribute, even though her contribution is small. Third, this study revealed that project maintianers believe that casual contributions are a healthy way of contributing to OSS (**RQ3**). Therefore, they can become even more motivated to do this kind of contribution.

Project owners. We found that although 28.64% of the casual contributions were related to fixing documentation issues, several other kind of contributions were performed (**RQ2**). Project owners can benefit from this finding, by labeling tasks specific for casual contributors. Similarly, some casual contributors are more comfortable on solving low effort tasks (**RQ3**). Thus, project owners can create specific roles for casual contributors (*e.g.,* casual translators), which could also foster more engagement. Also, we found 5 contributions that only added new test cases. These contributions happened only because the project had an existing testing infrastructure. This finding may encourage project owners to create and maintain a testing infrastructure, in case it does not exist. Still, we found that the majority of the contributions analyzed were bug fixes. However, only 24.13% of them were associated with a Github issue. We believe that, if project owners open more bug fix issues, new casual contributors would work on them. Finally, since several project maintainers do not have enough time to review casual contributions, they can introduce "contributions guidelines", so that newcomers can read and get acquainted with them, therefore reducing code review effort.

Reseachers. Researchers can also benefit from this study. As we found that "Time spent by the core members to review newcomers code" is the most common problem that the casual contributions bring (**RQ3**), researchers can introduce new techniques to ease source code review. Also, researchers can propose techniques aimed at assigning reviews to the core members that might be more familiar with the code.

Tool Builders. In this study we found that a significative proportion of the contributors of the analyzed projects are actually casual contributors (**RQ1**). Tool builders can take advantage of this finding and improve visualization tools. For instance, Github could provide a feature to present the "degree of casual contributions". Therefore, casual contributors could easily identify projects that are more likely to receive these kind of contributions. Tools builders can go further and provide mechanisms to notify core members when newcomers arrive in the project, so that core members can provide further assistance on the onboarding process.

CS Professors. As we found that several contributions do not require a high number of source code modifications (**RQ2**), professors can better motivate students to start collaborating with OSS projects. One way to do this is assigning students to work on real-world software issues. Also, since we found that "project owners fail to encourage the newcomers to stay" (**RQ3**), professors can help them in this direction by providing long-term open-source assignments for students. Although some students might quit the project after the course is done, some may get interested and stay longer.

V. Related Work

Park and Jensen [62] studied the information needs that newcomers have. The authors showed that visualization tools support the first steps of these newcomers. Also, the authors observed that the newcomers that use these tools finish their contributions faster, and have better code comprehension. Von Krogh *et al.* [78] studied the joining process of FreeNet project. They found that newcomers follow the project activities before making a contribution. Despite their interest on the onboarding process, they focused on the contributors that become active members of the project, not taking into consideration the ones that place casual contributions.

Some other studies analyzed how social factors influence the retention of newcomers on OSS projects [79], [41], [37]. These studies studied social networks (*e.g.,* mailing lists) in order to understand (1) with whom newcomers collaborate, and (2) how the network evolve along the years. Similarly, these studies do not focus on the contributors that do not focus on long-term commitment. Moreover, Jensen *et al.* [50] analyzed four projects to understand if newcomers are quickly answered, if their gender and nationality impact the kind of answer that they receive, and if the treatment they receive is similar to the ones that other members of the project receive.

Nakakoji *et al.* [59] studied four OSS projects aimed at understanding the evolution of its communities. Among the contributions, the authors coined the term *onion patch*, which refers to the 8 roles of the project members. The hypothesis is that new contributors start as a "lurker", then they join a mailing list, watch other interactions, afterwards they slowly become more involved in the project, contributing with code and becoming active member. In our study we observed that it is common to find casual contributors that are highly motivated by their "own itches", and do not necessarily follow this model or want to become active members.

Some parts of the literature focus on the forces of motivation that drive newcomers toward projects. Lakhani and Wolf [54], for example, found that extrinsic benefits primarily motivate new contributors, together with enjoyment, challenges and improving programming skills. Hars and Ou [48] reported that internal motivation plays a role, but note that external factors, such as building human capital and personal software solution needs, are more influential. Shah [70] distinguished between two different contributors: need-driven and hobbyists. In our study we could evidence that "scratching own itches" (personal and institutional needs) was by far the most reported motivation. This can be an indication that this kind of contribution is primarily driven by extrinsic motivation.

Finally, as regarding the casual contributors, several authors have acknowledged the existence and the growth of this behavior [64], [63], [46], [76]. However, even though some of these authors suggest that it is important to further understand the impact of this kind of contribution in the projects that receive it, as well as its problems and benefits, to the best of our knowledge, there is no prior work in the literature that addresses this topic in details.

VI. Threats to Validity

Internal Validity. First, although we mitigated the problem of contributors using different full names, therefore being wrongly categorized as casual contributors, the technique was not effective in solving all these problems. For instance, we found some contributors that used a common name (*e.g.,* Paul) as their full names. This fact prevents disambiguation techniques from being successful. One might suggest to disambiguate these contributors using the ones that Github shows on the project's webpage. However, although Github shows the total number of contributors of a given project, it lists only the top 100 most active ones, and a significant proportion of projects analyzed are far beyond this number (see Table II). Thus, using Github as our ground truth, we manually compared the total number of contributors that we found, with the ones that Github reports. We found that our study reports between 7% to 10% of additional contributors. However, we believe that this data is not sufficient to skew the main results of our study — as showed, on average, 48.98% of the contributors are casual ones. Second, we selected projects based on their popularity (number of stars) on Github. This means that these projects are not necessarily popular because they are highly used. For instance, we found a project that created the "Arnold Schwarzenegger programming language"[11]. Although this project is interesting, it is not likely that it will be adopted in practice.

External Validity. Although we investigated 275 popular OSS projects written in, at least, 17 different programming languages, we likely did not discover all possible characteristics of casual contributions. We are aware that each project has its singularities and that the OSS universe is large and deep, meaning the amount and the perception of the casual contributions can differ according to the project or the ecosystem. Our strategy was to focus on GitHub popular projects, and cannot be generalized to other projects hosted on other forges. With our methodology, we expected that similar analysis can be conducted by others when they become relevant.

VII. Conclusion

In this paper we conducted an in-depth study on the casual contributions made into 275 non-trivial OSS projects. For each project, we initially analyzed quantitative characteristics of these contributions. We found that casual contributors are rater common. More than 48% of the contributors are actually casual ones. Also, after a manual inspection of a representative sample of 384 casual contributions, we discovered that the contributions are far from being trivial. Indeed, several solutions require an in-depth knowledge of the application source code. Still, we asked casual contributors and project maintainers about what drives this behavior. Most of them suggested that they are motivated by their personal needs, *e.g.*, fixing bugs that block or impact the development of other projects. Yet, there are evidence that casual contributors do not become more active mainly because of time constraints. Finally, we found that although these contributions bring many benefits, they also cause some problems, mainly related to time required from core members reviewing the quality of code.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful comments. Gustavo is supported by CAPES, CNPq (455261/2015-6), and FACEPE (ACE-0323-1.03/15). Igor is supported by CNPq (477831/2013-3), and UTFPR. Marco is supported by CNPq, FAPESP, NAWEB, and NAPSoL.

References

[1] Acpi: Change package length error to warning. https://github.com/torvalds/linux/commit/1371c89. Accessed: 2015-11-07.

[2] Add more help messages to cloudcfg utility. https://github.com/kubernetes/kubernetes/commit/dd04554. Accessed: 2015-11-07.

[3] Add rubinius to build matrix with allowed failure. https://github.com/discourse/discourse/commit/53f35cc. Accessed: 2015-11-07.

[4] Add support for blacklisting certain content_types. https://github.com/thoughtbot/paperclip/commit/4b8dce4. Accessed: 2015-11-07.

[5] Add support for ipv6 remote hosts. https://github.com/apenwarr/sshuttle/commit/95c9b78. Accessed: 2015-11-07.

[6] added –get-id option to print video ids. https://github.com/rg3/youtube-dl/commit/1a2adf3. Accessed: 2015-11-07.

[7] Added failing test to demonstrate digest authentication failure. https://github.com/rails/rails/commit/53c1ae9. Accessed: 2015-11-07.

[8] Added flexiblecontexts extension in parser.hs to fix compilation. https://github.com/koalaman/shellcheck/commit/f054e2e. Accessed: 2015-11-07.

[9] Added single quote to the chars to escape. https://github.com/AFNetworking/AFNetworking/pull/713. Accessed: 2015-11-07.

[10] airchat.pl: Fixed broken, space-laden url/href. https://github.com/lulzlabs/AirChat/commit/03cd91d. Accessed: 2015-11-07.

[11] Arnold schwarzenegger based programming language. https://github.com/lhartikk/ArnoldC. Accessed: 2015-11-07.

[12] Bump mono-mdk to version 4.2.0. https://github.com/caskroom/homebrew-cask/commit/4984e28. Accessed: 2015-11-07.

[13] Change ios logging directory to use cache directory instead of documents directory. https://github.com/CocoaLumberjack/CocoaLumberjack/commit/3bf4585. Accessed: 2015-11-07.

[14] A curated list of awesome python frameworks, libraries and software. http://github.com/vinta/awesome-python. Accessed: 2015-11-07.

[15] Default fog_public option to true (as stated in documentation). https://github.com/thoughtbot/paperclip/commit/62a9f64. Accessed: 2015-11-07.

[16] Disable auto-saves until the data has been loaded. https://github.com/github/hubot-scripts/commit/2b20f1d. Accessed: 2015-11-07.

[17] Ensure hud overlaywindow is visible above other windows. https://github.com/TransitApp/SVProgressHUD/commit/eebd6ec. Accessed: 2015-11-07.

[18] feat(ngoptions): add support for disabling an option. https://github.com/angular/angular.js/commit/da9eac8. Accessed: 2015-11-07.

[19] fix: direct links for celery man. https://github.com/github/hubot-scripts/commit/a33c8ba. Accessed: 2015-11-07.

[20] fix personality(per_linux32) performance issue. https://github.com/torvalds/linux/commit/839052d. Accessed: 2015-11-07.

[21] Fixed 'type' being nil on windows 7 error. https://github.com/thoughtbot/paperclip/commit/6f2ca93. Accessed: 2015-11-07.

[22] jquery bracket library for organizing single and double elimination tournaments. http://github.com/teijo/jquery-bracket. Accessed: 2015-11-07.

[23] lyx.rb: update to 2.1.4. https://github.com/caskroom/homebrew-cask/commit/3046a4d. Accessed: 2015-11-07.

[24] ppc32: fix destroy_context() race condition. https://github.com/torvalds/linux/commit/ddca3b8. Accessed: 2015-11-07.

[25] Routing methods dsl refactored to get rid of explicit paths parameter. https://github.com/ruby-grape/grape/commit/d3f0c29. Accessed: 2015-11-07.

[26] A showcase of the best typefaces from the google web fonts directory. https://github.com/ubuwaits/beautiful-web-type. Accessed: 2015-11-07.

[27] The swift programming language in chinese. https://github.com/numbbbbb/the-swift-programming-language-in-chinese. Accessed: 2015-11-07.

[28] Update fr.coffee. https://github.com/codecombat/codecombat/commit/237b97a. Accessed: 2015-11-07.

[29] Update readme.md. https://github.com/thoughtbot/paperclip/commit/d49bca2. Accessed: 2015-11-07.

[30] Usb: digi_acceleport further buffer clean up. https://github.com/torvalds/linux/commit/5fea2a4. Accessed: 2015-11-07.

[31] Usb: ftdi_sio: Add support for ge healthcare nemo tracker device. https://github.com/torvalds/linux/commit/9c491c3. Accessed: 2015-11-07.

[32] Use constant-time string comparison for auth. https://github.com/reddit/reddit/commit/83058d4. Accessed: 2015-11-07.

[33] Uses a faster implementation for ensure_required_validations! https://github.com/thoughtbot/paperclip/commit/6b1f610. Accessed: 2015-11-07.

[34] Wip fog integration. https://github.com/thoughtbot/paperclip/commit/4047d32. Accessed: 2015-11-07.

[35] Github archive, jan 2015. Accessed on 2015-12-22.

[36] Github press, jan 2015. Accessed on 2015-12-22.

[37] C. Bird. Sociotechnical coordination and collaboration in open source software. In *Proceedings of the 2011 27th IEEE International Conference on Software Maintenance*, ICSM '11, pages 568–573, Washington, DC, USA, 2011. IEEE Computer Society.

[38] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining email social networks. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, MSR '06, pages 137–143, 2006.

[39] M. Burke, C. Marlow, and T. Lento. Feed me: Motivating newcomer contribution in social network sites. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 945–954, New York, NY, USA, 2009. ACM.

[40] A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Rev.*, 51(4):661–703, Nov. 2009.

[41] N. Ducheneaut. Socialization in an open source software community: A socio-technical analysis. *Computer Supported Cooperative Work*, 14(4):323–368, Aug. 2005.

[42] F. Fagerholm, P. Johnson, A. S. Guinea, J. Borenstein, and J. Mnch. Onboarding in open source projects. *IEEE Software*, 31(6):54–61, Nov. 2014.

[43] K. Fogel. *Producing Open Source Software: How to Run a Successful Free Software Project*. O'Reilly Media, first edition, Feb 2013.

[44] M. Folwer. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[45] M. Goeminne and T. Mens. Evidence for the pareto principle in open source software activity. In *In the Joint Porceedings of the 1st International workshop on Model Driven Software Maintenance and 5th International Workshop on Software Quality and Maintainability*, pages 74–82, 2011.

[46] G. Gousios, M. Pinzger, and A. v. Deursen. An exploratory study of the pull-based software development model. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 345–355, New York, NY, USA, 2014. ACM.

[47] G. Gousios, A. Zaidman, M. D. Storey, and A. van Deursen. Work practices and challenges in pull-based development: The integrator's perspective. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, pages 358–368, 2015.

[48] A. Hars and S. Ou. Working for free? motivations of participating in open source projects. In *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, pages 1–9. IEEE, 2001.

[49] G. Hertel, S. Niedner, and S. Herrmann. Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel. *Research Policy*, 32(7):1159–1177, July 2003.

[50] C. Jensen, S. King, and V. Kuechler. Joining free/open source software communities: An analysis of newbies' first interactions on project mailing lists. In *Proceedings of the 44th Hawaii International Conference on System Sciences*, HICSS '10, pages 1–10. IEEE, Jan. 2011.

[51] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian. The promises and perils of mining github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 92–101, 2014.

[52] B. Kitchenham and S. Pfleeger. Personal opinion surveys. In F. Shull, J. Singer, and D. Sjberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 63–92. Springer London, 2008.

[53] B. Kitchenham, S. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. El Emam, and J. Rosenberg. Preliminary guidelines for empirical research in software engineering. *Software Engineering, IEEE Transactions on*, 28(8):721–734, Aug 2002.

[54] K. Lakhani and R. Wolf. *Perspectives on Free and Open Source Software*, chapter Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects, pages 1–22. The MIT Press, Cambridge, Mass., 2005.

[55] J. Marlow, L. Dabbish, and J. Herbsleb. Impression formation in online peer production: Activity traces and personal profiles in github. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, CSCW '13, pages 117–128, 2013.

[56] N. McDonald and S. Goggins. Performance and participation in open source software on github. In *CHI '13 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '13, pages 139–144, New York, NY, USA, 2013. ACM.

[57] I. Moura, G. Pinto, F. Ebert, and F. Castor. Mining energy-aware commits. In *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*, pages 56–67, May 2015.

[58] M. Nagappan, T. Zimmermann, and C. Bird. Diversity in software engineering research. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 466–476, 2013.

[59] K. Nakakoji, Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye. Evolution patterns of open-source software systems and communities. In *Proceedings of the International Workshop on Principles of Software Evolution*, IWPSE '02, pages 76–85, New York, NY, USA, 2002. ACM.

[60] S. Nanz and C. A. Furia. A comparative study of programming languages in rosetta code. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 778–788, 2015.

[61] S. Oreg and O. Nov. Exploring motivations for contributing to open source initiatives: The roles of contribution context and personal values. *Computers in Human Behavior*, 24(5):2055–2073, Sept. 2008.

[62] Y. Park and C. Jensen. Beyond pretty pictures: Examining the benefits of code visualization for open source newcomers. In *Proceedings of the 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, VISSOFT '09, pages 3–10. IEEE, Sept. 2009.

[63] R. Pham, L. Singer, O. Liskin, F. Figueira Filho, and K. Schneider. Creating a shared understanding of testing culture on a social coding site. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 112–121, Piscataway, NJ, USA, 2013. IEEE Press.

[64] R. Pham, L. Singer, and K. Schneider. Building test suites in social coding sites by leveraging drive-by commits. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1209–1212, 2013.

[65] G. Pinto, I. Steinmacher, and M. Gerosa. Casual contributors — replication package. http://bit.ly/casual-contributors. Accessed: 2015-12-21.

[66] B. Ray, D. Posnett, V. Filkov, and P. Devanbu. A large scale study of programming languages and code quality in github. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 155–165, 2014.

[67] E. S. Raymond. *The Cathedral and the Bazaar*. O'Reilly & Associates, Inc., 1999.

[68] E. S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001.

[69] M. P. Robillard. What makes apis hard to learn? answers from developers. *IEEE Softw.*, 26(6):27–34, Nov. 2009.

[70] S. K. Shah. Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science*, 52(7):1000–1014, July 2006.

[71] I. Steinmacher, T. Conte, M. A. Gerosa, and D. F. Redmiles. Social barriers faced by newcomers placing their first contribution in open source software projects. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '15, pages 1–13, New York, NY, USA, Feb. 2015. ACM.

[72] I. Steinmacher, M. A. G. Silva, M. A. Gerosa, and D. F. Redmiles. A systematic literature review on the barriers faced by newcomers to open source software projects. *Information and Software Technology*, 59:67–85, Mar. 2015.

[73] I. Steinmacher, I. S. Wiese, A. P. Chaves, and M. A. Gerosa. Why do newcomers abandon open source software projects? In *Proceedings of the 2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '13, pages 25–32. IEEE, 2013.

[74] A. Strauss and J. M. Corbin. *Basics of Qualitative Research : Techniques and Procedures for Developing Grounded Theory*. SAGE Publications, 3rd edition, 2007.

[75] H.-T. Tsai and P. Pai. Why do newcomers participate in virtual communities? an integration of self-determination and relationship management theories. *Decision Support Systems*, 57:178–187, Jan. 2014.

[76] B. Vasilescu, V. Filkov, and A. Serebrenik. Perceptions of diversity on github: A user survey. In *Proceedings of the 2015 8th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE '15. IEEE, 2015.

[77] G. von Krogh, S. Haefliger, S. Spaeth, and M. W. Wallin. Carrots and rainbows: Motivation and social practice in open source software development. *MIS Quarterly*, 36(2):649–676, June 2012.

[78] G. von Krogh, S. Spaeth, and K. R. Lakhani. Community, joining, and specialization in open source software innovation: A case study. *Research Policy*, 32(7):1217–1241, 2003.

[79] M. Zhou and A. Mockus. Who will stay in the floss community? modelling participant's initial behaviour. *IEEE Transactions on Software Engineering*, 41(1):82–99, 2015.