Inadequate Testing, Time Pressure, and (Over) Confidence: A Tale of Continuous Integration Users

Gustavo Pinto
Federal Institute of Pará
Belém, PA, Brazil
gustavo.pinto@ifpa.edu.br

Marcel Rebouças
Federal University of Pernambuco
Recife, PE, Brazil
mscr@cin.ufpe.br

Fernando Castor
Federal University of Pernambuco
Recife, PE, Brazil
castor@cin.ufpe.br

***Abstract*—Continuous Integration (CI) prescribes that changes should be integrated into the main codebase as often as possible and that the system should be built frequently. To support CI, a number of software tools have been developed. However, little is known about the main reasons for build breakage and whether CI delivers its promise of early problem detection and smooth integration. To shed light on this issue, we conducted a survey with 158 CI users. Through a qualitative investigation we found that inadequate testing is the most common technical reason related to build breakage, whereas lack of time plays a role on the social reasons. Still, although some respondents reported that CI usage increases the confidence that the code is in a known state, some respondents also reported that there is a false sense of confidence when blindly trusting tests.**

***Keywords*-Continuous Integration; Build Breakage; Survey;**

I. Introduction

Continuous Integration (CI) is a core agile practice [1]. It is aimed at integrating code that is under development with the mainline codebase in a shared repository at least daily, leading to multiple integrations per day [2]. Each integration is properly verified by an automated build. Arguably, in a project that uses CI, errors can be detected quickly, their causes can be located with less effort, and the overall development process can be sped up. As a result, there is an increasing adoption of continuous integration by software development teams.

In spite of the increasing adoption, the large set of tools, and the well-known benefits, little is known about how software developers are dealing with the usage of continuous integration techniques. According to the literature [3], more research is needed to better understand the dynamics, process, benefits, and implications of CI usage. Starting from this premise, this paper presents an empirical study aimed at illuminating the the reasons for build breakage, and the benefits/problems of CI usage. Specifically, the question we are trying to answer is:

> **RQ.** What are the perceptions from CI users in terms of reasons for build breakage, and the benefits and problems of CI usage?

To provide answers to this question, we conducted an online survey with 158 CI users. Through quantitative and qualitative analyzes of these answers, our study produced a set of findings, many of which were unexpected. We discuss them in detail in Sections III—IV.

II. Survey Description

In order to investigate the perception of CI usage, we conducted an online survey. Our target population are software developers that have broken at least one build in a non-trivial open-source software.

A. Subjects

To select our subjects, we start looking at the most popular programming languages on Github, measured by the total number of files created in each language. Using this criteria, the chosen programming languages were: C, C++, C#, Clojure, CoffeeScript, Erlang, Go, Haskell, Java, JavaScript, Objective-C, Perl, PHP, Python, Ruby, Scala, and TypeScript. However, we removed CoffeScript and TypeScript since Travis did not offer support to these programming languages.

For each one of the remaining languages, we then selected the 50 most popular projects, measured by the number of stars, which is an explicit way for Github users to manifest their satisfaction with a project [4]. We also restricted our search to projects that have a `.travis.yml` file. This file stores all configuration required to run Travis' service. We ended up with an initial list of 750 travis-configured projects. When manually analyzing these projects, we observed that some of them do not properly fit for the purpose of this study, because:

- **Some projects are not software projects**. We found that several popular open-source projects are not software projects. Instead, these projects are used to share bookmarks[1], textbooks[2], and were mirrors of other repositories[3]. We excluded these projects.
- **Some projects are not active**. We found 30 projects that are not active. We believe it is important to focus on active projects only because we wanted to measure ongoing development. Therefore, we selected projects that have at least one commit and at least 2 committers in the last 12 months. We excluded these projects.

[1] `https://github.com/vinta/awesome-python`
[2] `https://github.com/SamyPesse/How-to-Make-a-Computer-Operating-System`
[3] `http://www.github.com/WordPress/WordPress`

After this manual process, we ended up with a list of 682 projects. For each one, we queried the Travis API[4] in order to retrieve builds' metadata. However, the Travis API did not answer 16 of our requests. Unfortunately, some of these requests are for projects such as `Angular.js` and `Rails`, which are highly active and have a long history of software builds. This fact reduced our final sample to 666 projects.

Overall, these projects performed about 737,000 builds, which 182,072 have failed. Our subjects constitute a random sample of 1,100 software developers, with valid email addresses, that have broken at least one build. We chose to focus on developers with broken builds because they might provide a fix for the build and, therefore, have a richer experience with CI than developers that have never faced such problems. We contacted participants individually by email, inviting them to participate in the survey.

B. Design

The survey used in this work was based on the recommendations of Kitchenham *et al.* [5], following the phases prescribed by the authors: planning, creating the survey, defining the target audience, evaluating, conducting the survey, and analyzing the results. We also employed a number of principles used to increase survey participation [6], such as sending personalized invitations, allowing the participant to be completely anonymous, and asking closed and direct questions. We set up the survey as an on-line questionnaire (it can be found at: `https://docs.google.com/forms/d/1qqSkJ4gOqTSiSsvGGO5M1A-0cQOrYqUodcU1IXtw_Qo/prefill`). Before sending the link to our subjects, we created a first draft of the survey and informally presented it to a number of colleagues. Based on their remarks, we refined some of the questions and explanations, mainly to make them more precise. Along with the instructions of the survey, we included some examples as an attempt to clarify our intent. Participation was voluntary and the estimated time to complete the survey was 10-15 minutes. Over a period of 30 days, we obtained 158 responses, resulting in 14.4% of response rate.

C. Questions

Our initial survey consisted of 15 questions, 7 of which were open. The survey was divided in five sections: (1) technical background of the participants, (2) experience with CI techniques, (3) CI fundamentals, (4) reasons for build breakage, and (5) benefits and problems of CI usage. In this paper, we report on sections 4 and 5.

(4) Reasons for build breakage. Here we asked what are the technical reasons (Q12) and social reasons (Q13) that might have influenced the build breakage.

(5) Benefits and Problems of CI usage. Finally, we asked their opinion about the benefits (Q14) and problems (Q15) related to the use of continuous integration techniques.

[4] `http://api.travis-ci.org`

III. Reasons for Build Breakage

A. Technical Reasons for Build Breakage

We found several technical reasons that might explain build breakage. Among the most common ones, there is ***inadequate testing*** (33 occurrences). Respondents mentioned that some tests are "*Badly written that fail with minor bugfixes*" or even "*not enough tests*". However, one respondent mentioned that they might be tempted to skip tests execution, since "*running a test suite may be too slow and skipped*". These respondents use CI as a way to speed up the development process, while still relying on the test suite.

Moreover, some respondents claimed that ***version changes*** might play a role (12 occurrences). For instance, "*[the] version of a language component is different, and the change made to the language cause breakage*". Another drawback reported is related to ***dependency management*** (8 occurrences). As one example, one respondent mentioned that "*people sometimes don't update dependencies, so the CI server detects errors that do not happen locally*". It is important to note that ***dependency management*** and ***version changes*** are two key activities of build systems. Therefore, developers that may face technical problems with the usage of build systems, may also face them with CI systems.

Still, we noticed that some developers are facing barriers with the ***intricacy of the code base*** (14 occurrences) and ***lack of domain knowledge*** (18 occurrences). These developers claimed "*unfamiliarity with the architecture of the code and overall module interactions*", which might be due to a "*lack of experience with the project*". This is particularly relevant for open-source projects, since newcomers often do not know how the project is organized or how to start contributing [7]. In addition, the social facilities brought by social coding websites such as Github and Bitbucket lowered the barriers for one placing a contribution to an open-source project. Therefore, casual contributors that happen to enjoy one particular open-source project might be tempted to contribute [8]. However, without proper guidance or documentation, such contributors might not be aware of the workflow (*e.g.,* "*I didn't know about the linter (or that CI was being used) until after I submitted my patch.*")[5]. As opposed to the ***intricacy of the code base***, some developers ***missed edge cases*** (5 occurrences), such as "*syntax errors, formatting errors if using a linter*", which, according to one respondent, are due to an "*underestimation of impact of small changes*".

Finally, other not so common technical reasons include: ***git usage*** (4 occurrences), ***flaky tests*** (3 occurrences), and ***timezones*** (2 occurrences). Ultimately, five developers either perceived no technical reason related to build breakage or cannot recall.

B. Social Reasons for Build Breakage

When analyzing the answers, we found that the most common social reason associated with build breakage is ***time***

[5] `Linters` are tools for verifying code style guidelines.

pressure (36 occurrences). One particular respondent mentioned that he self-imposed this time pressure, because of an "eagerness to help", as he mentioned: "*I'll just make that change right now!*".

Another social reason is the ***lack of testing culture***, as evidenced by 17 respondents. In this particular finding, although the majority of the respondents acknowledged that this is due to "*people not running the tests and build on their machines before pushing the changes*", we also found cases of "*poor testing and reviewing of the commit beyond the 'immediate problem' the developer is attempting to rectify*". Similarly, we found 8 respondents that believe that build breakage can be caused by ***carelessness*** on the part of developers (*e.g.,* "*just be happy I'm commiting to the project, somebody else can test if what I did works*"). Confluent with ***carelessness***, there is ***overconfidence*** on the part of 11 respondents. Before breaking the build, they thought that "*the change is alright*", "*this is such a trivial change*", or "*this is only a small fix, it should not break anything*".

Still, we found six respondents that claimed that the ***lack of communication*** hindered the solution of a task, leading to the build breakage. Indeed, most of the respondents mentioned ***lack of communication*** skills (*e.g.,* "*disincentive to ask folks for help*" or "*not knowing who to ask for help*"). Yet, two respondents believe in ***moving fast and breaking things***, as one of them clarified "*I'll merge a commit that isn't quite ready as a clear signal of intent to move in that direction.*" and seven respondents are convinced that ***it is fine to break the build***, as one respondent exemplified: "*CI is there for you not to be afraid for broken builds in a branch*". Notwithstanding, one respondent warned that it is fine "*as long as it's not merged into trunk/master until it's green*". Finally, five respondents mentioned that they are not aware of any social reason.

IV. Benefits and Problems of CI Usage

A. The Benefits of CI Usage

As for the benefits, we found a variety of reasons. The most common benefit is to ***catch problems as early as possible***, reported by 32 respondents. It is important to note that problems can be described in terms of new bugs and regressions. One respondent summarized this benefit as "*Being aware of when/where breakage occurs greatly accelerates solution*". Moreover, ***automation*** was perceived as one of the main benefits, as pointed out by 19 respondents. Automation, however, is not only related to automated testing, as one respondent highlighted, but can be described in terms of "*automated code quality enforcement, automated release cycles, automated deployment*". Generally speaking, automation is aimed at "*performing a wide range of manual steps that a human would not normally be bothered to check*".

Another benefit is related to improving ***software quality***, as described by 18 respondents. Although most of the respondents have mentioned "software quality" with no additional details, some respondents have mentioned other quality attributes such as (1) increased stability, (2) quality of code and test, and (3) quality of documentation. One of the anonymous respondents has detailed how CI can improve software quality: "*When you use CI, you have a good health check in your code base, and from time to time you can keep looking if any other dependencies didn't break your package. It is a warrant of quality of your code*". Another interesting benefit is the ***fast development cycle***. According to one respondent, this happens because "*[its] extremely fast development cycle [made it] easy to try out stuff & easy to add new testing (you don't need to coordinate with other people as much, which is time-expensive)*".

Furthermore, we found that 16 respondents mentioned ***cross-platform testing*** as an effective method for "*compiling for multiple different targets (x86,arm,window,linux etc)*". Another respondent complemented that "*cross-platform testing acts as a sort of documentation*", and "*[it can be configured] with moderate ease and in a fraction of the time*". Some respondents also mentioned that "*this [task] is not feasible or cost-effective to do manually*". Still, 10 respondents mentioned that continuous integration gives more ***confidence*** to perform required code changes, as one respondent described: "*Depends on the coverage, but some sort of confidence that introduced changes don't break the current behavior*". The same respondent went further and complemented that "*I assume the most critical parts of the system have been covered by test cases*". It is important to note that test cases can only ensure that *known* bugs stay fixed. As Dijkstra claimed in the 1970s [9], and extensively evaluated in the following decades (*e.g.,* [10]), "*program testing can be used to show the presence of bugs, but never their absence*" [9].

B. The Problems of CI Usage

As regarding the hidden problems associated with continuous integration usage, we found that 31 respondents are having a hard time ***configuring the build environment***. Although some respondents agreed that the burden placed by the configuration process is often doable (*e.g.,* "*Some extra overhead and complexity for setting them up / maintaining the configuration. Not usually a big problem*"), we found cases of which the configuration process is far from trivial (*e.g.,* "*I found it hard to setup some distributed/multicomponent tests. This partially can be resolved by the containers.*"). It is important to observe that, according to a recent study [3], a significant proportion of open-source projects that use CI perform 5 or less changes to their CI configurations. Also, many changes were due to the version changes of dependencies. Therefore, this additional overhead might not place a constant burden on the software development team. Overcoming such technical problems is particularly challenging for ***onboarding newcomers*** (9 occurrences) since they "*do not understand what CI does and what it doesn't*".

Another recurring problem is the ***false sense of confidence*** (25 occurrences). As opposed to the ***confidence*** benefit, respondents described the ***false sense of confidence*** as a situation of which developers blindly trust in tests. One respondent synthesized this problem as:

> "*Over-reliance on a passing build result can encourage a reviewer to merge code without a thorough review. The continuous integration pass is only as meaningful as the test coverage.*"

This over-reliance on software testing, in general, and unit testing, in particular, as a measure of quality has been thoroughly discussed in the software testing literature [11]. For instance, tests can be insufficient, have poor quality, or even incorrect. Furthermore, some respondents have associated this false sense of confidence to insufficient testing (*e.g.,* "*If test coverage is not sufficient, passing all the time doesn't mean much*"). However, high coverage percentages alone cannot ensure code quality, since intermittent, non-deterministics or unknown bugs may not be detected by the periodic maintenance tests [12]. In fact, there is a complex relation between test-coverage and defect-coverage [11].

Moreover, we observed that the use of continuous integration techniques require developers a certain level of ***discipline*** (12 occurrences), as one respondent mentioned "*it takes a lot of self-organization to work under pressure of 'master must build'*". In addition, one respondent claimed that such ***discipline*** might introduce a lack of focus, since "*instead of only focusing on the code of your application, you also focus on the code of your build system. Basically, a software engineer becomes an infrastructure engineer, which is much less interesting*". Likewise, ***additional effort*** was also evidenced by 3 other respondents (*e.g.,* "*Need more people sharing / Documenting and helping resolve problems / Making sure builds pass*").

Interestingly, one respondent drew attention to the fact that the benefit ***multiple environments*** can also be a problem, for instance "*If build fails it may be hard to debug it, especially if the problem doesn't occur in the local environment*". Additional problems that were highlighted in our research are the ***monetary costs*** (*e.g.,* "*Hosted services like travis are not free if you want to use them at bigger scale or you need to maintain by yourself*") and ***flaky tests*** (*e.g.,* "*Tests can sometimes be flaky, which is frustrating*"). Finally, 10 respondents mentioned that they have never experienced any problems with CI.

V. Related Work

Vasilescu *et al.* [13] found although that 92% of the selected projects have configured to use Travis-CI, 45% of them have no associated builds recorded in the Travis database. They also found that direct contributions (pushed commits) are more frequent than pull-requests. In a follow up study, Vasilescu *et al.* [14] found that CI helped to increase the number of accepted pull requests from core developers, and to reduce the quantity of rejected from non-core developers, without affecting code quality. While CI can help a reviewer to make decisions regarding pull-requests faster, our study provide evidence that it is important not to over-rely on its results. To the best of our knowledge, the work of Hilton *et al.* [3] is the closest work to ours. By analyzing builds and performing a survey, the authors collected evidence showing that CI reduces the time between releases and that it is widely adopted in popular projects. However, the authors do not go deeper, for instance, in analyzing the perceptions about the problems and challenges of using this development practice.

VI. Conclusions

Continuous integration is gaining increasingly adoption among software developers. However, few is known about the perception of these users about the fundamental concepts related to CI systems and their usage, the reasons for build breakage, and the benefits and problems associated to it. In this study we performed an user survey with 158 CI users to shed the light on these questions. Through a qualitative research analysis, we produce a list of findings about the CI usage, some which are not always obvious, such as the (over) confidence in CI systems, a lack of testing culture, and the self-imposed time pressure.

Acknowledgements. We would like to thank the anonymous reviewers for helping to improve this paper. This research was partially funded by CNPq (304755/2014-1 and 406308/2016- 0), FACEPE (APQ-0839-1.03/14), FACEPE PRONEX (APQ 0388-1.03/14), and PROPPG/IFPA.

References

[1] K. Beck and C. Andres, *Extreme Programming Explained: Embrace Change (2Nd Edition)*. Addison-Wesley Professional, 2004.

[2] P. Duvall, S. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk*, ser. A Martin Fowler signature book. Addison-Wesley, 2007.

[3] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in *ASE*, 2016, pp. 426–437.

[4] H. Borges, A. Hora, and M. T. Valente, "Understanding the factors that impact the popularity of GitHub repositories," in *ICSME*, 2016.

[5] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. E. Emam, and J. Rosenberg, "Preliminary guidelines for empirical research in software engineering," *IEEE Trans. Softw. Eng.*, vol. 28, no. 8, pp. 721–734, Aug. 2002.

[6] E. Smith, R. Loftin, E. Murphy-Hill, C. Bird, and T. Zimmermann, "Improving developer participation rates in surveys," in *CHASE*, May 2013, pp. 89–92.

[7] I. Steinmacher, I. S. Wiese, T. Conte, M. A. Gerosa, and D. F. Redmiles, "The hard life of open source software project newcomers," in *CHASE*, 2014, pp. 72–78.

[8] G. Pinto, I. Steinmacher, and M. A. Gerosa, "More common than you think: An in-depth study of casual contributors," in *SANER*, 2016, pp. 112–123.

[9] E. W. Dijkstra, "Structured programming," O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, Eds., 1972, ch. Chapter I: Notes on Structured Programming, pp. 1–82.

[10] H. Zhu, P. A. V. Hall, and J. H. R. May, "Software unit test coverage and adequacy," *ACM Comput. Surv.*, vol. 29, no. 4, pp. 366–427, Dec. 1997.

[11] L. Briand and D. Pfahl, "Using simulation for assessing the real impact of test coverage on defect coverage," in *ISSRE*, 1999, pp. 148–.

[12] M. Abramovici and P. S. Parikh, "WARNING: 100% fault coverage may be misleading!!" in *Proceedings International Test Conference 1992*, Sep 1992, pp. 662–.

[13] B. Vasilescu, S. van Schuylenburg, J. Wulms, A. Serebrenik, and M. G. J. van den Brand, "Continuous integration in a social-coding world: Empirical evidence from github," in *ICSM*, 2014, pp. 401–405.

[14] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in github," in *ESEC/FSE 2015*, pp. 805–816.