Building Test Suites in Social Coding Sites by Leveraging Drive-By Commits

Raphael Pham, Leif Singer, and Kurt Schneider
Leibniz Universität Hannover
Software Engineering Group
Hannover, Germany
{raphael.pham, leif.singer, kurt.schneider}@inf.uni-hannover.de

***Abstract*—GitHub projects attract contributions from a community of users with varying coding and quality assurance skills. Developers on GitHub feel a need for automated tests and rely on test suites for regression testing and continuous integration. However, project owners report to often struggle with implementing an exhaustive test suite. Convincing contributors to provide automated test cases remains a challenge. The absence of an adequate test suite or using tests of low quality can degrade the quality of the software product.**

We present an approach for reducing the effort required by project owners for extending their test suites. We aim to utilize the phenomenon of *drive-by commits*: capable users quickly and easily solve problems in others' projects—even though they are not particularly involved in that project—and move on. By analyzing and directing the *drive-by commit* phenomenon, we hope to use crowdsourcing to improve projects' quality assurance efforts. Valuable test cases and maintenance tasks would be completed by capable users, giving core developers more resources to work on the more complicated issues.

I. Introduction

GitHub[1] changed the way developers collaborate on social coding sites. Software projects (source code, documentation, tests, etc.) are located in one place and GitHub provides communication means for project members, reducing coordination efforts. The collaboration process is streamlined: public projects can be cloned by interested developers (*"forked"*), improved, and then be offered back to the original project owner (by *"sending a pull request"*). In our analysis of GitHub [1], we found different attributes of user interaction on GitHub that pose difficulties for quality assurance in projects. Project owners try to cope with differing education in testing and a bigger contribution community by resorting to automated tests and maintaining an exhaustive test suite. Having a test suite in place enables regression testing and heightens confidence in the software product. This in turn encourages new contributions: contributors are now more likely to alter code, as newly introduced problems with old functionality would become apparent instantly. Also, contributors often rely on existing tests as educational examples or even as the basis for creating their own tests: some contributors start their own tests by copying and pasting existing tests. Encountering tests in the codebase or seeing a test suite in place even communicates a demand for tested contributions. However, communicating testing culture in this voluntary environment of a social coding site properly remains a challenge [1]. Project owners often receive untested contributions and merge them into the project's main branch for various reasons. This way, the project accumulates technical debt: As testing is an important quality assurance effort [2], the project owner will have to provide suitable test cases later in development. A constrasting approach would be test driven development [3].

GitHub exhibits several attributes that allow for crowdsourcing mechanisms [4] [5]. For example, GitHub's high exposure is already being exploited by some companies. Industrial projects have been reported to release a version of their project on GitHub and to rely on the open source community to find and fix most of the inherent bugs. The core developers of the project could concentrate on more pressing issues [1].

We observed another interesting phenomenon: GitHub facilitates so-called *drive-by commits* (DBC): small changes that do not require a prolonged engagement with a project, yet provide some value for it. Developers providing such changes would not always be actively interested in a project, but might have stumbled upon it when browsing GitHub. When they had found, for example, a spelling error or a missing translation, they would make a quick correction, submit their commit as a pull request to the project, and forget the project again. This may due to the low barriers of GitHub and a project's high exposure to a huge number of potential contributors.

Actively guiding a project's community to deliver DBCs could help these open source teams to improve their workflow: suitable tasks are distributed to the community while the core development team takes care of more complicated issues. Also, many companies use open source software for developing their own products [6] [7] [8]. Such companies would benefit from higher quality and productivity in open source software.

II. Research Goals

Our research goal can be divided into two subgoals:

1) to understand the mechanisms of DBC better, and
2) to employ this knowledge for improving quality assurance efforts in open source development teams.

This would allow us to distribute a project's testing workload to uninvolved bystanders, enabling a project's core members to focus on complex and critical issues.

[1] https://github.com

978-1-4673-3076-3/13 © 2013 IEEE

ICSE 2013, San Francisco, CA, USA
New Ideas and Emerging Results

To better understand the supporting and facilitating circumstances for DBC, we will investigate how and when DBCs happen. Such an understanding would allow us to derive a model of the DBC. This would clarify the following questions:

1) What are the *preconditions* for DBCs?
2) Which circumstances *support* or *prevent* DBCs?
3) How can the DBC mechanism be applied in the domain of testing?

We will investigate how the DBC mechanism can be employed in a controlled manner to solve varying problems. In this case, we try to solve quality assurance problems of open source development teams on social coding sites, improving the process of building a test suite.

This effort should help the development of open source projects in several ways. The project is provided with a steadily growing test suite and the quality of the project should rise. The existence of a test suite in turn could trigger or help following contributors to provide tests themselves [1]. Also, if the creation of less complex test cases is distributed to the community, the core development team is able to care about more complex issues and drive the development of the project forward.

III. Preliminary Results

This section describes preliminary results gained from further analysis of interview data of our previous work [1].

A. The Drive-By Commit

Often, the core development team of a popular project on GitHub is comprised of a small group of developers while the periphery of less deeply involved developers is rather large. This is due to several reasons: GitHub encourages users to connect to projects and "follow" their development. Users can also follow other developers whose work they find interesting or inspiring. GitHub generally encourages collaboration by employing easy to use and low-barrier mechanisms to get involved with a project: forking a project can be done with several clicks all inside the Web browser. Lastly, projects and developers on GitHub are searchable and browsable by different criteria. These mechanisms allow projects on GitHub to attract and gather a large group of developers, active users, and passive by-standers. With each new contributor and pull request, this community grows and project owners observe a constant flux in contributors. This high exposure combined with a project's large periphery and GitHub's low-barrier mechanisms spawned the new mode of contributing to a project by *drive-by commit*.

In our previous research [1], interviewees reported a certain spin-up time to get acquainted with a project's conventions in order to start working on it. This, however, was not the case with DBCs—and said users pointed this out as one of GitHub's most prominent advantage.

Users linked the ease of use, complexity of the change, and assurance of correctness to DBC. For example, one interviewee recounted how he added a new language to a GitHub project as a DBC. He did so because it was easy for him to provide this language and it only took him one night. Furthermore, this project had an internationalization framework in place and the user knew exactly where to apply his changes. This reassured him: *"I just can operate on the promises that a well-developed system will just work and not have to do a whole lot of work to make that contribution."*

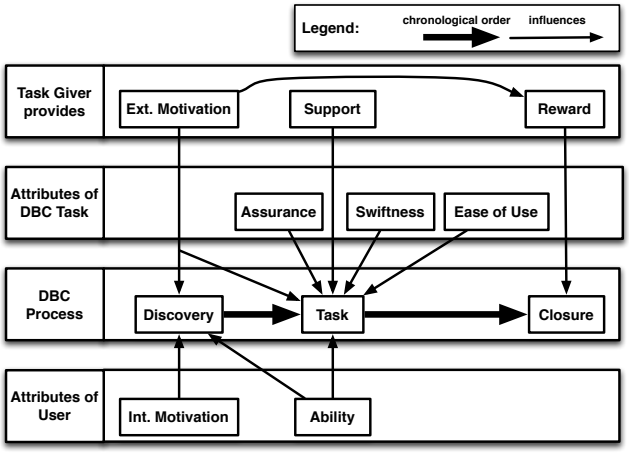

Fig. 1. The DBC Model

In our interviews [1] with active GitHub users about their testing behaviors, the topic of assurance of correctness—especially in the context of collaboration with others—came up repeatedly. Said user also said that not having to explicitly checking out the project, setting up an environment, and building it made things much easier. In this case, he could just "drop" his commit, felt assured that it worked, and moved on.

We validated that ease of use can facilitate DBCs. 496 active GitHub users rated the following statement on a five item Likert Scale [9]: *"Since it is so easy to send a pull request, I contribute more changes that I would not have engaged in otherwise."*. Value one represented most disagreement and value five most agreement. The median of this distribution was value 4 (154 answers), the modus (value 5) accumulated 209 answers.

B. A Model of the Drive-By Commit

In our interviews with active GitHub users [1], the interaction mode of DBC showed several similarities. Reoccurring themes of DBC in action were: little effort, swift, focussed, and generally low-barrier tasks. We extracted these similarities and formed a preliminary model of the DBC (see Fig. 1).

The DBC process that we encountered on GitHub involved the *Discovery* of an open *Task* and its straight-forward resolving. *Discovery* should either present potential contributors with open and suitable tasks, or provide means for them to discover such. Suitability of a task is dependent on a user's *Ability*. The step of discovery should take this into account.

Flooding a user with tasks that do not fit her ability might impede existing motivations to help. We therefore believe that the discovery process needs to support users in finding tasks that they are actually capable of solving.

Lastly, sending a pull request with a new commit to a project provides *Closure* in the DBC process. When designing interventions that aim to facilitate DBCs, this step should provide the user with a defined and clear end of the task.

C. Influencing Factors

We extracted different levels of **Influencing Factors** for the occurrence of DBCs in this process: attributes of the task itself and user attributes. Users seemed more at ease to provide DBC when they believed in the correctness of their contributions. Providing some form of *Assurance* during the tasks would support this need. This may be done by either giving the user the opportunity to check correctness herself or by providing automatic checks and opportunities for correction.

Swiftness describes several attributes of the task that can be useful when selecting tasks for a potential DBC community. Users described DBCs to be fairly quick and focussed activities. Any disturbance from the main task was perceived as counter-productive. This includes any form of prolonged pre- or post-processing and any effort connected with this:

1) The potential contributor will not set up any environment or build any software in order to develop the DBC.
2) Believing that their actions could cause a follow-up duties should cause most users to refrain from committing. Nonetheless, some degree of extra-work for a DBC can probably be elicited if it happens before the *Closure* event—such as providing a commit message. This remains to be investigated.
3) The task should not need any start-up time and design of the task should keep tight focus on the DBCs value.
4) The task should be clearly defined and the user should be informed what is expected of her.

Ease of Use refers mainly to a user interface that should be easy to learn and use.

On GitHub, users creating DBCs reported an *intrinsic motivation* to improve a project, even though they were not deeply involved with it. This was often coupled with a strong sense of capability: they knew that they could improve on a certain issue fairly easily and quickly.

We want to actively employ the DBC mechanism and introduce the role of the *Task Provider* to our model. Currently, DBCs may include an intrinsic motivation, such as the need to "scratch an itch" (to improve something that is bothering the user). Regarding testing, this need may not be equally developed. Therefore, we introduce *extrinsic motivators* that influence users to solve testing tasks. These could be used to trigger the user to search or discover a project that offers open tasks. This could be facilitated by offering a *Reward* that assigned at the closure step.

D. Introducing the DBC Model to GitHub

GitHub provides low-barrier mechanisms for cloning and merging source code repositories. Developers can start collaborating with a few clicks, all from within the browser. The popular GitHub project Travis-CI[2] tries to automate continuous integration for GitHub projects and eases the testing effort.

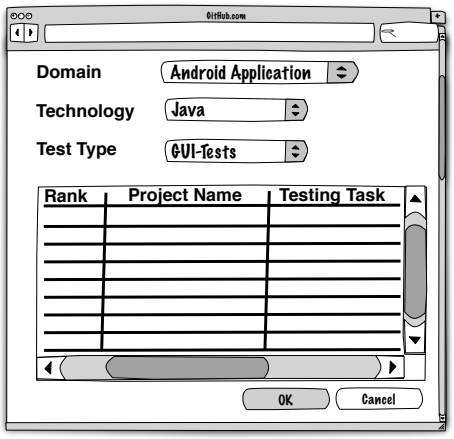

Fig. 2. The Search Site

However, even cloning a project locally might hinder the DBC mechanism. In order to enable DBCs on a larger scale, we propose to simplify the GitHub collaboration process even more. As we want the user to solely concentrate on creating a test case as a DBC, we plan to hide any step that we can automate from the user.

The overall idea is as follows: we introduce new means for the interested user to search for "testing tasks" (Discovery). When the user has found a suitable task, we deploy everything needed for providing a test case (the whole project, existing test suite, ...) in the background. The user is presented with a minimalistic interface for the sole purpose of providing the test case (Task). When everything is done, the user receives visual feedback and a reward (Closure).

IV. Implementation Sketches

The activity of *discovering* a new testing task for a DBC can be supported by a web site that allows developers to search for projects in need of testing efforts (see Fig. 2).

Here, the user can filter projects by technology, domain, or other attributes. This way, she can choose which kind of tasks she wants and is able to solve (**Ability**). All results are ranked by an *internal score*. Projects with a higher need for testing help will be rated with a higher score and are pushed up. This should give these projects a higher exposure and lead to more test cases being delivered by DBC. The need for testing help or the assessment of the state of the test suite is done by an internal metric that we are still developing.

The internal score and thus the rating of a project could be influenced by the project owner: if a project owner solves open testing tasks for other projects (in the role of a developer, driven by the DBC mechanism) she gains internal score points as a reward. She can spend these to push her own project up (**ext. Motivation**).

[2]https://travis-ci.org

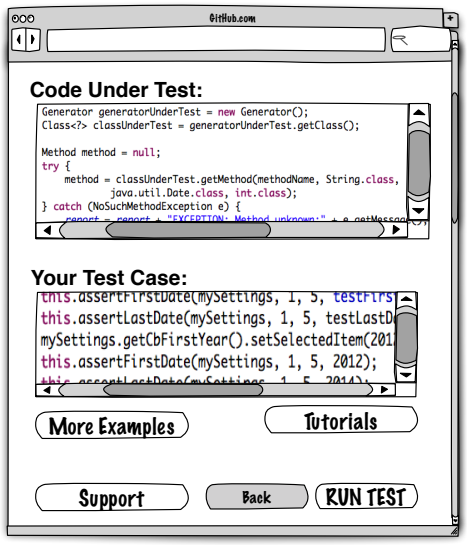

Fig. 3. The Simplistic Test Case Interface

A project owner testing other projects instead of putting that effort towards better tests for her own project seems counter-intuitive at first, but provides some benefits:

1) By engaging in testing of other projects, the project owner can activate a larger number of potential contributors. This could potentially lead to more test cases being added by the crowd than the number of test cases he would have been able to provide himself.
2) Test cases generated by DBC would be written by strangers—and could potentially provide much needed distance or lead to different test approaches.

When the user has a testing task she is interested in solving, she clicks on this entry and is presented with an in-browser view of the project that needs to be tested. The user can browse this project and a goal-oriented overview of the testing state of the project could be shown: a map or dashboard showing untested classes or functions or specifically marked code snippets. When the user has found a suitable place to add a test case, she can press a button "Add a new test case". If this task of browsing proves to be counter-productive, the user could be taken directly to the missing test case location.

When the "Add a new test case" button is pressed, the current project is deployed in the background: it is checked out into a virtual machine, continuous integration services like Travis-CI are run, and the user is presented with a simple web-based editor interface to enter her new test case (see Fig. 3).

This interface either features other test cases of this class or has one of them pre-entered in the text-editor. This way, the user gets to know the way test cases are handled in this project and can use it as a basis for his own test case—a practice we heard of in several interviews [1]. When the test case is complete, it is automatically run and the user is informed of the result. When the user accepts, she is informed of the testing score she has earned and the project's statistics overall (*Closure*). After that, the search site for open testing tasks could be displayed again to restart the process anew.

Project owners could provide a specific configuration for the virtual machine that is started when a user accepts to write a test case. This would allow for certain customizations. If project owners do not wish to create such a file manually, an application could derive a project's required configuration by analyzing it, generate a suitable configuration file for a virtual machine, and send it to project owner as a pull request.

V. Conclusions and Outlook

While researching GitHub [1], we found the *drive-by commit* phenomenon: users who are not deeply involved with a project stumble across small issues, resolve them in a quick manner, and move on. These users recognize the issue and are capable of resolving it fairly quickly. We assume this to be a specialized case of crowdsourcing. Right now, this behaviour is not guided or leveraged in a controlled manner, even though GitHub provides the prerequisites to do so.

In this paper, we presented a preliminary model for the DBC mechanism. We use this model to construct a new view of GitHub that systematically directs this phenomenon towards improving test suites for GitHub projects. To illustrate the viability of our approach, we provided concrete examples of how this new view could be implemented. However, some steps are not yet defined and are the subject of future work. Currently, our approach focusses on the creation of test cases — although other testing related tasks could also be considered.

Our approach can potentially enable the generation of test cases by an uninvolved crowd of software developers. This would leave the core development team free to solve more pressing issues while still holding up quality assurance.

Our next research steps include the definition of a suitable metric to define a project's need for test cases. We are also in the process of developing a feasibility prototype.

References

[1] 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 35th Intern. Conf. on Software Engineering (to appear)*, 2013. [Online]. Available: http://se.uni-hannover.de/pub/File/pdfpapers/Pham2012.pdf
[2] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*. Wiley, 2011.
[3] N. Nagappan, E. Maximilien, T. Bhat, and L. Williams, "Realizing quality improvement through test driven development: Results and experiences of four industrial teams," *Empirical Software Engineering*, vol. 13, pp. 289–302, 2008.
[4] A. Kittur, E. H. Chi, and B. Suh, "Crowdsourcing user studies with mechanical turk," in *Proceedings of the SIGCHI Conf. on Human Factors in Computing Systems*, ser. CHI '08, 2008, pp. 453–456.
[5] D. C. Brabham, "Crowdsourcing as a model for problem solving: An introduction and cases," *Convergence February 2008 vol. 14 no. 1 75-90*.
[6] C. Ebart, "Open source software in industry," *Software, IEEE*, vol. 25, no. 3, pp. 52 –53, may-june 2008.
[7] B. Lundell, B. Lings, and E. Lindqvist, "Open source in swedish companies: where are we?" *Information Systems Journal*, vol. 20, no. 6.
[8] D. Pranic and Z. Pozgaj, "Usage of open source software in public administration of republic of croatia," in *MIPRO, 2010 Proceedings of the 33rd Intern. Convention*, may 2010, pp. 1316 –1321.
[9] R. Likert, "A technique for the measurement of attitudes." *Archives of psychology*, 1932.