Working with Open Source Development Data

Considerations triggered by a study of bug scenarios

Matthijs den Besten
Oxford e-Research Centre &
e-Horizons Institute
University of Oxford
Oxford, UK
matthijs.denbesten@oerc.ox.ac.uk

Hela Masmoudi
University Pierre et Marie
Curie
Paris, France
masmoudi_hela@yahoo.fr

Jean-Michel Dalle
University Pierre et Marie
Curie
Paris, France
jean-michel.dalle@upmc.fr

ABSTRACT

The retrieval and preparation of public data on software development calls for more than just technical skills. In addition, care and judgement are needed to avoid disproportionate costs to the providers of data or unnecessary embarrassment to the participants tracked in the data. Taking the extraction of bug scenarios as a use case, we illustrate these concerns and discuss how they could be translated into social requirements that would help to make retrieval and preparation a sustainable exercise. In particular, we call for more efforts to establish institutional repositories of public data on software development and, besides, we suggest that reviewers could play a role in making sure that empirical research is performed in a way that does not bring the long-term relationship between software developers and researchers in jeopardy.

1. BUG SCENARIOS

The extraction of bug scenarios from bug repositories has a lot of potential as it helps shed a light on creative communities as problem-solving organizations. The quality of the output of such commmunties is, by definition, linked to their ability to solve problems and enhance the quality of the goods they produce. What are the processes that they then set up? How are they related to the emergent processes that also characterize these communities, and that for instance drive the attention of developers toward more complex and “interesting” pieces of code? Recent inquiries have shown that there exist superbugs [5]. Could appropriately improved processes help creative communities get rid of part of the superbug problem? But also, what is the influence of the tools that creative communities use and implement? Such ’bug trackers’ are now commonplace and, besides the well-known Bugzilla, a new generation is emerging. Could innovative solutions and functionalities be identified and implemented so as to foster bug resolution within the context of creative communities? These would be interesting questions to address.

Not only is research on bug management relevant to understand how open-source software communities develop code and solve associated problems, or to inform the design of well-adapted bug tracking systems, research on debugging also addresses the more general issue of online problem solving — a problem that is of an increasing relevance nowadays. However, only a limited number of contributions have addressed this issue and even fewer have focused on it. Early contributions notably include work by Mockus and Herbsleb [11] and by Crowston and colleagues [4, 3], for whom bug resolution was one among the relevant characteristics of open-source software projects. A few other investigations have followed, in a software engineering flavour, suggesting for instance how bug reports could be assigned automatically [1]. Also related are investigations under the auspices of the economics of information security [12] and discussions of information practices [14].

What we propose here is to extract bug scenarios from bug repositories, much like earlier work done by Ripoche [13]. The extraction has three stages. First, we want to develop a scenography and an automated detection of scenes and episodes (sequences of scenes). Second, we want to develop a database based on several bug repositories connected to different open-source projects and use the scenography of stage 1 to automatically classify bug resolution scenarios in the database. Furthermore, based on the measures developed in stage 3, we would like to suggest rules according to which inefficient scenarios could be detected and fixed. Third, we want to measure the efficiency of different classes of scenarios, using survival analysis and related econometrics techniques, in terms of how quickly and completely bugs get fixed. Scenes would be characterized by various elements such as their duration, and the types of contributors involved. The identification of the context in which a bug appears could be done by adopting a technique of “bug dressing” that has been developed elsewhere [2]. In addition, text analysis could provide interesting clues. For instance, ineffective tensions can be revealed by the use of specific paradigms such as discouragement (“I give up”), conflict (“This is stupid!”) and misunderstanding (“I don’t understand”), while other paradigms are cues for synergy, be it encouragement (“Impressive, John!”), thanks (“Thanks to John, I found it.”), or something else. Standard classification techniques could then be used to produce tentative classification of scenarios.

Eventually this kind of analysis could produce the specs of a module that could be inserted in all interested forges (collaborative development environments) and to help improve the bug resolution process by through the early identification of deviant scenarios.

2. CONCERNS

The retrieval and preparation of bug scenarios depends on active and passive collaboration of at least three groups of people. First of all, there are the software developers whose bug resolution practices are studied. For them, it is obviously important that the retrieval of data does not impose an unreasonable cost such as a big increase in bandwidth charges. But, in addition, developers may feel uneasy with the idea that their practices, and in particular, their individual behaviour are being showcased at conferences and in scientific journals. It would not seem unreasonable that researchers would go to some lengths in order to insure that the confidentiality of developers is kept. For, as Waskul puts it, "private interactions persist in spite of public accessibility" [16]. At the same time, however, researchers have to think about the research community as scholars should be able to check and replicate research. Finally, the research should produce results at a reasonable cost and in a reasonable amount of time, at least to keep research funders happy.

These general concerns come out to the fore when we consider a step-by-step procedure to retrieve and prepare bug scenarios for bug repositories. First of all, we have to find bug repositories and ideally, we would need to find the code repositories that are associated with the bug repositories so that we can check how the bug resolution affects the final code (I). Next, we need to retrieve bug reports for a selection or for all bugs in the repository (II). Then, we need to analyze these reports and publish our findings (III). Finally, we need to make sure that our bug scenarios remain available after the end of the research project (IV).

Step I, the retrieval of repositories, depends on the will of software development communities to make their data available or on the ability of researchers or their backers to archive raw development data before they disappear from the Internet. For code repositories, we are already experiencing a small drama as open source communities switch from CVS to other systems and fail to maintain access to their old repositories. Bug repositories like Bugzilla pose an additional difficulty to investigators as they are Web-based and generally do not provide straightforward means to get a hold of all their contents. Of course, one can use a technique known as "scraping" in order to download the relevant pages. However, a site like bugzilla.mozilla.org does not approve of this technique and has put a file "robots.txt" on its web-site which prohibits all access to crawlers to most of its site. This in itself is perfectly understandable since the automated downloading of large amounts of data that crawlers do tends to take up a lot of bandwidth and comes at a sometimes considerable cost for the provider of the site. Should we contact the web-site owners to get explicit permission for our scraping? Or could we get away with scraping provided it was sufficiently non-obtrusive and perhaps as slow as downloading by hand? Really downloading the bug-reports by hand would make the process too costly and would take too long.

The concerns associated with step II, the retrieval of bug reports from the repositories, are more about transparency and replicability. For instance, if we would only want to select the bugs reports that relate to bugs that result in changes in the code base, we would have to do an analysis of the comments that are associated with commits in the repository and extract the numbers of the bugs that are referenced there. Unfortunately, however, in most open source projects, there are no clear guidelines on how to reference bugs in commits and even if these guidelines are there, they are not always strictly adhered to. For instance, in mozilla, some prefix the bug number with "b=", others insert the word "bug" and yet others use a verb like "fixes". To identify bugs on the basis of comments like these, inevitably, one misses out on some of the bug references and inevitably, one will identify some numbers as references when in fact they are not. And so the data will be biased as only the bugs associated with certain code committers will be found. The concern here then, is how to account for this bias and to place the appropriate caveats whenever aggregate data are represented. To some extent this concern can be mitigated by given other researchers full access to the code that was used to extract the data, but then, these other researchers also have to understand the code.

The concerns associated with step III, the analyis of the bug resolution processes, are about preventing harm to the individuals we are studying. When people report a bug and participate in the resolution of that bug, they do not necessarily expect that someone will later look back at the interactions and analyze their behaviour. Especially when the bug resolution process goes nowhere, it would be highly unconfortable for the participants if they were put on the spot and if their behaviour was showcased at conferences and in journals. Yet, at the same time, it is important for researchers to be able to give concrete examples of the scenes that are identified. It is one thing to say a large crowd sometimes just indicates a low signal to noise ratio. It would be altogether better if one could point to a specific example, say Mozilla bug 8427, where there is a lot of commentary, but just a few people who propose solutions. Sometimes, one observes a parallel discussion. Sometimes one can identify a "clan" that works seperately on the resolution of the bug and ignores whatever anyone else has to say. For instance, in bug 94349 of Mozilla that seems to be the case. Should we try to contact Scott, Mitesh, Brian and Conrad, who form the clan in this bug, and obtain there permission before exposing them as such? And how about people who seem incapable of collaboration?

These concerns of step III are exacerbated once we turn to step IV and make our scenario database available to the public. From a research perspective, it would be good to not only store abstract scenes but also the actual episodes in the bug resolution processes from which they are derived. But even though we as researchers are only interested in the processes and mechanisms that lead to deviant scenarios, others may use the same data in order to identify the persons involved. And that, in turn may harm the reputation of those persons, possibly without justification. A concern of a different nature is how to make sure that the data will actually become available to the public and remain available. This concern becomes especially acute once project funding runs out and researchers move to different topics. Here, an institutional repository might provide a solution, but again someone has to maintain and pay for it.

3. GUIDELINES

The gamut of concerns described above can be addressed in a variety of ways — ideally in consultation with the developers, researchers, and funding bodies who have a stake. To help in this process, there are a number of guidelines that we can consult. A good starting point is the overview of the debate about the ethics of Internet research written by Eynon, Fry, and Schroeder [6]. Specific guidelines for crawling of web-sites are discussed by Thelwell and Stuart [15] and guidelines on the preparation are proposed by King [9], among others. These guidelines are, however, not guidelines that can be adopted without thinking. For instance, King is very thorough with respect to the removal of person-identifiers as he is concerned with data that are very sensitive such as exchanges within a depression self-help group. It would seem doubtful that a same level of meticulousness would be needed for less sensitive topics such as the resolution of bugs. At the same time, in a way, King is not rigorous enough as he does allow for quotations where the person-identifier is obscured. Yet, in many cases, this won't be sufficient as search engines will lead you straight back to the source on the basis of the quote alone and allow you to identify its originator [10]. For retrieval, the issues are not black and white either. For instance, Thelwell and Stuart themselves already note that the existing robots guidelines are a bit out-dated and recommend to "follow the robots guidelines, but re-evaluate the recommendations for crawling speed."

4. SUPPORT

Since guidelines in themselves are unlikely to become useful and general enough to apply to a large number of situations, it is important to develop other kinds of support for scholars who try to carry out empirical research.

There are several important ways in which life could be made easier for researchers trying to do empirical research on software development like the extraction of bug scenarios. Here we propose two. First, it would be greatly beneficial, if reviewers of paper submissions to conferences on software development could take into consideration concerns of cost and confidentiality of software developers and provide guidance and comments to the authors of these papers in trying to strike the right balance. Second, it would help if more alternative sources of information were available. Initiatives like FLOSSMole [8] are a good start and common needs have been described [7]. But, in addition, more support should be sought to safeguard original data on software development in institutionally backed repositories before they disappear from view and to provide a place to researchers to deposit their data and extraction and analysis code for others to inspect and reuse.

5. REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *Proceedings of the International Conference on Software Engineering*, 2006.

[2] J. Brézillion and P. Brézillion. Context modelling: Task model and model of practices. In *Proceedings of the 6th International and Interdisciplinary Conference on Modelling and Using Context*, Roskilde, Denmark, 2007.

[3] K. Crowston, J. Howison, and H. Annabi. Information systems success in free and open source software development. *Software Process: Improvement and Practice*, 2006.

[4] K. Crowston and B. Scozzi. Coordination practices for bug fixing within FLOSS development teams. In *Proceedings of the First International Workshop on Computer Supported Activity Coordination*, Porto, Portugal, April 2004.

[5] J.-M. Dalle and M. den Besten. Different bug fixing regimes? A preliminary case for superbugs. In *Proceedings of the Third International Conference on Open Source Systems*, Limerick, Ireland, June 2007. To appear.

[6] R. Eynon, J. Fry, and R. Schroeder. The ethics of internet research. In N. Fielding, R. Lee, and G. Black, editors, *Handbook of Online Research Methods*. Sage, Forthcoming.

[7] L. Gasser, G. Ripoche, and R. J. Sandusky. Research infrastructures for empirical science of F/OSS. In *Proceedings of ICSE Workshop on Mining Software Repositories*, 2004.

[8] J. Howison, M. Conklin, and K. Crowston. FLOSSmole: A collaborative repository for FLOSS research data and analyses. *International Journal of Information Technology and Web Engineering*, 1(3):17–26, 2007.

[9] S. A. King. Researching internet communities: Proposed ethical guidelines for the reporting of results. *The Information Society*, 12:119–127, 1996.

[10] H. McKee and J. E. Porter. The ethics of digital writing research: A rhetorical approach. *College of Composition and Communication*, in press.

[11] A. Mockus, R. T. Fielding, and J. D. Herbsleb. Two case studies of open source software development: Apache and mozilla. *ACM Transactions on Software Engineering and Methodology*, 11:309–346, 2002.

[12] A. Ozment and S. E. Schechter. Milk or wine: Does software security improve with age? In A. D. Keromytis, editor, *Proceedings of the 15th USENIX Security Symposium (USENIX06)*, pages 93–104, 2006.

[13] G. Ripoche and J.-P. Sansonnet. Experiences in automating the analysis of linguistic interactions for the study of distributed collectives. *Computer Supported Cooperative Work*, 15:149–183, 2006.

[14] R. J. Sandusky, L. Gasser, and G. Ripoche. Information practices as an object of DCP research. In *Distributed Collective Practices Workshop in CSCW*, 2004.

[15] M. Thelwall and D. Stuart. Web crawling ethics revisited: Cost, privacy, and denial of service. *Journal of the American Society for Information Science and Technology*, 57(13):1771–1779, 2006.

[16] D. Waskul and M. Douglass. Considering the electronic participant: Some polemical observations on the ethics of on-line research. *The Information Society*, 12:129–139, 1996.