An Initial Characterization of Bug-injecting Development Sessions

Irina Diana Coman
The Most Serene Republic
United Kingdom
irina.coman@gmail.com

Alberto Sillitti
Innopolis University
Russian Federation
a.sillitti@innopolis.ru

Giancarlo Succi
Innopolis University
Russian Federation
g.succi@innopolis.ru

ABSTRACT

Even experienced developers rigorously testing their code and using state of the art tools and practices, inject every now and then bugs into the code. There is a huge amount of literature about the characterization of such bugs including the effectiveness of the reports and the fixes, the time required to fix them, etc. Existing works have already identified several factors considered to influence directly the bug injection. However, there is no support for the claims made so far using data coming from industrial, bug-injecting development sessions. This paper aims at filling this gap by analyzing industrial bug-injecting development sessions from several points of view. It investigates 49 bug-injecting development sessions evaluating and discussing three alleged, developers-centered main causes of bug injection: expertise, knowledge of code, and distraction. Additionally, the paper provides insights into the complete lifetime of bugs from injection to the fix and discusses implications for bug prediction.

CCS CONCEPTS

• **Software and its engineering** → Software defect analysis; **General and reference** → Empirical studies

KEYWORDS

Bugs, empirical software engineering.

1 INTRODUCTION

The presence of bugs in the code has been among one of the major problems in software development since the very early era of computers. Despite that, more than 40 years of research and development in software engineering have produced very refined tools and processes, bugs are still an important issue. All developers introduce bugs that are not found until much later. Since bugs are the result of software engineers making mistakes while coding, the key question is what causes software engineers to make such mistakes [5] [6] [8].

We analyze the details of 49 industrial bug-injecting development sessions using the data coming from a non-invasive software metrics collection tool [1] [4]. We identify and track the complete lifetime of bugs and we investigate bug-injecting sessions focusing on three main factors [2] [3]: developer experience, knowledge of the code, and level of distraction.

We consider 3 types of data: bug information coming from the issue-tracking system, code information coming from the code repository, and product and process information from the metrics collection tool.

We focus our attention only on the bugs that were detected and fixed in the considered 7 months and that were introduced in the 31 months in which the team had the metrics collection tool installed. This means that the bugs that take longer to be detected or that are not fixed in this time-frame are not considered at all in this study.

ICSE'18 Companion, May 27-June 3, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5663-3/18/05.
https://doi.org/10.1145/3183440.3195069

2 RESULTS

Developer and day of the week. As other studies have found, some links between the author of a change and the day of the week [9], we investigate who are the authors of each bug and in what day of the week the bug was injected.

Regarding the day of the week of the fixes and injection, we first consider each commit as a single fix or bug-injection.

Knowledge of code. Lack of knowledge of the code is considered one of the bug-inducing factors. We are using two measures to evaluate the knowledge of the code. One is the time elapsed since the same developer changed the file for the last time. The second is the time that the developer spent working with that file (both reading and modifying it) computed as percentage of the total computer-interaction time of that developer. However, the second type of information is available only for the main developers of the team, which reduces the set of bugs considered to 27.

We compute over the previous month before the bug injection, how much time developers spend accessing and modifying the actual files in which they will inject bugs. The developers either did not work with them at all during the month before bug-injection or they interacted with them for less than 2% (that is less than 1 hour and a half) of their total working time.

Activities. Based on the active software application used by the developer, we can distinguish 3 main activities: software development (including coding, debugging, testing, building, code management, design), internet-browsing, and communication (e-mail and instant messaging). The activities are based on the ones identified in [4] [7].

Interruptions. We consider two levels of interruptions: coarse-grained and fine-grained.

Coarse-grained interruptions are basically any breaks in the development session. A break occurs when the developer stops interacting with his workstation (i.e., leaves the computer, answers the phone, talks to colleagues etc.).

Fine-grained interruptions are switches between two high-level activities. As communication (as e-mail or instant messaging) was the main type of activity that intruded on the work of developers, we considered that a fine-grained interruption occurs whenever a developer switches without any break from development to communication. A manual investigation of a random set of such communication events showed that the developer was not initiating the communication session himself, but rather he was the one receiving either an e-mail alert or an instant message. Out of the 27 development sessions considered, only 24 had any fine-grained interrupts.

Expertise. From the point of view of their level of expertise, the developers in our study can be classified in three groups: novices, experts, and occasional experts.

The three groups of developers do not seem however to map directly to three groups based on the number of bugs injected by each developer. Overall, most of the developers seem to introduce between 1 and 4 bugs in the time interval considered.

The novices are not really introducing obviously more bugs than the other developers. However, taking also into consideration the number of changes that they make, some of them seem to have a higher percentage of bug-injections in their changes. Still, this observation does not hold as well for the other novices.

Although they might not introduce more bugs, it seems that some of the novices are fixing more.

Overall, the number of bugs injected by each developer seem to indicate that there are differences between individual developers, but these differences might not be that obvious and are not strictly linked to the expertise.

Knowledge. We are considering mainly two distinct types of knowledge: knowledge of the code maintained, and programming knowledge needed to perform the task (such as API, algorithms, patterns, etc.). To evaluate the code knowledge, we consider the amount of time developers interacted with a piece of code and the time elapsed since their last change of a piece of code.

To evaluate the second type of knowledge, we can get a clue from the types of activities performed during each development session. The developers in this study were using the Internet mainly for looking up information that they required for development, such as accessing API docs, development examples, MSDN library, etc. Therefore, the amount of Internet usage is an indicator to some extent of how much additional programming knowledge the developers considered needed for the task at hand.

The bug-injecting development sessions are not uniform with respect to the percentage of time spent on the Internet which varies from less than 1% to 36%. Thus, it seems that potential knowledge issues that might lead to bug-injection are likely to be related to more complex knowledge than the type of specific, available knowledge that can be relatively easily obtained in a session of browsing the Internet.

Developers injected a large number of bugs in files that they never accessed before. However, another relatively big group of bugs are introduced in files with which the authors worked just the day before. This suggests that bugs are mostly injected either in code that developers simply do not know or in code with which they work quite a lot.

Distraction. We consider two different types of distraction: external to the environment and internal. For the external distraction, we are considering the proximity of the weekend as a disturbing factor. This was also considered in a previous study by Sliwerski *et al.* [9]. For the internal distraction, we are considering two types of interrupts: fine-grained and coarse-grained.

Considering the number of bugs introduced in each day of the week, the most error-prone days seem to be rather in the middle of the week (Wednesdays and Thursdays) than at the end or beginning. Fridays seem to be mainly the day for fixes, with Saturdays obviously being for extra work when needed. This is partially due to the team having sometimes to meet very strict business deadlines.

This result is quite contrasting with existing results from the literature that observed high probabilities of bug-injection on Fridays [9]. The external distraction factors might in fact be so variable from an environment to another that it is questionable whether their influence can be directly compared in different cases.

3 CONCLUSIONS AND FUTURE WORK

Although this data set is not suitable to perform a formal evaluation of the impact of each factor on bug-injection, we consider that such evaluation is a needed future work to be done on large, industrial data sets. The results from this study can also serve for informing or evaluating future hypotheses of factors that cause bug-injection.

REFERENCES

[1] A. Bykov, V. Ivanov, A. Rogers, A. Shunevich, A. Sillitti, G. Succi, A. Tormasov, J. Yi, A. Zabirov, D. Zaplatnikov. A New Architecture and Implementation Strategy for Non-Invasive Software Measurement Systems., In 33rd ACM Symposium on Applied Computing (SAC 2018), Pau, France, 9 - 13 April 2018.

[2] I. D. Coman, A. Sillitti. An Empirical Exploratory Study on Inferring Developers' Activities from Low-Level Data. In 19th International Conference on Software Engineering and Knowledge Engineering (SEKE 2007), Boston, MA, USA, 9 - 11 July 2007.

[3] I. D. Coman, A. Sillitti. Automated Identification of Tasks in Development Sessions. In 16th IEEE International Conference on Program Comprehension (ICPC 2008), Amsterdam, The Netherlands, 10 - 13 June 2008.

[4] I. D. Coman, A. Sillitti, G. Succi. A case-study on using an Automated In-process Software Engineering Measurement and Analysis system in an industrial environment. In 31st International conference on Software Engineering (ICSE 2009), Vancouver, BC, Canada, 16 - 24 May 2009

[5] I. D. Coman., P. N. Robillard, A. Sillitti, G. Succi. Cooperation, Collaboration and Pair-Programming: Field Studies on Back-up Behavior. In Journal of Systems and Software, Elsevier, Vol. 91, No. 5, pp. 124 - 134, May 2014.

[6] E. Di Bella, A. Sillitti, G. Succi. A multivariate classification of open source developers. In Information Sciences, Elsevier, Vol. 221, pp. 72 - 83, February 2013.

[7] S. Kim, T. Zimmermann, K. Pan, and E. James Jr. Whitehead. Automatic identification of bug-introducing changes. In ASE '06: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering, pages 81–90, Washington, DC, USA, 2006.

[8] T. Remencius, A. Sillitti, G. Succi. Assessment of software developed by a third-party: A case study and comparison. In Information Sciences, Elsevier, Vol. 328, pp. 237 - 249, January 2016.

[9] J. Sliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? Ín MSR '05: Proceedings of the 2005 international workshop on Mining software repositories, pages 1–5, New York, NY, USA, 2005. ACM.