Toward a Benchmark Repository for Software Maintenance Tool Evaluations with Humans

Matúš Sulír
Technical University of Košice
Košice, Slovakia
matus.sulir@tuke.sk

Abstract

To evaluate software maintenance techniques and tools in controlled experiments with human participants, researchers currently use projects and tasks selected on an ad-hoc basis. This can unrealistically favor their tool, and it makes the comparison of results difficult. We suggest a gradual creation of a benchmark repository with projects, tasks, and metadata relevant for human-based studies. In this paper, we discuss the requirements and challenges of such a repository, along with the steps which could lead to its construction.

CCS Concepts • **General and reference** → *Experimentation.*

Keywords benchmarking, programming, techniques, tools

ACM Reference Format:
Matúš Sulír. 2019. Toward a Benchmark Repository for Software Maintenance Tool Evaluations with Humans. In *Proceedings of the 2019 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH Companion '19), October 20–25, 2019, Athens, Greece.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3359061.3362776

1 Introduction

To evaluate a programming technique or tool, controlled experiments with human participants are often performed. The participants are divided into two groups, one using the tool/technique being evaluated and the other using the baseline. They are asked to perform the supplied maintenance tasks on a given project, such as fixing a bug or implementing a feature. Performance measures, e.g., the time to complete the tasks and their correctness, are collected and compared.

A crucial decision in this context is what project and tasks to select. Only 15% of studies use real tasks from issue trackers, and even in this case, they are sometimes modified to fit the purpose [3]. The rest of the studies use made-up tasks, which can favor the evaluated tool, be unrealistic and thus decrease external validity: *It would be surprising if the researcher could not come up with a single scenario where the new technique would prove itself somehow 'better' than an existing technique [2].* Furthermore, since everyone uses a different project and task for evaluation, it is difficult to compare techniques or perform data synthesis in systematic reviews. Although multiple software engineering artifact repositories exist [1, 6], they are specialized or designed mainly for automated studies and not focused on experiments with humans. For these reasons, we advocate the creation of a repository of benchmarks for software maintenance tool evaluations with human participants. It should contain a set of projects and tasks which the researchers could use to evaluate tools, along with the results of already performed experiments.

2 Projects

Currently, researchers trying to select a project for use in an experiment can sift through a long list at websites such as GitHub, trying to find a project fulfilling all general criteria: 1) it is an engineered software project – not, e.g., a tutorial, 2) it can be successfully and effortlessly compiled from source, 3) the problem domain is general enough to be understandable by the participants, 4) it is relatively self-contained, 5) it does not require extensive manual configuration and setup, 6) it has automated tests with sufficient coverage. The selected project must also fulfill experiment-specific criteria, such as: 1) it is written in the given programming language, 2) it is either a library, a web application, or a mobile application, etc., 3) the project either uses some specific technology/framework or does not use it, 4) it has a suitable size, depending on the kind of experiment being performed. Based on our experience, this searching process can last even a few days.

We envision a website with a sample of projects fulfilling all of the general criteria. For each such project, specific criteria are listed, so the researcher can select a project according to the needs. For example, we can search for a medium-sized non-Android Java library. Note that selecting such specific criteria lowers the chance that multiple researchers will use the same project. Therefore, the researcher can also enter a simpler query, such as "any Java project" – and the system will always return the same project. If we are not satisfied with the suggestion, the system can offer an alternative.

SPLASH Companion '19, October 20–25, 2019, Athens, Greece

This is the author's version of the work. It is posted here for your personal use. Not for redistribution. The definitive Version of Record was published in *Proceedings of the 2019 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH Companion '19), October 20–25, 2019, Athens, Greece*, https://doi.org/10.1145/3359061.3362776.

3 Tasks

There are two challenging criteria for a task: 1) It should be representative of the tasks performed in practice. Since industrial tasks tend to be confidential, our best match is tasks from the issue trackers of open source projects, first automatically filtered and then manually curated. 2) The task should exercise the behavior tested in an experiment. For example, a bug in a single thread is useless for an experiment focused on multi-threaded debugging, even if it may be relevant with respect to the tasks occurring in practice.

To help researchers fulfill the second criterion, the benchmark repository would contain not only task descriptions copied from issue trackers, but also sets of task properties. The researcher could then filter the tasks according to the properties relevant for a given experiment. Now a difficult question arises – what properties can a task have?

First, we can assign each task a course-grained category from a fixed set, such as a bug fix or feature addition. Then it is possible to divide the categories into more fine-grained ones, e.g., bugs into memory, concurrency bugs, etc. Care must be taken to find the right level of granularity, since too many categories would be impractical.

We can also ask: What activities should a developer perform in order to solve the task? By aggregating data from multiple developers working on the same task, we could obtain typical patterns of developers' behavior for the given task. What exactly is an activity, though?

It is possible to distinguish a fixed number of high-level activities, such as "understanding" or "writing" [4]. Such activities can be obtained manually from screen recordings and think-aloud protocols, or automatically from IDE (integrated development environment) interaction traces [7].

We can also include low-level actions recorded in the IDE and other windows during a task. Information about recorded events would be particularly useful when the tested tool focuses on certain actions in the user interface of an IDE. For example, if for a certain task, we found that the majority of developers spent at least 50% of time in a debugger, it is a suitable task for the evaluation of a debugger enhancement.

Many researchers (starting with [8]) studied what questions developers ask when programming. We could obtain a list of frequently asked questions for a particular task, based on the data from think-aloud protocols. Such data would help with the task selection, particularly if the evaluated tool is focused on answering a specific question.

At our envisioned benchmark website, the researcher will be able to use a specific query, such as "a bug-fixing task when the text search was often performed and questions about variable changes were asked" – or a general query, e.g. "any bug-fixing task". More general queries increase the chance of finding a suitable task.

4 Creation Steps

Our vision can be realized on multiple levels, each useful even on its own: 1) Structured demonstration: A group of researchers gathers to try their tools on the same project, comparing their results and experience. Such events already occurred [9], but they are rare. 2) Contest: Besides demonstrations, the participants also compete based on a set of criteria (see, e.g., DocGen – http://dysdoc.github.io). 3) Benchmark repository: In the information visualization community, such a repository was created using data from contests [5]. Existing data from already performed experiments could also be added, in case they are sufficiently complete.

To illustrate how the envisioned repository could look like in the future, a simple static demo is at http://sulir.github.com/humanbench. A researcher will select a project and tasks based on criteria. Then it would be possible to download a container with the project, its dependencies, an IDE, and tasks descriptions. If no project/task fulfills the criteria, it should be possible to add a new one. Finally, the researchers should upload the results of their experiments to the repository, so it would be possible to see and compare them.

Acknowledgments

This work was supported by Project VEGA No. 1/0762/19 Interactive pattern-driven language development.

References

[1] M. Böhme, E. O. Soremekun, S. Chattopadhyay, E. Ugherughe, and A. Zeller. 2017. Where is the Bug and How is It Fixed? An Experiment with Practitioners. In *ESEC/FSE '17*. ACM, 117–128. https://doi.org/10.1145/3106237.3106255

[2] S. Greenberg and B. Buxton. 2008. Usability Evaluation Considered Harmful (Some of the Time). In *CHI '08*. ACM, 111–120. https://doi.org/10.1145/1357054.1357074

[3] A. J. Ko, T. D. LaToza, and M. M. Burnett. 2015. A practical guide to controlled experiments of software engineering tools with human participants. *Emp. Softw. Eng.* 20, 1 (2015), 110–141. https://doi.org/10.1007/s10664-013-9279-3

[4] T. D. LaToza, G. Venolia, and R. DeLine. 2006. Maintaining Mental Models: A Study of Developer Work Habits. In *ICSE '06*. ACM, 492–501. https://doi.org/10.1145/1134285.1134355

[5] C. Plaisant, J.-D. Fekete, and G. Grinstein. 2008. Promoting Insight-Based Evaluation of Visualizations: From Contest to Benchmark Repository. *IEEE T. Vis. Comput. Gr.* 14, 1 (2008), 120–134. https://doi.org/10.1109/TVCG.2007.70412

[6] D. Rodriguez, I. Herraiz, and R. Harrison. 2012. On Software Engineering Repositories and Their Open Problems. In *RAISE '12*. IEEE, 52–56. https://doi.org/10.1109/RAISE.2012.6227971

[7] T. Roehm and W. Maalej. 2012. Automatically Detecting Developer Activities and Problems in Software Development Work. In *ICSE '12*. IEEE, 1261–1264. https://doi.org/10.1109/ICSE.2012.6227104

[8] J. Sillito, G. C. Murphy, and K. De Volder. 2008. Asking and Answering Questions During a Programming Change Task. *IEEE T. Softw. Eng.* 34, 4 (2008), 434–451. https://doi.org/10.1109/TSE.2008.26

[9] S. E. Sim and M.-A. D. Storey. 2000. A Structured Demonstration of Program Comprehension Tools. In *WCRE '00*. IEEE, 184–193. https://doi.org/10.1109/WCRE.2000.891465