Software Evolution and Quality Data from Controlled, Multiple, Industrial Case Studies

Aiko Yamashita†, S. Amirhossein Abtahizadeh*, Foutse Khomh†, and Yann-Gaël Guéhéneuc*
*Department of Computer and Software Engineering, Polytechnique de Montréal
†Centrum Wiskunde & Informatica, and Oslo and Akershus University College of Applied Sciences
aiko.fallas@gmail.com

I. INTRODUCTION

Many moderator factors can impact the outcome of a software engineering activity [1]. Experimenters must control them (or take them into account by measuring them) to avoid spurious relationships between the variables of interest. The moderator factors: programming skill and learning effect are important factors to consider because software development is intrinsically a human endeavor in which developers play a major role [2]. Equally, type of task have been shown to be an important moderator in program comprehension [3]. However, it is not possible to easily control for these factors in real-life settings because of their very nature: their values depend on the contexts of the systems, which are often beyond the experimenters’ controls. In this paper, we present a data set that has two particularities. First, it involves six professional developers and four real-life, industrial systems. Second, it was obtained from controlled, multiple case studies where the moderator variables: programming skill, maintenance task, and learning effect were controlled for. This data set is relevant to experimenters studying evolution and quality of real-life systems, in particular those interested in studying industrial systems and replicating empirical studies.

Index Terms—software quality, software evolution, software defects, software replicability, case study, empirical study, moderator factors, industrial data, replication, code smells.

II. BACKGROUND OF THE DATA SET

This data set is derived from a controlled, multiple, industrial case study conducted by Simula Research Laboratory in 2008, with the objective of investigating the effect of code smells on the evolution and quality of real-life, industrial software systems. Previous work describes in detail the context and the methodology of the study [5].

A. Systems

The four systems forming part of this data set, and involved in the study described in [5] originally come from a previous study conducted by Simula Research Laboratory in 2003 [6] to investigate software project replicability. For the study, Simula issued a tender for the development of a new web-based information system. Based on the bids, they hired four Norwegian software consultancy companies to develop independently a version of the system, using the same requirements specification. This resulted in four systems (hereon denoted as A, B, C and D) with near-identical features, but with dissimilar size, design, and implementation. The systems encompass a

<table>
<thead>
<tr>
<th>System</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java</td>
<td>8205</td>
<td>26679</td>
<td>4983</td>
<td>9960</td>
</tr>
<tr>
<td>JSP</td>
<td>2527</td>
<td>2018</td>
<td>4591</td>
<td>1572</td>
</tr>
<tr>
<td>Others</td>
<td>371</td>
<td>1183</td>
<td>1241</td>
<td>1018</td>
</tr>
<tr>
<td>Total</td>
<td>11103</td>
<td>29880</td>
<td>10815</td>
<td>12550</td>
</tr>
</tbody>
</table>

TABLE I LOC PER FILE TYPE FOR ALL FOUR SYSTEMS.
A. Data Description and Collection

The dataset comprises of four parts: 1) Software evolution history, 2) Software defects, 3) Code smells and change-related metrics (of both the initial and final versions of the artifacts), and 4) Task dates.

- **Software evolution history**: As mentioned previously, there were six developers involved in the study. Each was given access to one SVN repository containing the two systems they were assigned to. Consequently, there are six repositories in total. Each repository contains the evolution of two of the four systems, as depicted in Fig. 1. In total, there are 12 code evolution histories, each displaying a certain combination of the variables: system, developer and round. The repositories were initially in SVN, but later on were migrated to Git (see section III-B for further details).

- **Software defects**: In the same way as each developer was given access to an SVN repository, they were given access to one issue tracking system (Trac). Each Trac project was in addition, integrated with the corresponding SVN repository. Each issue tracking system registered the defects corresponding to the two systems each developer worked on. In total, there are 12 excel files of software defects, each set displaying a certain combination of the variables: system and developer, where the file format is: "Defects_Dev\{1/2/3/4/5/6\}Sys\{A/B/C/D\}.xlsx. Trac treats every issue as a “Ticket”, and each Ticket contains the following information: Ticket ID, Status, Resolution, Severity, Priority, Created, Modified, Summary, and Description. The description of the maintenance tasks were not included as tickets in Trac, but rather given as a specification document to the developers.

- **Code smells and change metrics**: The following code smells were detected in the four systems: Data Class, Data Clumps, Duplicated code in conditional branches, Feature Envy, God (Large) Class, God (Long) Method, Misplaced Interface, and Interface Segregation Principle (ISP) Violation. Note that these metrics are available only for Java files due to limitations in the tool. For each code smell instance identified, the following information was extracted: Smell name, smell description/details, Filename, and System. The code smells were extracted from the original (untouched) version before they were subjected to changes by the developers ("InitialSmells.xls"), and they were also measured on the 12 resulting versions after the developers finalized the maintenance tasks ("FinalSmells.xls"). Consequently, the last data contains in addition the following information: System, Developer, Round. For detecting the code smells, two commercial tools were used: Borland Together and InCode; they both use the detection strategies (metrics-based interpretations of code smells) by Marinescu [7]. As for code change metrics ("Changes.xls"), the following metrics were calculated for each commit revision: Programmer, Revision No., Date, Full path, Filename, File extension, System, Action Type (i.e.,...
A. How to access the data?

The evolution history is available at the following url: http://opendata.soccerlab.polymltl.ca/git/explore/projects. The researcher should then create an account in order to access, but it should be granted automatically. The defect data, alongside the secondary data, can be accessed via Zenodo, at the following url: https://zenodo.org/record/293719. The script for anonymising git is available at: https://goo.gl/B2XM9z.

B. How has the data been used?

The data presented in this paper has been used in the doctoral dissertation by Yamashita [5]. In addition, it has been used in the work by Yamashita & Counsell [9], where it was investigated if code smells can be used as system-level indicators of maintainability. Analyses reported in [10, 11] investigate whether code smells can be used as indicators of problematic artifacts, and to which extent can code smells uncover maintenance problems in general. Sjøberg et al., [12] and Soh et al., [13] also use this data set in conjunction to additional data (e.g., interaction traces) to quantify the effect of code smells on maintenance effort at different granularity levels. Finally, Yamashita & Moonen [14] also use this dataset to investigate the phenomenon of inter-smell relations in Object Oriented systems.

C. Potential usage scenarios

a) Analysis of “repeated defects” in a multiple case study: There is evidence that developers introduced similar defects while working on the same system, which hints that some defects are “meant to happen”. It could be interesting to examine these defects and investigate the properties of the system leading to the introduction of those defects.

b) Studies on the impact of different metrics/attributes on software evolution: This data has primarily been used to investigate the effect of code smells on software maintainability, but other metrics/attributes can be extracted from the systems, and validated across the different cases.

c) Further studies of inter-smells: Explanatory and predictive models built based on the notion of inter-smells –by using techniques such as association mining or clustering can be contrasted to traditional, file-based analyses.

d) Cost-benefit analysis of smell removal: The repositories can be mined for the refactorings performed, and can be contrasted to evaluate which refactorings actually “paid off” from quality and—or change size perspectives.

e) Benchmarking of tools/methodologies: The data set and the underlying systems can be used for benchmarking purposes, when evaluating new tools for metrics detection, defect extraction, or any other methodologies.

f) Task/context extraction: A possible improvement on the data set concerns the accuracy of the time when a given task was performed. Currently, the data only defines the date on which a developer was working on a given task/system (with the already mentioned “overlap” issue). It could be valuable to experiment with techniques/tools/methods that can allow identifying the exact context (e.g., task) at the time of each commit. Such techniques have industrial applications such as the one reported by [15].

1 https://about.gitlab.com/gitlab-com/  2 https://goo.gl/LXIKUE

3 In conjunction with other qualitative data collected during the same study.
D. Challenges and Limitations

a) Context of the study: The external validity of any results stemming from this data are contingent to the context of the study, in this case: medium-sized, Java-based, three-layered architecture, web-based, information systems.

b) Tasks were individual: The software professionals completed the project individually, i.e., not in teams or pair programming. This can affect the applicability of results obtained from this data in highly collaborative environments.

c) Time frame: The data does not fully represent a long-term maintenance project with large tasks, given the size of the tasks and the shorter maintenance period covered in the study. However, tasks resemble backlog items in a single sprint/iteration within the Agile context.

d) The age of the systems: The technology used in this study is already nearly 10 years (14 if the original study [16] is considered). However, there are still many industrial systems which are even older than 14 years, and the technology involved is still quite relevant to current software projects.

e) Tool availability: Unfortunately, the tools used for detecting the code smells are not available anymore, thus researchers would need to resort to alternative tools for extracting code smells, with possibly different results.

f) No explicit corrective tasks: The tasks considered explicitly in the study design are only of adaptive and perfective nature. Thus, the corrective tasks manifested in the study are not controlled for (i.e., they were generated from the defects originally existing in the systems—which were of diverse nature, and side-effects from developers’ changes).

g) Date accuracy for the task: As previously mentioned, the data specifies the dates for which a developer worked in a specific task/system. However, if the developer did several commits the same day, it is not always straightforward to determine to which task the commit corresponds to (in particular commits concerning the same system).

h) Quality of the defect reports and commit logs: Although developers were instructed to report the defects with as much information as possible, this was not always the case. Also, not all the commit logs were associated with an issue (Ticket) ID. This may require in some cases, mining techniques to link a commit to a bug fix.

i) Realism of the study: Is natural to believe that there will be a trade-off between the degree of realism and the degree of control in such type of studies (for a more detailed discussion on this issue, see [5]). We believe the systems and tasks belong to a realistic setting, and special care was put in order to ensure as much as possible, a realistic project.

V. CONCLUSION AND FUTURE WORK

We presented a data set that has two particularities: first, it involves six professional developers and four real-life, industrial systems. Second, it was obtained from controlled, multiple case studies designed to control to the factors: programming skill, task and learning effect. In future work, we plan to: 1) release more data from this study, 2) provide concrete guidelines for sharing diverse types of data from software engineering studies, and 3) present a proposal for a platform that could constitute a more intuitive approach for sharing research data.

REFERENCES


