Replication and Benchmarking in Software Analytics

Harald Gall

University of Zurich, Switzerland
http://seal.ifi.uzh.ch
@ BENEVOL, Mons, 17 Dec 2013
Panel

Empirical answers to fundamental software engineering problems

Wednesday, August 21
17:00-18:15, Column Hall

Chair
Bertrand Meyer (ETH Zurich, Switzerland, and ITMO, Russia)

Panelists
Harald Gall (University of Zurich, Switzerland)
Mark Harman (University College London, UK)
Giancarlo Succi (Free University of Bolzano-Bozen, Italy)

Abstract
For all the books on software engineering, and the articles, and the conferences, a remarkable number of fundamental questions, so fundamental that just about software project runs into them, remain open. At best we have folksy rules, some possibly true, others doubtful, and others – such as “adding people to a software project delays it further” – wrong to the point of absurdity. Researchers in software engineering should, as their duty to the community of practicing software practitioners, try to help provide credible answers to such essential everyday questions. The purpose of this panel discussion is to assess what answers are already known through empirical software engineering, and to define what should be done to get more.
The Screening Plant of a SW Miner
SQA Mashup Teaser
Roadmap for the talk

- Challenges of Software Mining Studies
- Mining Studies: Where are we now?
- Software Analytics: Replication and Benchmarking
- An Infrastructure for Software Analytics
I. Challenges of Software Mining Studies
Which data sources?

- Evolution analysis data repositories à la PROMISE
  - Flossmole, Sourcerer, Ultimate Debian DB
  - Provide benchmark (raw) data
- Interactive online web platforms that provide various analyses
  - Boa, FOSSology, Alitheia core, Ohloh
  - Analyses offered by design
  - Data produced is best used within the system
- Industrial project data (not widely accessible)
What kind of studies?

- **Source code**
  - Which entities co-evolve/co-change?
  - How to identify code smells or design disharmonies?

- **Bugs and changes**
  - Who should / how long will it take to fix this bug?
  - When do changes induce fixes?
  - Predicting bugs and their components?

- **Project and process**
  - Do code and comments co-evolve?
  - Who are the experts of a piece of code?
Example: Bug Prediction

Using Code Churn vs. Fine-Grained Changes

Predicting the Types of Code Changes

Using the Gini Coefficient for Bug Prediction

Predicting the Method

Using developer networks for Bug Prediction
Performance of bug prediction

- Learn a prediction model from historic data
- Predict defects for the same project
- Hundreds of prediction models / learners exist
- Models work fairly well with precision and recall of up to 80%.

<table>
<thead>
<tr>
<th>Predictor</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Release Bugs</td>
<td>73.80%</td>
<td>62.90%</td>
</tr>
<tr>
<td>Test Coverage</td>
<td>83.80%</td>
<td>54.40%</td>
</tr>
<tr>
<td>Dependencies</td>
<td>74.40%</td>
<td>69.90%</td>
</tr>
<tr>
<td>Code Complexity</td>
<td>79.30%</td>
<td>66.00%</td>
</tr>
<tr>
<td>Code Churn</td>
<td>78.60%</td>
<td>79.90%</td>
</tr>
<tr>
<td>Org. Structure</td>
<td>86.20%</td>
<td>84.00%</td>
</tr>
</tbody>
</table>

Example: Code Ownership

C. Bird, N. Nagappan, B. Murphy, H. Gall, P. Devanbu, Don't touch my code! Examining the effects of ownership on software quality, ESEC/FSE '11
Actionable Findings

- "Changes made by minor contributors should be reviewed with more scrutiny."
- "Potential minor contributors should communicate desired changes to developers experienced with the respective binary."
- "Components with low ownership should be given priority by QA."

C. Bird, N. Nagappan, B. Murphy, H. Gall, P Devanbu, Don't touch my code! Examining the effects of ownership on software quality, ESEC/FSE '11
Studies and Issues

- Bug predictions do work, cross-project predictions do not really work
- Data sets (systems) need to be “harmonized”

Open issues:

- Replicability of studies
- Benchmarks to be established
II. Software Mining Studies: Where are we now?
Nature of Studies

- **Replication**
  - Less than 20% can be replicated, from all the empirical studies published in MSR 2004-2009
    [G. Robles: Replicating MSR: A study of the potential replicability of papers published in the Mining Software Repositories proceedings. MSR 2010]

- **Data availability**
  - Raw data for OSS is easily available straight from publicly available sources
  - (Pre)Processed data is not yet widely available

Data preparation / tailoring to stakeholders
Performance/Time variance

- Bug prediction performance varies over time
- OpenOffice 2001–2008
- monthly slices
- conceptual drift!
- phases of stability!

III. Software Analytics: Where to go from here?
What is missing?

- Replication
- Large-scale comparative studies
- Preprocessing and Learners
- Calibration
- Benchmarking
- Line up of essential questions
- Adopting technologies from other fields
Replicability Evaluation

- Mining Studies of MSR 2004 - 2011
  - 84 (49%) experimental/empirical studies
  - 88 (51%) non-experimental studies (new methods, tools, case studies, visualizations, etc.)

- Studies classified into 6 categories and manually checked if they can be replicated with SOFAS:
  - Version History Mining, History Mining, Change Analysis, Social Networks and People, Defect Analysis, Bug Prediction
MSR Replication with SOFAS

The empirical study of every selected paper was then inspected in detail to assess if and how it could be replicated using SOFAS. As a result, the studies were then rated as fully supported, partially supported, and not supported. A study was considered fully supported only if the same results could be replicated or if all the necessary data could be calculated with the exception of its final aggregation or interpretation. On the other hand, a study was considered partially supported if its results could not be replicated out of the box, but the ground data from which they are derived could be calculated. At last, a study was deemed not supported if no or very little ground data could be calculated.

Replication is usually divided in two main categories: exact and conceptual. Exact replication is when the procedures of the experiment are followed as closely as possible. Conceptual replication is when the experimental procedure is not followed strictly, but the same research questions or hypotheses are evaluated, e.g., different tools, algorithms are used or some of the variables are changed. In this paper, we did not distinguish between exact and conceptual replication. A study was considered replicable whenever it could be replicated, either conceptually or exactly replicate, using a service or a combination of services currently available in SOFAS.

### Table 1

<table>
<thead>
<tr>
<th>Research category</th>
<th>Number of papers</th>
<th>Fully replicable papers</th>
<th>Partially replicable papers</th>
<th>Non replicable papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version History Mining</td>
<td>8 (9%)</td>
<td>4</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>History Mining</td>
<td>17 (20%)</td>
<td>0</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>Change Analysis</td>
<td>13 (15%)</td>
<td>5</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>Social Networks and People</td>
<td>19 (22%)</td>
<td>6</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>Defect Analysis</td>
<td>19 (22%)</td>
<td>8</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>Bug Prediction</td>
<td>8 (9%)</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>84 (100%)</td>
<td>25 (30%)</td>
<td>27 (32%)</td>
<td>32 (38%)</td>
</tr>
</tbody>
</table>

The results of such replicability assessment are reported in Table 1. The level of replicability is spread quite evenly across the different research categories and there is no category for which SOFAS was more successful than the rest. The only exceptional category is Historical Mining as no full replicability could be achieved for it. This is mostly likely due to the fact that it is a quite broad category encompassing very diverse studies each needing their own very specific analyses to calculate the final results needed. If such analyses are not present in SOFAS, the studies can not be replicated. Nevertheless, 0% of such studies are partially supported, this is because...
Replicability

- **Full replication**: 30% of the studies can be fully replicated out of the box
- **Partial replication**: 32% of the studies can be partially replicated
- As evaluation, we fully replicated “Do time of day and developer experience affect commit bugginess?” by J. Eyolfson, L. Tan and P. Lam, MSR 2011
The replication of the study

- We replicate the study to verify the 4 main findings.
- We extend the study by testing the findings for additional OSS projects:
  - Apache HTTP, Subversion, and VLC
- We analyze the results:
  - Do we achieve the same results?
  - Can the original conclusions also be drawn for the additionally investigated projects?
Analysis Workflow

Repository Info

GitVersion History Extractor
Extract the entire history of the provided project

Version History URL

Git-based Bugs-Fixes Linker
Extract all the relevant change couplings (exceeding the given threshold) for the given history

Query results
Find all the bug-introducing and bug fixing commits

Query results
Aggregate buggy commits by the day of the week

Query results
Aggregate buggy commits by the hour of the day

Query results
Extract the commit frequency and experience of the all the users who introduced bugs

Query results
Extract the commit frequency and experience of the all the users who introduced bugs

Manual aggregation
Aggregate buggy commits by developers experience and commit frequency

Manual aggregation
Extract the commit frequency and experience of the all the users who introduced bugs
Replication results

Percentage of buggy commits

- We confirmed the results of the original study with slight differences (different heuristic and date of analysis)
- The additional projects exhibit similar values (22-28%)

<table>
<thead>
<tr>
<th>Project</th>
<th># commits</th>
<th># bug-introducing commits</th>
<th># bug-fixing commits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Study</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux</td>
<td>268820</td>
<td>68010 (25%)</td>
<td>68450</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>38978</td>
<td>9354 (24%)</td>
<td>8410</td>
</tr>
<tr>
<td>Extended Study</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apache Http Server</td>
<td>30701</td>
<td>8596 (28%)</td>
<td>7802</td>
</tr>
<tr>
<td>Subversion</td>
<td>47724</td>
<td>12408 (26%)</td>
<td>10605</td>
</tr>
<tr>
<td>VLC</td>
<td>47355</td>
<td>10418 (22%)</td>
<td>10608</td>
</tr>
</tbody>
</table>
Influence of time of the day

- We confirmed the results of the original study
- The amount of buggy commits are particularly high between midnight and 4 AM and tends to then drop below average (morning and/or early afternoon)

- Windows of low bugginess greatly vary between projects
- Commit bugginess follows very different patterns
Replication results /3

- Influence of developer
  - We confirmed the results of the original study
  - A drop in commit bugginess is evident with the increasing amount of time a developer has spent on a project

- Influence of day of the week
  - We confirmed the results of the original study
  - Different weekly patterns in the additional projects
Interpretation of results

Feasibility

- We can replicate 30% of the analyzed studies and compute the ground data needed for another 32%.
- The studies we can replicate all use historical data extracted from different repositories.

Scalability

- The approach can scale up to very many of projects.
- Once the analysis workflow is defined, it can be automatically run with different project repositories.
- Still, limitation is total execution time (Apache HTTP ~ 8 hrs).
Interpretation of results

- Extensibility
  - We only focused on the replication of existing studies
  - The results and ground data produced by SOFAS analyses can be fed to other services, used by third-party analyses and tools or combined with data from other sources.
  - Do time of day, developer experience and file ownership affect commit bugginess?
    - e.g. taking into account code ownership measured using the Gini coefficient [Giger, 2011]
To get to the next level ...

- Support for replicability & systematic analysis workflows
- Calibration of data preprocessing
- Performance measures & performance criteria for studies
- Conclusion stability of studies
IV. Replicating Software Mining Studies with SOFAS
SOFtware Analysis Services

- SOFAS = RESTful service platform by G. Ghezzi
- using software evolution ontologies
- enabling the composition of analysis workflows
Current SOFAS services

- **Data Gatherers**
  - Version history extractor for CVS, SVN, GIT, and Mercurial
  - Issue tracking history for Bugzilla, Trac, SourceForge, Jira

- **Basic Services**
  - Meta-model extractors for Java and C# (FAMIX)
  - Change coupling, change type analyses
  - Issue-revision linker
  - Metrics service

- **Composite services**
  - Evolutionary hot-spots
  - Highly changing Code Clones
  - and many more ...
Analyse your code!

Enter the URI of a publicly accessible git or SVN repository, and we'll run some analyses on it. Please make sure that your repository contains at least two releases. Since many visualizations care about the evolution of the project, analysing a single release will not suffice.

Type: git
URL: 
Project Name: 
Submit
During version control extraction, your repository is cloned and prepared for the analysis. All tagged releases are downloaded so they can be parsed by the FAMIX service. *completed!*

The FAMIX meta-model extraction service is able to understand and extract the static structure of the source code, creating a logical model for other services to work with. *analysing...*

From the FAMIX model, the object-oriented metrics service and size & complexity metrics service calculate different metrics regarding the structure, layout and complexity of the code.

Using the metric data gathered in previous steps, the Code disharmonies service is able to detect many kinds of "code smells" of the software project.

Finally, the relevant information is collected from the different services, synthesized and prepared for visualization. Just a few more minutes, and the results will be ready.
V. Mashing Up Software Analytics
Data for Stakeholders
Multiple Stakeholder Mining

- We need to tailor information to the information needs of stakeholders, such as developers, testers, project managers, or quality analysts.
  - Study their needs beyond typical developer needs (‘questions developers ask’ by Sillito et al.).
  - Devise prototypes to elicit that information needs, for example, SQA-Mashup for Integrating Quality Data.
A Mashup of Software Project data
- commit & issue & build & test data
- all in mashups, integrated, easy to access
- however, filtered to the information needs of stakeholders

Most recent paper
- Martin Brandtner, Emanuel Giger, Harald Gall, Supporting Continuous Integration by Mashing-Up Software Quality Information, CSMR-WCRE 2014, Antwerp, Belgium

Available in Win 8 App Store
- http://goo.gl/ZUWrvm
A Developer’s view
A Tester's view
A project timeline

Change Distiller - Timeline
Mashup pipe configuration
VI. Conclusions
Merry Christmas!