SQUAAD: Software Quality Understanding by Analysis of Abundant Data

Sponsor: DASD(SE)

By
Mr. Pooyan Behnamghader
6th Annual SERC Doctoral Students Forum
November 7, 2018
FHI 360 CONFERENCE CENTER
1825 Connecticut Avenue NW
8th Floor
Washington, DC 20009

www.sercuarc.org
Outline

➢ Why Is Studying Software Quality Evolution at Commit-Level Important for Systems Engineering?
➢ A Scalable and Efficient Approach for Compiling and Analyzing Commit History
  ○ Why Is Compilability Important?
  ○ Aims
  ○ Method
  ○ Research Questions
  ○ Data Collection
  ○ Results
  ○ Discussion of Benefits and Risks
➢ SQUAAD
  ○ Distribution
  ○ Deployments
➢ Key Takeaways
Why Is Understanding Software Quality Evolution at Commit-Level Important?

Commit History Over a Period of 9 Years
Why Is Compilability Important?

➢ Uncompilable code
  ○ Symptom of careless development.
  ○ Static bytecode analysis, and dynamic analysis.
  ○ Uncompilability in post-development analysis due to compilation environment issues.

➢ Compilation over commit history is a major challenge
  ○ Alexandru et al. (2017) declare the unavailability of the compiled versions
    ■ main unresolved source for the manual effort in software evolution analysis.
  ○ Tufano et al. (2017) mine the commit-history of 100 Apache projects
    ■ 62% of all commits are currently not compilable.
Compilability and Commit-Level Mining Approaches

A Single Compile Error Breaks the Build for the Whole Software.

➢ Analyzing only impacted files
  ○ Reduce the cost and complexity.
  ○ Not suitable for compilation.

➢ Analyzing the whole software
  ○ Suitable for compilation.
  ○ Many un compilable commits.
Aims

We intend to demonstrate if analyzing changes in a module (instead of the whole software) results in
➢ Achieving a high compilation ratio, and
➢ A better understanding of software quality evolution.

Although the Whole Puzzle Is Incomplete Because of One Missing Piece, the Main Part(s) Are Complete and Understandable.
Method

➢ Targeting a software module and studying its evolution
  ○ To identify
    ■ Impactful commits that change that module.
    ■ Ancestry relationships between them: Impact-parent → Impact-child.
  ○ To fix
    ■ The compilation environment issues (e.g., missing dependencies) and to identify commits that are uncompilable because of a developer’s fault.
  ○ To scale
    ■ The analysis by distributing revisions over the cloud using SQUAAD.

➢ Empirical evaluation
  ○ To demonstrate the efficiency of our approach.
  ○ Four research questions.
Research Questions

➢ How efficient is our approach in reaching the maximum compilation and identifying uncompilable commits?

➢ How efficient is our approach in identifying change in quality metrics?
We Study 37838 Distinct Revisions Committed by 1998 Developers

<table>
<thead>
<tr>
<th>Org.</th>
<th>Timespan</th>
<th>Sys.</th>
<th>Developers</th>
<th></th>
<th>Commits</th>
<th>LOC (MS)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>All</td>
<td>Impactful</td>
<td>All</td>
<td>Impactful</td>
</tr>
<tr>
<td>Apache</td>
<td>01/02-02/18</td>
<td>38</td>
<td>1424</td>
<td>937</td>
<td>46952</td>
<td>22627</td>
</tr>
<tr>
<td>Google</td>
<td>08/08-01/18</td>
<td>18</td>
<td>1217</td>
<td>771</td>
<td>19249</td>
<td>11527</td>
</tr>
<tr>
<td>Netflix</td>
<td>05/11-01/18</td>
<td>12</td>
<td>641</td>
<td>290</td>
<td>11379</td>
<td>3684</td>
</tr>
<tr>
<td>Total</td>
<td>01/02-02/18</td>
<td>68</td>
<td>3282</td>
<td>1998</td>
<td>77580</td>
<td>37838</td>
</tr>
</tbody>
</table>

➢ System selection criteria

- Official systems of Apache, Netflix, Google on GitHub.
  - For Apache, less than 3000 commits
- Build tools:
  - Ant, Maven, Gradle
  - Not Android, Bazel, Eclipse
- No manual installation of other tools for compilation.
- More than 100 distinct revisions of the core module.
➢ **Average system compilability ratio**
  - 98.4% for Apache, 98.1% for Google, 93.9% for Netflix.

➢ **Commit compilability ratio**
  - 98.4% for Apache, 99.0% for Google, 94.3% for Netflix.
## Maximum Compilation Over Commit History Minimizes Missing Data

### Low Ratio of Missing Data

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td>1.55</td>
<td>16810</td>
<td>69.80</td>
<td>53.39</td>
<td>45.42</td>
<td>2.77</td>
<td>5.97</td>
</tr>
<tr>
<td>(A)ffiliated</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(E)xternal</td>
<td>1.58</td>
<td>4966</td>
<td>85.11</td>
<td>71.07</td>
<td>57.34</td>
<td>3.25</td>
<td>8.46</td>
</tr>
<tr>
<td>P</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.559</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Google</td>
<td>0.45</td>
<td>8734</td>
<td>77.59</td>
<td>59.18</td>
<td>44.74</td>
<td>0.28</td>
<td>2.74</td>
</tr>
<tr>
<td>A.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E.</td>
<td>3.09</td>
<td>2510</td>
<td>77.34</td>
<td>61.38</td>
<td>53.03</td>
<td>2.02</td>
<td>6.24</td>
</tr>
<tr>
<td>P</td>
<td>0</td>
<td>1</td>
<td>0.429</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Netflix</td>
<td>7.52</td>
<td>1610</td>
<td>78.45</td>
<td>62.05</td>
<td>54.60</td>
<td>1.30</td>
<td>8.82</td>
</tr>
<tr>
<td>A.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E.</td>
<td>4.42</td>
<td>1511</td>
<td>77.22</td>
<td>60.73</td>
<td>49.80</td>
<td>1.32</td>
<td>8.87</td>
</tr>
<tr>
<td>P</td>
<td>0.001</td>
<td></td>
<td>0.963</td>
<td>0.974</td>
<td>0.079</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Benefits of Focusing on a Module

➢ Provides a more **complete** view of evolution for some commit-level mining tasks.

➢ Facilitates **manual inspection** of individual data points which is a labor intensive task.

➢ Reduces the **cost** and **complexity** of the mining task.

➢ Provides a more **accurate** analysis by omitting irrelevant changes.
Risk: We Skip Important Details in Other Modules

➢ We can select a combination of modules as the target to include all important details.

➢ The target can be
  ○ As small as a plug-in.
  ○ As large as a complete software system.
We Distribute the Analysis Over the Cloud or an On-Premise Server

Execution Host for Static/Dynamic Program Analysis

1. Software Repository
2. Software Revision
3. Software Orchestration Server
4. Software Evolution Data
5. Software Quality Evolution Database

Private/Public Cloud
On-Premise Server

Visualization Server (SQUAAD Web)
Research papers


Organizations

- “A recent use of the tools was to quickly analyze some NASA mission problem identification software for use in a Navy mission problem avoidance training class.” – SERC Report 2017
- Analysis of open-source systems for Huawei Technologies Co.
- MITRE and SEI are interested.
Key Takeaways

➢ Focusing on a module (instead of only impacted files or the whole software) to mine commit history
  ○ Achieves
    ■ High compilation ratio
    ■ More complete and accurate
    ■ Less complex and costly
    ■ Less manual effort
  ○ Enables
    ■ Uncompilability analysis
    ■ Byte-code static analysis (e.g., architecture recovery)
    ■ Dynamic analysis (e.g., test coverage)

➢ Our cloud-based distributed solution to analyze software evolution (SQUAAD)
  ○ Enables
    ■ In depth analysis of the commit history to study the impact of every change.
    ■ Analysis a large body of software, and from a variety of domains.