A Scalable and Efficient Approach for Compiling Commit History

Pooyan Behnamghader and Barry Boehm
USC Center for Systems and Software Engineering

Research Task / Overview

Compilation over commit history:
- Uncompilable code is a symptom of careless development.
- Some static, and all dynamic program analysis techniques depend on byte-code availability.

Two approaches for analyzing commit history:
- Measure quality metrics only in every commit’s impacted files.
- Compile and analyze the whole software after every commit.

Focusing on impacted files is not suitable for compilation, and compiling the whole software after every commit results in a low compilation ratio.

How can we achieve a high compilation ratio?

Data & Analysis

We Achieve High Compilation Ratios.

<table>
<thead>
<tr>
<th>Frequency (%)</th>
<th>Apache</th>
<th>Google</th>
<th>Netflix</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 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.

Analysis of uncompilability:
- We identify 303 sequences of uncompilable commits and study their characteristics (i.e., length, and duration, and number of developers).
- We create a model to predict uncompilability based on commit metadata (i.e., time, message, and author) with an F1-score of 0.89 and an AUC of 0.96.

Goals & Objectives

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.

Methodology

Approach:
- We focus on an evolving module (target).
- We compile and analyze only the distinct revisions of the target and omit other modules to prevent their errors from breaking the build.
- We reach the maximum compilation over commit history for the target module and identify all commits that are uncompilable as a result of a developer’s fault during development.

Algorithms:
- Identifying distinct revisions of the target and ancestry relationships between them.
- Distributing the analysis over the cloud.
- Reaching the maximum compilation and identifying the reason(s) for uncompilability.

Evaluation:
- We conduct a large-scale empirical study on 37838 distinct revisions of the core module of 68 systems across Apache, Google, and Netflix to assess their compilability.

Future Research

Bytecode analysis over commit history:
- Architecture evolution.
- Code coverage evolution.
- Uncompilability over commit history:
  - Taxonomy on why developers commit broken code.

Contacts/References

Pooyan Behnamghader: pbehnag@usc.edu, Barry Boehm: boehm@usc.edu