A Scalable and Efficient Approach for Compiling Commit History

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

SYSTEMS ENGINEERING RESEARCH CENTER

USC

Research Task / Overview

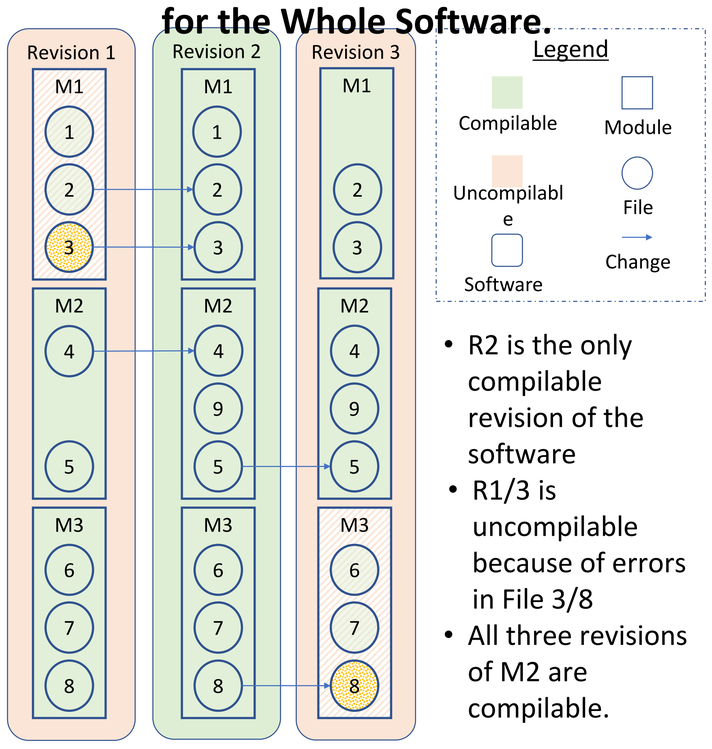
Compilation over commit history:

- 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.

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



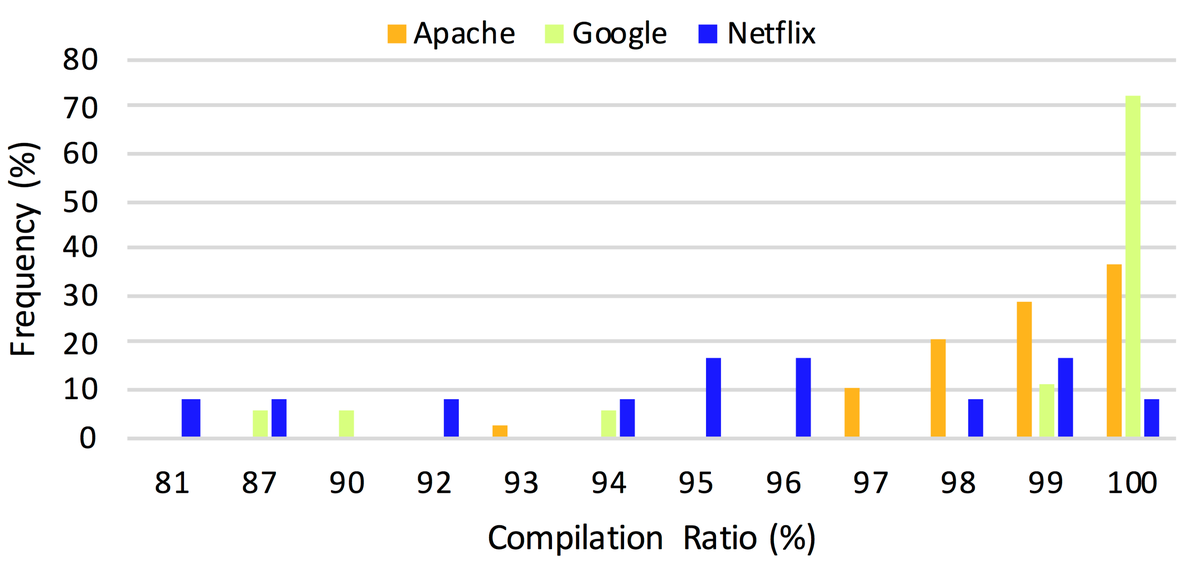
- R2 is the only compilable revision of the software
- R1/3 is uncompilable because of errors in File 3/8
- All three revisions of M2 are compilable.

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.



- 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.

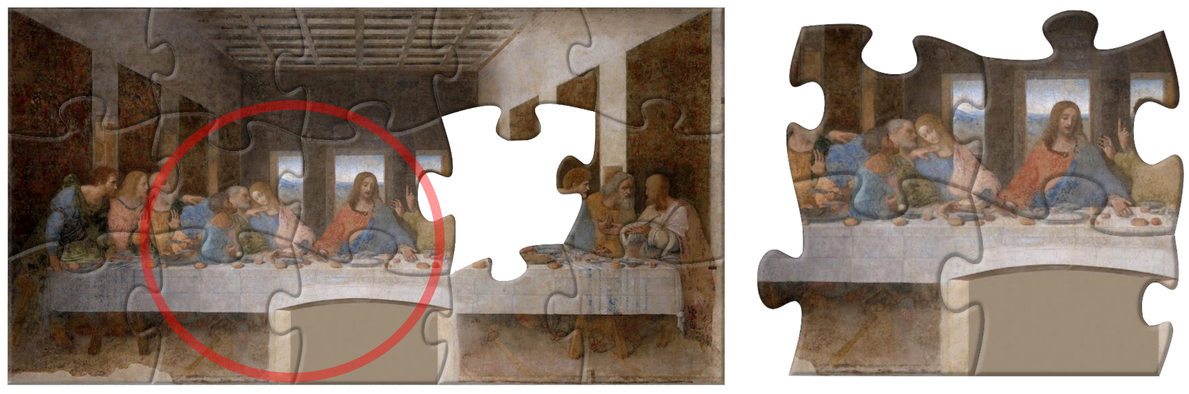
Analysis of uncompilability:

- We identify 303 sequences of uncompiled 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: pbehnamg@usc.edu, **Barry Boehm**: boehm@usc.edu

Pooyan Behnamghader, Patavee Meemeng, Iordanis Fostiropoulos, Di Huang, Kamonphop Srisopha, and Barry Boehm. 2018. **A scalable and efficient approach for compiling and analyzing commit history.** In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '18). ACM, New York, NY, USA, Article 27, 10 pages.

SERC Doctoral Students Forum and Sponsor Research Review, November 7 & 8, 2018