SQUAAD: Software Quality Understanding by Analysis of Abundant Data

Sponsor: DASD(SE)

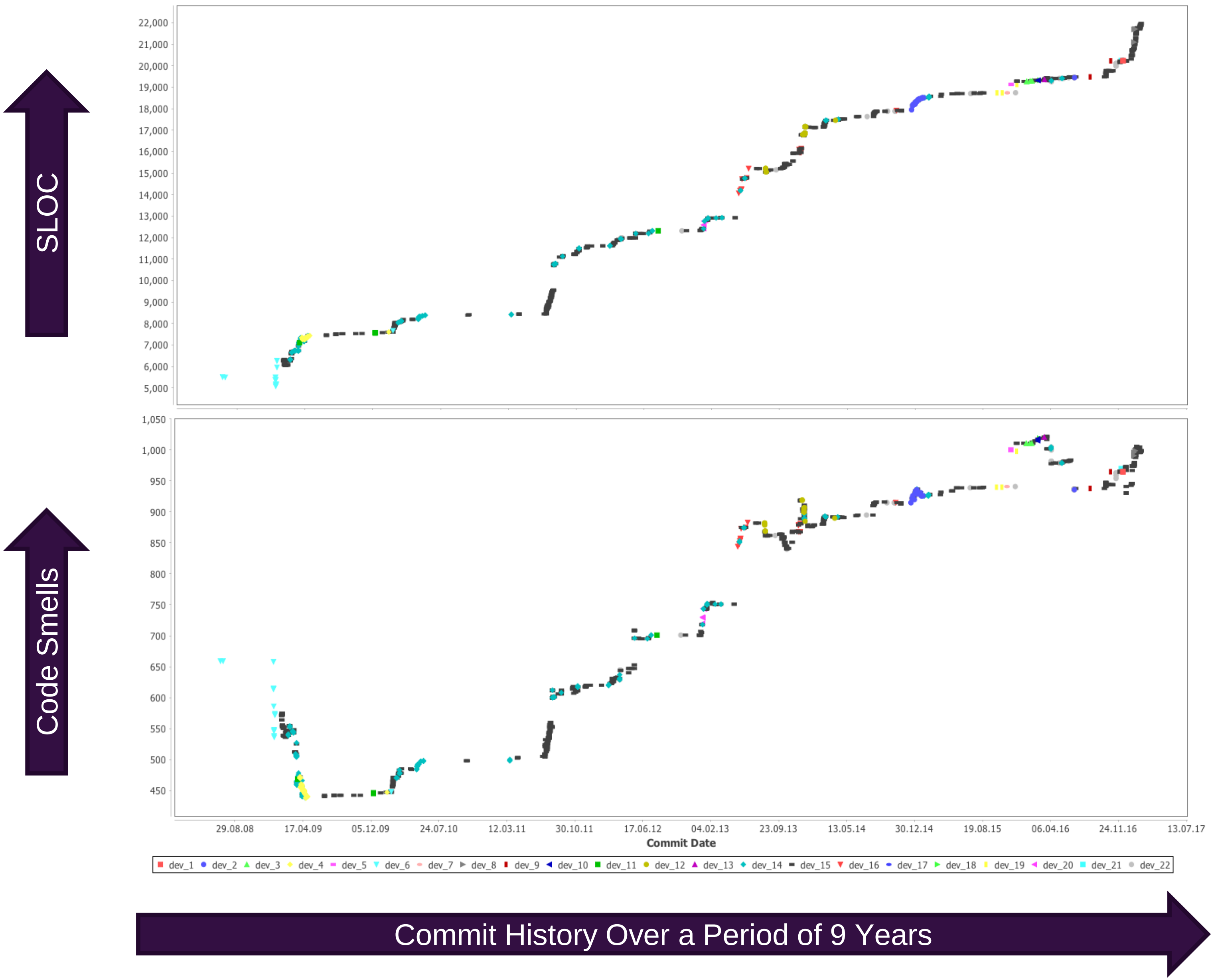
By
Mr. Pooyan Behnamghader
6th Annual SERC Doctoral Students Forum
November 7, 2018
FHI 360 CONFERENCE CENTER
1825 Connecticut Avenue NW
8th Floor
Washington, DC 20009

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?

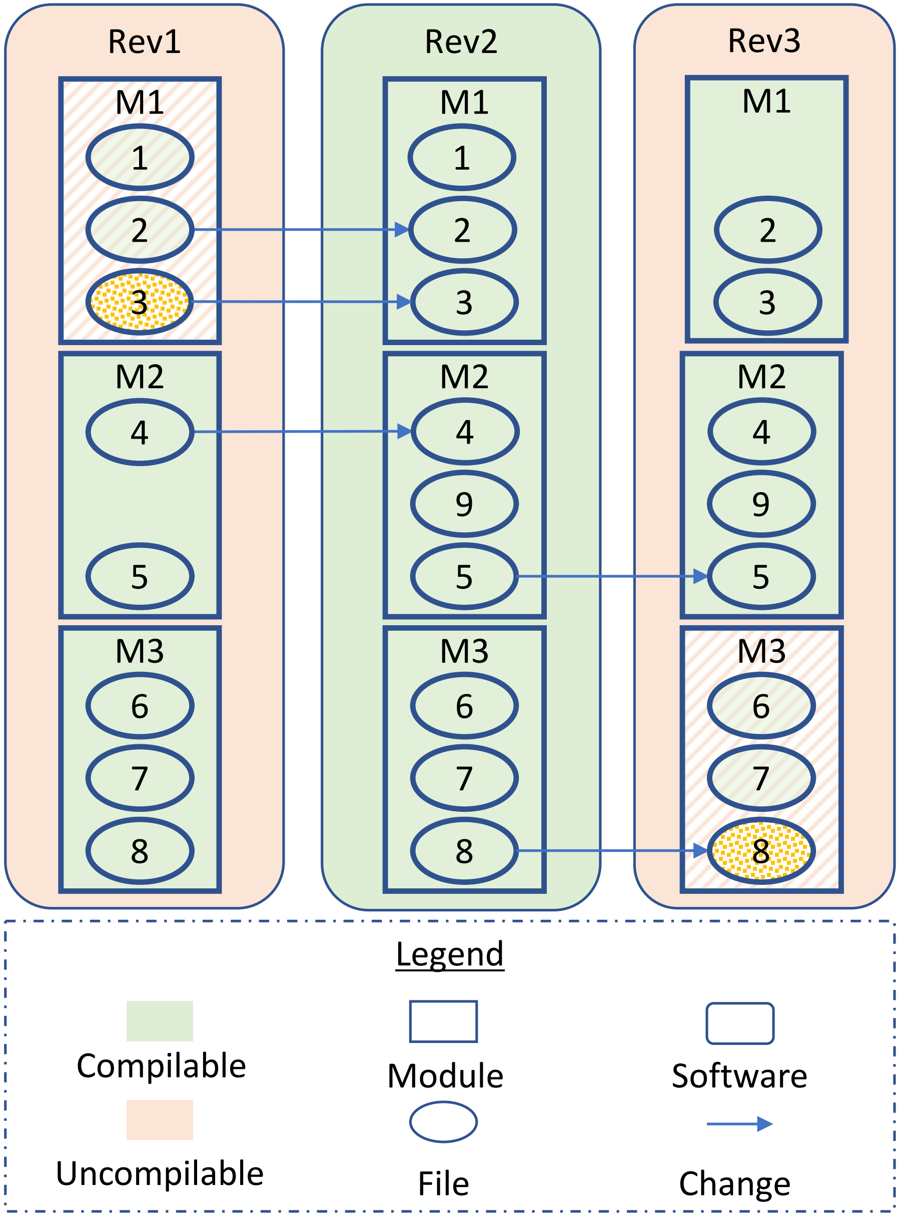


Why Is Compilability Important?

- 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.



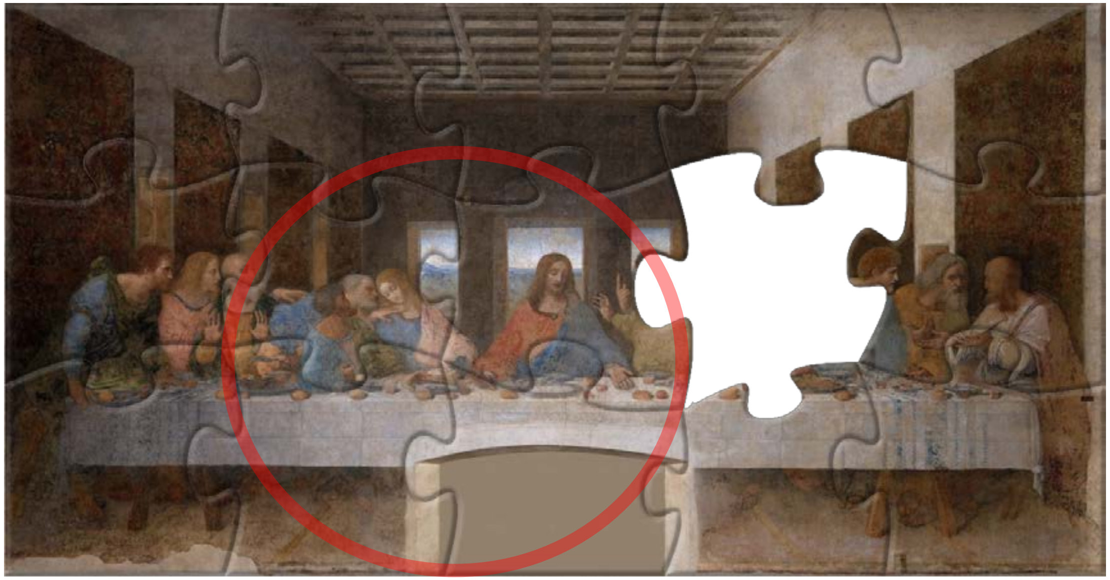
- ➢ Analyzing only impacted files
  - Reduce the cost and complexity.
  - Not suitable for compilation.
- ➢ Analyzing the whole software
  - Suitable for compilation.
  - Many uncompilable commits.

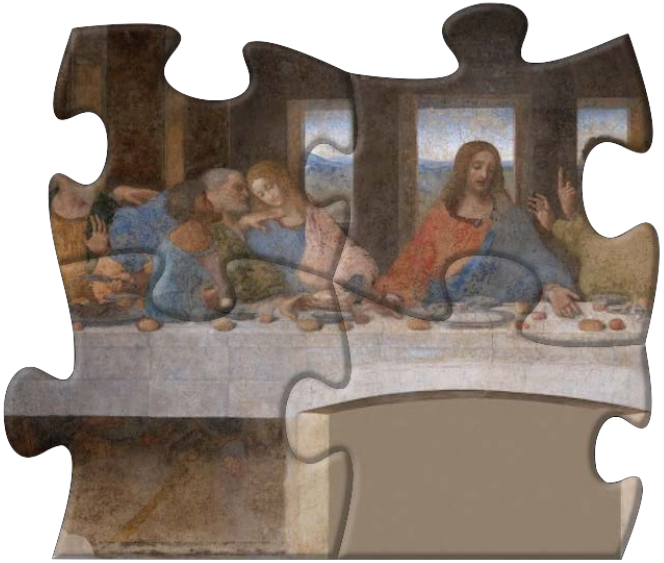
Aims

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?

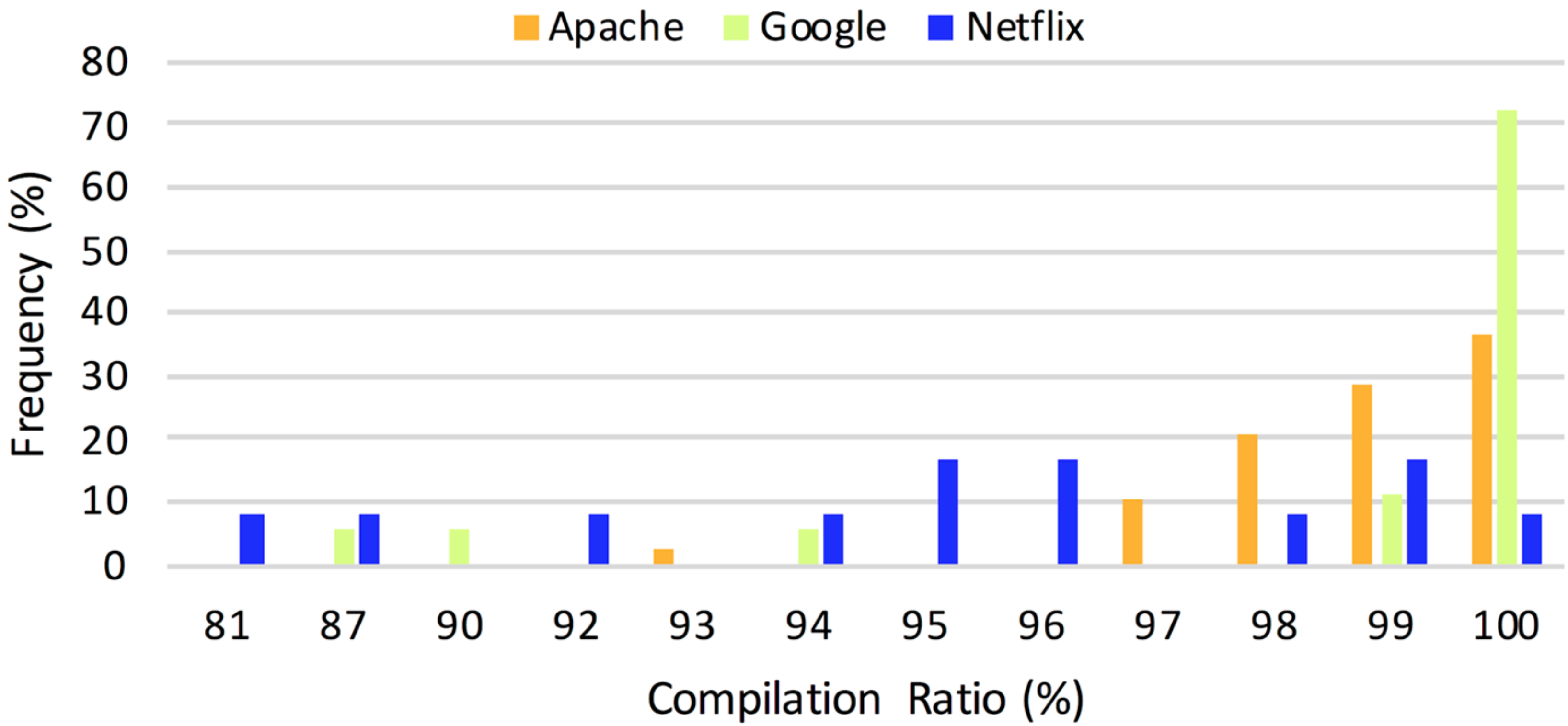
- How efficient is our approach in identifying change in quality metrics?

We Study 37838 Distinct Revisions Committed by 1998 Developers

Org.	Timespan	Sys.	Developers		Commits		LOC (MS)
			All	Impactful	All	Impactful	
Apache	01/02-02/18	38	1424	937	46952	22627	734
Google	08/08-01/18	18	1217	771	19249	11527	760
Netflix	05/11-01/18	12	641	290	11379	3684	37
Total	01/02-02/18	68	3282	1998	77580	37838	1531

- 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.

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.

Maximum Compilation Over Commit History Minimizes Missing Data

Low Ratio of Missing Data

Large Amount of Collected Data

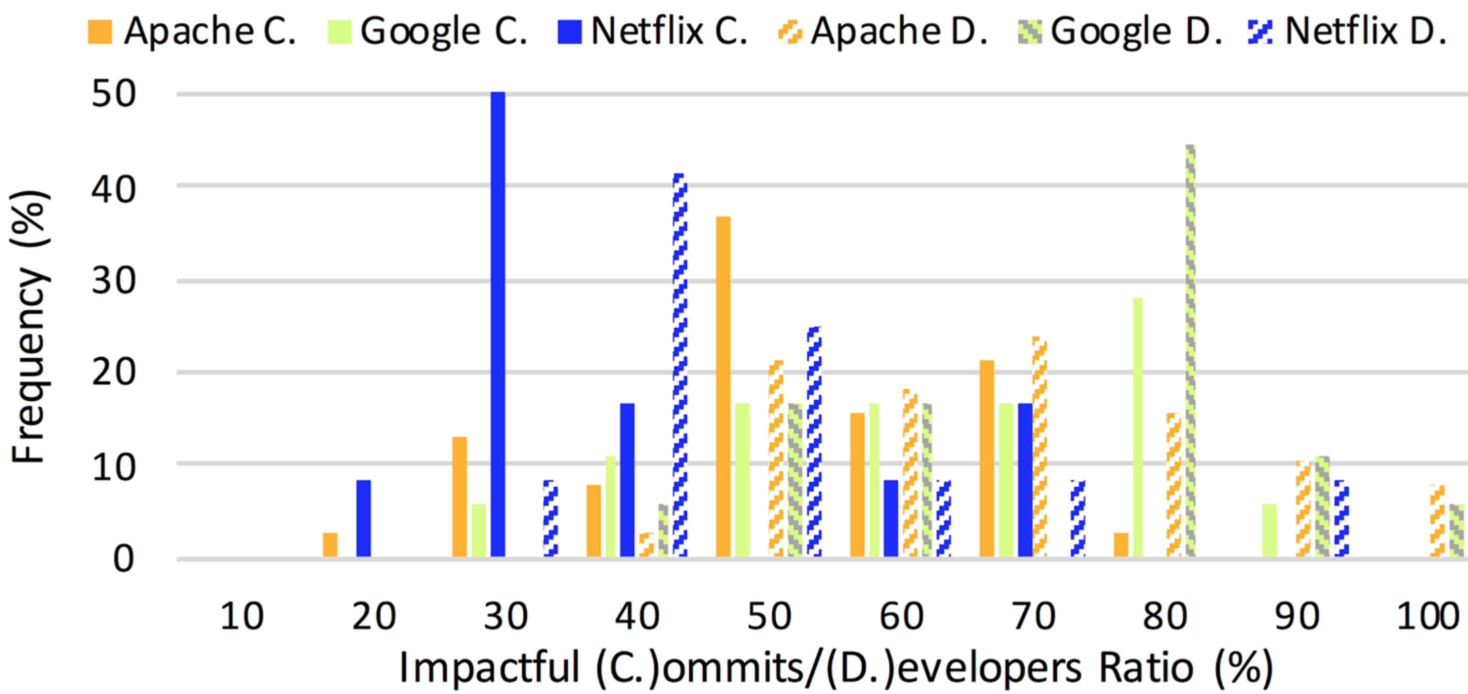
Statistical Significant Difference Between Two Groups of Commits/Developers; Even for Quality Metrics With Low Change Frequency Ratios.

Affiliation		Missing (%)	Commits (#)	Quality Metric Change Ratio (%): Lines of Code	Code Complexity	Code Smells	Malicious Code	Vulnerabilities
Apache	(A)ffiliated	1.55	16810	69.80	53.39	45.42	2.77	5.97
	(E)xternal	1.58	4966	85.11	71.07	57.34	3.25	8.46
	P	1		**0**	**0**	**0**	0.559	**0**
Google	A.	0.45	8734	77.59	59.18	44.74	0.28	2.74
	E.	3.09	2510	77.34	61.38	53.03	2.02	6.24
	P	**0**		1	0.429	**0**	**0**	**0**
Netflix	A.	7.52	1610	78.45	62.05	54.60	1.30	8.82
	E.	4.42	1511	77.22	60.73	49.80	1.32	8.87
	P	**0.001**		0.963	0.974	0.079	1	1

Benefits of Focusing on a Module

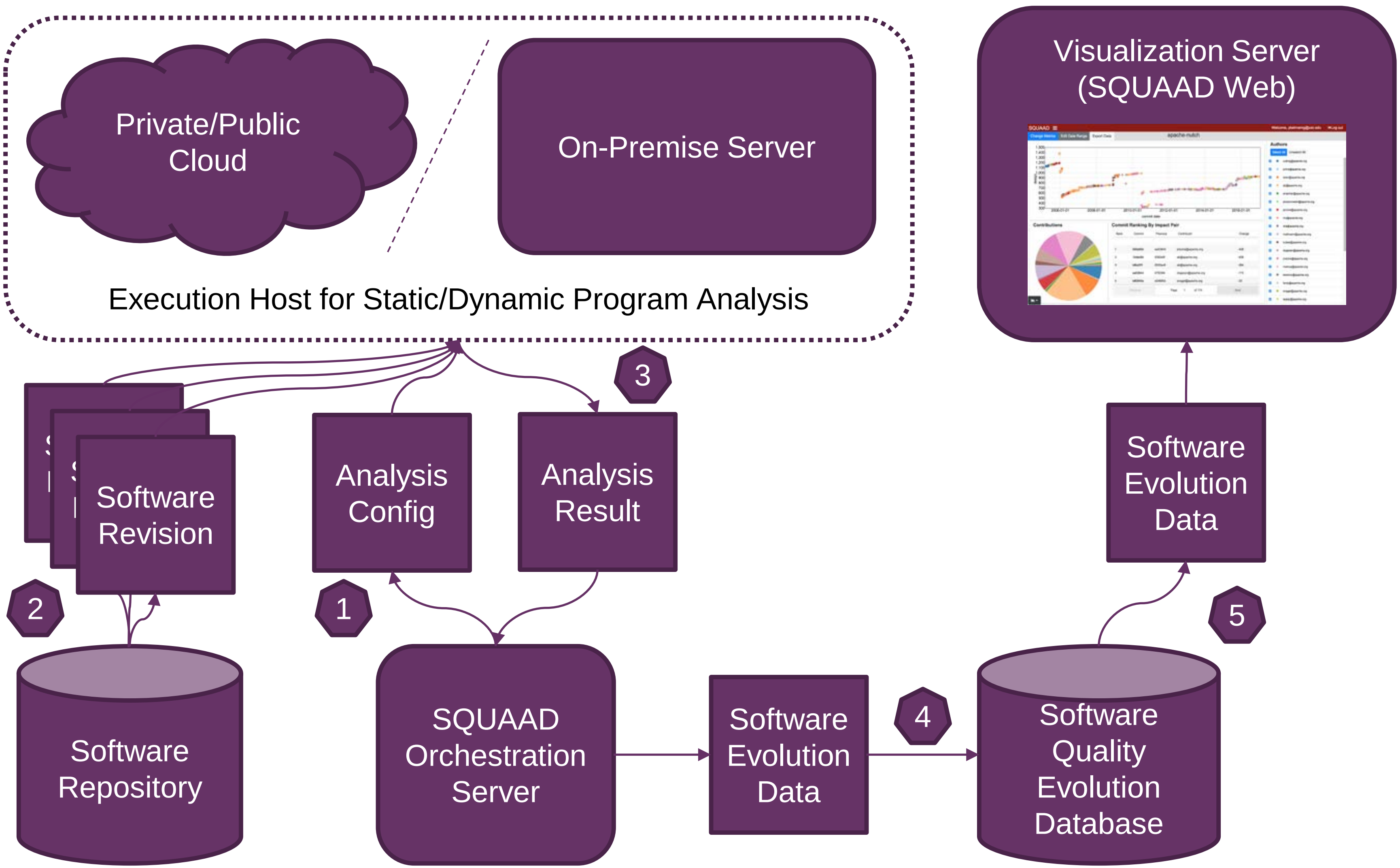
- Provides a more **complete** view of evolution for some commit-level mining tasks.
- Facilitates **manual inspection** of individual data points which is a labor intensive task.
- Reduces the **cost** and **complexity** of the mining task.
- Provides a more **accurate** analysis by omitting irrelevant 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



SQUAAD’s Deployments

- Research papers
  - B. Boehm and P. Behnamghader “**Anticipatory Development Processes for Reducing Total Ownership Costs**” 2018. Systems Engineering. 2018 (Accepted)
  - Behnamghader, Pooyan, et al. "**A scalable and efficient approach for compiling and analyzing commit history.**" Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ACM, 2018.
  - Alfayez, Reem, et al. "**An exploratory study on the influence of developers in technical debt.**" Proceedings of the 2018 International Conference on Technical Debt. ACM, 2018
  - P. Behnamghader and B. Boehm. 2018. **Towards Better Understanding of Software Maintainability Evolution**. In 2018 Conference on Systems Engineering Research (CSER 2018). Charlottesville, USA.
  - Alfayez, Reem, et al. "**How does contributors involvement influence open source systems.**" Software Technology Conference (STC), 2017 IEEE 28th Annual. IEEE, 2017.
  - Behnamghader, Pooyan, et al. "**Towards Better Understanding of Software Quality Evolution through Commit-Impact Analysis.**" Software Quality, Reliability and Security (QRS), 2017 IEEE International Conference on. IEEE, 2017.
  - Behnamghader, Pooyan, et al. "**A large-scale study of architectural evolution in open-source software systems.**" Empirical Software Engineering 22.3 (2017): 1146-1193.
- 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.