Large Scale Evolutionary Analysis on Software Systems

Pooyan Behnamghader
pbehnamg@usc.edu
USC Center for Systems and Software Engineering

Outline

- Introduction
  - Recent contributions
- Large Scale Evolutionary Analysis Example
  - A Large-Scale Study of Architectural Evolution in Open-Source Software Systems
  - Tool Requirements, Architectural Change, Evolutionary Trend Example
- Scale and Challenges
- ARCADE-Controller (ATLAS)
- New Results
  - New Results: FindBugs, PMD, CheckStyle, SonarQube, UCC, SLOCCount
- Discussion

Introduction

Recent contributions:

- A Large-Scale Study of Architectural Evolution in Open-Source Software Systems.
  - Pooyan Behnamghader, Duc Le, Joshua Garcia, Daniel Link, Arman Shahbazian and Nenad Medvidovic.
  - Journal of Empirical Software Engineering. (Accepted Sept. 20, 2016/In Press.)
  - Domains: Software Architecture Recovery, Mining Software Repository
- Using Visual Symptoms for Debugging Presentation Failures in Web Applications
  - Sonal Mahajan, Bailan Li, Pooyan Behnamghader, William G. J. Halfond
  - 2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)
  - Domains: Software Testing, Web Interface Analysis

Both studies initially had **scalability** issues

Large Scale Evolutionary Analysis Example

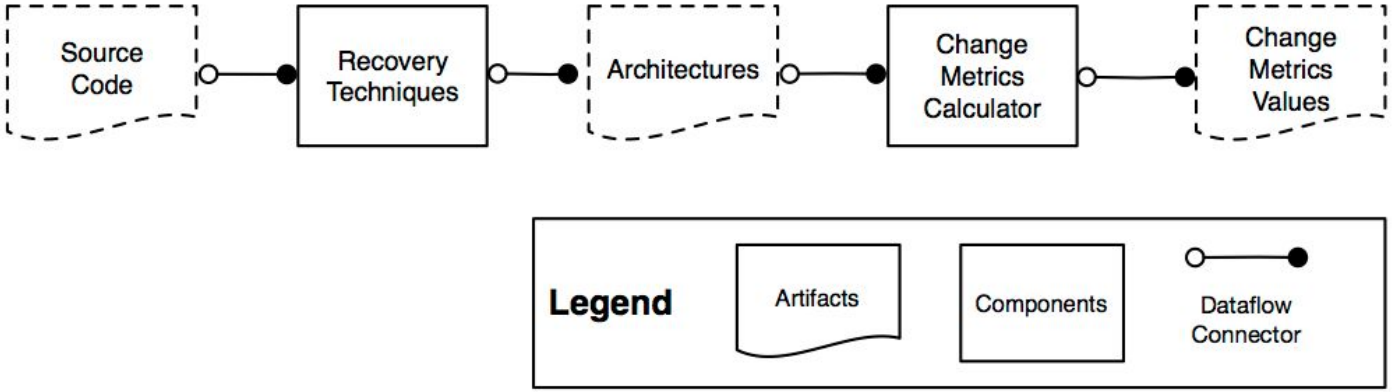
A Large-Scale Study of Architectural Evolution in Open-Source Software Systems

Research Questions:

- RQ1: To what extent do architectures **change** at the system level?
- RQ2: To what extent do architectures **change** at the component level?
- RQ3: Do architectural **changes** at the system and component levels occur concurrently?
- RQ4: Does significant architectural **change** occur between minor system versions within a single major version?

Analysis Tool Requirements

- Architecture recovery techniques to extract architectural models from implementation level artifacts.
- Metrics to calculate system level and component level architectural changes.



Architectural Change

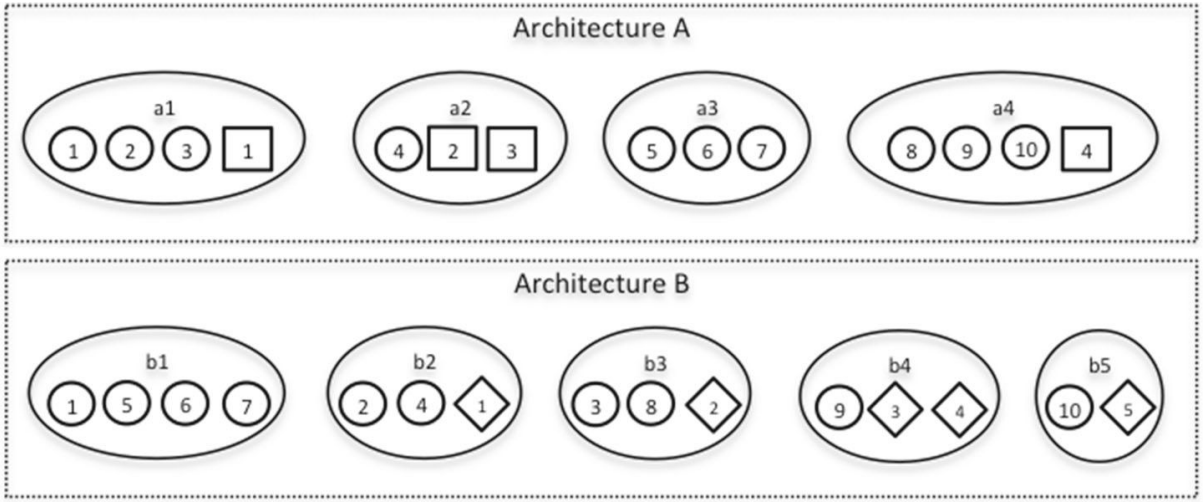
What does architectural change mean?

Architecture A

- 4 clusters
- 14 entities

Architecture B

- 5 clusters
- 15 entities



Evolutionary Trend Example

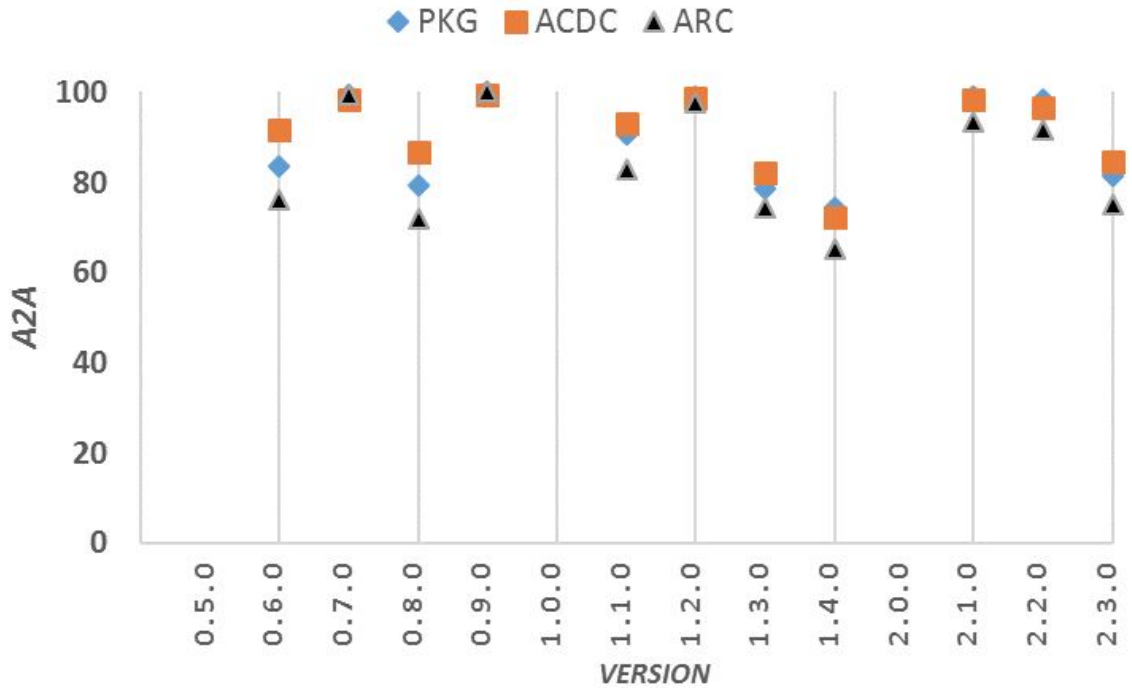
Architectural changes among minor versions of apache-ant-ivy

Recovery techniques:

- PKG (gross organization)
- ACDC (module dependency)
- ARC (semantic view)

Change metric:

- A2A (System-level)



The Scale of the Study

The largest study of architectural recovery and architectural evolution to date:

- 23 subject systems
- 931 examined system versions
- 140 MSLOC analyzed code
- 2793 analyzed architectural models
- Comparing pairs of architectural models using two change metrics

The challenges in this scale:

- Dealing with issues of each case study
- Comparing the results of multiple analyses on same data points
- Collaborating in a team

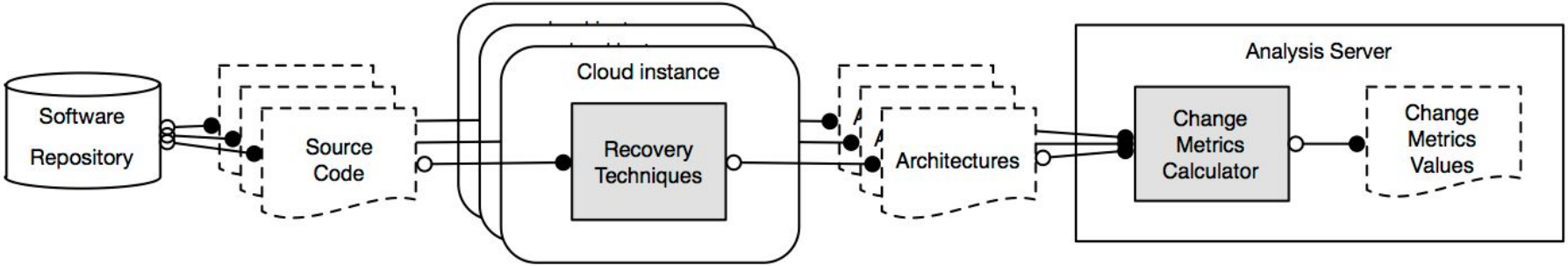
Challenges

- Case Study
  - Different structure/modules/build automation tool for each system/version
    - e.g., the “core” module locates in a different subdirectory in older versions.
  - Unsuitable case studies
    - e.g., the architecture is too small to be meaningful for some specific analysis.
- Analysis
  - Implementing tool for a new technique that does not have any ground-truth
    - e.g. your algorithm is wrong or there is a bug in the implementation.
      - You realize it when you see weird evolutionary patterns in your data
  - Using already existing of-the-shelf implementation of a solid technique
    - The implementation is simply buggy. (e.g. non-deterministic implementation)
- Teamwork!

It Would be Much Easier if...

- For each case study we could declare
  - the remote repository.
  - build command(s).
  - interesting modules, subsystems, or packages.
  - interesting version sets
- For each analysis we could declare
  - how to prepare the environment for the analysis.
  - how to run analysis on each system/version.
  - how to interpret and compare the results, and generate statistics.
- We could define a portable workflow to automatically run the study
  - on a powerful remote server.
  - distributed over the cloud.

ARCADE-Controller (ATLAS)



- Cloud Instances
  - Download and compile the source code
  - Run the analysis on the subject system
  - Send the artifacts to the analysis server
- Analysis Server
  - Compares the artifacts using change metrics
  - Gets the statistics

ARCADE-Controller (ATLAS) - cont.

- The ability to define a solid workflow for the analysis
  - Replicability
- The ability to employ cloud-computing power to run large-scale analyses in a reasonable amount of time
  - Scalability
- The ability to use the same subject systems for different analyses
  - Data Consistency and Reusability
- The ability to run each analysis in a cloud instance as a sandbox
  - Running static (e.g., ARCADE) and dynamic (e.g., FieryEye[1]) analysis

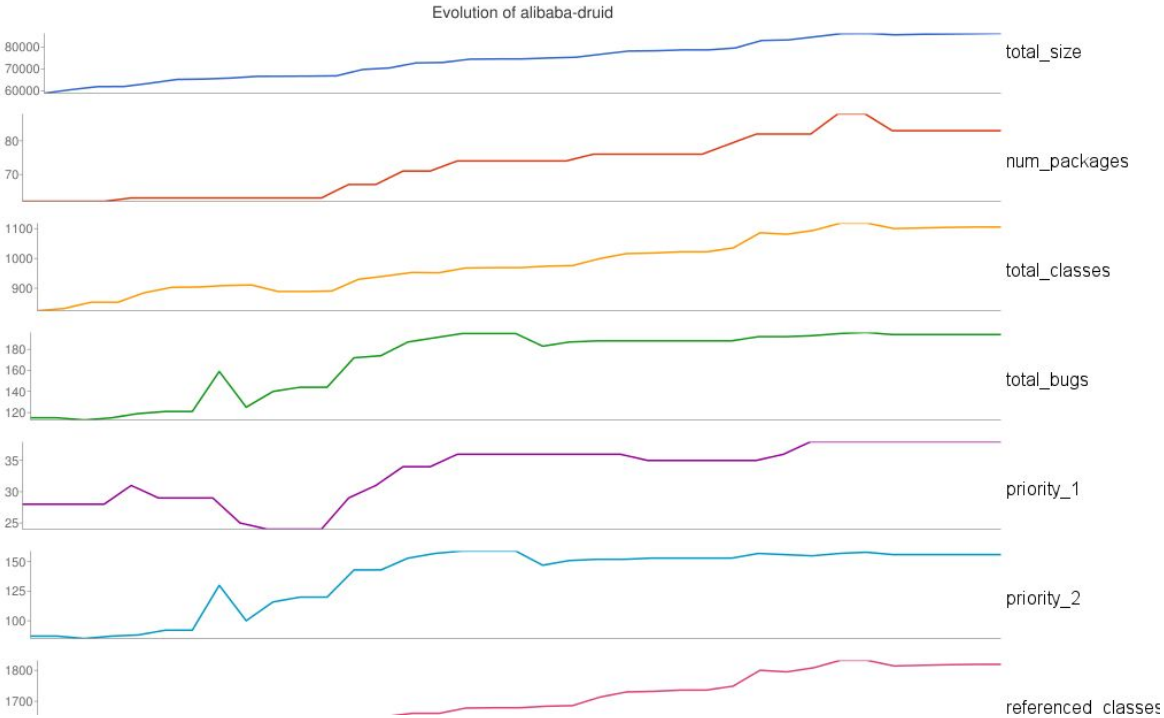
1. FieryEye is dynamic web-interface analyzer. We used the same technique to resolve the scalability issues for FieryEye.

New Results

- Architectural evolution
  - Architecture recovery and architectural changes: ACDC, ARC, PKG, A2A, CVG
- All it takes from downloading the source code from the repository to generating the statistics is “push a button”!
  - Both on a local machine or on the cloud
- There are already several subject systems configured in the framework
  - The capability to collect a large amount of data without much effort!
- How about the evolution of other aspects (e.g., defects, style, debt) of a software system?
  - FindBugs, PMD, CheckStyle, SonarQube, UCC, SLOCCount

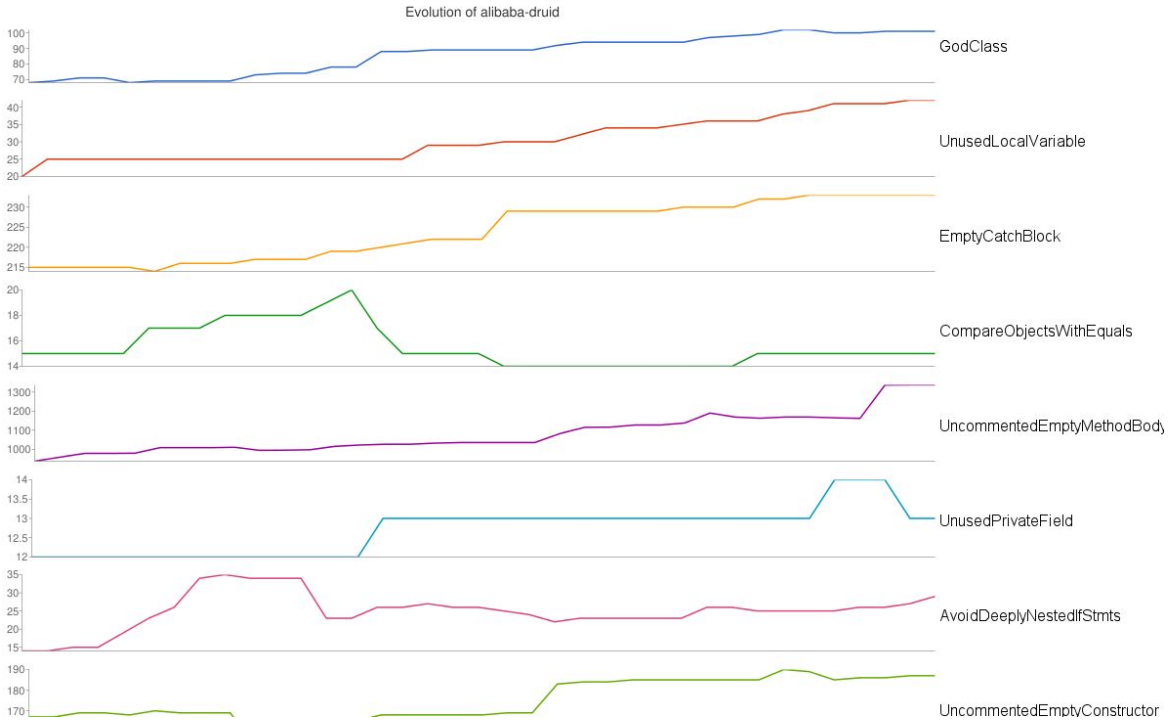
FindBugs

- Finds bugs in Java programs.
- Analyze programs compiled for any version of Java.
- Requires binary releases



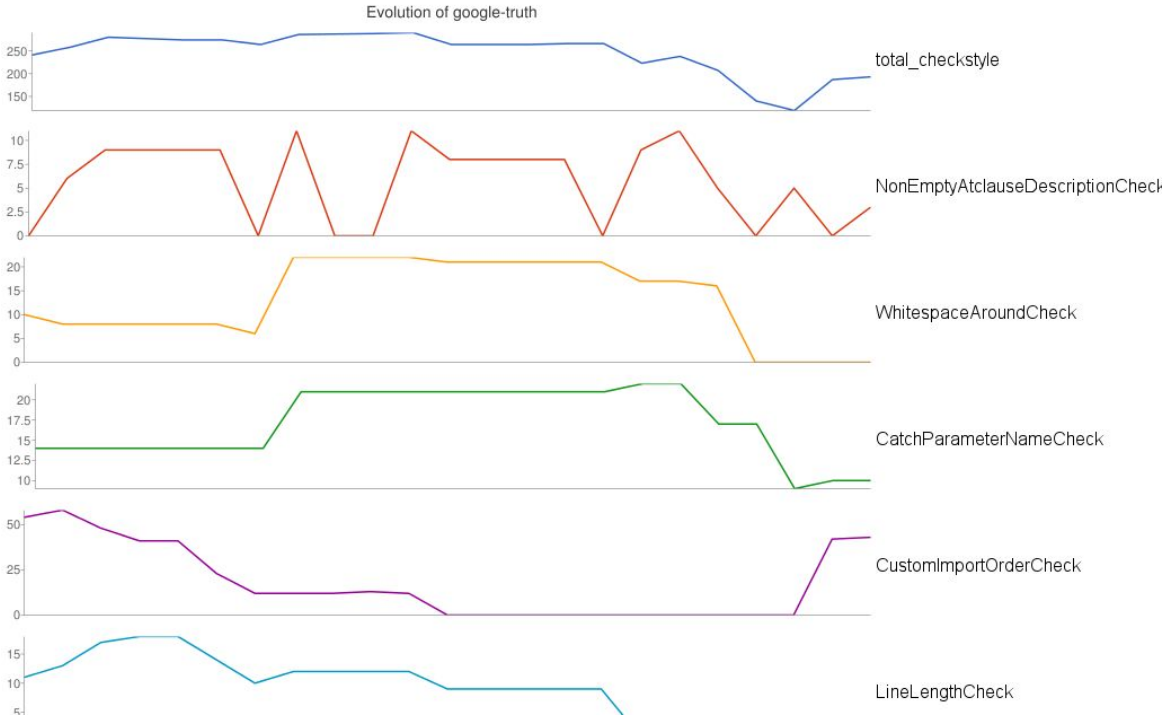
PMD

- Finds common programming flaws like:
  - unused variables
  - empty catch blocks
  - unnecessary object creation
  - ...
- Source code analyzer
- Supports a variety of languages



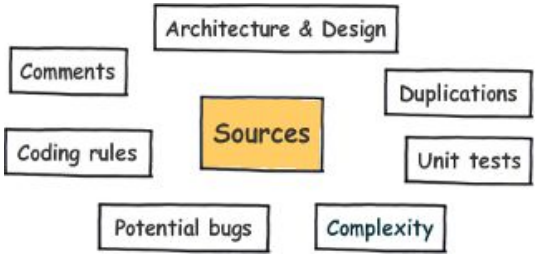
Checkstyle

- Helps programmers write Java code that adheres to a coding standard.
- Highly configurable
  - Sun Code Conventions
  - Google Java Style

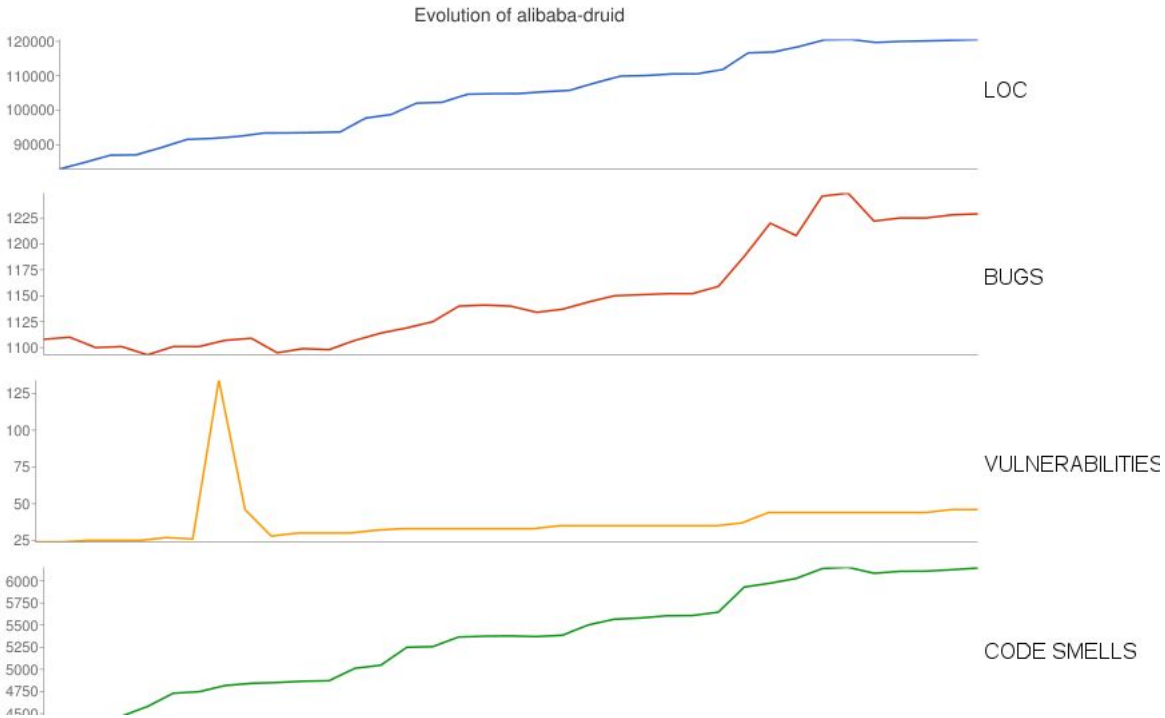


SonarQube

- Manages code quality.
- Covers the 7 axes of code quality:

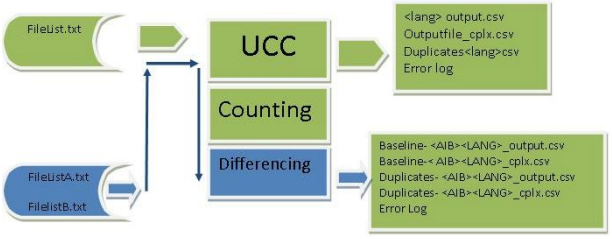


http://www.sonarqube.org

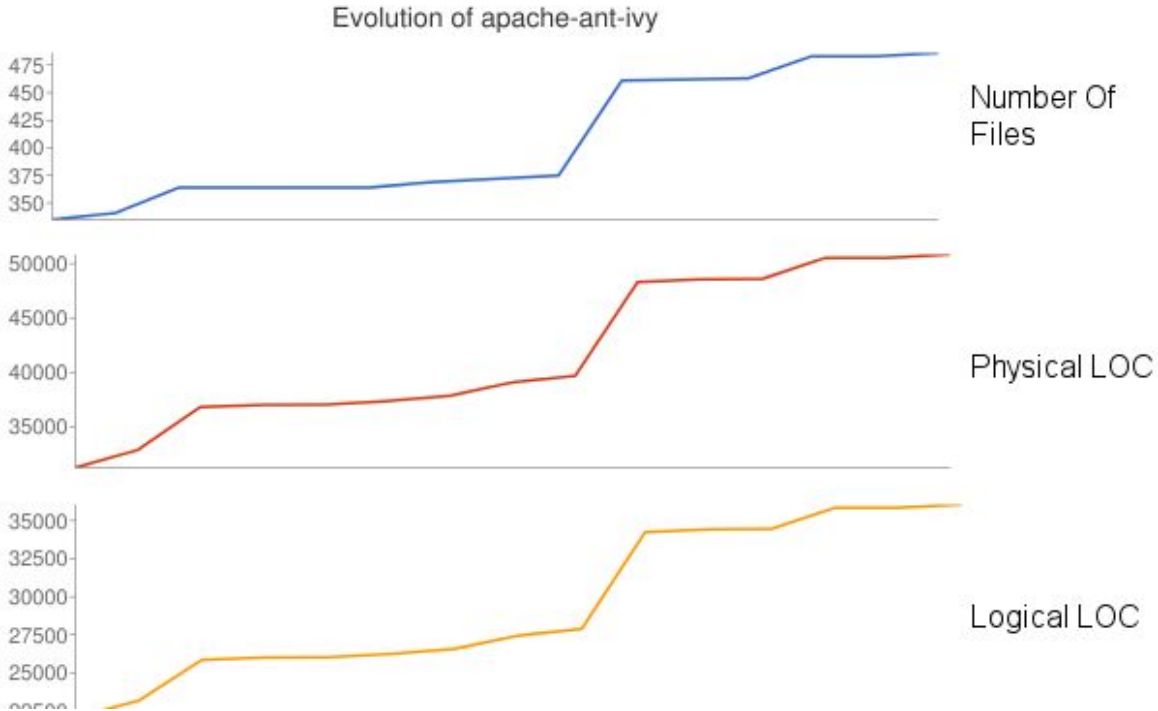


UCC (Unified Code Count)

- Counts, compares, and collects logical differentials between two versions of the source code of a software product.
- USC Center for Systems and Software Engineering

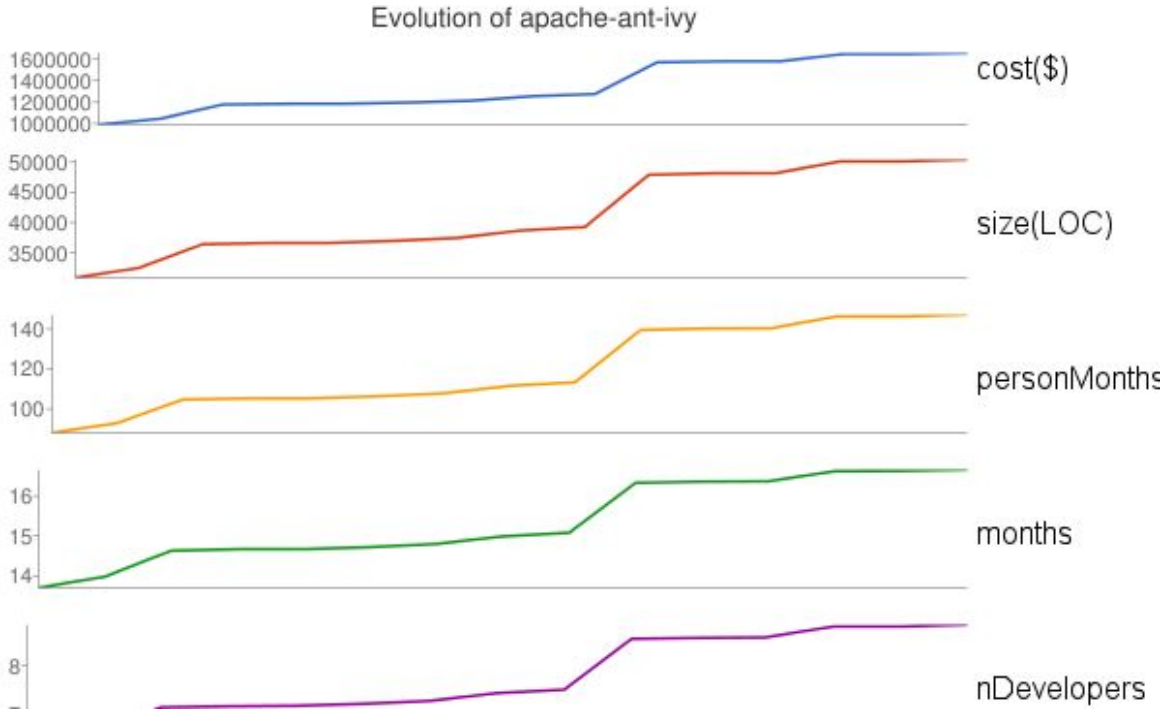


https://en.wikipedia.org/wiki/File:Block_Diagram_of_UCC.JPG



SLOCCount (calculates basic COCOMO model)

- Counts physical source lines of code.
- Basic COCOMO model
  - Person-Months = 2.4 * (KSLOC**1.05)
  - Months = 2.5 * (person-months**0.38)
  - Estimated Average Number of Developers= Effort/Schedule
  - Average salary = $56,286/year, overhead = 2.40.



Discussion

- Suggestions
  - What other static analysis tools can be added to the framework?
  - What dynamic analysis tools can be added to the framework?
  - Would it be interesting if we extend the framework to study difference between commits and the impact of each developer?