Studying the test process in open source software systems

Andy Zaidman, **Bart Van Rompaey,**

Serge Demeyer, Arie van Deursen

***TU Delft*, University of Antwerp**

13de Testdag, 29 November 2007

Delft University of Technology

SERG

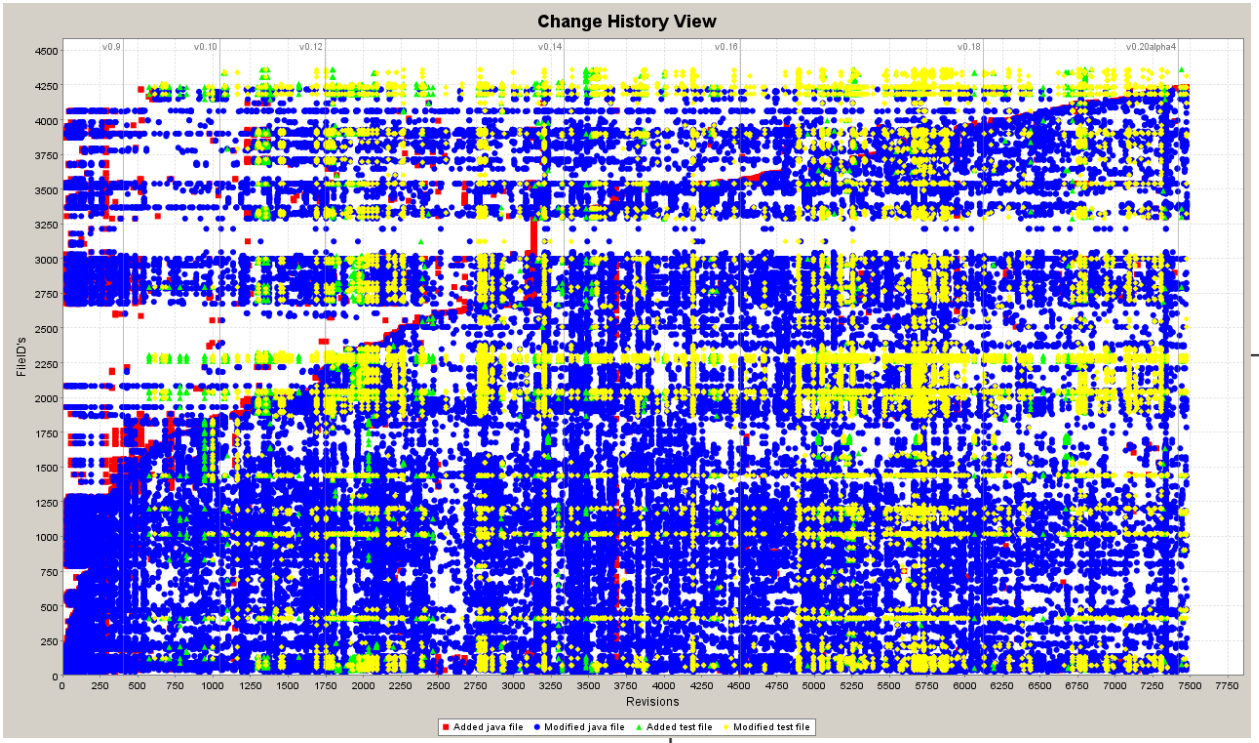
How do YOU test?

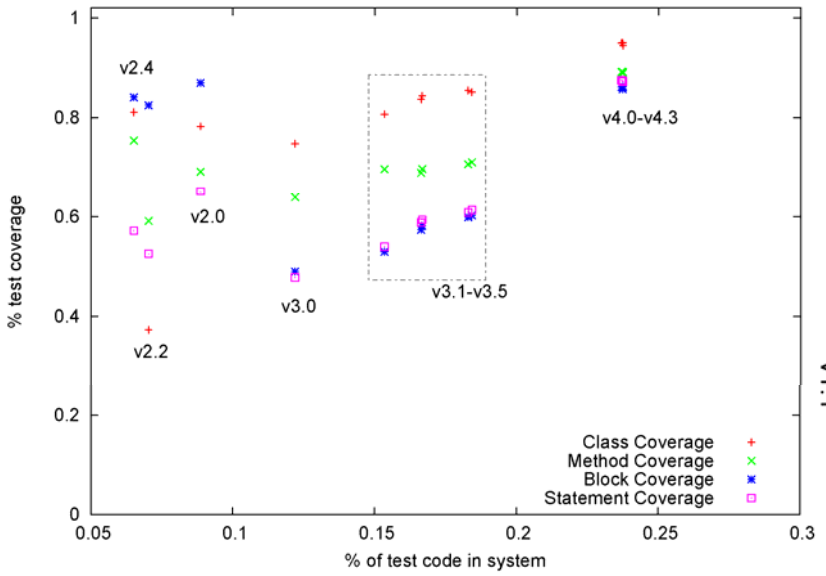
How does your team test?

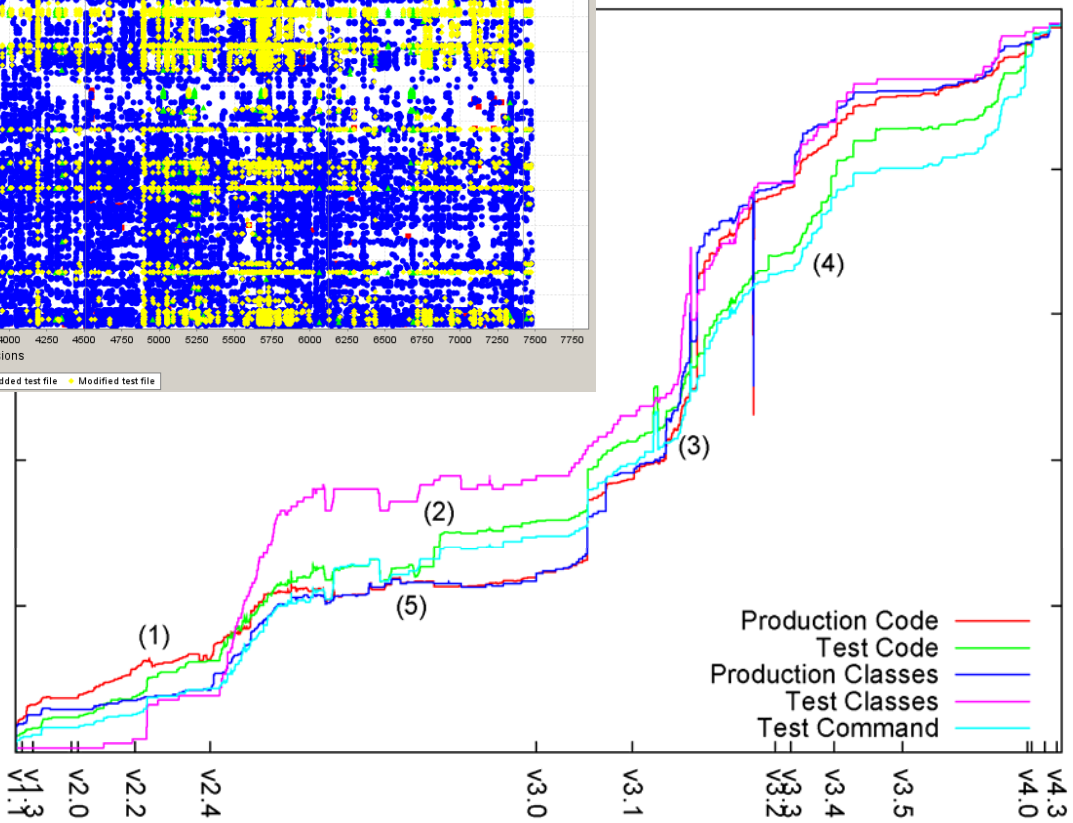
WHY do you want to know?

TU Delft

Delft University of Technology


Change History View
v0.9
v0.10
v0.12
v0.14
v0.16
v0.18
v0.20alpha4
FileID's
Revisions
Added java file
Modified java file
Added test file
Modified test file


v2.4
v2.0
v2.2
v3.0
v3.1-v3.5
v4.0-v4.3
% test coverage
% of test code in system
Class Coverage
Method Coverage
Block Coverage
Statement Coverage


(1)
(2)
(3)
(4)
(5)
Production Code
Test Code
Production Classes
Test Classes
Test Command
v1.1
v1.3
v2.0
v2.2
v2.4
v3.0
v3.1
v3.2
v3.3
v3.4
v3.5
v4.0
v4.3

Well, why would you…

… want to know the “quality” of the tests?

- When maintaining the system
  → can you trust the existing tests?
- When judging the overall quality of a software system
  → is it good enough for *our* standards?
- When monitoring the test process
- It is a **selling argument**!
  → indication of trustworthiness

What about… test coverage?

Nice, but somewhat hollow measure…

- Easy to keep artificially high
  → high-level tests versus ‘unit tests’
  → unit tests can serve as ‘defect locators’
- What about boundary testing?
  → coverage tells you nothing…

“Test health”

→ determine the long-term quality of your test-suite.

- If the quality is constant → less worrying for the future
- If the quality is fluctuating → possible problematic (future) evolution

→Tells something about the *process*

→How to determine?

"Test health" vs. "Co-evolution"

- How well does the test writing effort follow normal development?

- In terms of:
  - **Commit behavior**, i.e., when are tests adapted?
  - **Metric-trends**, i.e., determine impact of changes to production code and test code
  - **Test coverage ratio trends**, i.e. the ratio of test coverage versus test code

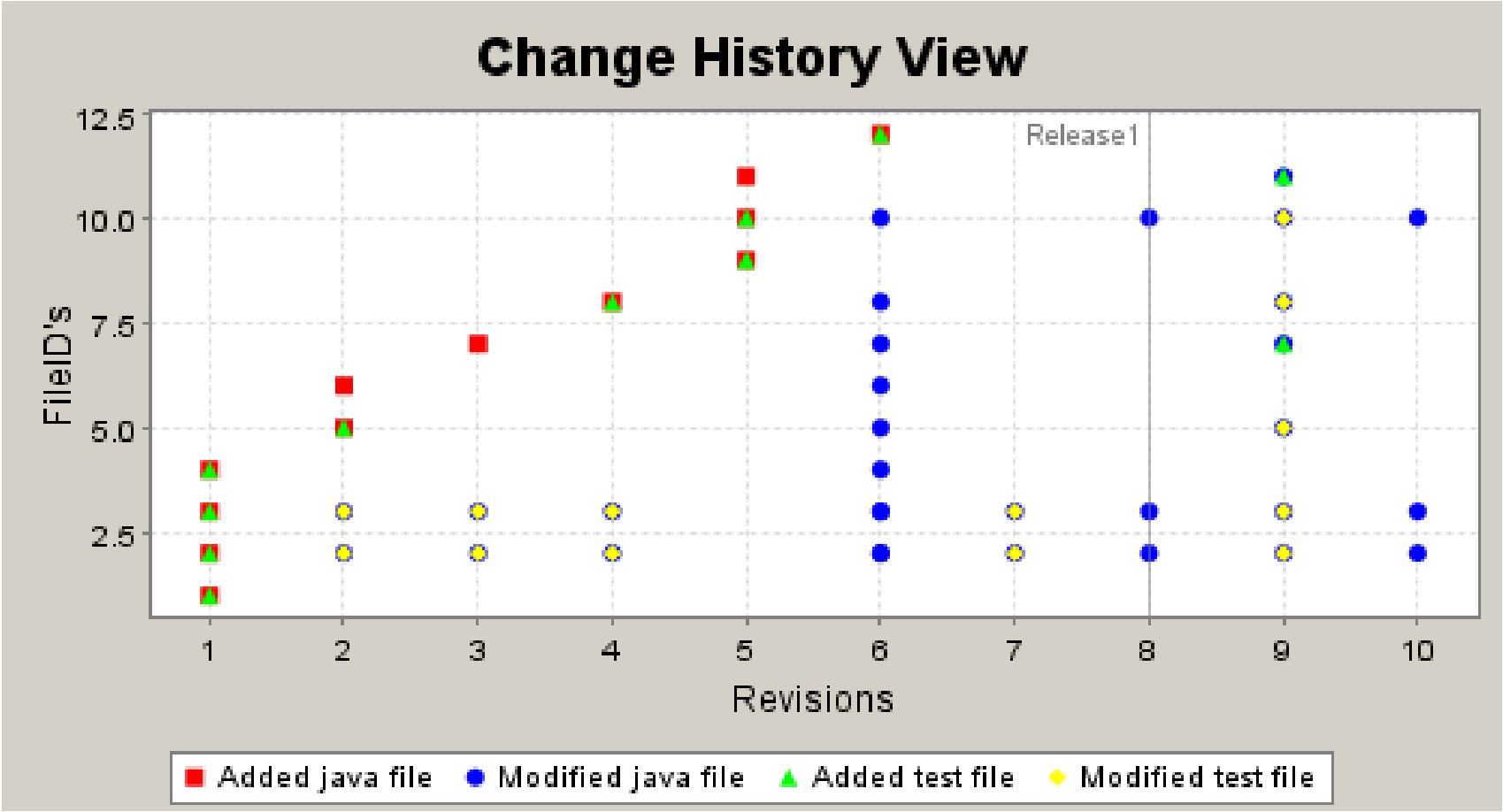
How? Use *historical* data

- Development history is recorded by the *version control system* (VCS)
  - Subversion, CVS, Microsoft Visual SourceSafe, ClearCase, …

- Separate production code from test code

→ study

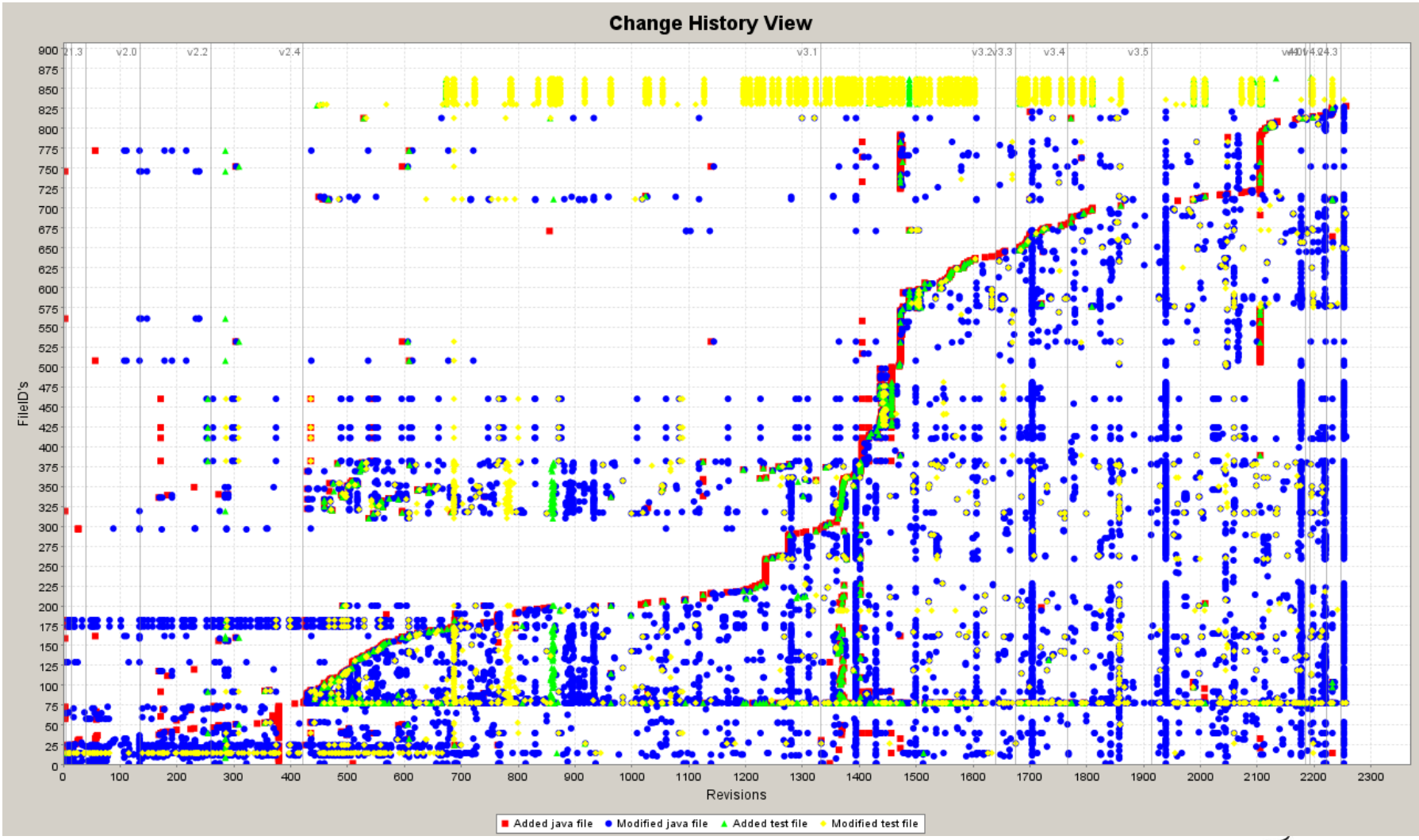
Example change history view



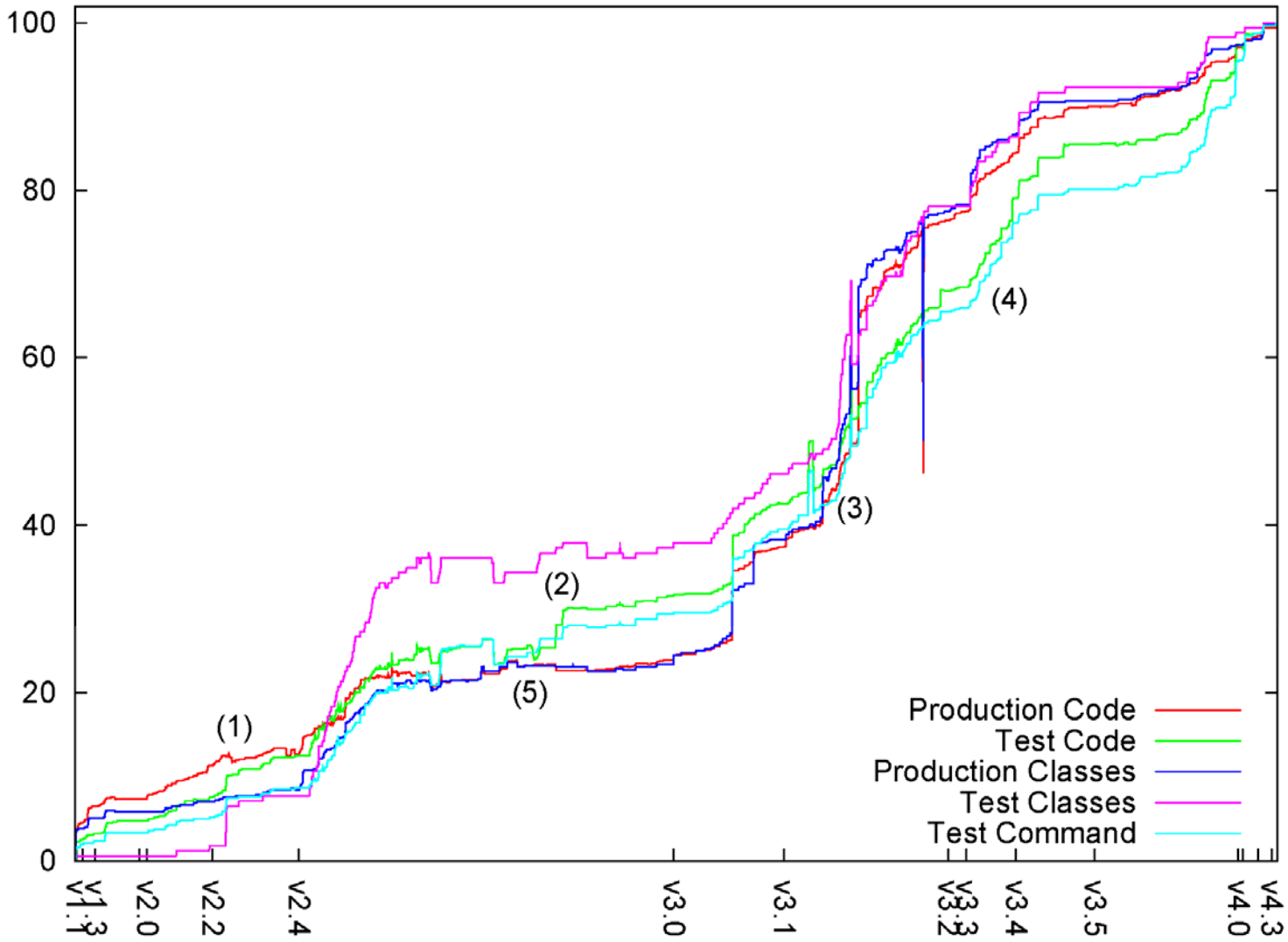
Two subject software systems

- Checkstyle
  - Java source-code style checker (and automatic improver)
  - ~ 6 years development, 2260 commits, 6 developers, 738 classes, 47 kSLOC
- ArgoUML
  - UML drawing application
  - ~ 7 years development, 7477 commits, 42 developers, 1533 classes, 130 kSLOC

Checkstyle change history view

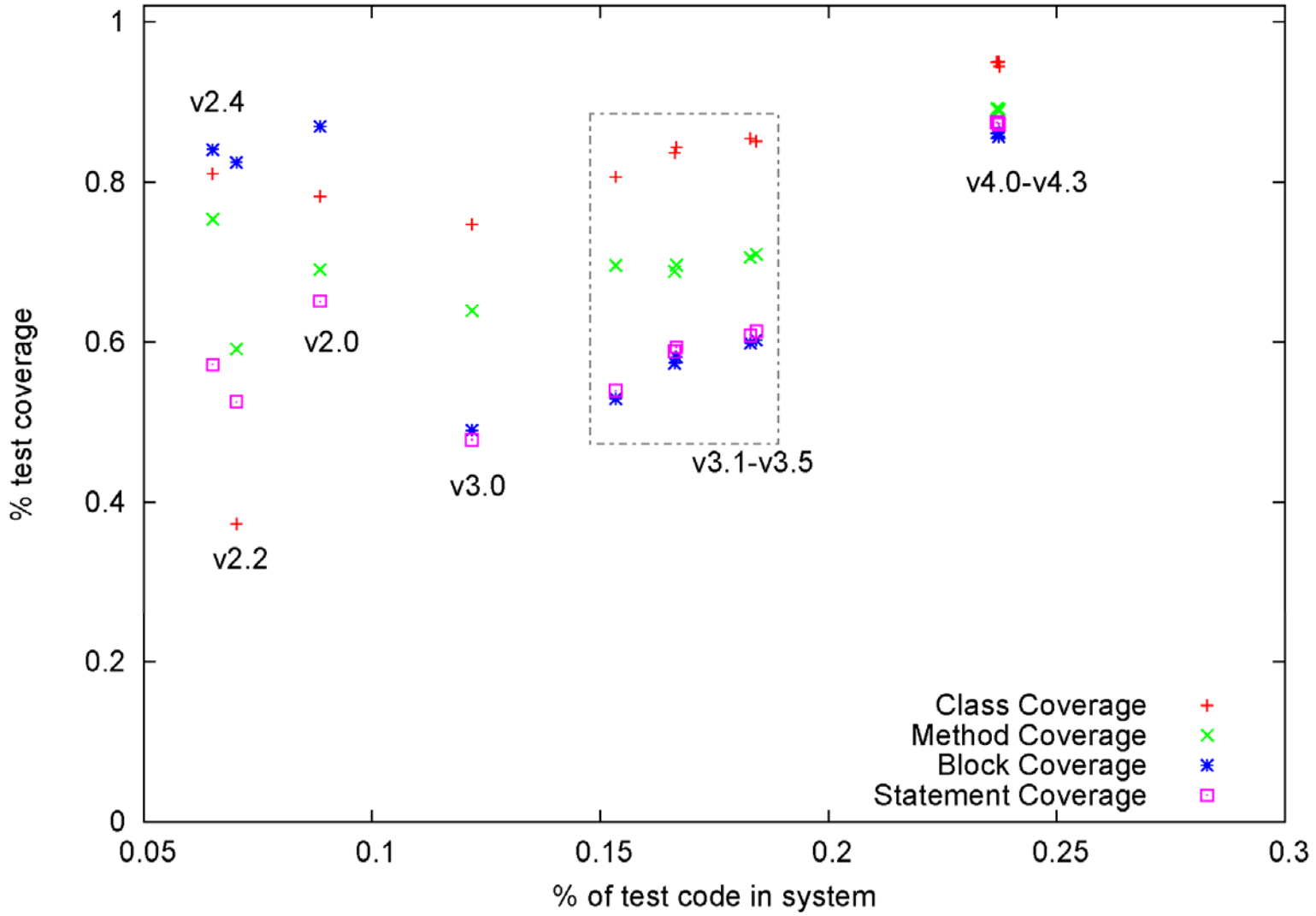


Checkstyle growth history view

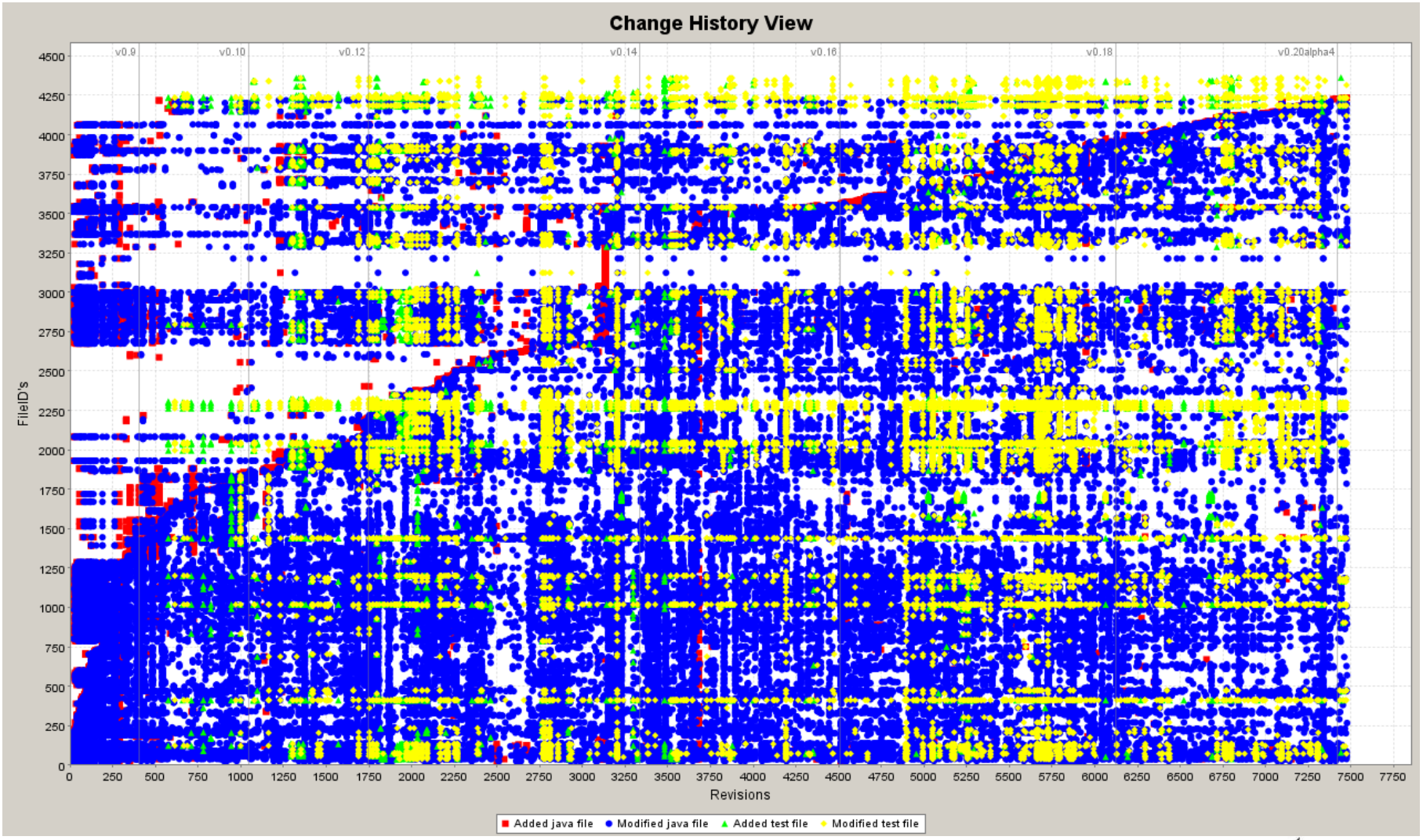


TU Delft

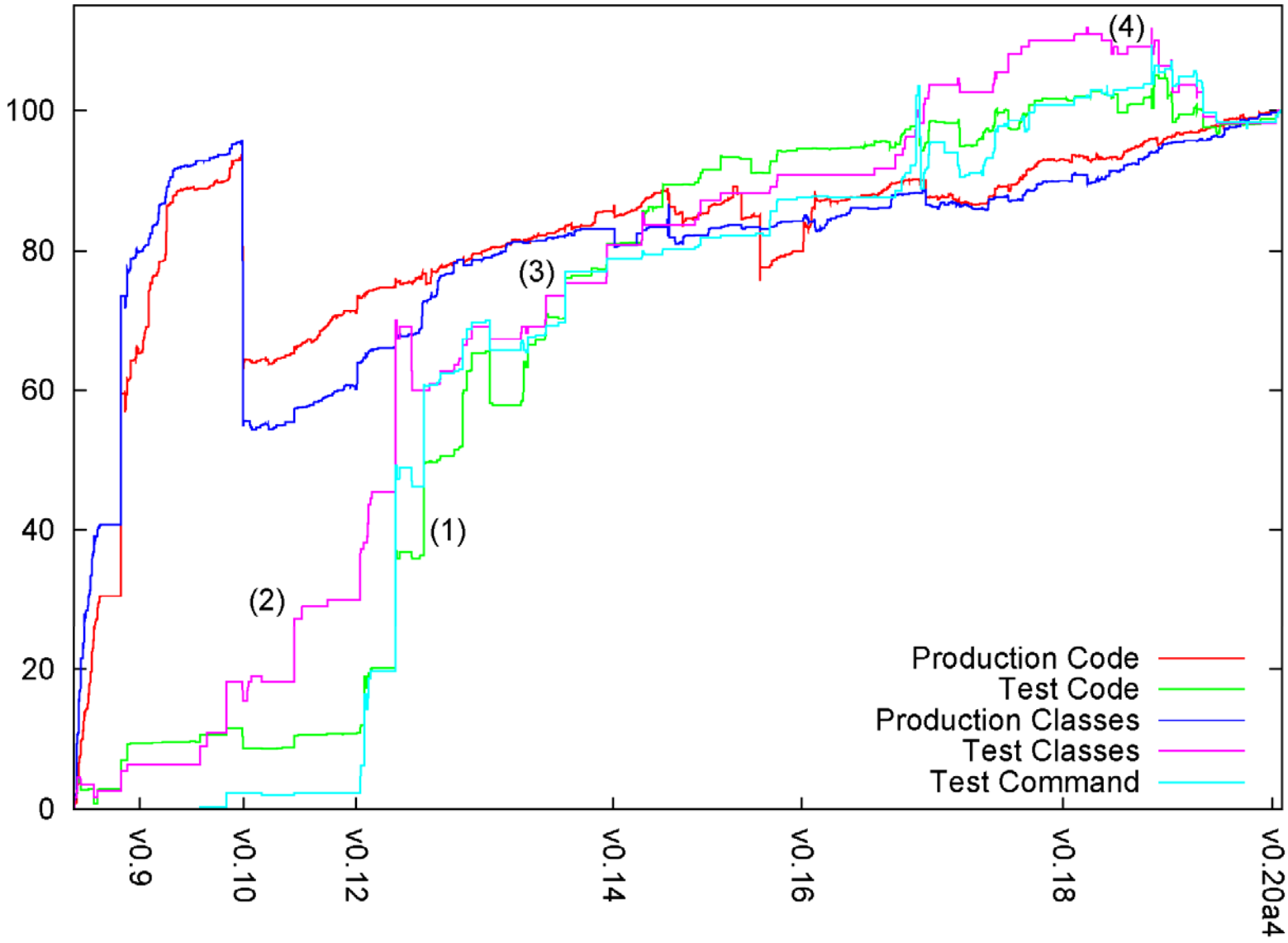
Checkstyle test quality evolution



ArgoUML change history view



ArgoUML growth history view



Conclusion?

- In our case studies: phased testing
  - Sometimes 6 months of no testing!
  - How bad?
    - Sometimes tests just less effective
    - Sometimes the tests don't compile any more
- How does it happen in other environments?
  - Companies? →currently working with **SIG**
  - Controlled open source projects (Apache)?

SERG

What NOW?

Starting up a new research project that builds upon these techniques.

Looking for industrial partners…

INTERESTED?

Come talk to us!

TU Delft
Delft University of Technology

a.e.zaidman@tudelft.nl

http://www.st.ewi.tudelft.nl/~zaidman

bart.vanrompaey2@ua.ac.be

http://www.win.ua.ac.be/~bvromp

Delft University of Technology