Assertion-Driven Development: Assessing the Quality of Contracts using Meta-Mutations

Thomas Knauth, Christof Fetzer, Pascal Felber

Denver, CO, USA, 2009-04-04

Forecast

- weak contract completeness
- completeness varies widely for mature JML classes and programs developed by students
- better tools are needed to help develop sound/complete contracts

Motivation

- Is this a good contract for a square root function?

```
//@ require x >= 0;
//@ ensure \result * \result == x;
long square_root(long x) { ... }
```

Motivation (2)

- What's wrong with this contract?

```
//@ require x >= 0;
//@ ensure \result * \result == x;
long square_root(long x) { ... }
```

- contract only correct if square root of x is a natural number
- **writing correct self-checks is non-trivial!**

Weak contract completeness

- **weak contract completeness** is capability to detect mutants in a given implementation
- lower bound = detected mutants / all mutants
- upper bound = detected mutants / non-equivalent mutants
- goal: 100% upper bound when lower bound saturated

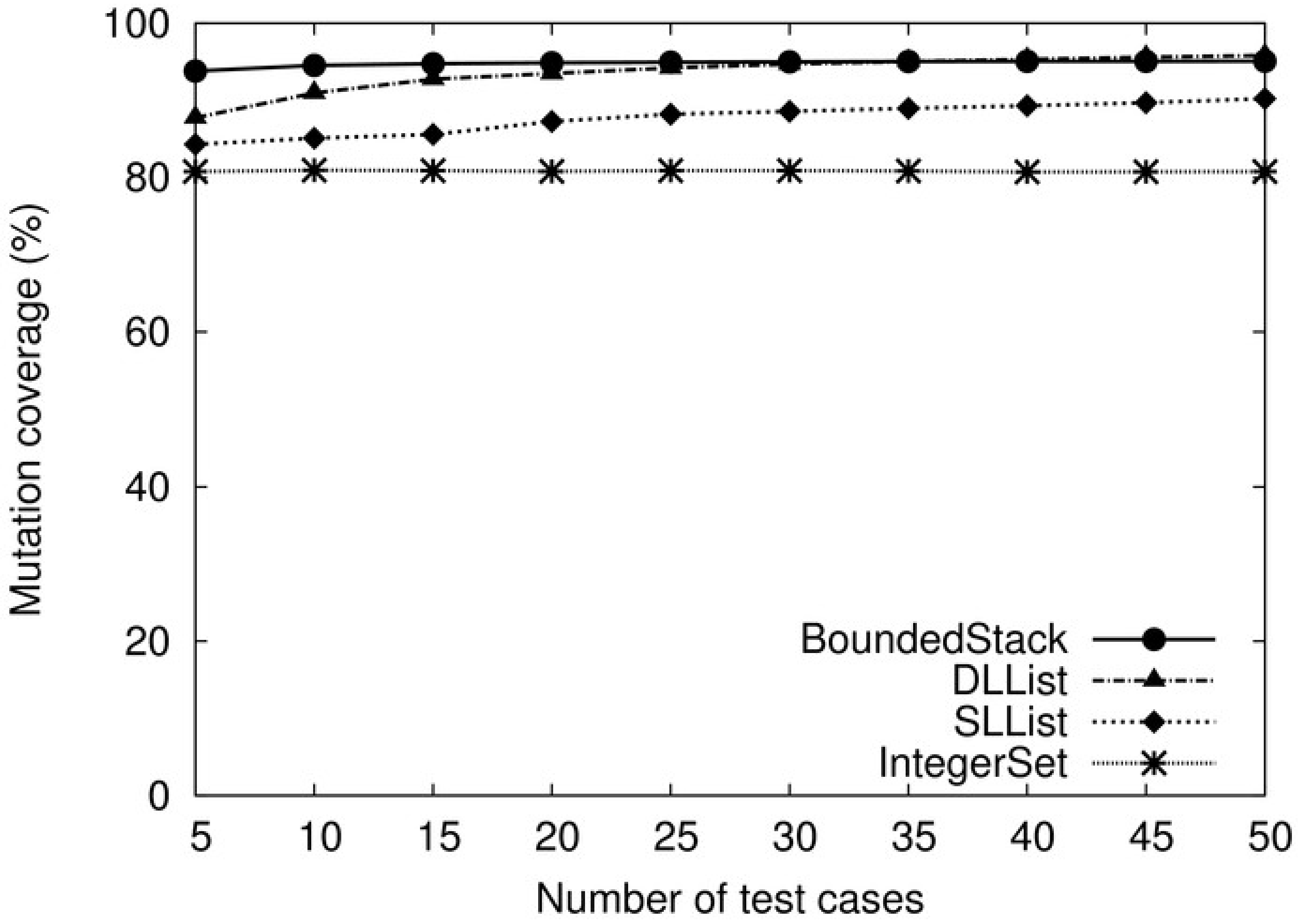
Experiments

- apply approach to set of JML classes
  - generate random sequence of method calls
- 19 students develop program with contracts
  - generate random test inputs
- generate meta-mutants in either case

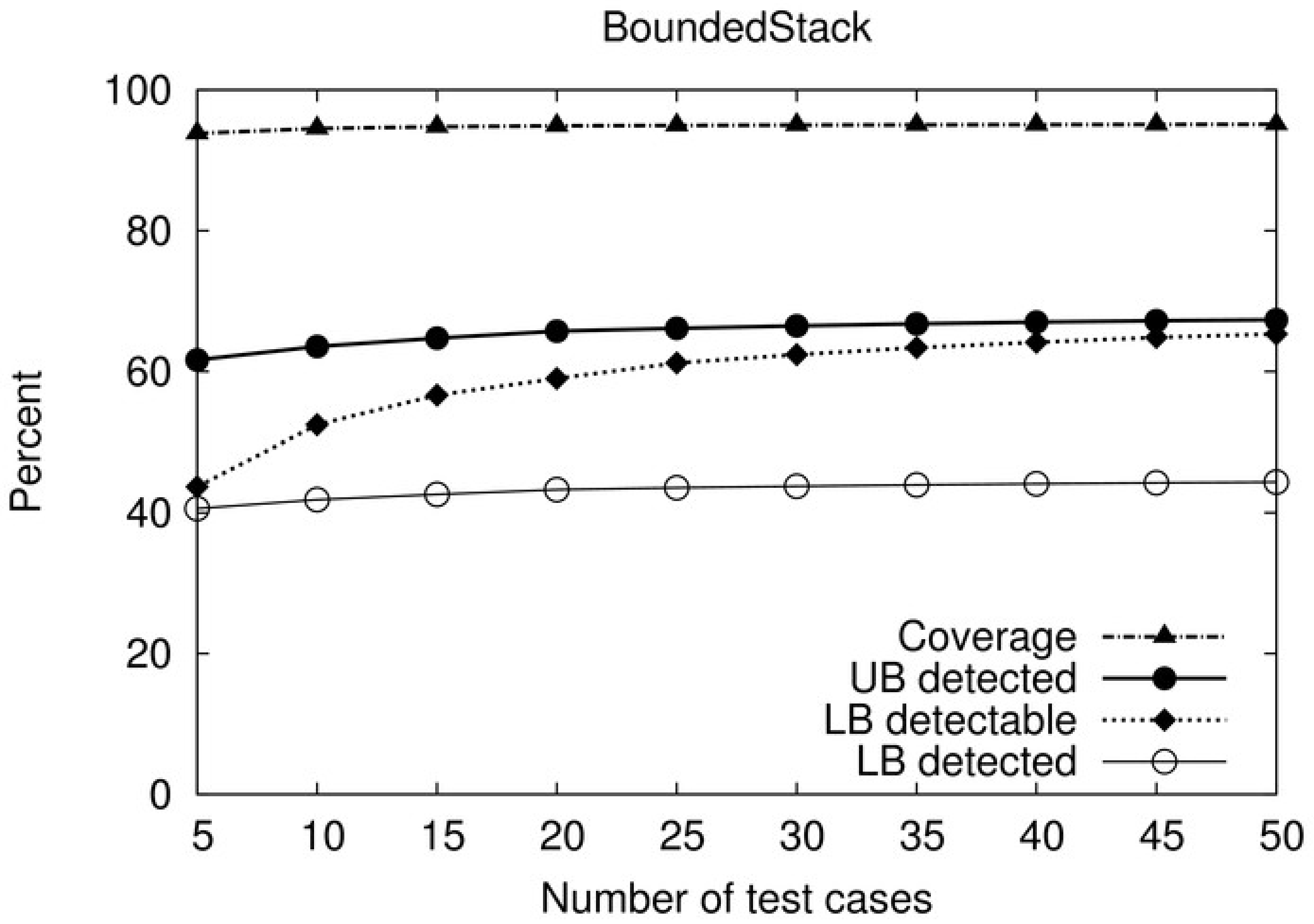
Meta-mutation

- implemented as Eclipse plug-in
- mutation operators loosely based on previous work
  - not all operators are possible when meta-mutating, e.g. swap access modifiers
- mutations can be switched on individually based on ID
- source code mutation has better accuracy than mutating binary code

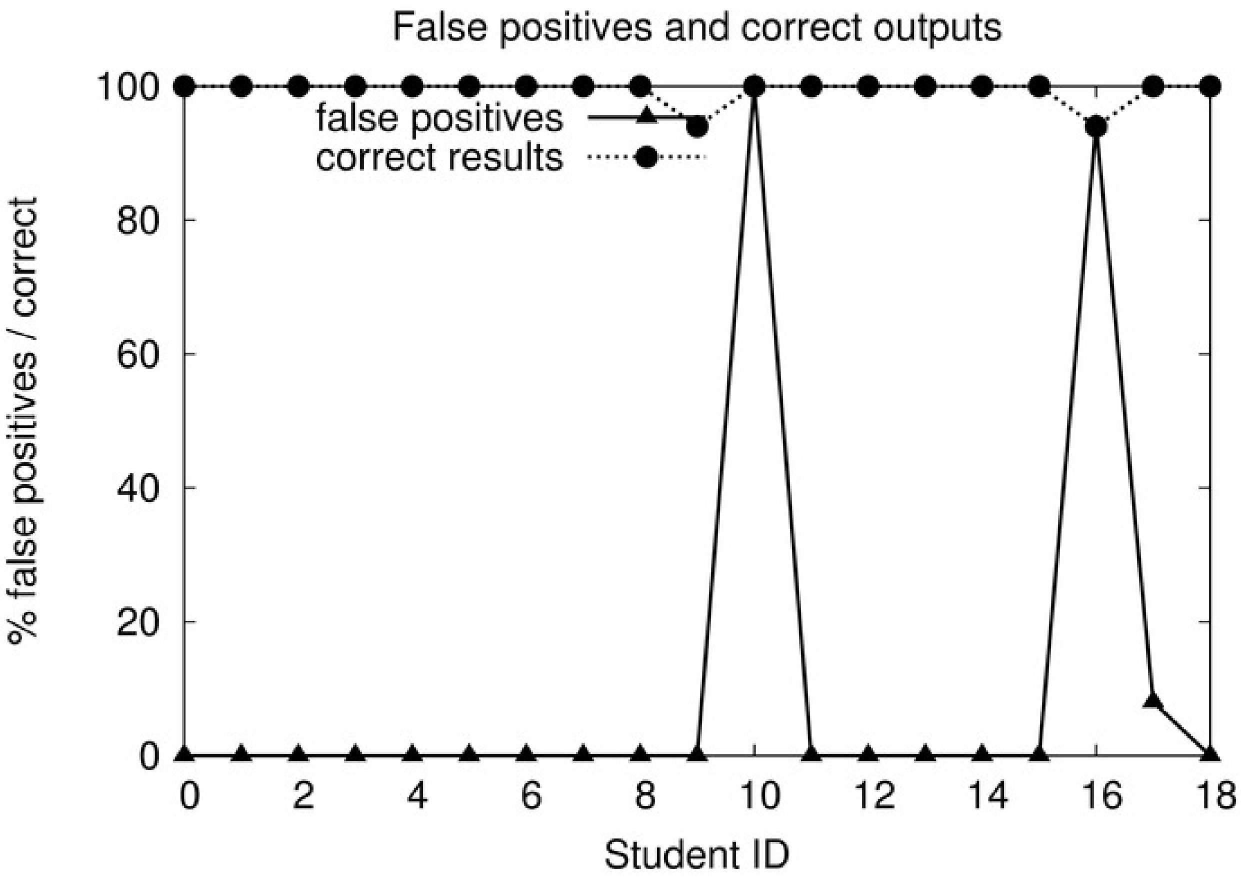
Mutation coverage



JML BoundedStack

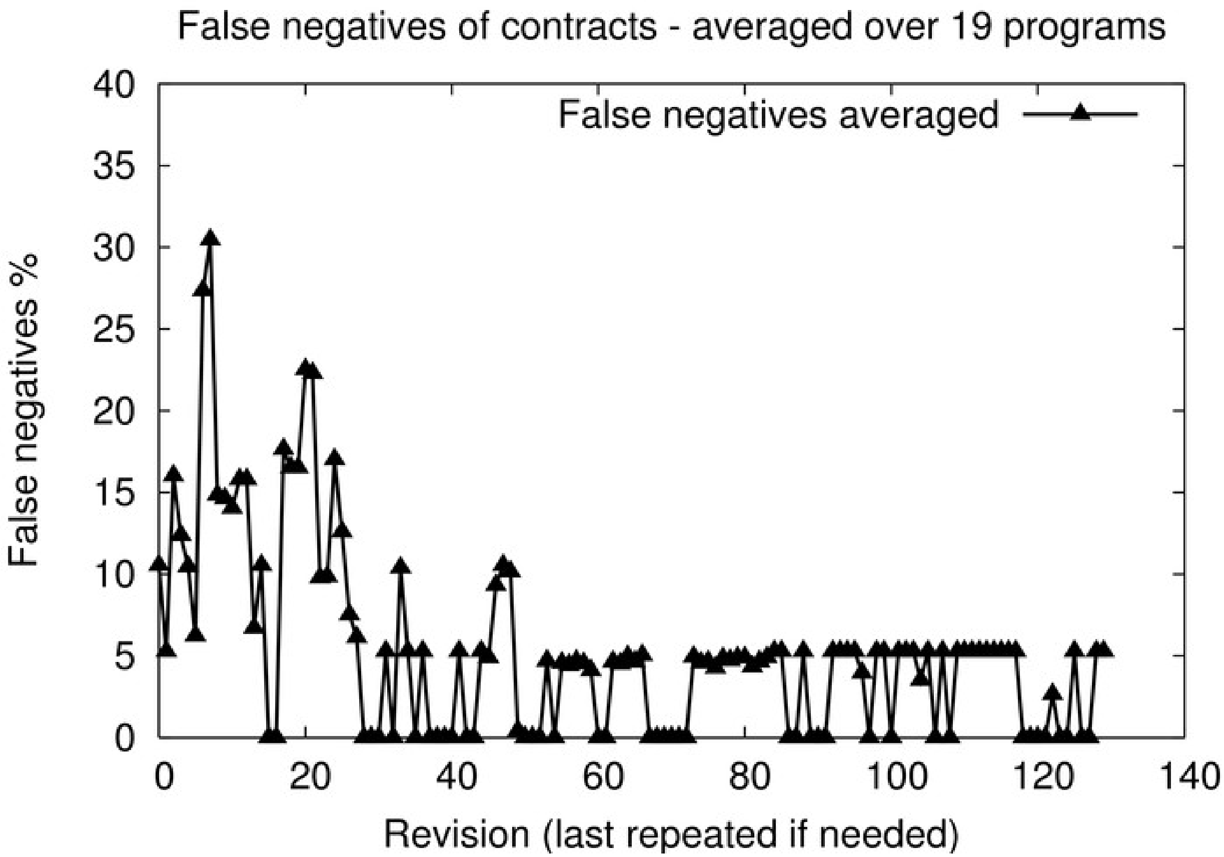


False positive (FP) and correctness



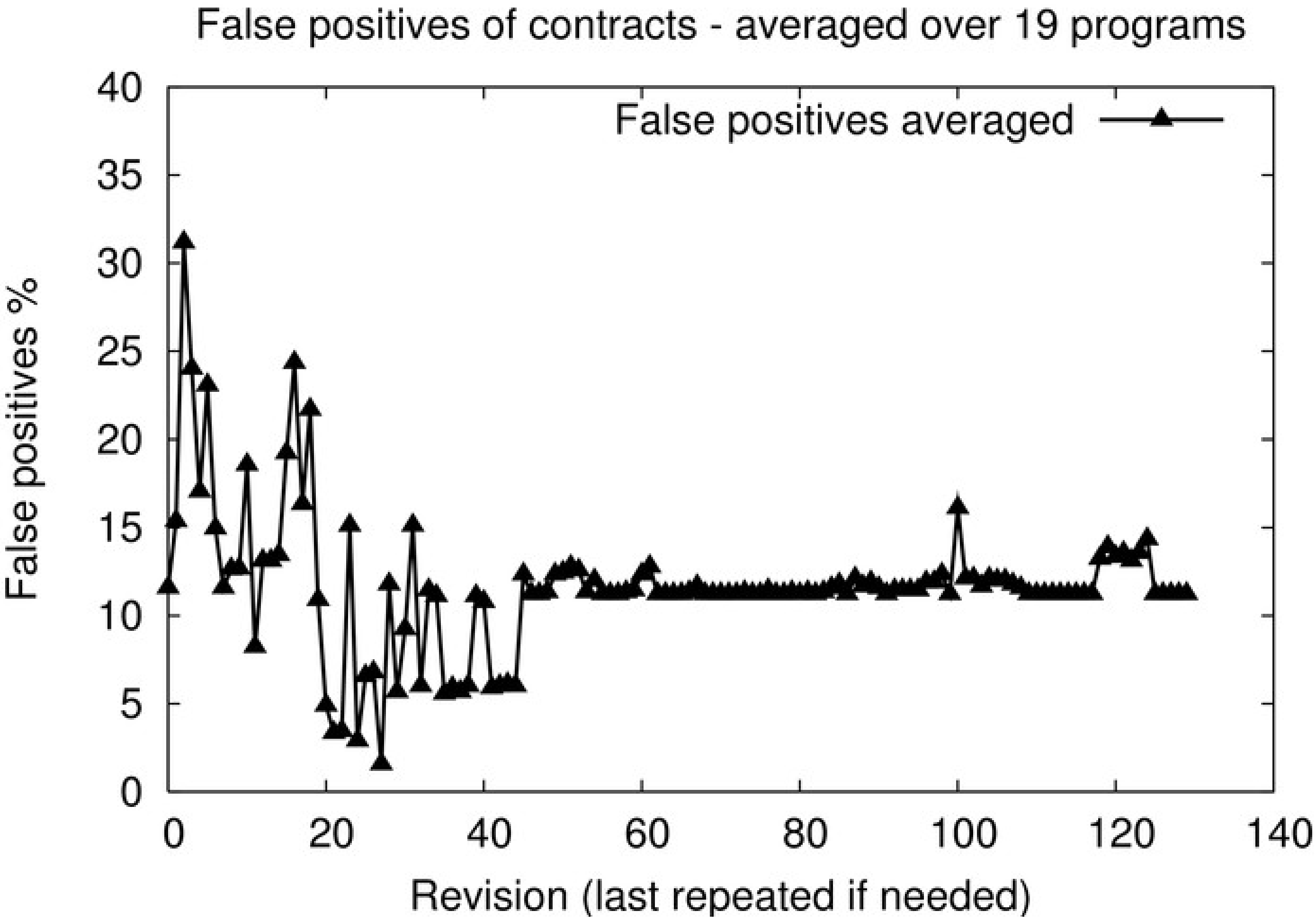
- FP = program correct, contract raising alarm

False negatives (FN) all revisions

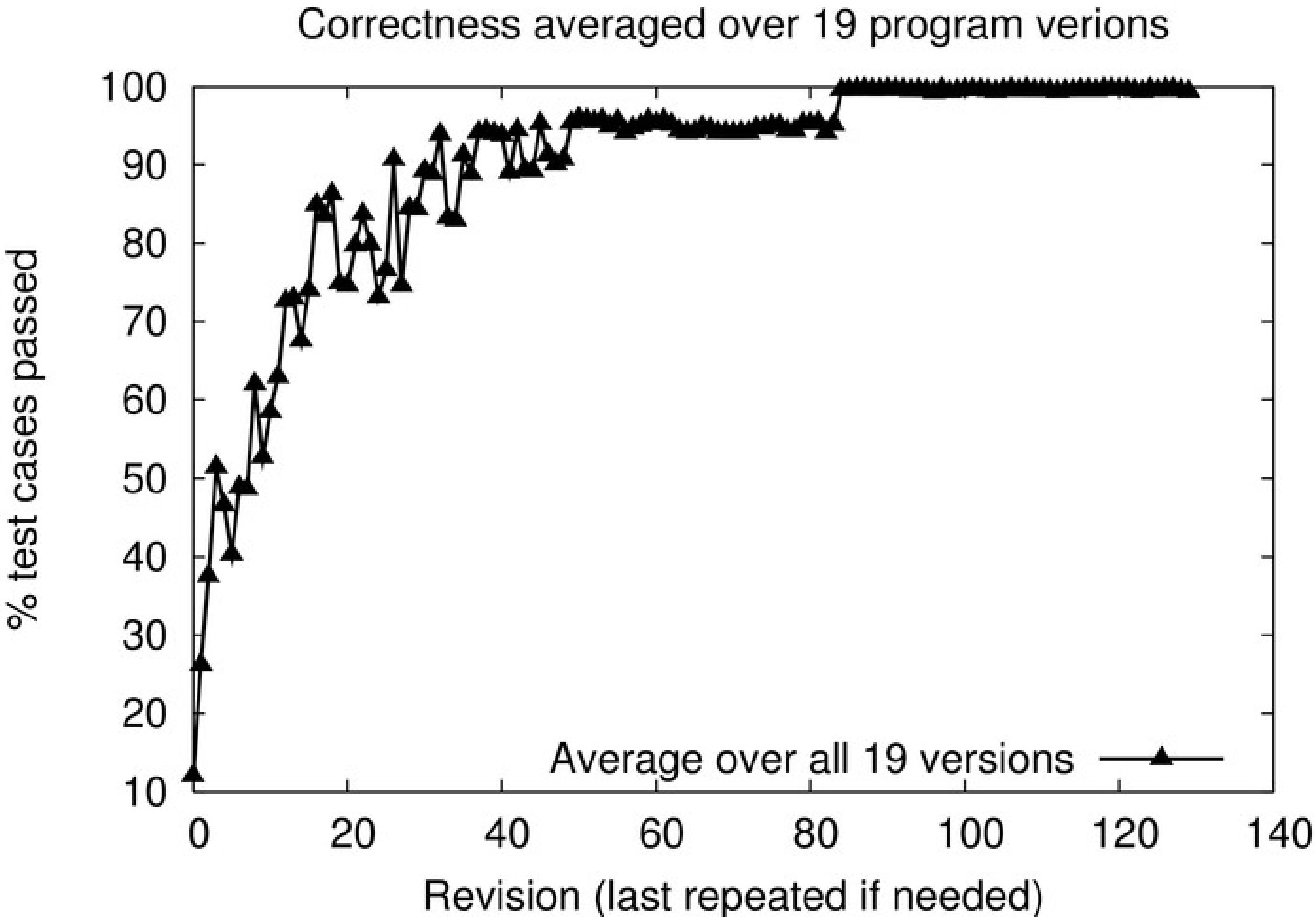


- FN = program wrong, contract silent

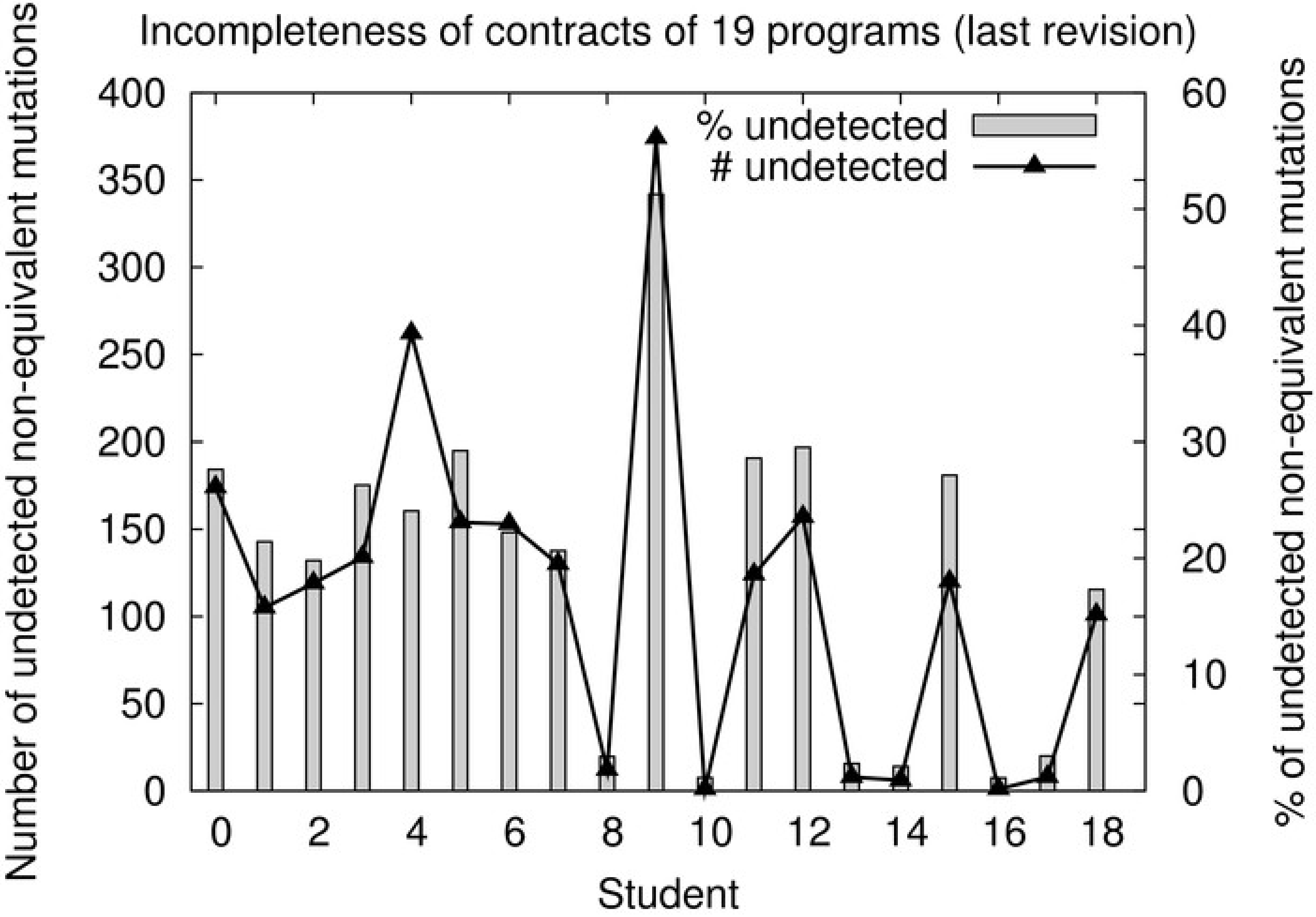
False positive all revisions



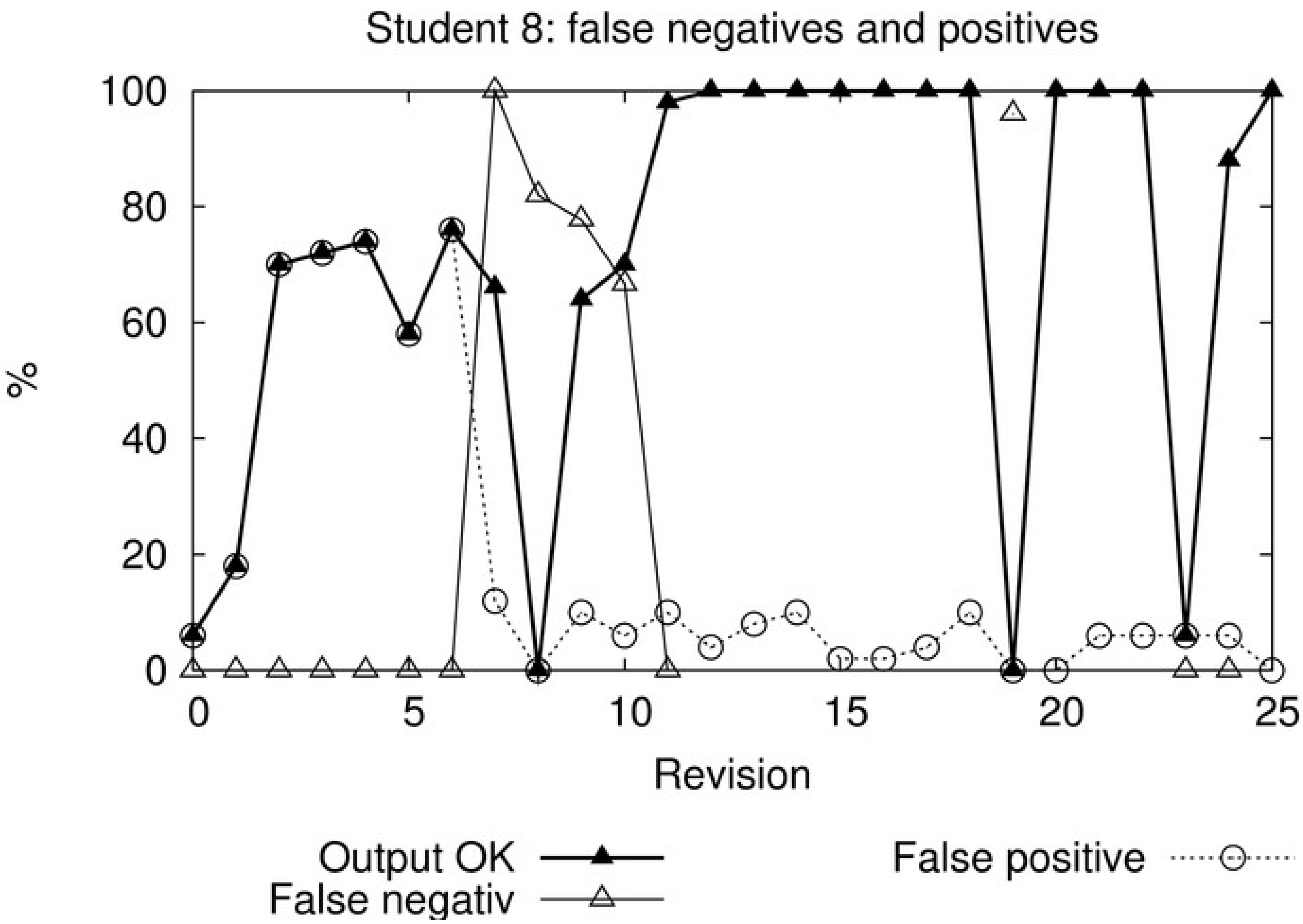
Correctness all revisions



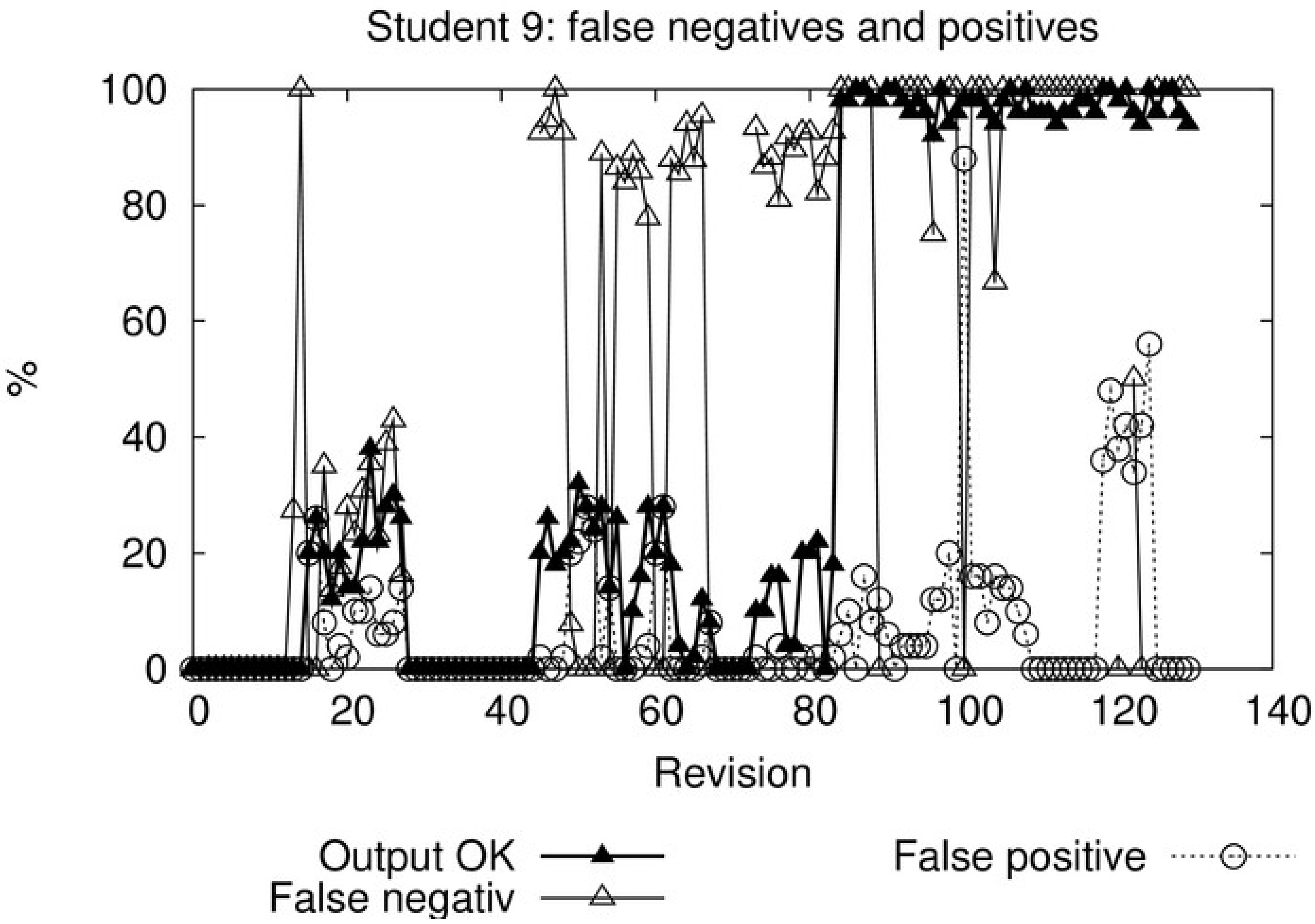
Contract (In)Completeness (LB!)



FN and FP student 8



FN and FP student 9



Future work

- compute upper bound and non-equivalent mutants for student programs
- parallelize contract evaluation to speed up self-checks

Summary

- writing correct self-checks is non-trivial
- upper/lower bound on completeness varies widely
- tools for developing sound/complete self-checks are needed