Automatic Repair of Real Bugs in Java: A Large-Scale Experiment on the Defects4J Dataset

Matias Martinez, Thomas Durieux, Romain Sommerard, Jifeng Xuan, Martin Monperrus

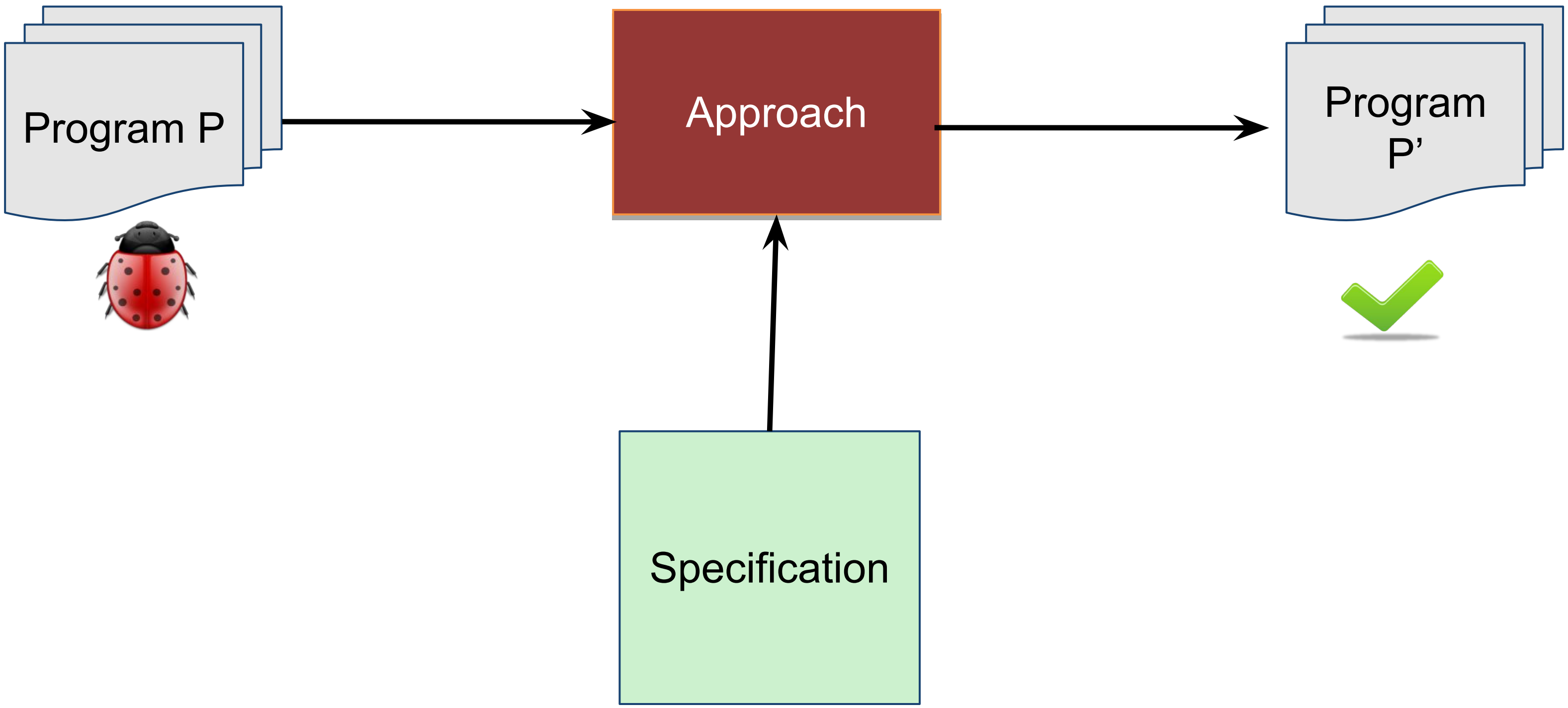
Automatic Software Repair

Automatic software repair is the transformation of an **unacceptable** behavior of a program execution into an **acceptable** one according to a **specification**.

The specification is a set of expected behaviors defined by

- natural language documents
- formal logic formulas
- test suites

Automatic Software Repair



Automatic software repair

Test-suite based repair approaches:

All test passes → no bug anymore
One or more failing test cases → bug

Automatic Software Repair

Type of automatic software repair:

- **state repair**: modifying the program state during the execution

- **behavioral repair**: modifying the program code

RQ: Is a test-suite alone enough to drive automatic repair of real applications?

Experimentation

What do we need for this experiment?

1. An appropriate bug dataset
2. Implementation of automatic software repair approaches

An appropriate bug dataset?

Requirements:

- Large dataset
- Real bugs
- With test-suites
- Independent
- Easy to use

Defect4j

Defined by R. Just et al.
224 real bugs from 4 large Java projects

Project	#Bugs	Source KLoC	Test KLoC	#Test cases
Commons Lang	65	22	6	2,245
JFreeChart	26	96	50	2,205
Commons Math	106	85	19	3,602
Joda-Time	27	28	53	4,130
Total	**224**	**231**	**128**	**12,182**

An appropriate implementation?

Requirements:

- Available
- Target Java bugs
- Test-based approach
- Easy to use

Repair Approaches

Selected approaches for our experiment:

- **GenProg** (Le Goues, Weimer et al.) Genetic programming, synthesizes patches by reusing existing code
- **Kali** (Qi et al.) removes statements and blocks, adds return statements
- **Nopol** (Xuan, Monperrus et al.) uses SMT to synthesize *if conditions* and *missing preconditions*

GenProg Approach

GenProg by *Le Goues, Weimer et al.*

A genetic programming approach
Reuse existing code to synthesize the patches

Patch Operators:

- Delete statement
- Replace statement
- Move Statement
- Copy Statement

Implemented in jGenProg

Kali Approach

Kali by *Qi et al*.

Patch by removing code

Patch Operators:

- Remove statements
- Remove block
- Add “return” statements

Implemented in jKali

Nopol Approach

Nopol by Xuan, Monperrus et al.

Patch conditions and missing pre-condition

Use angelic value to determine the value of the condition

Synthesize patches with a SMT solver

Natively implemented in Java

Methodology

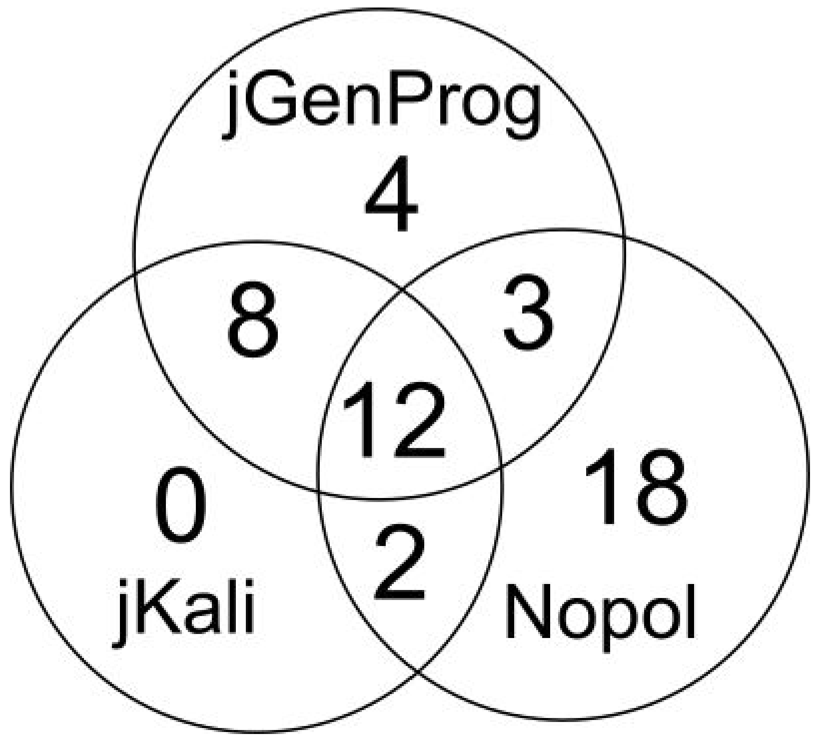
1. Execute jGenProg, Nopol, jKali on all bugs

2. Manual analysis for correctness

3. Open-science: full experimentation data and scripts
   (https://github.com/Spirals-Team/defects4j-repair)

Results

17 days of computational time
84 patches
47/224 (21%) bugs repaired



Test-suite adequate patches

The test-suite adequate patches only:

- pass the failing test cases
- pass the other test cases

The patches may overfit the test-suite

Methodology of manual analysis

Methodology:

- Classify in 3 categories: correct, incorrect, unknow
- Analysis done in parallel by 2 people
- Discussion to reach a consensus

Classification:

- Human patch = Generated patch → correct
- Obvious invalid → invalid
- Otherwise compare the execution flow and the state in a debugger

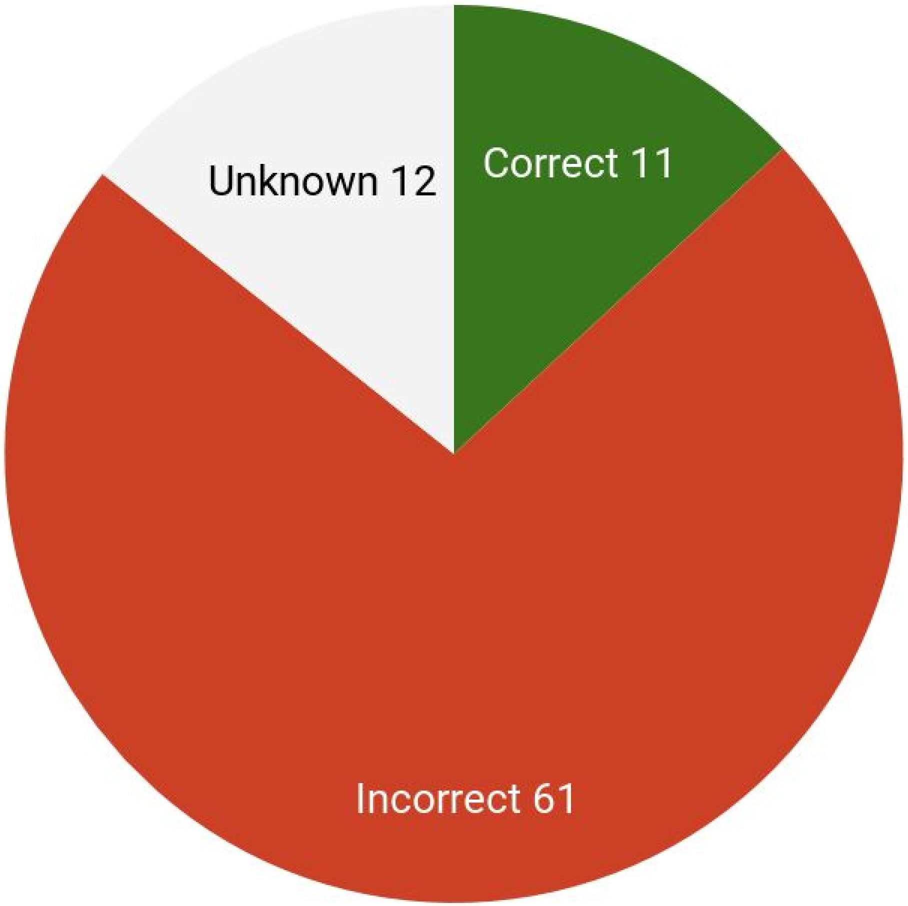
Results of manual analysis

Manually analyzed all patches

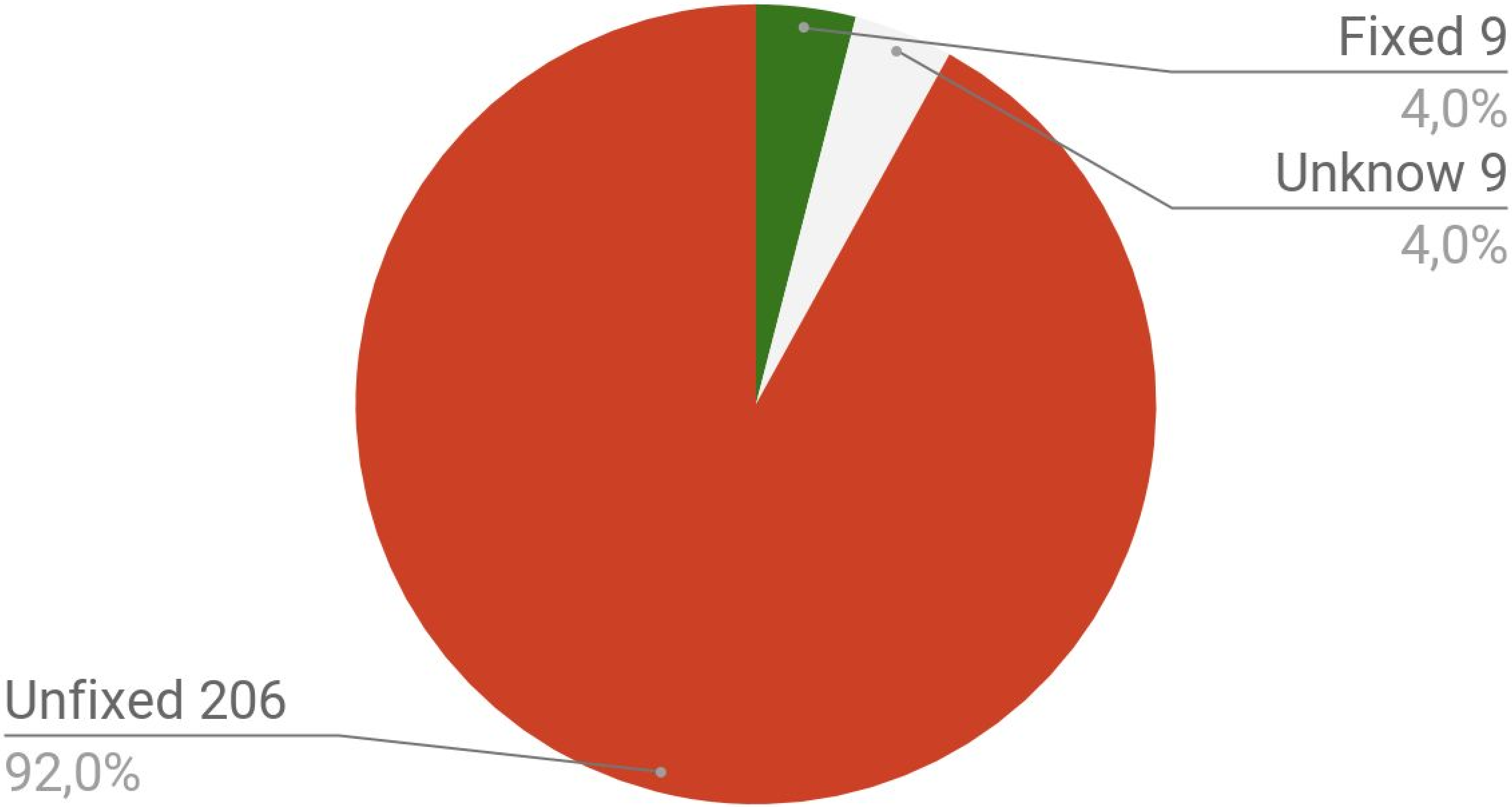
11 patches considered as correct:

- jGenProg 5
- jKali 1
- Nopol 5

Correctness of patches



Correctly fixed bugs



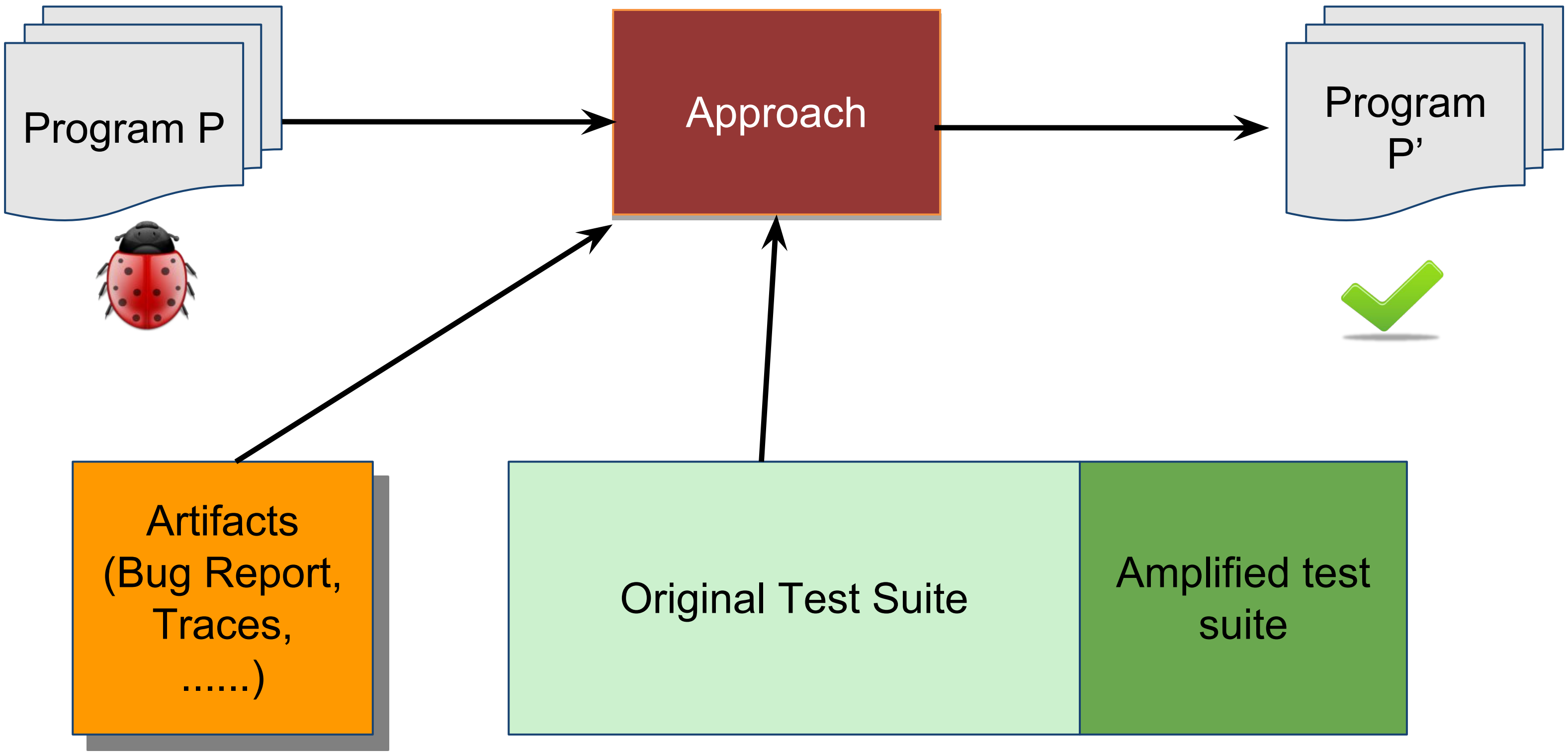
Lessons learned

Need to improve manual analysis:

- Time consuming
- Subject to errors
- Do not scale

Test-suite is not enough to drive alone the patch generation

Automatic software repair now



Novel repair systems since 2015

Prophet (ICSE’16): Ranking candidate patches based on human written patches

HDRepair (ICSME’16): Mining patch patterns from project history

ACS (ICSE’17): Extracting method specification from JavaDoc and failing test case

Genesis (ESEC/FSE’17): Learning code transformation from successful patches

Amplified test-suite

J. Yu et al. (2017): Test case generation using Evosuite for amplifying the test suite .

Xinyuan Liu et al. (2017): generate new test inputs to enhance the test suites and use their behavior similarity to determine patch correctness

Jinqiu Yang et al. (FSE’17): Opad (Overfitted PAtch Detection). Opad uses fuzz testing to generate new test cases, and employs two test oracles (crash and memory-safety) to enhance validity checking of automatically-generated patches.

Conclusion

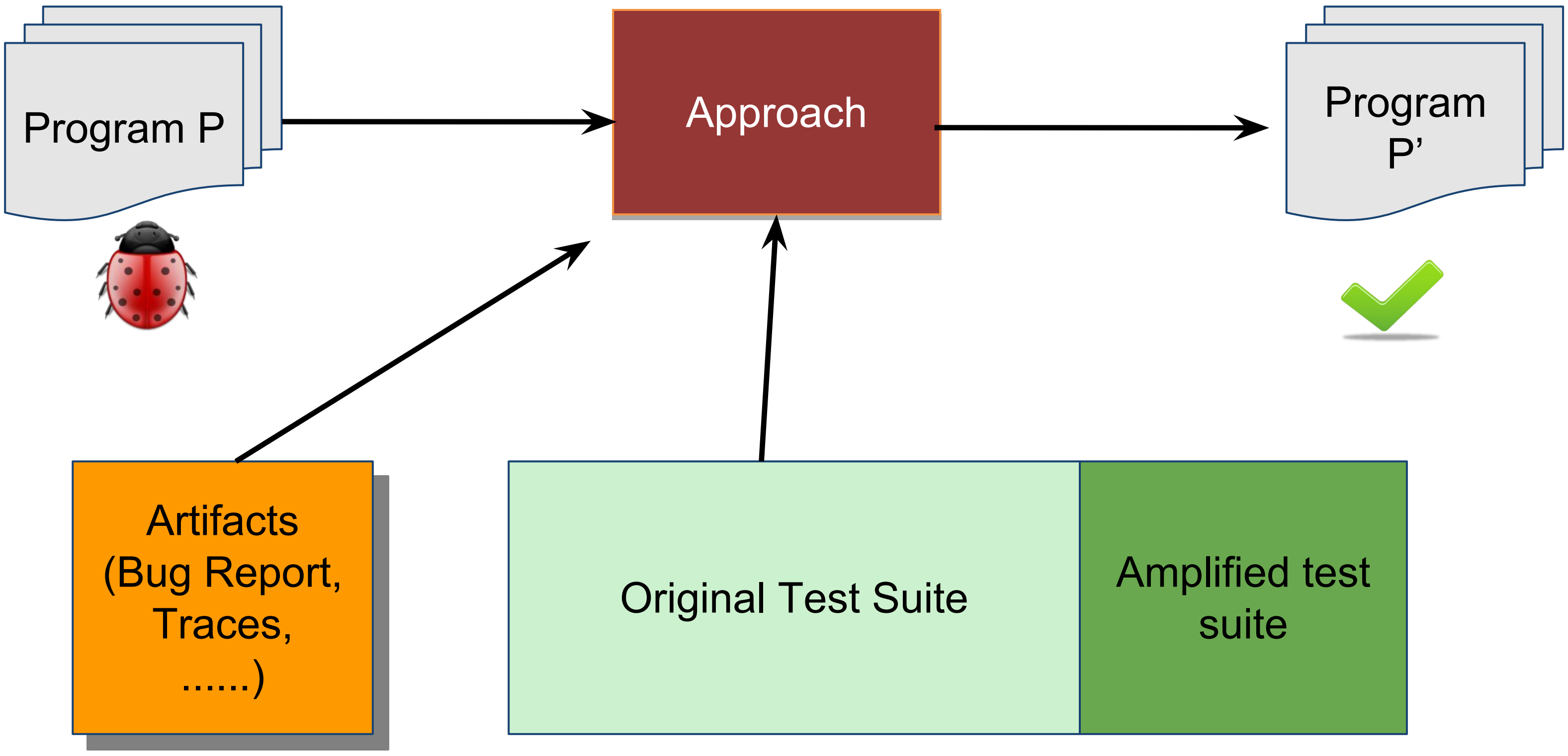
Is a test-suite alone enough to drive automatic repair of real applications?

No, the community is moving in two directions:

1) to automatically improve test-suites

2) to exploit external artifacts to guide the patch search or to filter the generated patches

Questions?


Program P
Approach
Program
P’
Artifacts
(Bug Report,
Traces,
……)
Original Test Suite
Amplified test
suite

Example- Kali

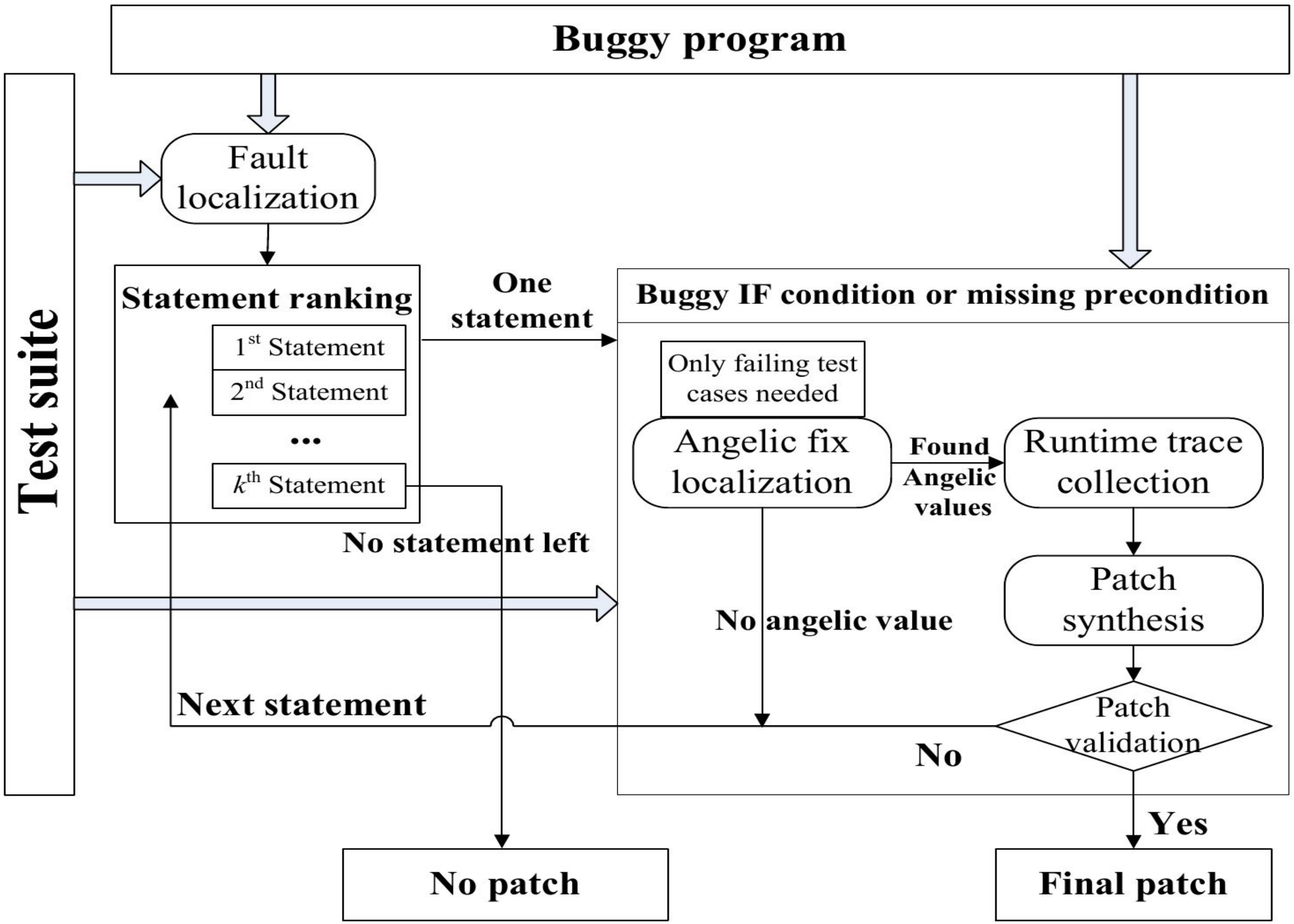
```
public class DiscreteDistribution....
T[] sample(int sampleSize) {
   if (sampleSize <= 0) {
    throw new NotStrictlyPositiveException([...]);
   }
   // MANUAL FIX:
   // Object[] out = new Object[sampleSize];
   T[] out = (T[]) Array.newInstance(singletons.get(0).getClass(),
sampleSize);
   for (int i = 0; i < sampleSize; i++) {
     // KALI FIX: removing the following line
     out[i] = sample();
   }
 return out;
}
```

```
public void testIssue942() {
   List<Pair<Object,Double>> list = new ArrayList<Pair<Object, Double>>();
   list.add(new Pair<Object, Double>(new Object() {}, new Double(0)));
   list.add(new Pair<Object, Double>(new Object() {}, new Double(1)));
   Assert.assertEquals(1, new
DiscreteDistribution<Object>(list).sample(1).length);
}
```

Example- Nopol

```
void stop() {
  if (this.runningState != STATE_RUNNING
          && this.runningState != STATE_SUSPENDED) {
                throw new IllegalStateException(...);
}
// MANUAL FIX:
// if (this.runningState == STATE_RUNNING)
// NOPOL FIX:
// if (stopTime < StopWatch.STATE_RUNNING)
          stopTime = System.currentTimeMillis();
          this.runningState = STATE_STOPPED;
}
```

Nopol



Buggy program
Test suite
Fault localization
Statement ranking
1st Statement
2nd Statement
…
kth Statement
One statement
Buggy IF condition or missing precondition
Only failing test cases needed
Angelic fix localization
Found Angelic values
Runtime trace collection
Patch synthesis
Patch validation
No angelic value
No statement left
Next statement
No
Yes
No patch
Final patch

References

- J. H. Perkins, G. Sullivan, W.-F. Wong, Y. Zibin, M. D. Ernst, M. Rinard, S. Kim, S. Larsen, S. Amarasinghe, J. Bachrach,M. Carbin, C. Pacheco, F. Sherwood, and S. Sidiroglou. Automatically Patching Errors in Deployed Software. In Proceedings of the 22nd Symposium on Operating Systems Principles (SOSP),2009
- Z. Qi, F. Long, S. Achour, and M. Rinard. An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In Proceedings of ISSTA. ACM, 2015
- C. Le Goues, T. Nguyen, S. Forrest, and W. Weimer. Genprog: A generic method for automatic software repair. Software Engineering, IEEE Transactions on, 38(1):54–72, 2012
- D. Kim, J. Nam, J. Song, and S. Kim. Automatic patch generation learned from human-written patches. In Proceedings of the 2013 International Conference on Software Engineering, pages 802–811, 2013
- R. Abreu, P. Zoeteweij, and A. J. Van Gemund. On theaccuracy of spectrum-based fault localization. In Testing: Academic and Industrial Conference Practice and ResResearch Techniques MUTATION,2007, pages 89–98. IEEE, 2007
- V. Dallmeier, A. Zeller, and B. Meyer. Generating fixes from object behavior anomalies. In Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering,pages 550–554. IEEE Computer Society, 2009.
- V. Debroy and W. E. Wong. Using mutation to automatically suggest fixes for faulty programs. In Proceedings of the 2010
- Third International Conference on Software Testing, Verification and Validation, ICST ’10, pages 65–74, Washington, DC, USA, 2010.IEEE Computer Society.
- F. DeMarco, J. Xuan, D. Le Berre, and M. Monperrus. Automatic repair of buggy if conditions and missing preconditions with smt. In Proceedings of the 6th International Workshop on Constraints in Software Testing, Verification, and Analysis, pages 30–39. ACM, 2014.
- R. Just, D. Jalali, and M. D. Ernst. Defects4J: A database of existing faults to enable controlled testing studies for Java programs. In Proceedings of the International Symposium on Software Testing and Analysis (ISSTA), pages 437–440, July 23–25 2014.