# CS 563:
# Software Maintenance and Evolution

# Automated Program Repair

Oregon State University, Spring 2024

# Motivation

**606 SOFTWARE FAILURES**

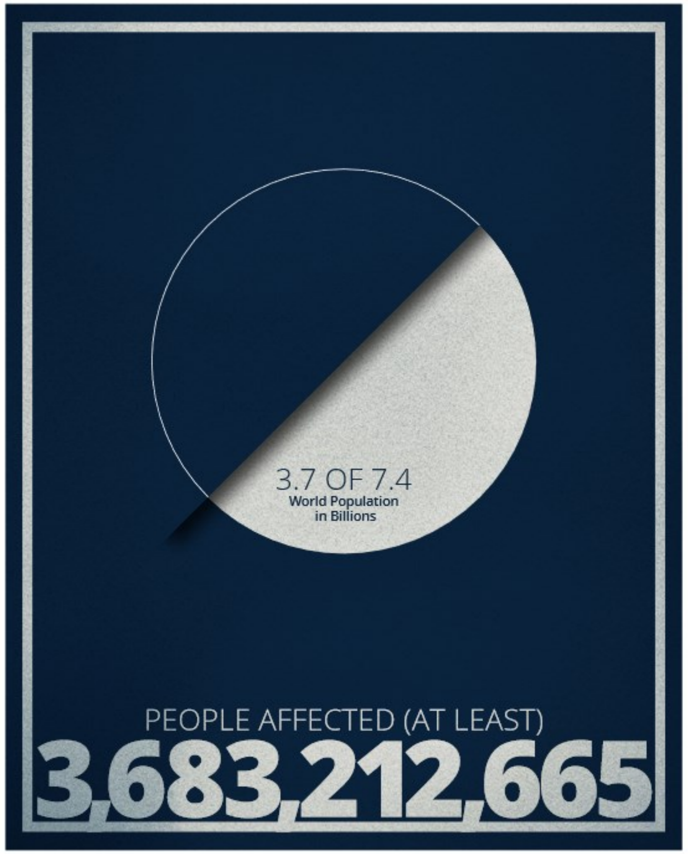
**1177 NEWS STORIES**

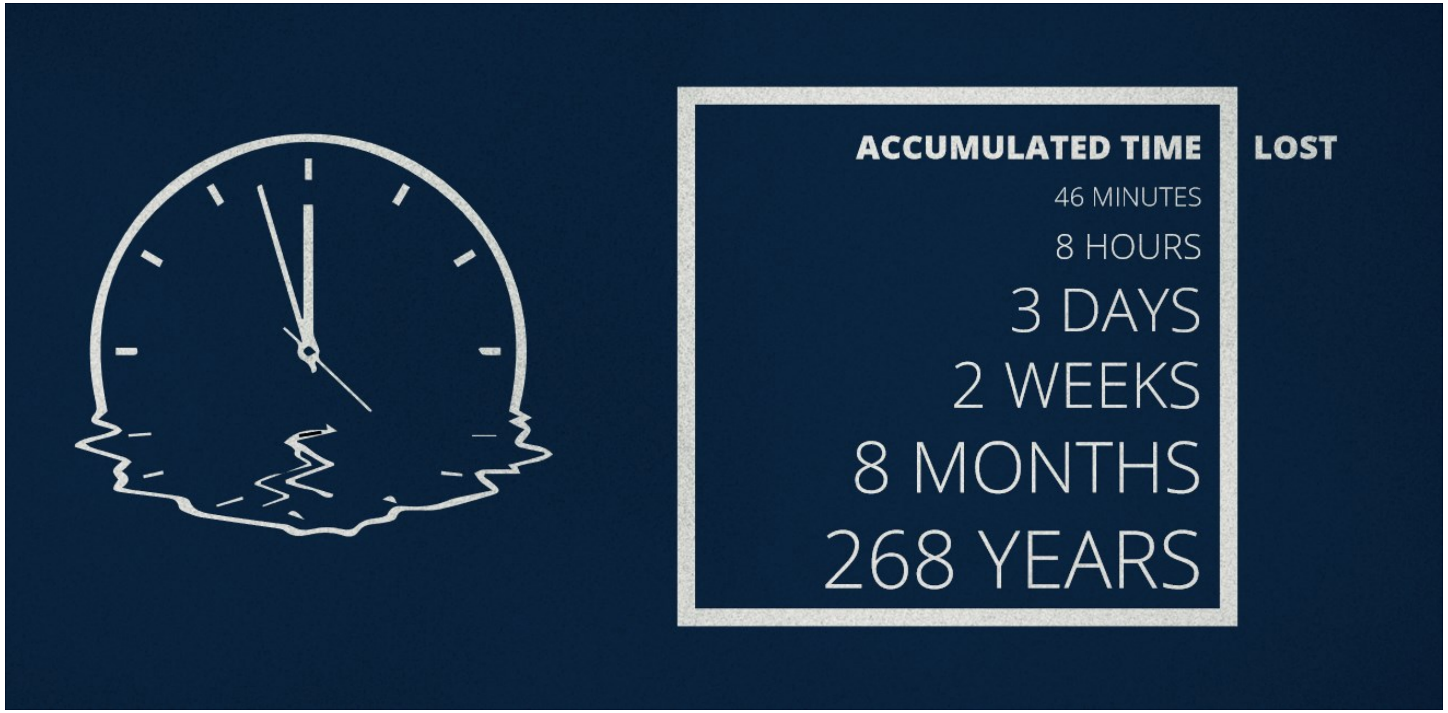
**314 COMPANIES**

**LO$$E$ FROM SOFTWARE FAILURES (USD)**

**1,715,430,778,504**

ONETRILLIONSEVENHUNDREDFIFTEENBILLIONFOURHUNDREDTHIRTYMILLIONSEVENHUNDREDSEVENTY-EIGHTTHOUSANDFIVEHUNDREDFOUR



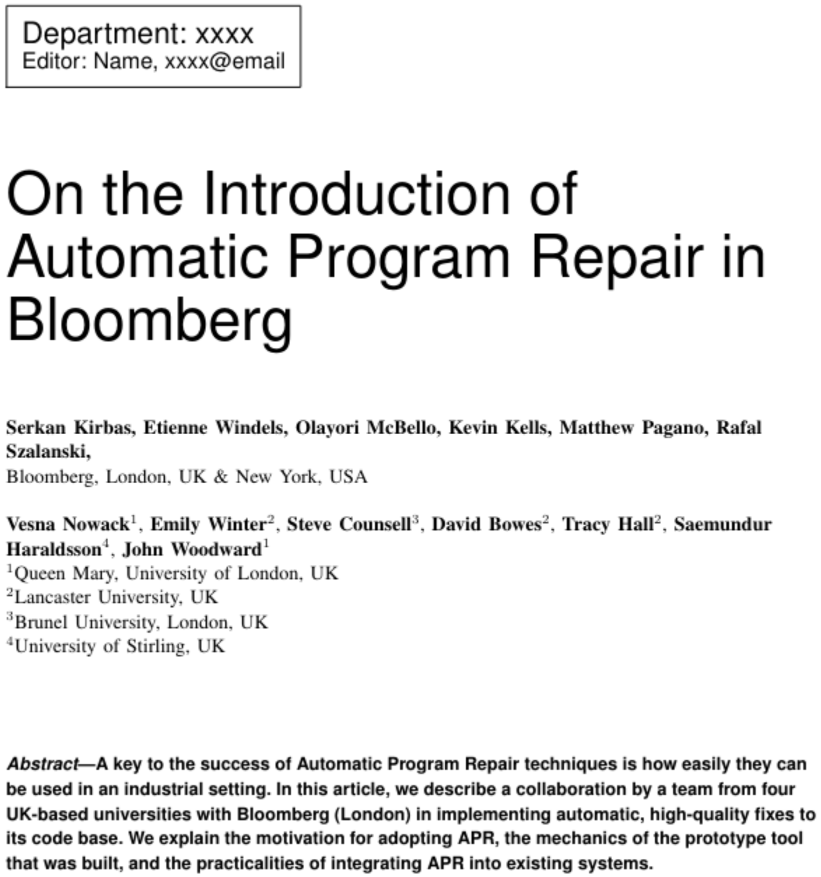


Source: Software Fail Watch: 5th Edition, 2018.
Available at: https://www.tricentis.com/software-fail-watch

*“Legacy software testing tools that were developed two decades ago—some which are still heavily in use today—were never intended to support high levels of quality in today's rapid release cycles. It’s time for a change.”*

Source: Software Fail Watch: 5th Edition, 2018. Available at: https://www.tricentis.com/software-fail-watch

# Automatic Program Repair



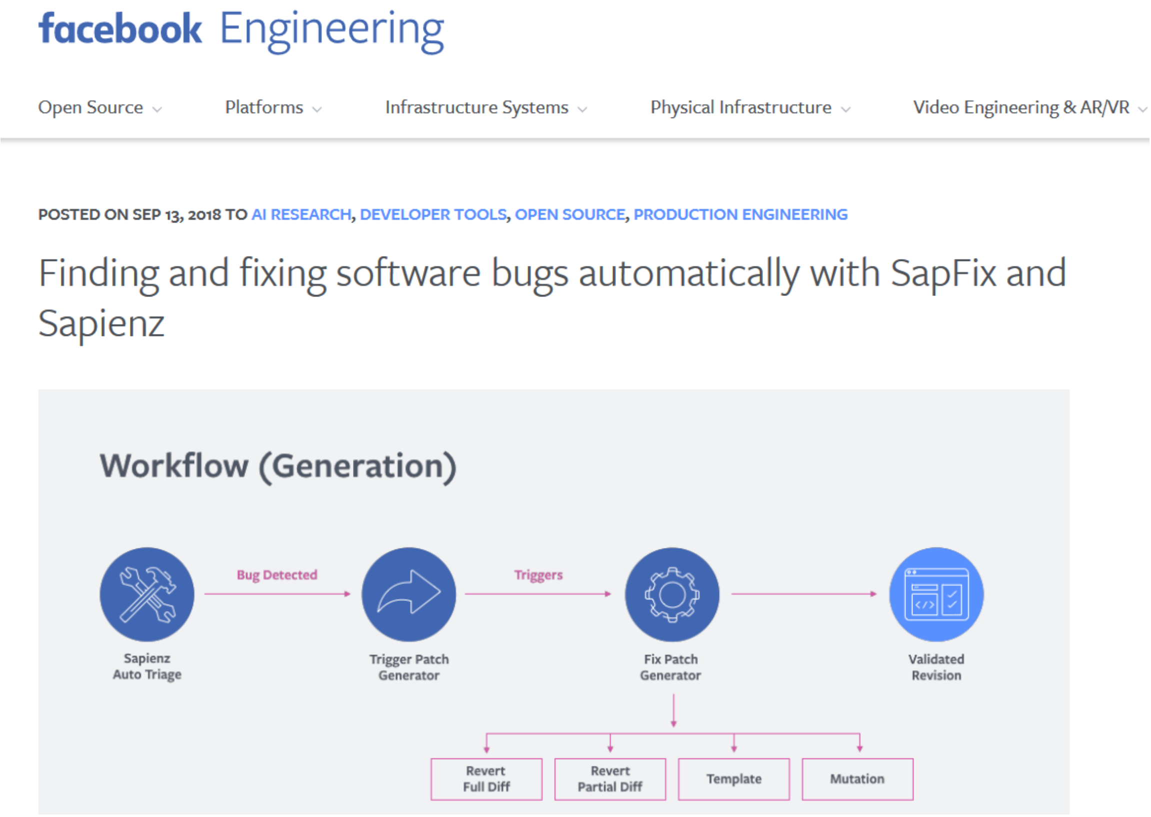
Department: xxxx
Editor: Name, xxxx@email

On the Introduction of Automatic Program Repair in Bloomberg

Serkan Kirbas, Etienne Windels, Olayori McBello, Kevin Kells, Matthew Pagano, Rafal Szalanski,
Bloomberg, London, UK & New York, USA

Vesna Nowack[1], Emily Winter[2], Steve Counsell[3], David Bowes[2], Tracy Hall[2], Saemundur Haraldsson[4], John Woodward[1]
[1]Queen Mary, University of London, UK
[2]Lancaster University, UK
[3]Brunel University, London, UK
[4]University of Stirling, UK

*Abstract*—A key to the success of Automatic Program Repair techniques is how easily they can be used in an industrial setting. In this article, we describe a collaboration by a team from four UK-based universities with Bloomberg (London) in implementing automatic, high-quality fixes to its code base. We explain the motivation for adopting APR, the mechanics of the prototype tool that was built, and the practicalities of integrating APR into existing systems.



facebook Engineering

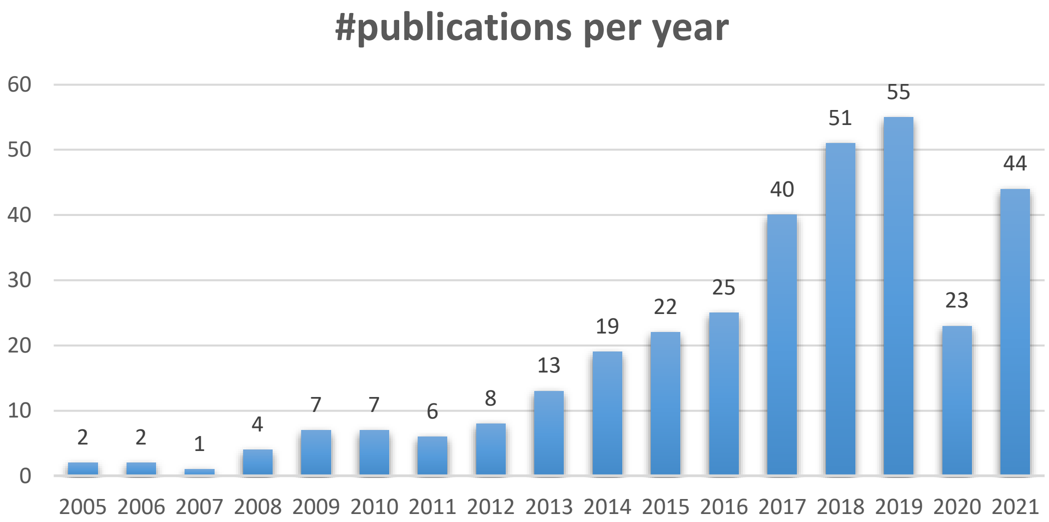
Open Source ⌄ Platforms ⌄ Infrastructure Systems ⌄ Physical Infrastructure ⌄ Video Engineering & AR/VR ⌄

POSTED ON SEP 13, 2018 TO AI RESEARCH, DEVELOPER TOOLS, OPEN SOURCE, PRODUCTION ENGINEERING

Finding and fixing software bugs automatically with SapFix and Sapienz

Workflow (Generation)

# State of the Art



ssFix
kPAR
SPR
GenProg
DeepFix
Refazer
SapFix
ACS
SOSRepair
NPEFix
Genesis
GetaFix
AllRepair
SequenceR
SketchFix
Repairnator
Clara
Kali
LSRepair
HistoricalFix
SearchRepair
CoderAssist
AVATAR
Nopol
QACrashFix
RSRepair
SimFix
JAID
NoFAQ
LeakFix
AutoFix
MintHint
Footpatch
SarfgenBugZoo
Prophet
Astor
CapGen
SOFix
SemFix
JFix
Angelix

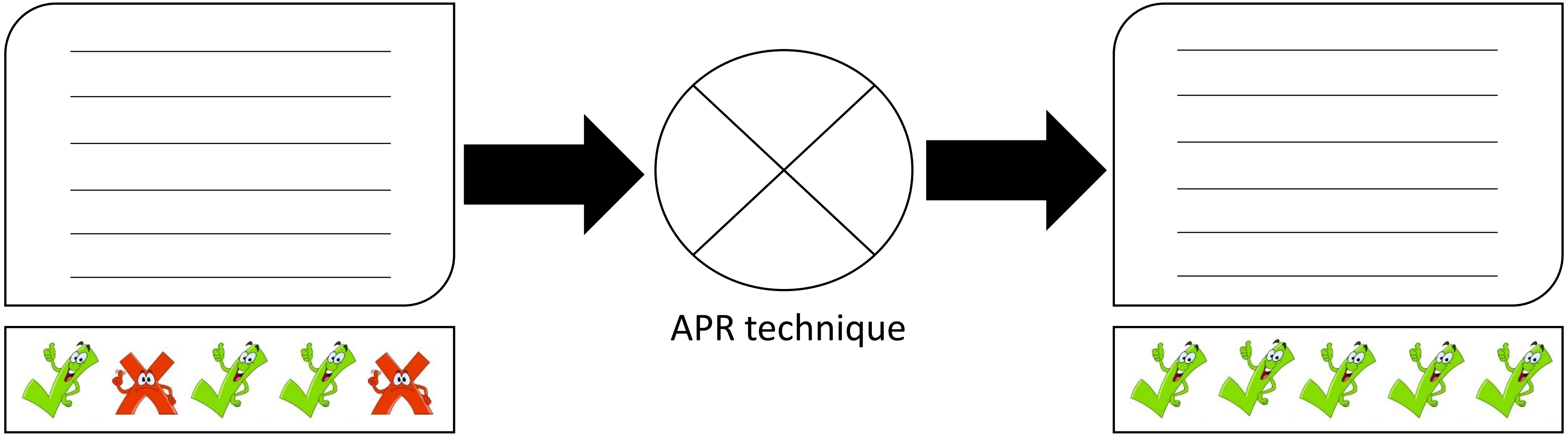
40+ repair techniques

Codeflaws
DroixBench
IntroClass
ManyBugs
QuixBugs
BugSwarm
Defects4J
Bugs.jar
Bears

10+ defect benchmarks

http://program-repair.org

# Automatic Program Repair

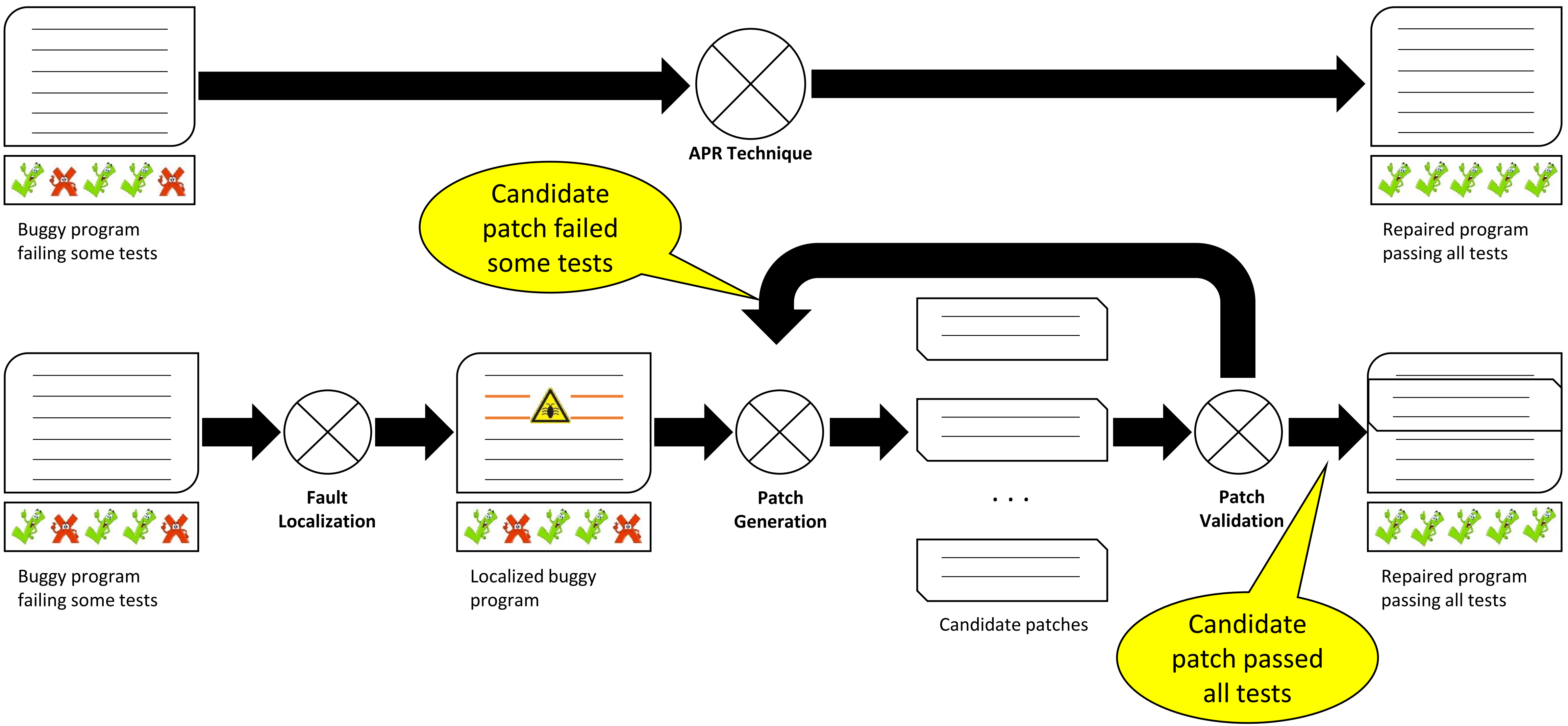


APR technique

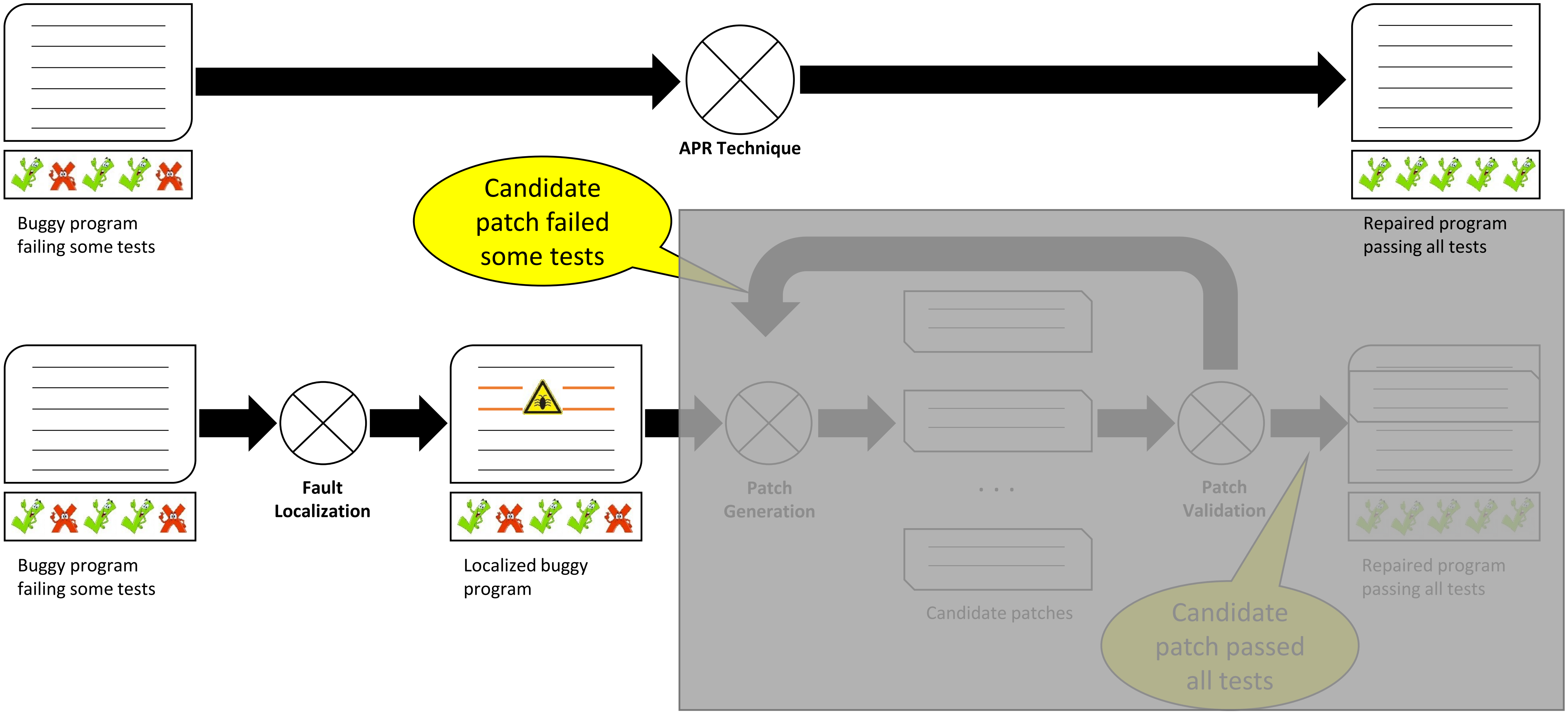
Buggy program
failing some tests

Repaired program
passing all tests

# Program Repair Process


APR Technique
Buggy program failing some tests
Repaired program passing all tests
Candidate patch failed some tests
Fault Localization
Patch Generation
Patch Validation
. . .
Buggy program failing some tests
Localized buggy program
Candidate patches
Repaired program passing all tests
Candidate patch passed all tests

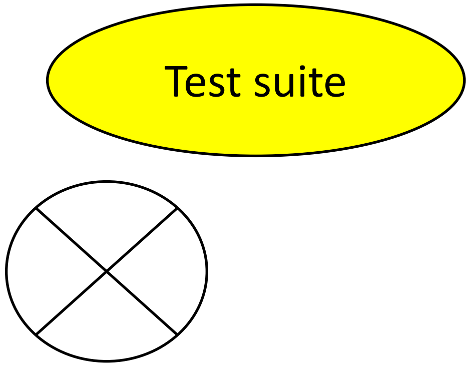
# Program Repair Process


APR Technique
Buggy program failing some tests
Repaired program passing all tests
Candidate patch failed some tests
Buggy program failing some tests
Fault Localization
Localized buggy program
Patch Generation
. . .
Candidate patches
Patch Validation
Repaired program passing all tests
Candidate patch passed all tests

# Fault Localization

**Fault Localization:** *Automatically determining program elements (such as statements or methods) that are defective and cause software failure.*

# Fault Localization Techniques

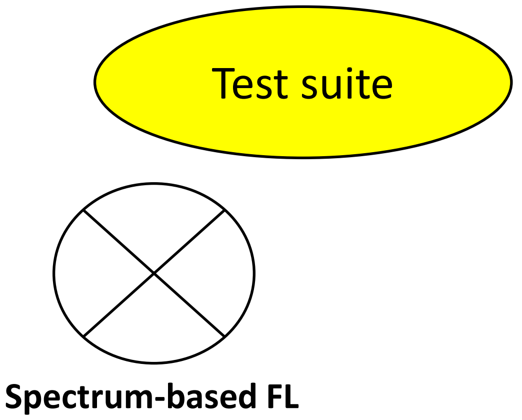


Test suite

**Spectrum-based FL**

Effect of passing and failing tests on *program element*

# Fault Localization Techniques



Effect of passing and failing tests on *program element*

**Source program**

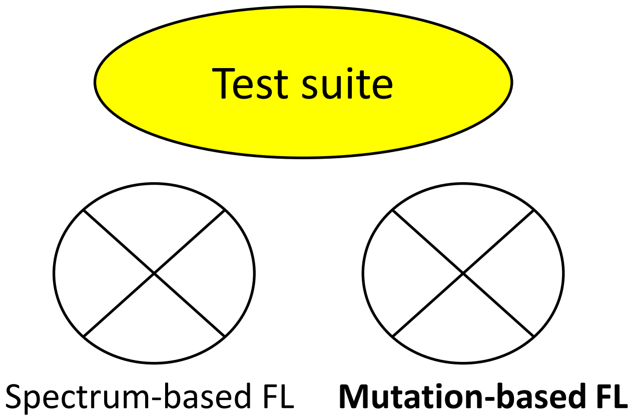
code statement 1
code statement 2
…
**code statement i**
…
code statement m

**Test suite**

| | |
|---|---|
| Test 1 | pass |
| Test 2 | fail |
| … | … |
| Test k | pass |
| … | … |
| Test n | fail |

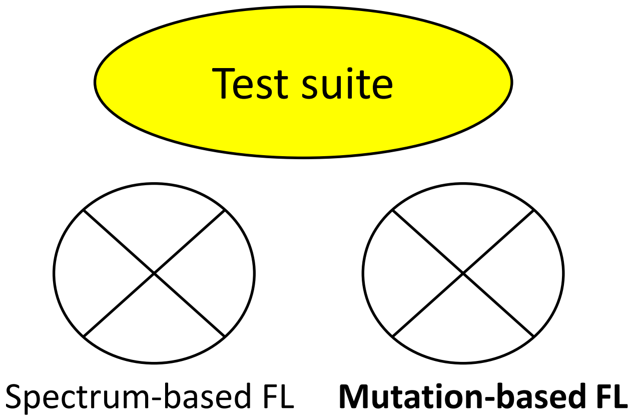
- Count the number of passing and failing tests which execute a code statement I
- **Suspiciousness score(code statement i) = *f* (#passing tests executing stmt i, #failing tests executing stmt i)**

# Fault Localization Techniques



Effect of *program element* on passing and failing tests

# Fault Localization Techniques



Effect of *program element* on passing and failing tests

**Source program**

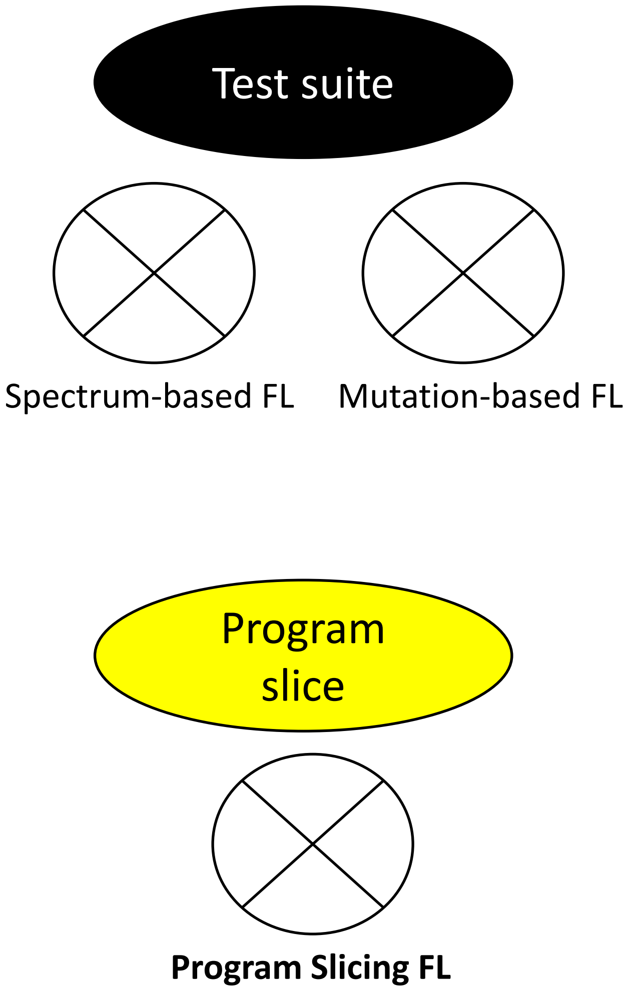
code statement 1
code statement 2
…
**code statement i**
…
code statement m

**Test suite**

| | |
|---|---|
| Test 1 | pass |
| Test 2 | fail |
| … | … |
| Test k | pass |
| … | … |
| Test n | fail |

- Mutate code statement i (for e.g., change ‘a = a + 1; ’ to ‘a = a - 1’)
- count the total number of passing and failing tests
- **Suspiciousness score (code statement i) = *f*(total #passing tests, total #failing tests)**

# Fault Localization Techniques



Test suite

Spectrum-based FL

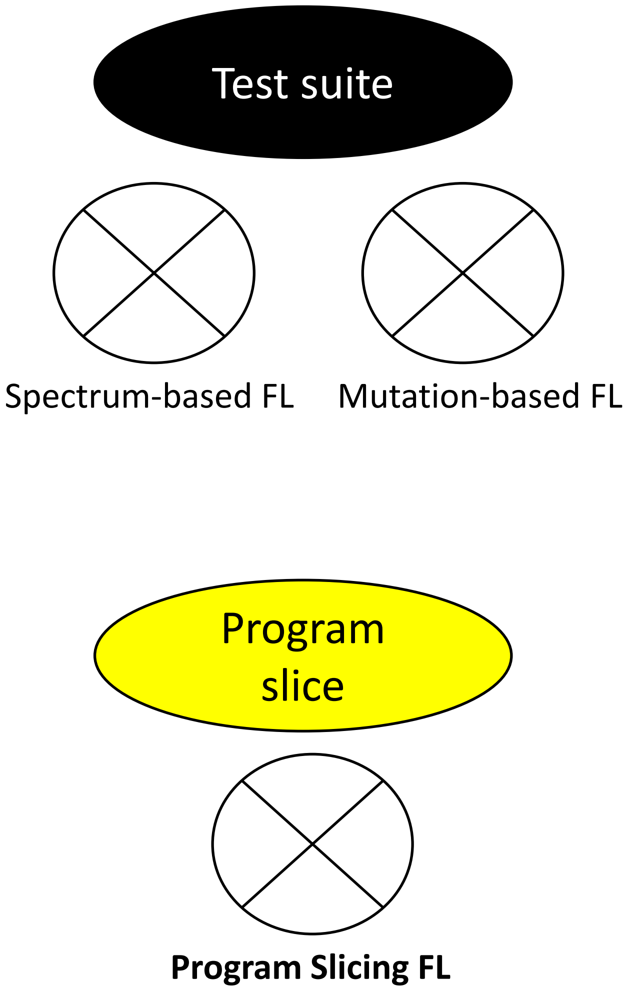
Mutation-based FL

Program slice

**Program Slicing FL**

Reduce program to a minimal form while still maintaining a given behavior exhibited using test suite

# Fault Localization Techniques



Reduce program to a minimal form while still maintaining a given behavior exhibited using test suite

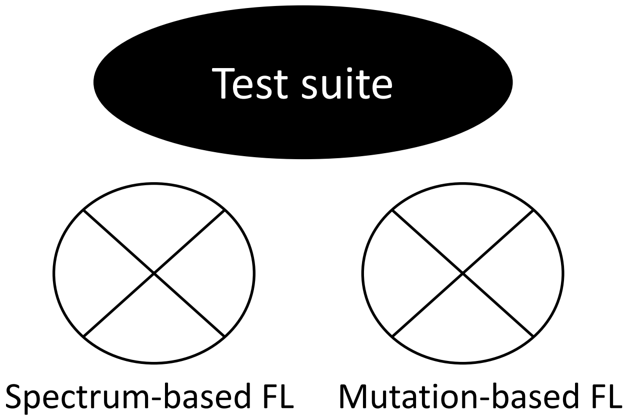
**Source program**

code statement 1
…
code statement i-1
code statement i
code statement i+1
…
Code statement m

**Test suite**

| | |
|---|---|
| Test 1 | pass |
| Test 2 | fail |
| … | … |
| Test k | pass |
| … | … |
| Test n | fail |

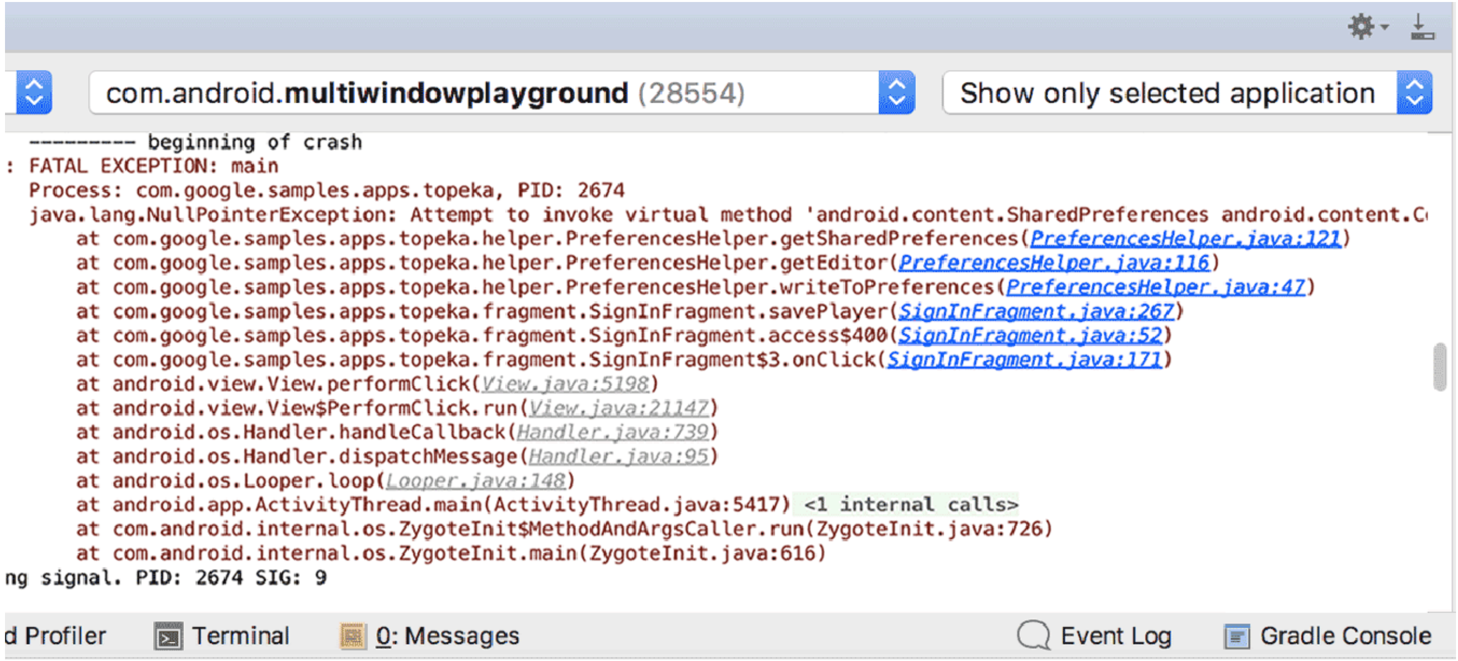
# Fault Localization Techniques



Test suite

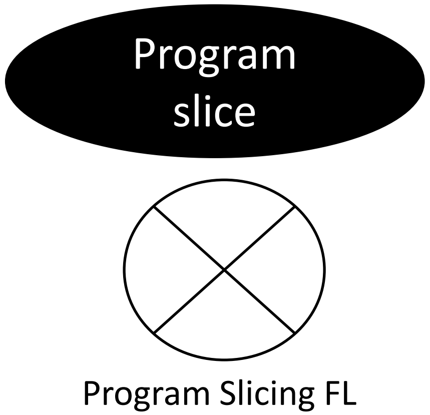
Spectrum-based FL

Mutation-based FL



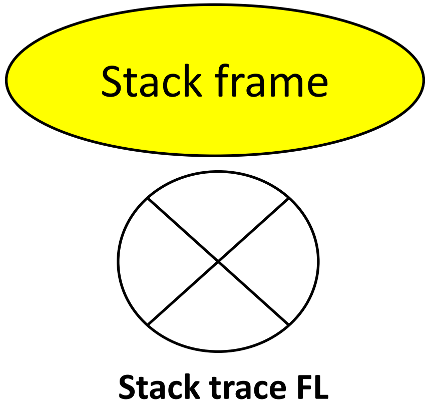
com.android.multiwindowplayground (28554)
Show only selected application
d Profiler
Terminal
0: Messages
Event Log
Gradle Console

```
 --------- beginning of crash
: FATAL EXCEPTION: main
  Process: com.google.samples.apps.topeka, PID: 2674
  java.lang.NullPointerException: Attempt to invoke virtual method 'android.content.SharedPreferences android.content.C
      at com.google.samples.apps.topeka.helper.PreferencesHelper.getSharedPreferences(PreferencesHelper.java:121)
      at com.google.samples.apps.topeka.helper.PreferencesHelper.getEditor(PreferencesHelper.java:116)
      at com.google.samples.apps.topeka.helper.PreferencesHelper.writeToPreferences(PreferencesHelper.java:47)
      at com.google.samples.apps.topeka.fragment.SignInFragment.savePlayer(SignInFragment.java:267)
      at com.google.samples.apps.topeka.fragment.SignInFragment.access$400(SignInFragment.java:52)
      at com.google.samples.apps.topeka.fragment.SignInFragment$3.onClick(SignInFragment.java:171)
      at android.view.View.performClick(View.java:5198)
      at android.view.View$PerformClick.run(View.java:21147)
      at android.os.Handler.handleCallback(Handler.java:739)
      at android.os.Handler.dispatchMessage(Handler.java:95)
      at android.os.Looper.loop(Looper.java:148)
      at android.app.ActivityThread.main(ActivityThread.java:5417) <1 internal calls>
      at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:726)
      at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:616)
ng signal. PID: 2674 SIG: 9
```



Program slice

Program Slicing FL

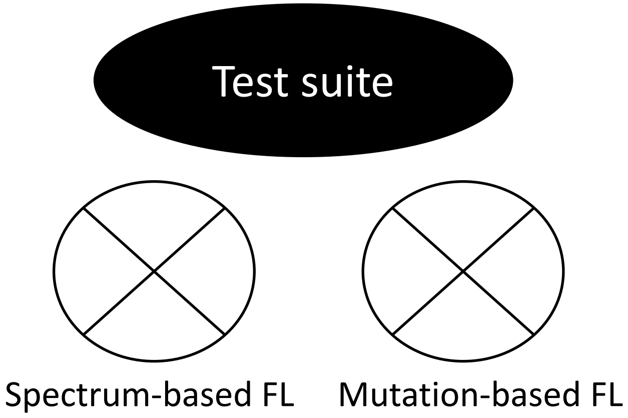


Stack frame

**Stack trace FL**

Use the list of active stack frames during execution of a program

# Fault Localization Techniques



**Source program**

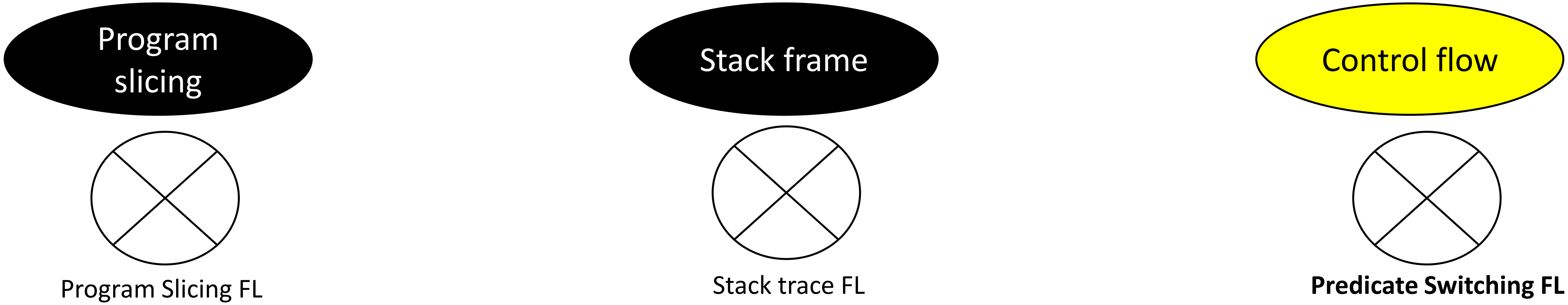
```
If (cond 1){
…
} else if (cond 2) {
…
```

**Test suite**

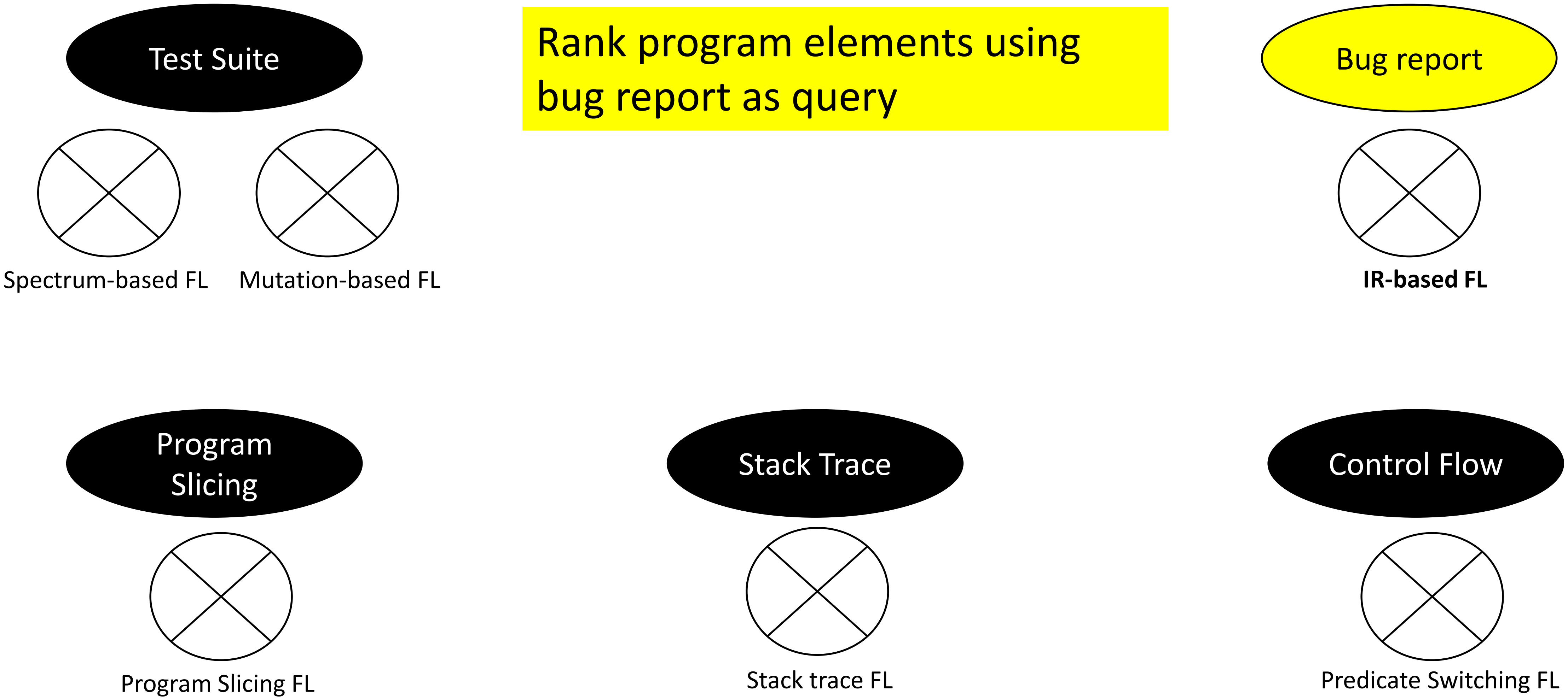
| | |
|---|---|
| Test 1 | pass |
| Test 2 | fail |
| … | … |
| Test n | fail |

Effect of predicate (conditional expression) on failing tests

- Mutate  if condition (for e.g.,  change ‘if (a - b > 0) ’ to ‘ if (a – b < 0)’)
- count the total number of passing and failing tests
- **Suspiciousness score (cond i) = *f*(total #passing tests, total #failing tests)**



# Fault Localization Techniques



Rank program elements using bug report as query

Test Suite

Spectrum-based FL

Mutation-based FL

Bug report

**IR-based FL**

Program Slicing

Program Slicing FL

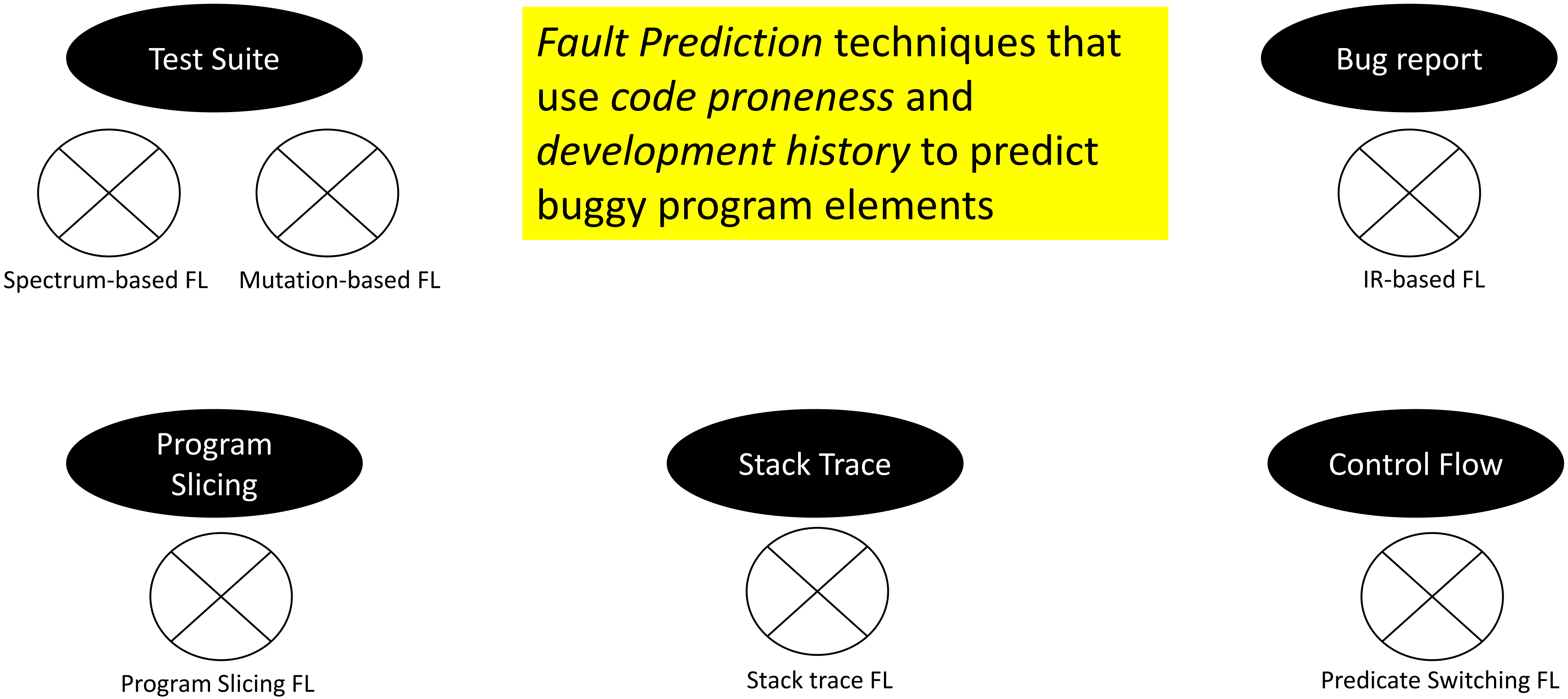
Stack Trace

Stack trace FL

Control Flow

Predicate Switching FL

# Fault Localization Techniques



Test Suite

Spectrum-based FL

Mutation-based FL

*Fault Prediction* techniques that use *code proneness* and *development history* to predict buggy program elements

Bug report

IR-based FL

Program Slicing

Program Slicing FL

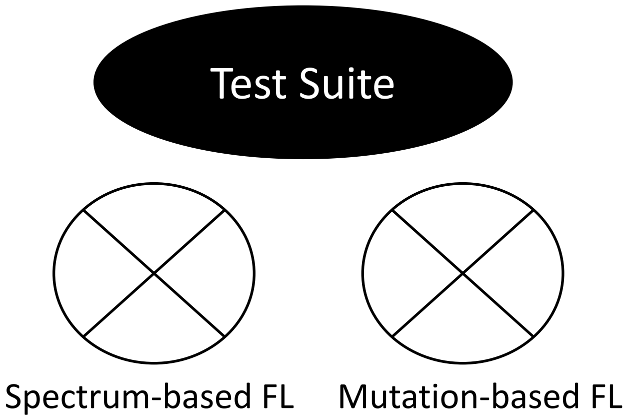
Stack Trace

Stack trace FL

Control Flow

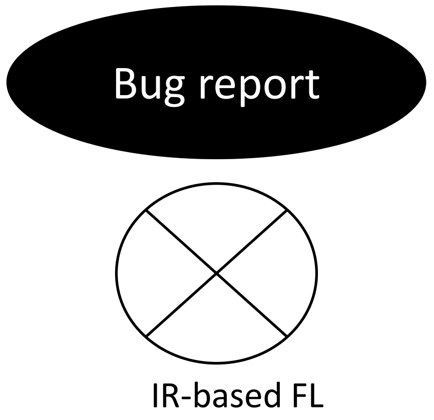
Predicate Switching FL

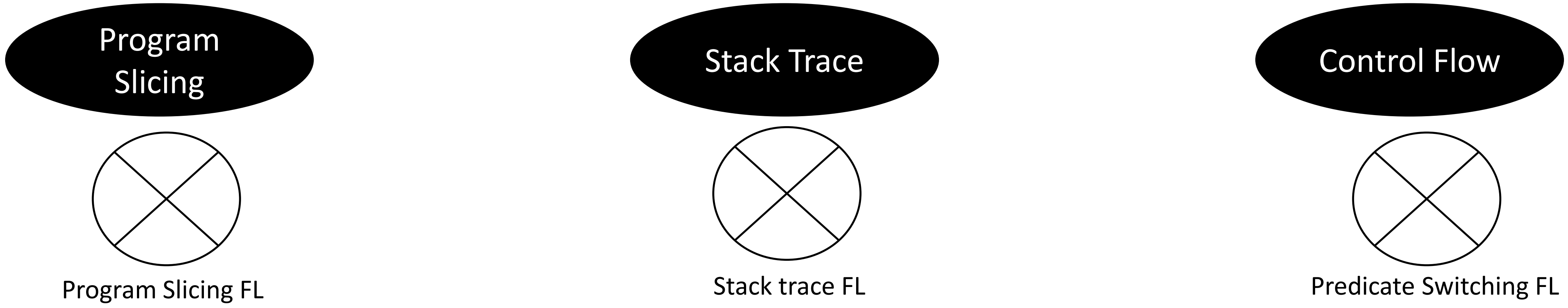
# Fault Localization Techniques



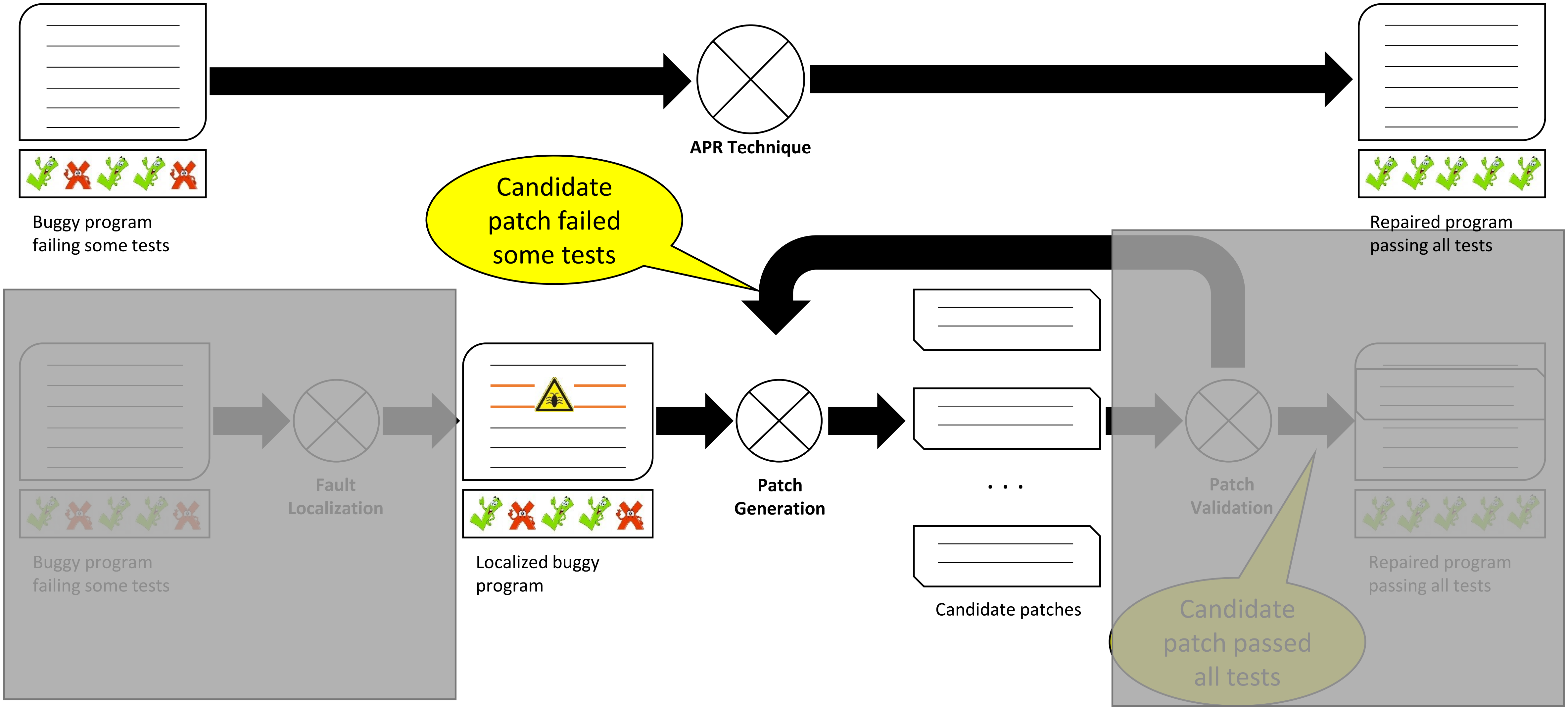
None of these techniques is the best technique!

Automated program repair techniques typically use SBFL





# Program Repair Process

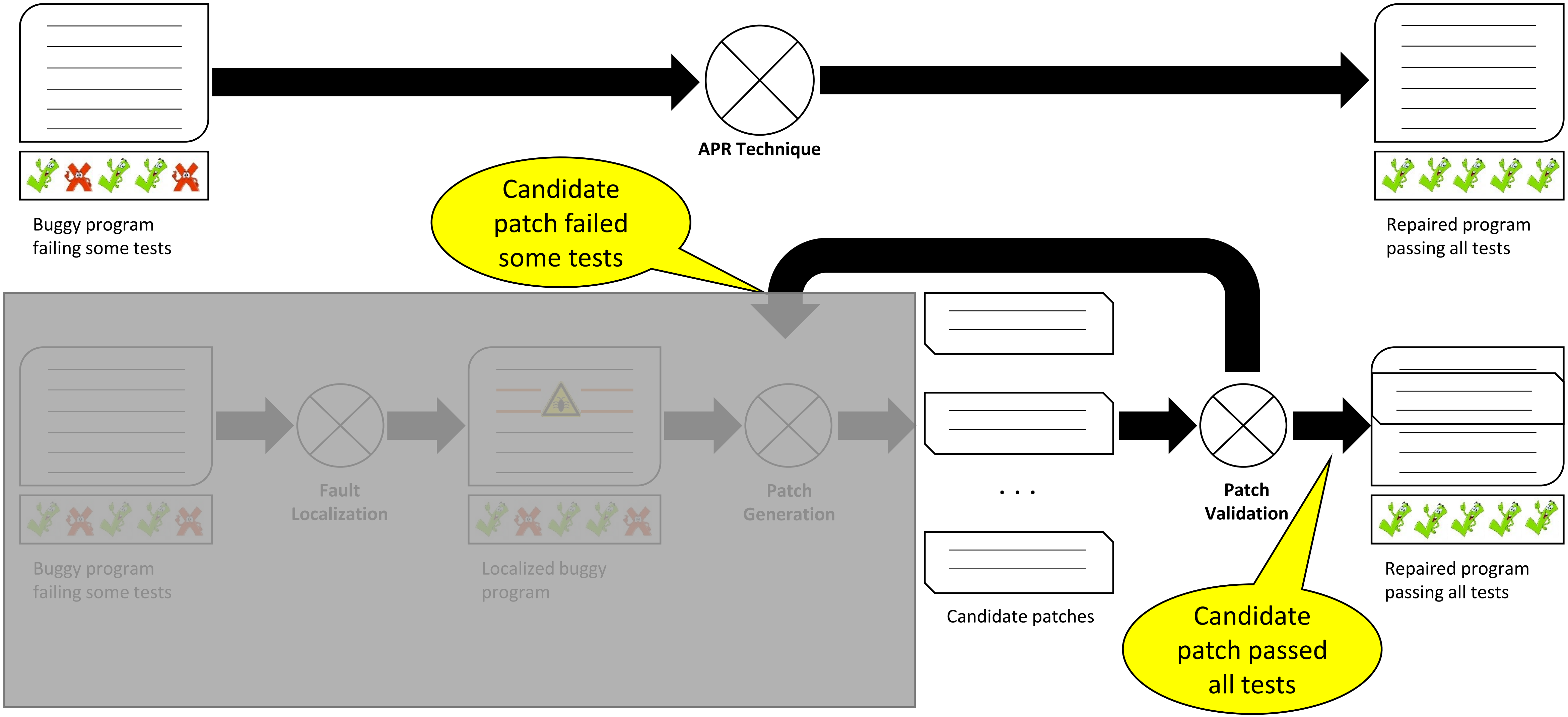

APR Technique
Buggy program failing some tests
Repaired program passing all tests
Candidate patch failed some tests
Buggy program failing some tests
Fault Localization
Localized buggy program
Patch Generation
. . .
Candidate patches
Patch Validation
Repaired program passing all tests
Candidate patch passed all tests

# Patch Generation

Three classes of repair techniques:

- **Generate-and-Validate (G&V) or Heuristics-based**
  - Use search-based software engineering (e.g., find code snippet which is similar to buggy code) to generate patch
  - E.g., GenProg, SimFix, Tbar, Arja
- **Constraint-based or Synthesis-based**
  - Builds formal constraints (e.g. SMT) and uses constraint solvers (e.g. Z3) to generate patch
  - E.g., SearchRepair, SOSRepair, Angelix
- **Learning-based**
  - Use deep learning techniques to *translate* buggy source code into patched source code (similar to translating one natural language to another)
  - E.g., Cure, Recoder, SequenceR

# Program Repair Process


APR Technique
Buggy program failing some tests
Repaired program passing all tests
Candidate patch failed some tests
Fault Localization
Patch Generation
Buggy program failing some tests
Localized buggy program
. . .
Candidate patches
Patch Validation
Repaired program passing all tests
Candidate patch passed all tests

# Cobra Effect

- *When an attempted solution to a problem makes the problem worse, as a type of unintended consequence.*

# Cobra Effect

- *When an attempted solution to a problem makes the problem worse, as a type of unintended consequence.*
- What has this to do with program repair?

# Patch Validation

```
Int triangle(int a, int b, int c) {
    if (a <= 0 || b <= 0 || c <= 0)
        return INVALID;
    if (a == b && b == c)
        return EQUILATERAL;
    if (a == b || b != c)        // bug!
        return ISOSCELES;
    return SCALENE;
}
```

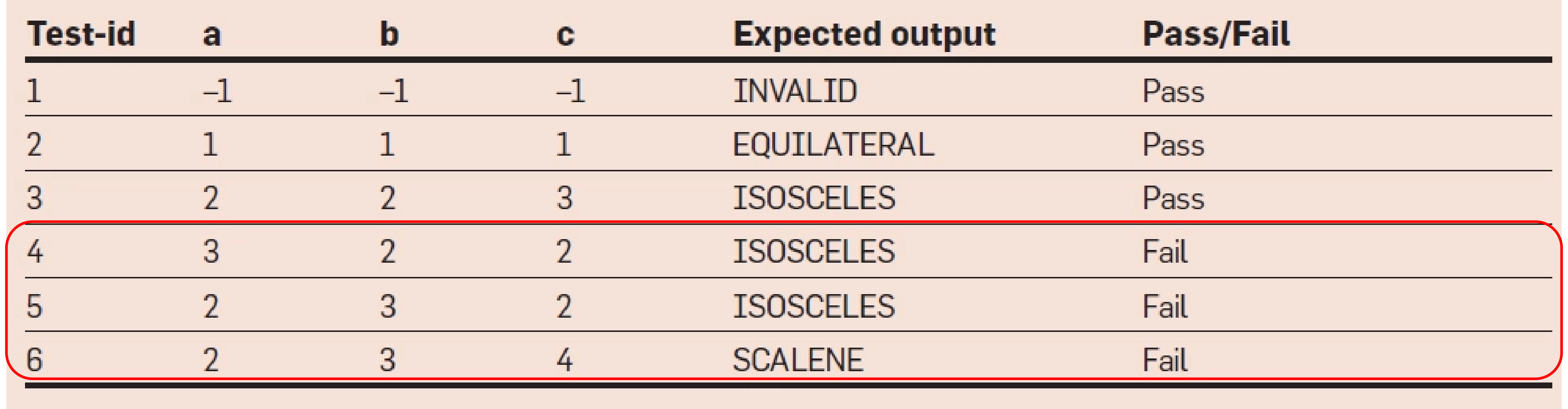
**correct patch**

```
- if (a == b || b != c)
+ if (a == b || b == c || a == c)
      return ISOSCELES
```

**overfitted/plausible patch**

```
- if (a == b || b != c)
+ if (c == 2 || c == 3)
      return ISOSCELES
```

Can we automatically identify over-fitted patches ?



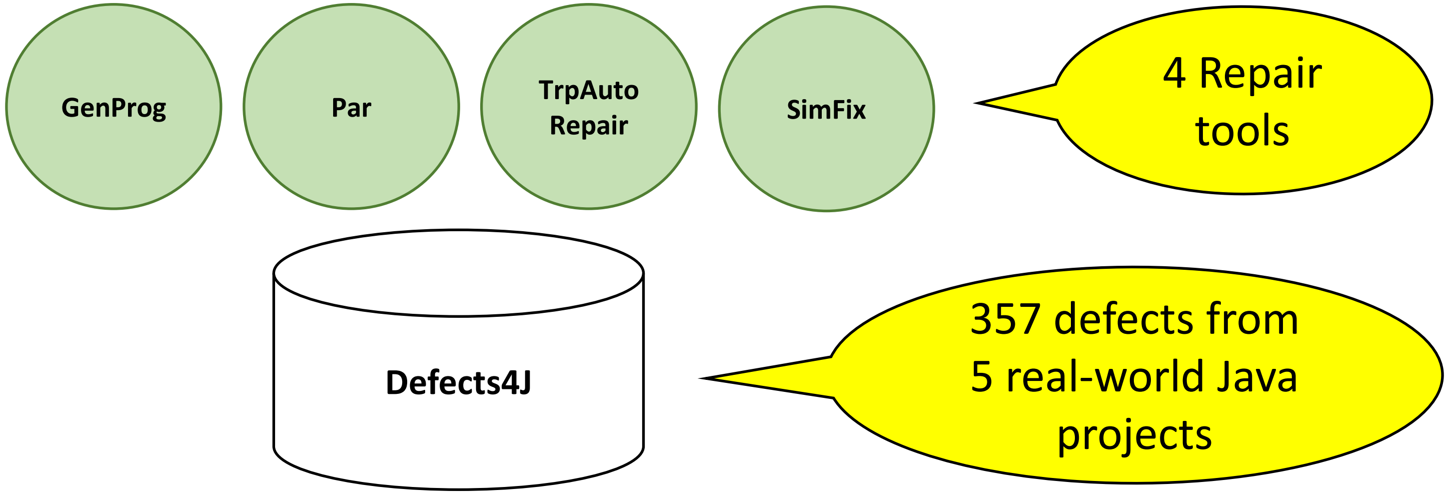
| Test-id | a | b | c | Expected output | Pass/Fail |
|---|---|---|---|---|---|
| 1 | –1 | –1 | –1 | INVALID | Pass |
| 2 | 1 | 1 | 1 | EQUILATERAL | Pass |
| 3 | 2 | 2 | 3 | ISOSCELES | Pass |
| 4 | 3 | 2 | 2 | ISOSCELES | Fail |
| 5 | 2 | 3 | 2 | ISOSCELES | Fail |
| 6 | 2 | 3 | 4 | SCALENE | Fail |

Example source: Le Goues et al., Automated program repair. CACM, Nov 2019

# How to evaluate patch “correctness”?

- Use automatically generated *high-quality* evaluation test-suite to measure the patch quality/correctness
- Patch Quality = #tests passed/total #tests

# Quality of Heuristics Repair Techniques



4 Repair tools

357 defects from 5 real-world Java projects

Manish Motwani, Mauricio Soto, Yuriy Brun, René Just, and Claire Le Goues, Quality of Automated Program Repair on Real-World Defects, IEEE TSE, 2022.

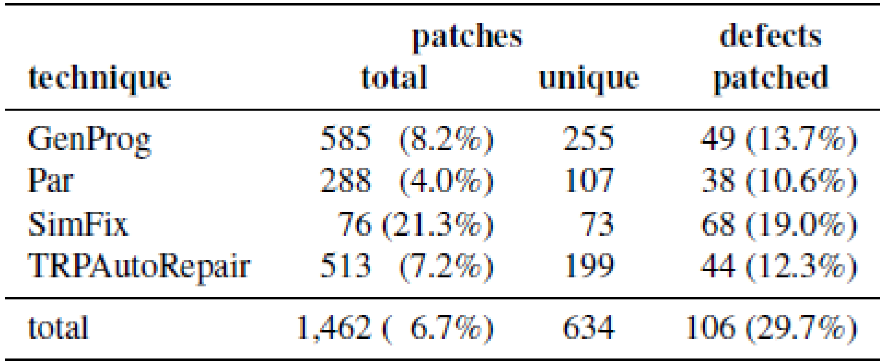
# Quality of Heuristics Repair Techniques

**RQ1**: Do G&V techniques produce patches for real-world Java defects?

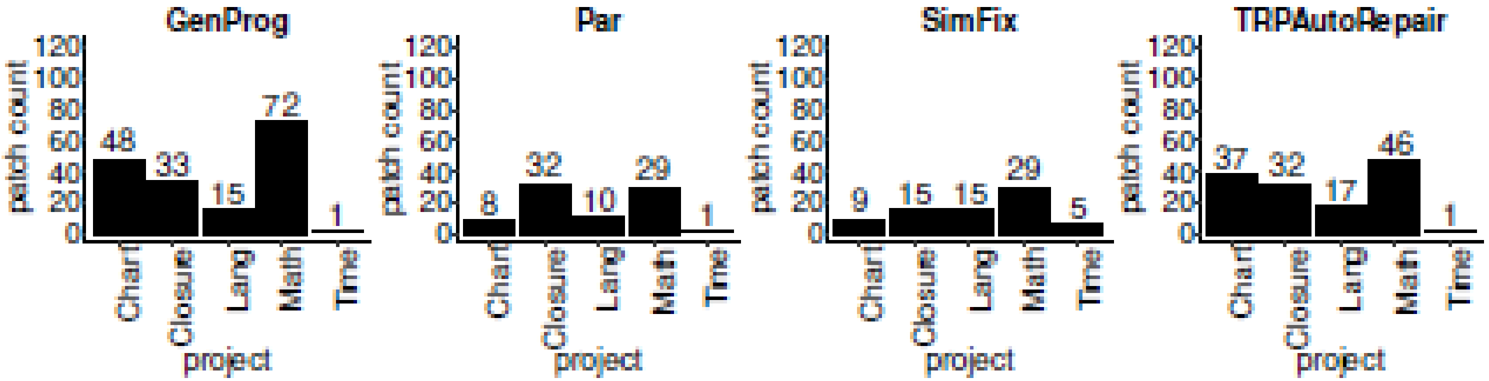
Manish Motwani, Mauricio Soto, Yuriy Brun, René Just, and Claire Le Goues, Quality of Automated Program Repair on Real-World Defects, IEEE TSE, 2022.

# Quality of Heuristics Repair Techniques

**RQ1**: Do G&V techniques produce patches for real-world Java defects?



| technique | patches<br>total | unique | defects<br>patched |
|---|---|---|---|
| GenProg | 585 (8.2%) | 255 | 49 (13.7%) |
| Par | 288 (4.0%) | 107 | 38 (10.6%) |
| SimFix | 76 (21.3%) | 73 | 68 (19.0%) |
| TRPAutoRepair | 513 (7.2%) | 199 | 44 (12.3%) |
| total | 1,462 ( 6.7%) | 634 | 106 (29.7%) |



**RA1**: Yes, although less often than for C defects (19% vs ~50%).

Manish Motwani, Mauricio Soto, Yuriy Brun, René Just, and Claire Le Goues, Quality of Automated Program Repair on Real-World Defects, IEEE TSE, 2022.

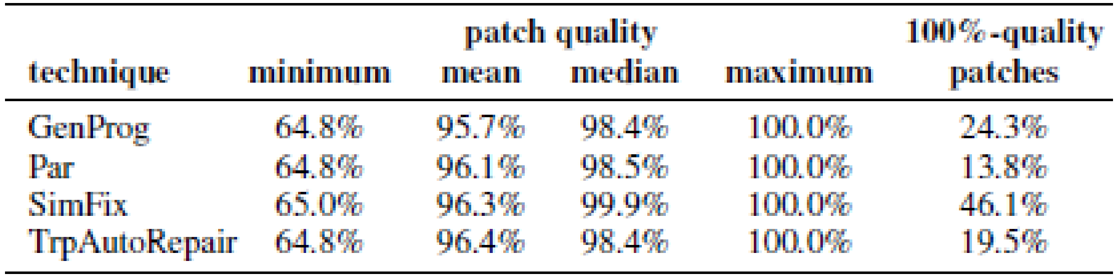
# Quality of Heuristics Repair Techniques

**RQ2**: How often and how much do the patches produced by repair techniques overfit to the developer-written test suite and fail to generalize to the evaluation test suite?

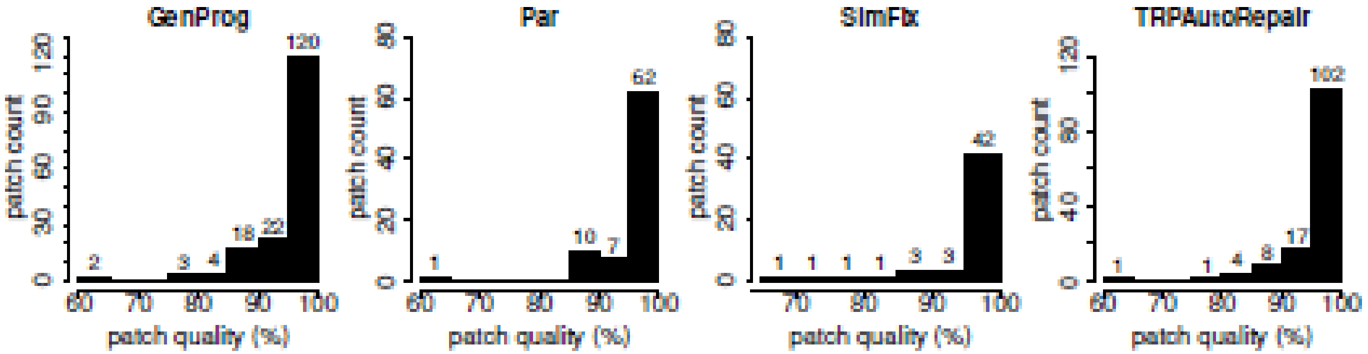
Manish Motwani, Mauricio Soto, Yuriy Brun, René Just, and Claire Le Goues, Quality of Automated Program Repair on Real-World Defects, IEEE TSE, 2022.

# Quality of Heuristics Repair Techniques

**RQ2**: How often and how much do the patches produced by repair techniques overfit to the developer-written test suite and fail to generalize to the evaluation test suite?



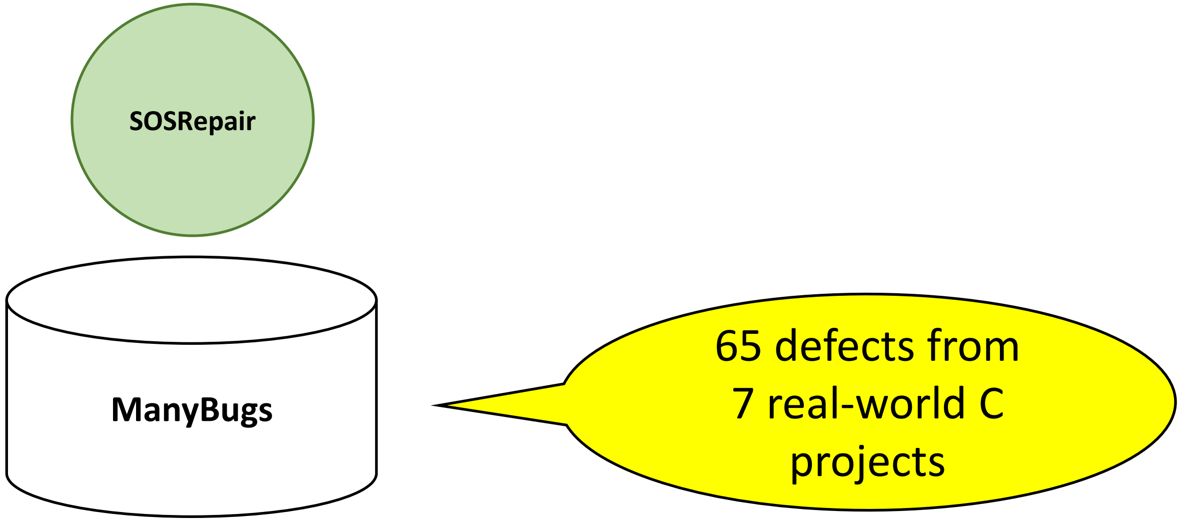
| technique | minimum | patch quality<br>mean | median | maximum | 100%-quality patches |
|---|---|---|---|---|---|
| GenProg | 64.8% | 95.7% | 98.4% | 100.0% | 24.3% |
| Par | 64.8% | 96.1% | 98.5% | 100.0% | 13.8% |
| SimFix | 65.0% | 96.3% | 99.9% | 100.0% | 46.1% |
| TrpAutoRepair | 64.8% | 96.4% | 98.4% | 100.0% | 19.5% |



**RA2**: Often. Only between 13.8% and 46.1% of the patches pass 100% of evaluation test suite.

Manish Motwani, Mauricio Soto, Yuriy Brun, René Just, and Claire Le Goues, Quality of Automated Program Repair on Real-World Defects, IEEE TSE, 2022.

# Quality of Synthesis-based Repair Technique



65 defects from 7 real-world C projects

*SOSRepair patches 22 (34%) out of 65 defects and of these 22 patches, 9 (41%) pass all independent tests*

*Key-Finding: Manually improving fault localization allows SOSRepair to patch 23 (35%) of the defects, of which 16 (70%) pass all independent tests*

Afsoon Afzal, Manish Motwani, Kathryn T. Stolee, Yuriy Brun, and Claire Le Goues, SOSRepair: Expressive Semantic Search for Real-World Program Repair, IEEE TSE, 2021

# How can we improve the quality of repair techniques?

# Idea-1: Improve Fault Localization

Test Suite

Spectrum-based FL

Mutation-based FL

Combine multiple techniques which use different software artifacts to identify buggy program elements.

Bug report

IR-based FL

Program Slicing

Program Slicing FL

Stack Trace

Stack trace FL

Control Flow

Predicate Switching FL

# Idea-2: Improve Patch Generation

Target specific class of defects for e.g., fixing bugs in if conditionals or fixing null pointer exceptions

Use learned fix patterns instead of hard-coded ones

Use novel search strategies to produce the correct patch first

Use novel test sampling strategies to test candidate patches on a sample of test suite and reduce computation time and resources

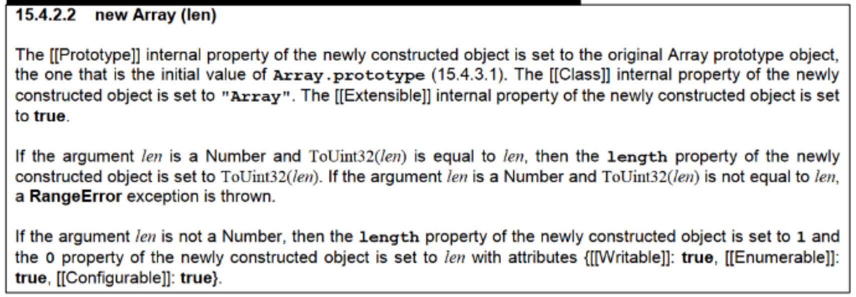
Use information from other artifacts such as contracts/specifications and bug reports

# Idea-3: Improve Patch Validation

Can we improve tests using information from artifacts that are available in real scenario when defect occurs?

Yes! we can use Natural Language Processing (NLP) techniques to generate executable tests from natural-language software artifacts

## Structured Informal Specification



**15.4.2.2 new Array (len)**

The [[Prototype]] internal property of the newly constructed object is set to the original Array prototype object, the one that is the initial value of `Array.prototype` (15.4.3.1). The [[Class]] internal property of the newly constructed object is set to `"Array"`. The [[Extensible]] internal property of the newly constructed object is set to **true**.

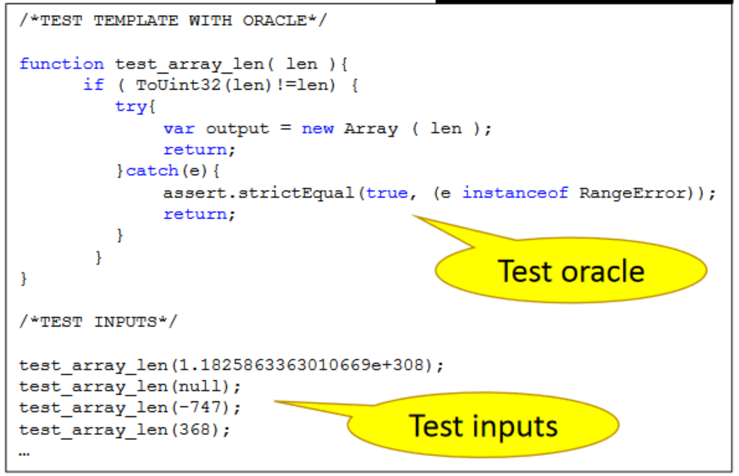
If the argument *len* is a Number and ToUint32(*len*) is equal to *len*, then the `length` property of the newly constructed object is set to ToUint32(*len*). If the argument *len* is a Number and ToUint32(*len*) is not equal to *len*, a **RangeError** exception is thrown.

If the argument *len* is not a Number, then the `length` property of the newly constructed object is set to `1` and the `0` property of the newly constructed object is set to *len* with attributes {[[Writable]]: **true**, [[Enumerable]]: **true**, [[Configurable]]: **true**}.

Swami

swami.cs.umass.edu

## Executable Test



```
/*TEST TEMPLATE WITH ORACLE*/

function test_array_len( len ){
      if ( ToUint32(len)!=len) {
         try{
             var output = new Array ( len );
             return;
         }catch(e){
             assert.strictEqual(true, (e instanceof RangeError));
             return;
         }
       }
}

/*TEST INPUTS*/

test_array_len(1.1825863363010669e+308);
test_array_len(null);
test_array_len(-747);
test_array_len(368);
…
```

Test oracle

Test inputs

# What did we learn?

- Three step process of automatic program repair
  - Fault localization
  - Patch generation
  - Patch validation
- Patch overfitting (repair quality) is a real concern
  - More than 50% of the patches produced by repair techniques overfit.
- Ideas to improve the repair quality by:
  - Improving fault localization
  - Improving patch generation
  - Improving patch validation
- For the latest updated research in program repair, visit http://program-repair.org/
- You will learn how GenProg repairs real-world bugs in HW2.

# Announcements

- HW2 is released today and is due on Wednesday, May 29, 11:59 PM
- Next class, we will have project plan presentations.
  - Each project group will give **20 min** presentation.
  - The presentation should **cover all the aspects of your project except results**.
  - Remember to **use key principles** while organizing your talk and text in your report.
  - Utilize office hours tomorrow from 2-3 PM if you want to discuss something.