

# Software Testing: From Practice to Research

Marcelo d'Amorim



UNIVERSIDADE  
FEDERAL  
DE PERNAMBUCO

Rio Cuarto, Argentina, February 2020

<https://star.cin.ufpe.br>

# Software Testing

Activity of finding bugs in Software

# Software Testing

Activity of finding bugs in Software

But, what is a bug in software? Code that violates software spec.

# Software Testing

## Activity of finding bugs in Software

But, what is a bug in software? Code that violates software spec.

But, what is a spec? Definition of what a system should (and should not) do.

# It is highly popular and expensive!

- Highly used in industry (code inspection also popular)
- It is expensive \*
  - Human: Writing test cases is time consuming
  - Computation: Running test cases is time consuming

\* NIST Report 2002: <http://www.nist.gov/director/planning/upload/report02-3.pdf>

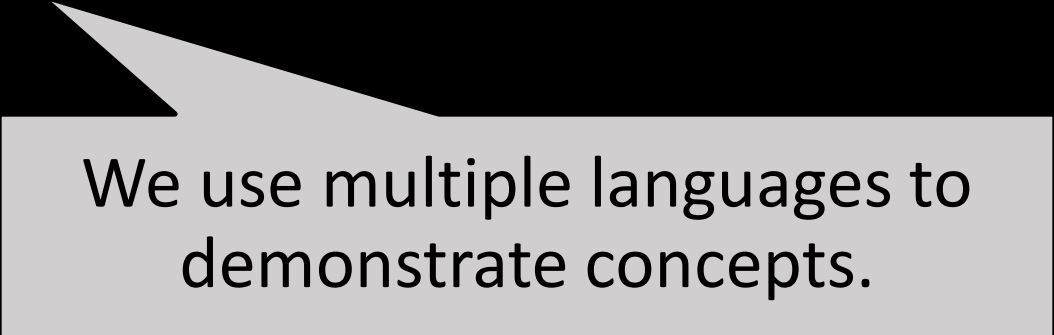
# Goals of this course

- Demonstrate popular concepts and tools used in industry (80%)
- Discuss problems in Software Testing research (20%)

<https://github.com/damorim/testing-cin-minicourse>

# Requirements

- Have basic knowledge of RCS (e.g., Git)
- Took some programming classes and projects



We use multiple languages to demonstrate concepts.

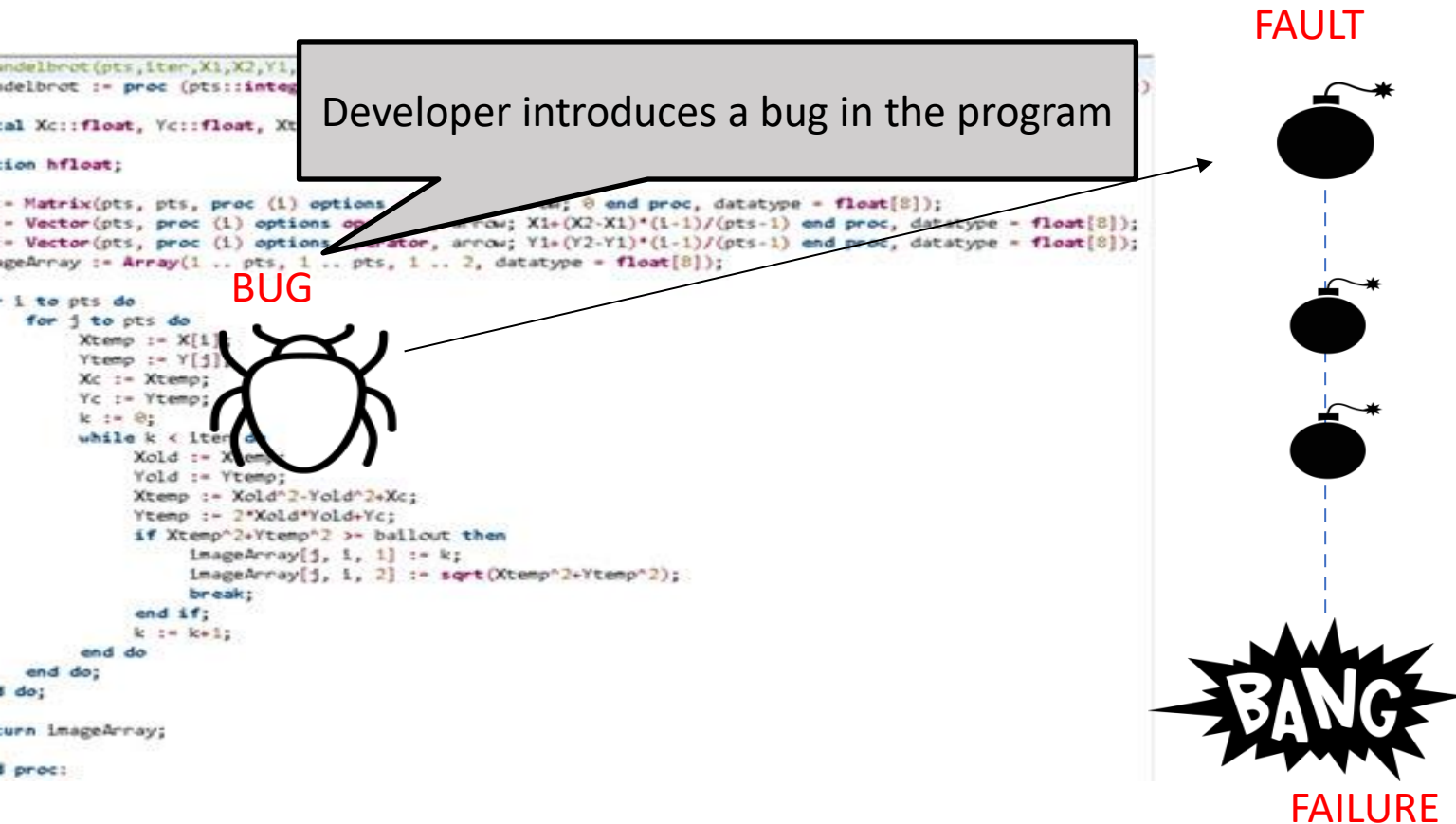
# GitHub Repository

<https://tinyurl.com/s4emvrg>

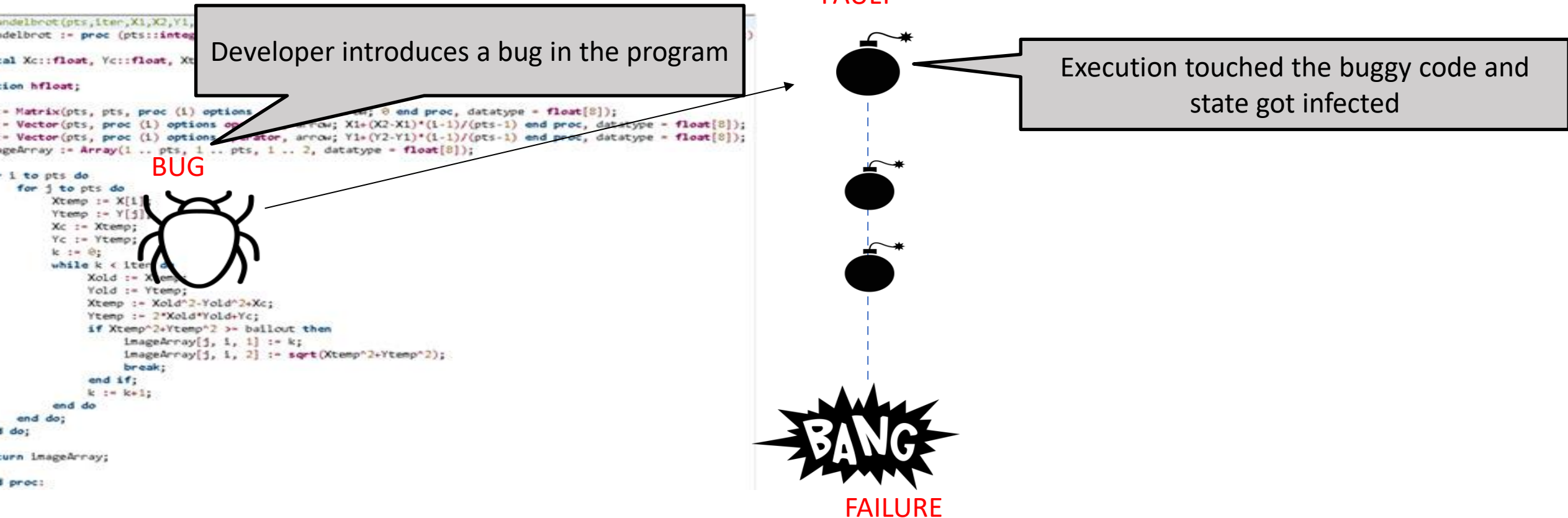


# Preliminaries

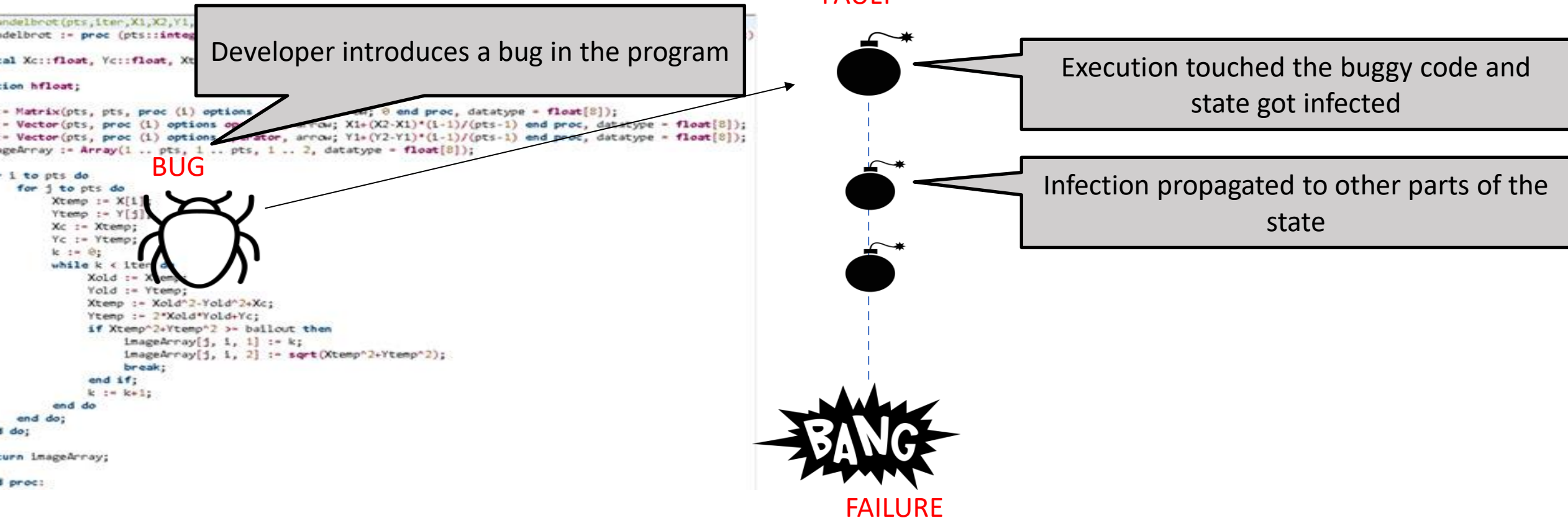
# The lifecycle of a bug



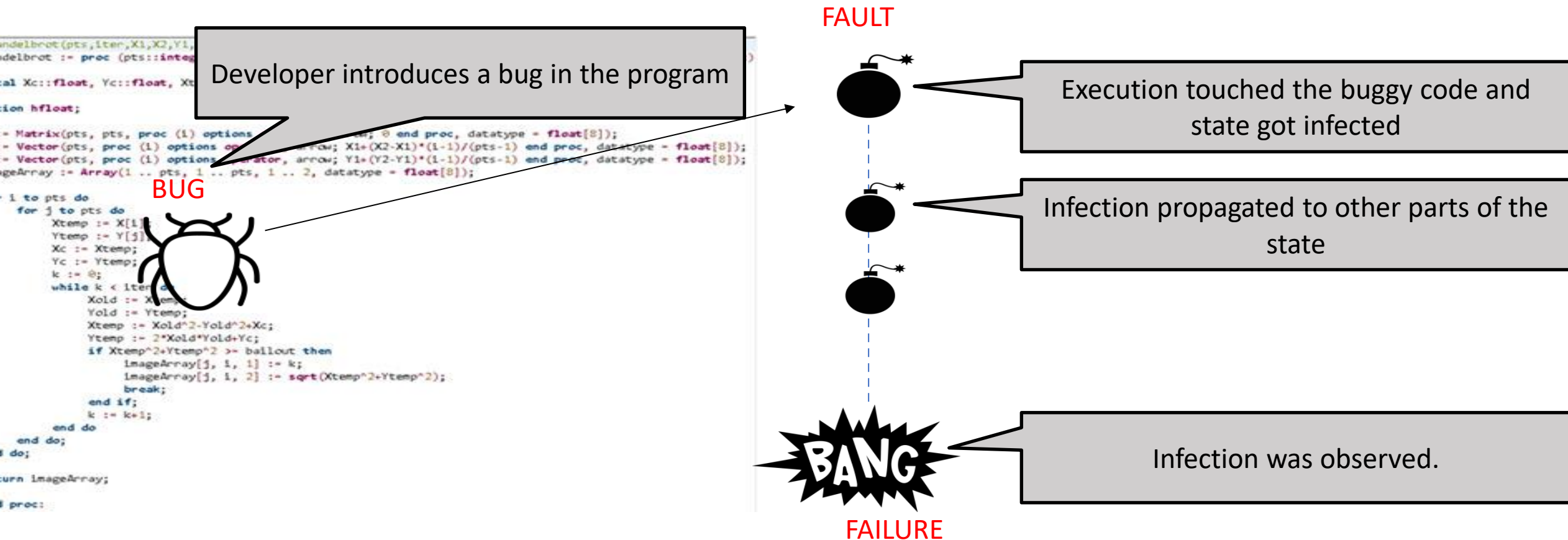
# The lifecycle of a bug



# The lifecycle of a bug



# The lifecycle of a bug



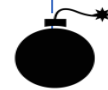
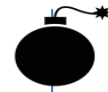
# The lifecycle of a bug

```
delbrot(pts,iter,X1,X2,Y1,Y2,bailout)
delbrot := proc (pts::integer, iter::integer, X1::float, X2::float, Y1::float, Y2::float, bailout::float)
local Xc::float, Yc::float, Xtemp::float, Ytemp::float, k::int, Xold::float, Yold::float, i::int, j::int,
        hfloat;
        hfloat := 0;
        i := 0;
        j := 0;
        Xc := X1;
        Yc := Y1;
        Xtemp := Xc;
        Ytemp := Yc;
        k := 0;
        while k < iter do
            Xold := Xtemp;
            Yold := Ytemp;
            Xtemp := Xold^2-Yold^2+Xc;
            Ytemp := 2*Xold*Yold+Yc;
            if Xtemp^2+Ytemp^2 >= bailout then
                ImageArray[j, 1, 1] := k;
                ImageArray[j, 1, 2] := sqrt(Xtemp^2+Ytemp^2);
                break;
            end if;
            k := k+1;
        end do;
        return ImageArray;
end proc;
```

BUG



FAULT

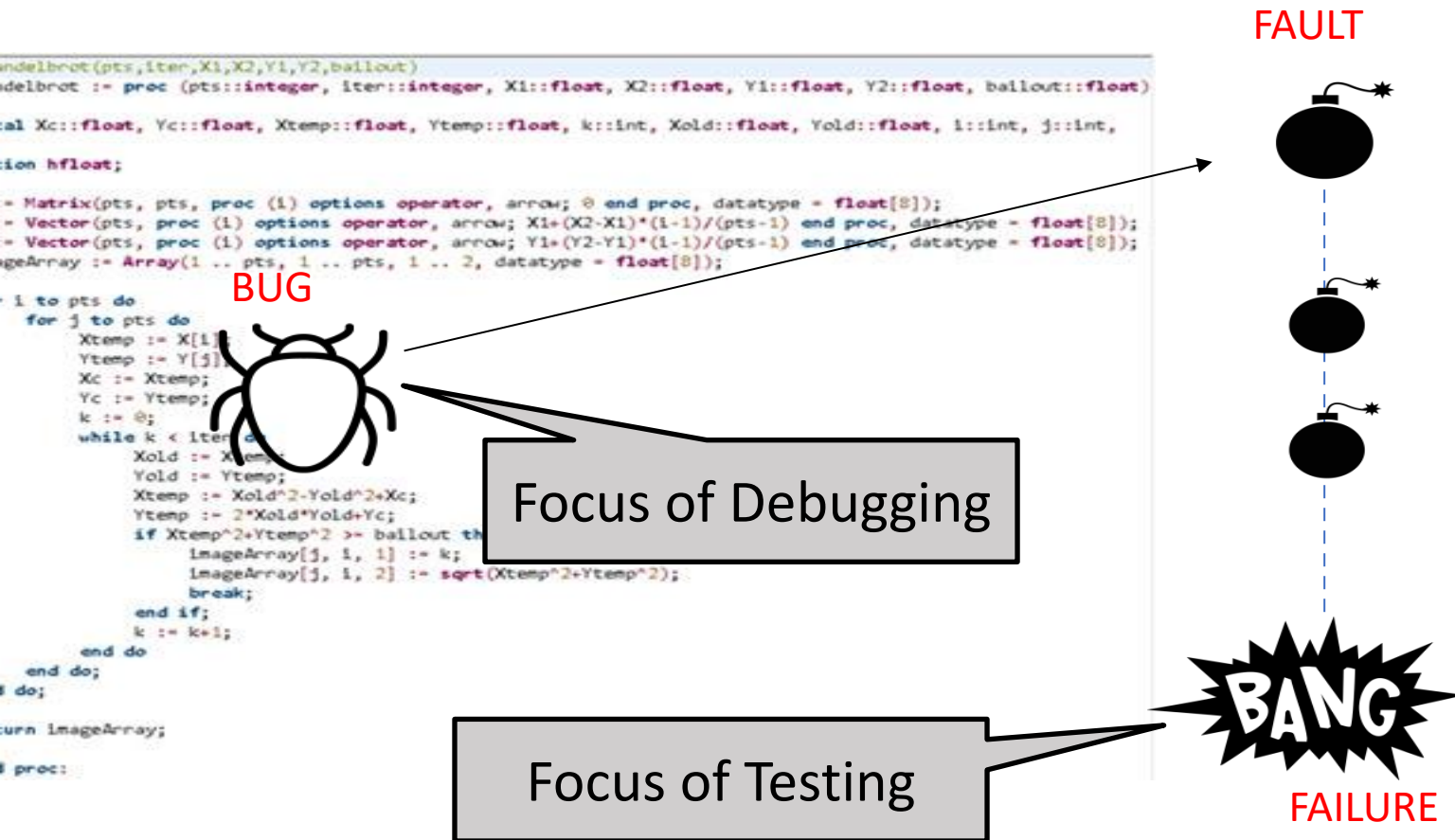


BANG

FAILURE

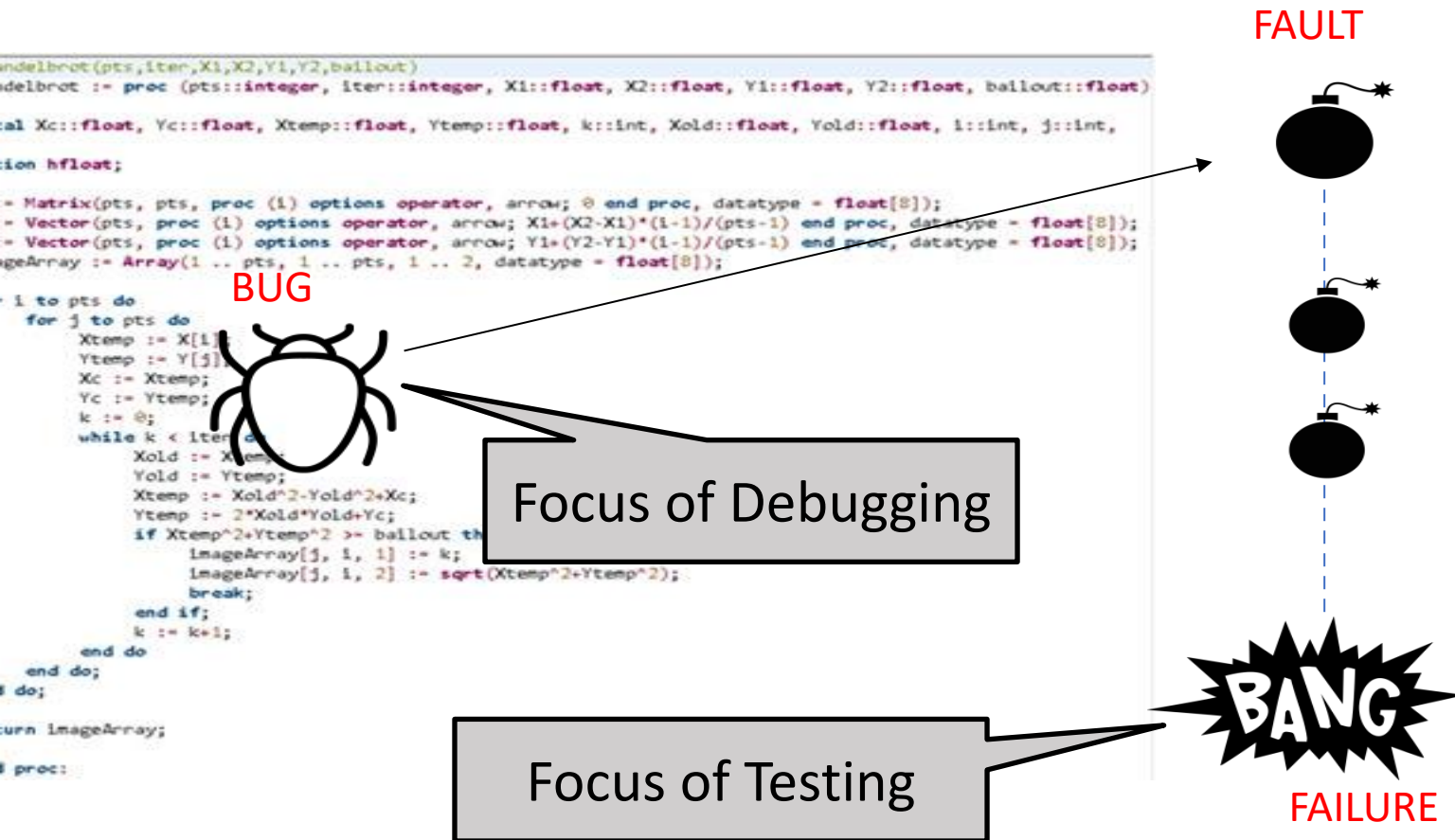
Focus of Testing

# The lifecycle of a bug





# The lifecycle of a bug



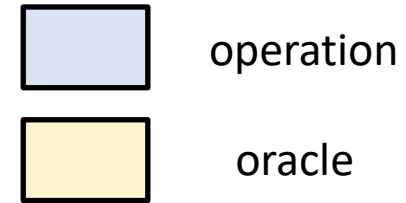
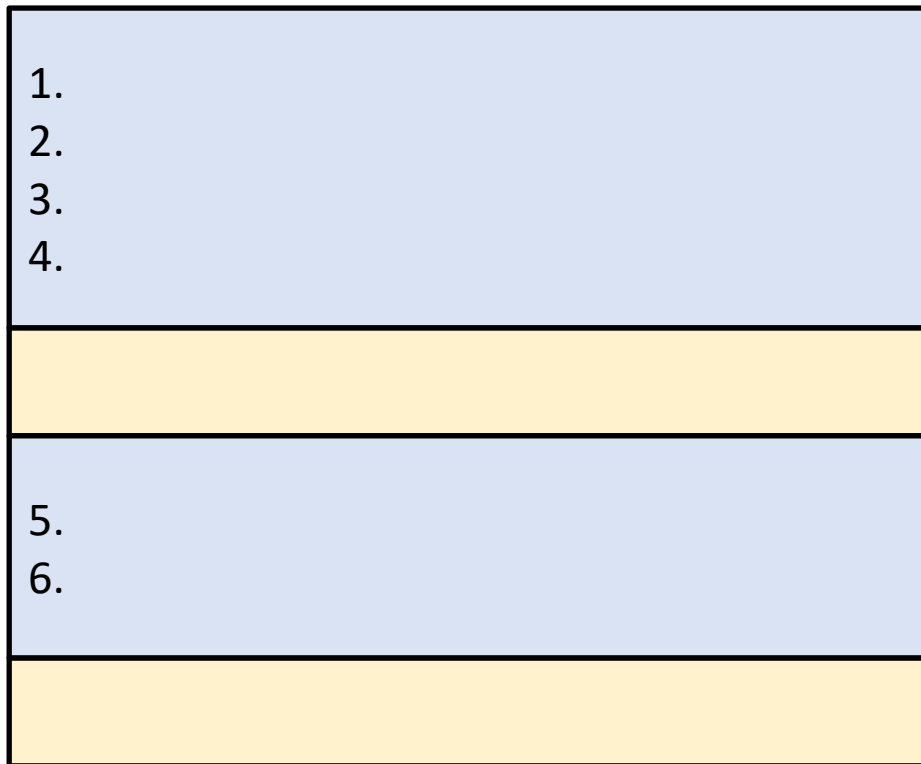
## Scientific Debugging:

1. Observation (e.g., failure)
2. Build Hypothesis
3. Predict behavior based on 2
4. Make an experiment
5. Observation
6. If hypothesis inconsistent then goto 2 (revise)  
else fix code



# Anatomy of a typical test case (to catch a bug)

aTest():

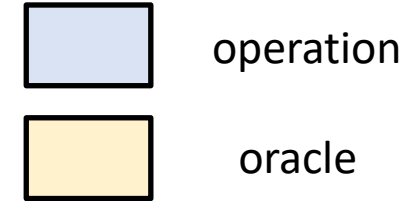


Oracle compares  
expectation with  
observation

# Anatomy of a typical test case (to catch a bug)

aTest():

Typically, test functions declare  
no parameters



Oracle compares  
expectation with  
observation

# Example: Calculator (with Java JUnit)

```
public class Calc {  
    public static int add(int a, int b) {  
        return a + b;  
    }  
    public static int sub(int a, int b) {  
        return a - b;  
    }  
}
```

```
import org.junit.*;  
public class CalcTest {  
    @Test  
    public void testCancel() {  
        int x = 5, y = 10;  
        int res = sub(add(x, y), y);  
        Assert.assertEquals(res, x);  
    }  
}
```

**Automated** test case  
(aka test script)



operation



oracle

# Test Automation

- + Reduces time to execute tests (\*)
- + Improves reproducibility
- Needs to maintain code of the test case
- Inadequate for testing usability, for example

Although manual testing is important (e.g., exploratory testing), it is rare not to see automated tests in serious software today.

# Characterization of Software Testing

- Based on the objective of the testing effort
  - Functional (Testing), Load, Performance, Security, Usability, etc.
- Based on the object tested
  - System (Testing), Integration, and Unit
- Based on code visibility
  - Black-box (Testing), White-box

Testing is a very general concept—applicable in a variety of contexts

# Characterization of Software Testing

- Based on the objective of the testing effort
  - Functional (Testing), Load, Performance, Security, Usability, etc.
- Based on the object tested
  - System (Testing), Integration, and Unit
- Based on code visibility
  - Black-box (Testing), White-box

Example: UI Testing with Selenium



Testing is a very general concept—applicable in a variety of contexts

# Internal versus External Software Quality

- Internal Quality is concerned with quality of the code artifact
  - E.g., modularity (e.g., cohesion and coupling), legibility, etc.
- External Quality is concerned with externally observable aspects
  - E.g., security, performance, functionality, etc.

# Internal versus External Software Quality

- Internal Quality is concerned with quality of the code artifact
  - E.g., modularity (e.g., cohesion and coupling), legibility, etc.
- External Quality is concerned with externally observable aspects
  - E.g., security, performance, functionality, etc.

Software Testing focuses on External Quality




# Software Testing Process

# When to Start Writing Test Cases?

- Top-down: As soon as you know your requirements
  - Test-Driven Development (TDD)
    - Downside--writing auxiliary code
- Bottom-up: As soon as you develop code
  - Unit -> Integration -> System

# When to Start Writing Test Cases

- Top-down: As soon as you know your requirements
  - Test-Driven Development (TDD)
    - Downside--writing auxiliary code
- Bottom-up: As soon as you develop code
  - Unit -> Integration -> System



Observation 1: Your tests should reflect your requirements (functional or not) even if they are implemented afterwards

# When to Start Writing Test Cases

- Top-down: As soon as you know your requirements
  - Test-Driven Development (TDD)
    - Downside--writing auxiliary code
- Bottom-up: As soon as you develop code
  - Unit -> Integration -> System

Observation 1: Your tests should reflect your requirements (functional or not) even if they are implemented afterwards

Observation 2: No need to wait until the last function of the system is implemented to implement system and integration tests. Use mock libraries!

# When to Start Writing Test Cases?

- Top-down: As soon as you know your requirements
  - Test-Driven Development (TDD)

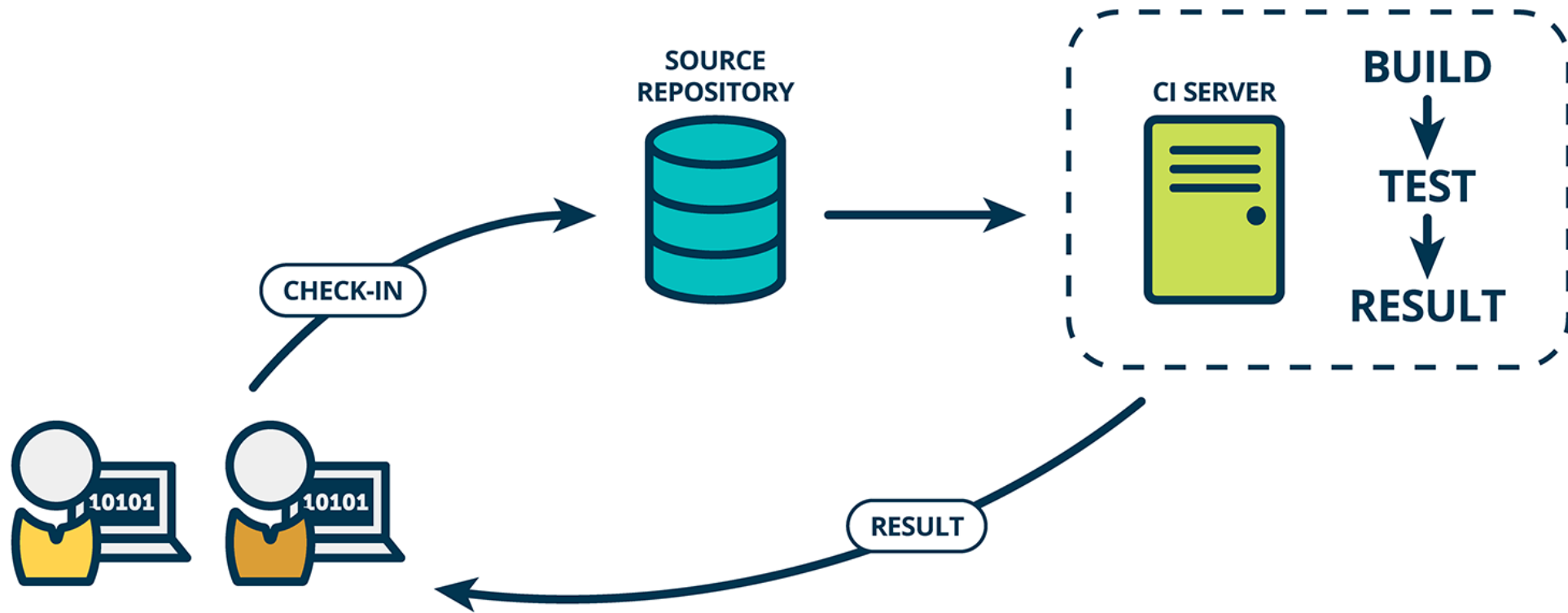
- Decision of when to start writing unit/integration/system tests varies with the project and team.

# Regression Testing

- Regression is the observation that a feature/functionality that used to work is malfunctioning
- Regression testing is the activity of running automated tests regularly with the goal of detecting regressions

# Continuous Integration (CI)

Solution to automate Regression Testing



To Remember



# To Remember 1: Testing cannot prove correctness

“Program testing can be used to show the presence of bugs, but never to show their absence!” --Edsger Dijkstra



It is important to assess “how good” your test suite is!  
(There are techniques for that. We will see.)

# To Remember 2: There is no Silver Bullet

Various other aspects are important to assure external quality

- Internal Quality
- Choice of Language and Tools
- Development Process
- (Trained and Motivated) People

# Brief on Software Specification

# Software Specification

- Definition of what a system should (and should not) do
- Implementation is the realization of a specification

# Assume-Guarantee Contract

- Form of specification of a software module (e.g., a function)
- It consists of a set of constraints describing
  - Assumptions on how the module will be used
  - Guarantees that the module provides

# Example: Power

What can happen if pow is called with negative y?

```
int pow(int x,int y)
{
    int r = 1;
    while(y > 1) {
        if (y % 2 == 1) {
            r = x * r;
        }
        x = x * x;
        y = y / 2;
    }
    return r * x;
}
```

# Example: Power

What can happen if pow is called with negative y?

```
int pow(int x,int y)
{
    int r = 1;
    while(y > 1) {
        if (y % 2 == 1) {
            r = x * r;
        }
        x = x * x;
        y = y / 2;
    }
    return r * x;
}
```

Consider  $x = 2, y = -2$

$$2^{-2} = 1/(2^2) = 1/4$$

So, pow should return 0 (note integer arithmetic)

Instead, it returns  $1 * 2 = 2$

# Example: Power

What can happen if pow is called with negative y?

```
int pow(int x,int y)
{
    int r = 1;
    while(y > 1) {
        if (y % 2 == 1) {
            r = x * r;
        }
        x = x * x;
        y = y / 2;
    }
    return r * x;
}
```

Consider  $x = 2, y = -2$

$$2^{-2} = 1/(2^2) = 1/4$$

So, pow should return 0 (note integer arithmetic)

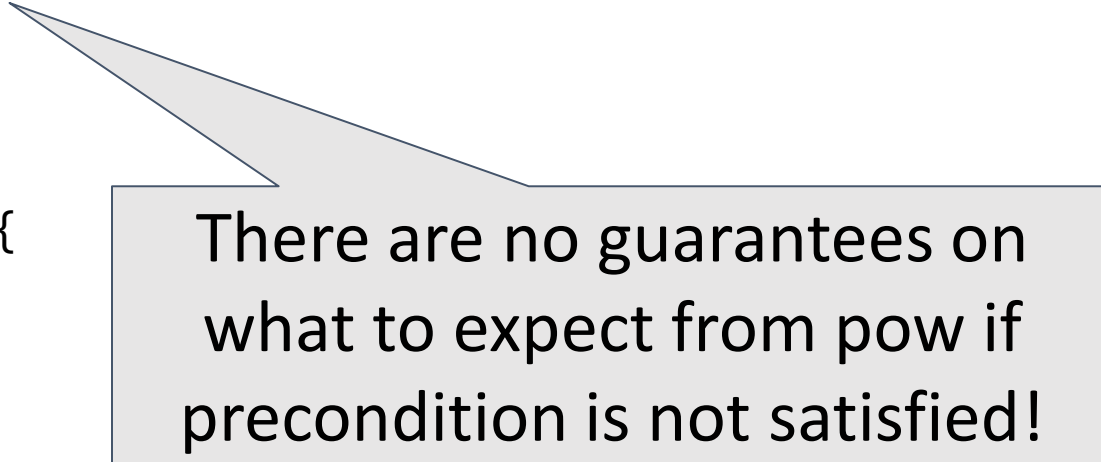
Instead, it returns  $1 * 2 = 2$

Is this a bug?



# Imposing Precondition

```
int pow(int x,int y)
//@requires y >= 0;
{
    int r = 1;
    while(y > 1) {
        if (y % 2 == 1) {
            r = x * r;
        }
        x = x * x;
        y = y / 2;
    }
    return r * x;
}
```



There are no guarantees on what to expect from pow if precondition is not satisfied!

# Promising a Postcondition

```
int pow(int x, int y)
//@requires y >= 0;
//@ensures \result == RPOW(x,y);
{
    int r = 1;
    while(y > 1) {
        if (y % 2 == 1) {
            r = x * r;
        }
        x = x * x;
        y = y / 2;
    }
    return r * x;
}
```

```
int RPOW (int x,int y)
//@requires y >= 0;
{
    if(y == 0)
        return 1;
    else
        return x * RPOW (x, y-1);
}
```

# Promising a Postcondition

```
int pow(int x, int y)
//@requires y >= 0;
//@ensures \result == RPOW(x,y);
{
    int r = 1;
    while(y > 1)
        if (y % 2 == 0)
            r = x * r;
        else
            x = x * x;
        y = y / 2;
    }
    return r * x;
}
```

**Assume-Guarantee**  
contract for function pow.

```
int RPOW (int x,int y)
//@requires y >= 0;
{
    if(y == 0)
        return 1;
    else
        return x * RPOW (x, y-1);
}
```

# Exercise

Specify contract (i.e., pre and postcondition) for function sort.

```
void sort(int[] ar)
```

The expression  $\backslash\text{old}(\text{var})$  denotes the value of var at the entry of the function call.

# Solution

Specify contract (i.e., pre and postcondition) for function sort.

```
void sort(int[] ar)
//@requires ar != null
//@ensures ASC(ar) && PERM(ar, \old(ar))
```

```
boolean ASC(int[] ar) // returns true iff contents of array are in ascending order
```

[illegible]

# (Take Home) Exercise

Specify contracts for the functions of a stack data structure.

```
class Stack<T> {  
    int size();  
    void push(T t);  
    T pop()  
}
```

# Software Specification

- Definition of what a system should (and should not) do
  - Implementation is the realization of a specification
- Writing specs is expensive
  - It is rare for developers to fully specify a system
  - However, it is common to partially specify systems (e.g., **test assertions**)

# Quiz

- Is it possible to use testing to demonstrate that there are no bugs in code?
- From what observation software debugging typically starts?
- Name one benefit of using automated tests.
- What is a regression?
- What is regression testing?
- What is continuous integration?



# Practice!

“For the things we have to learn before we can do them, we learn by doing them.” --Aristotle, The Nicomachean Ethics

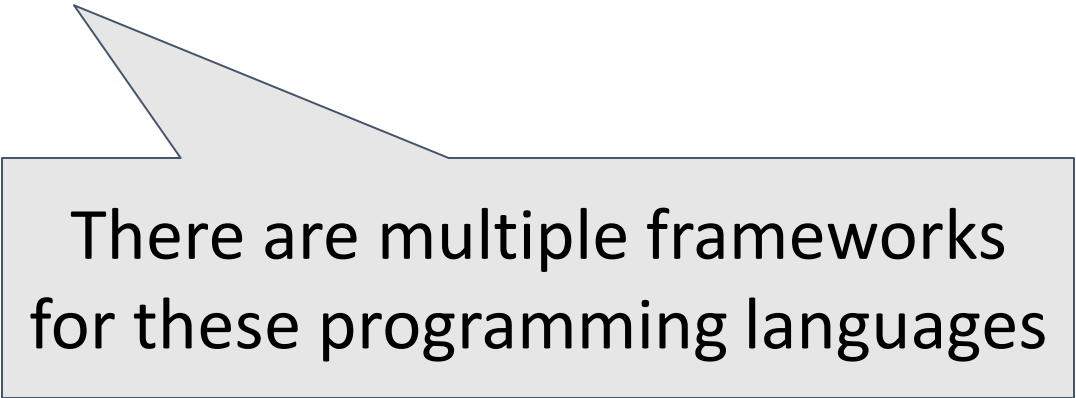
# Agenda

- |                                     |  |   |
|-------------------------------------|--|---|
| • Test frameworks                   |  | Test Infrastructure (to create and run tests) |
| • Build systems                     |  |   |
| • Coverage                          |  | Test adequacy                                 |
| • Mutation testing                  |  |   |
| • Combinatorial testing             |  | System testing                                |
| • UI testing                        |  |   |
| • Behavior-Driven Development (BDD) |  | Test design                                   |
| • Fuzzing                           |  |   |
|                                     |  | Test input generation                         |

Test Infrastructure

# Test Frameworks

- Tools that enable developers to write automated tests
- Available in most modern languages
  - For example: Java's JUnit, Python's unittest, C#'s NUnit, Ruby's rspec



There are multiple frameworks  
for these programming languages

# Example of functionalities in Java's JUnit

- Test fixtures to configure the system before/after running the test
- Customizable test runners
- Rich library of assertion functions (e.g., `assertNull(var)`)
- Ability to group test differently (e.g., slow/fast) for selective execution
- ...

# Example of functionalities in Java's JUnit

- Test fixtures to configure the system before/after running the test
- Custom
- Rich li
- Ability to group test differently (e.g., slow/fast) for selective execution
- ...

Other test frameworks offer very similar functionality

# Example of functionalities in Java's JUnit

- Test fixtures to configure the system before/after running the test
- Customizable test runners
- Rich library of assertion functions (e.g., `assertNull(var)`)
- Ability to group test differently (e.g., slow/fast) for selective execution
- ...

Practice – Section 1 in our repo

# Exercise 1

Introduce an exception `ElementNotFoundException` to indicate element not found in the binary search

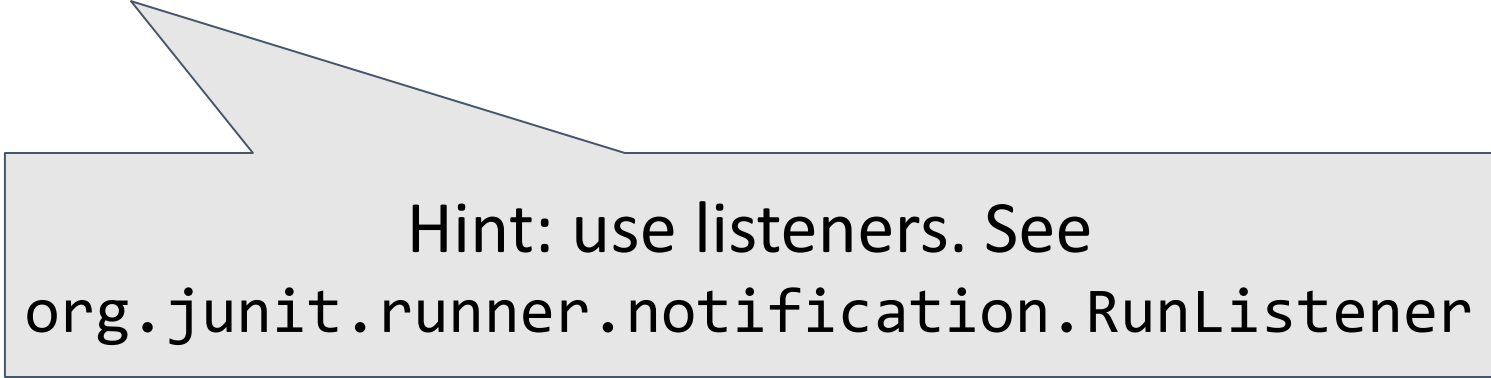


## Exercise 2

Change the method setup in `BinarySearchTest` to create an array with 100K elements and enforce each test to finish in no more than 1s.

# (Take Home) Exercise

Log every test that executed for more than .5 s



Hint: use listeners. See  
`org.junit.runner.notification.RunListener`

# Build Systems

- Automate common tasks during the development process
  - E.g., compile, execute test suites, assemble, deploy, generate reports
- Various implementations
  - E.g., make, ant, maven, gradle

# Build Systems

- Automate common tasks during the development process
  - E.g., compile, execute test suites, assemble, deploy, generate reports
- Various **Practice – Section 2 in our repo**
  - E.g., make, ant, maven, gradle

# Exercise 1

Add the following snippet to your build.gradle script. Then, run the command `$> gradle printClasspath`

```
## add this to build.gradle to print classpath
task printClasspath {
    doLast {
        configurations.testRuntimeClasspath.each { println it }
    }
}
```

# Exercise 2

Follow the tutorial at the link below to configure your build.gradle file to generate test execution logs.

Tutorial: <https://tinyurl.com/uyy3blb>

# (Take Home) Exercise 3

Create a repo on GitHub with some tiny application containing test cases. Then, configure your repo to use Travis CI.

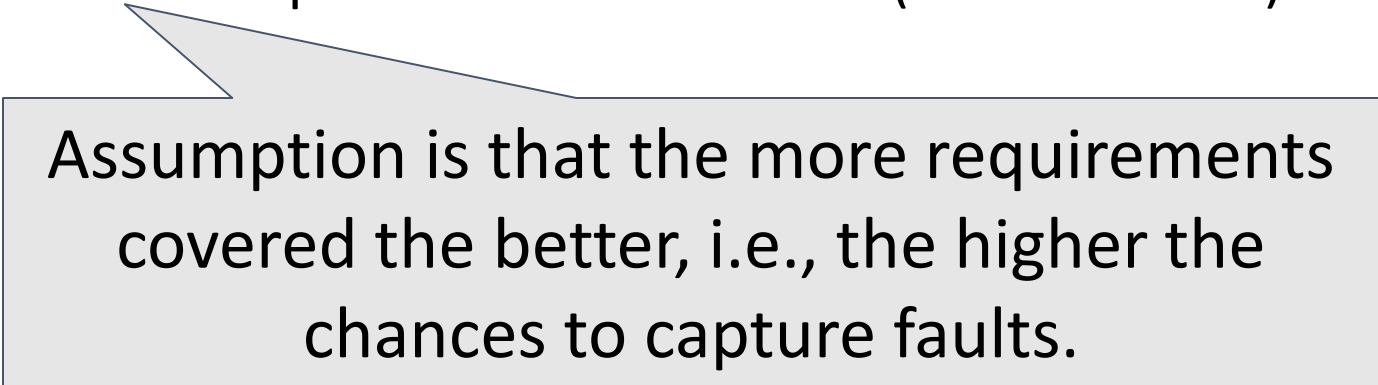
Tutorial: <https://tinyurl.com/s722a4p>

Test Adequacy



# Test Adequacy Criterion

- Important to determine how satisfactory a test suite is
  - Recall that Testing cannot prove correctness!
- General idea:
  - Define test requirements
  - Measure how many of these requirements are fulfilled (or **covered**)



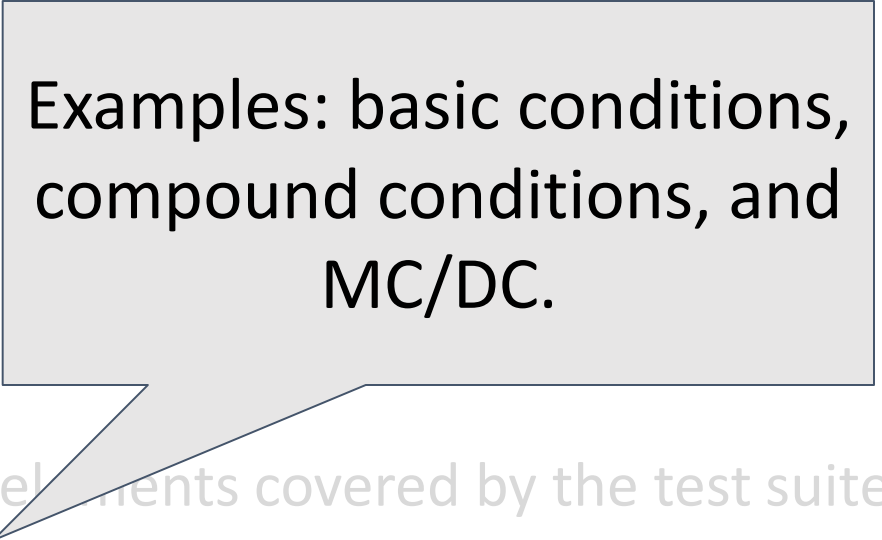
Assumption is that the more requirements covered the better, i.e., the higher the chances to capture faults.

# Test Adequacy Criterion

Examples: statement, line, basic-block, branch, function, acyclic paths, etc.

- Various approaches to measure coverage
  - **Structural coverage** measures amount of code elements covered by the test suite
  - **Logical coverage**: measures extent to which conditionals are covered
  - **Dataflow coverage**: measures extent to which data flows are covered
  - **Mutation coverage** measures amount of (injected) faults covered by the test suite
  - ...

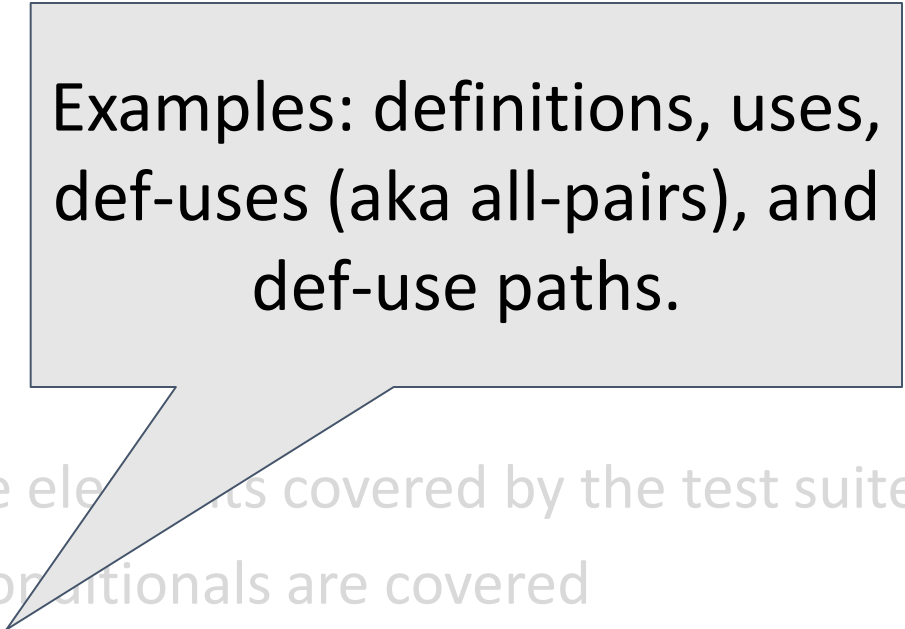
# Test Adequacy Criterion



Examples: basic conditions, compound conditions, and MC/DC.

- Various approaches to measure coverage
  - **Structural coverage** measures amount of code elements covered by the test suite
  - **Logical coverage:** measures extent to which conditionals are covered
  - **Dataflow coverage:** measures extent to which data flows are covered
  - **Mutation coverage** measures amount of (injected) faults covered by the test suite
  - ...

# Test Adequacy Criterion

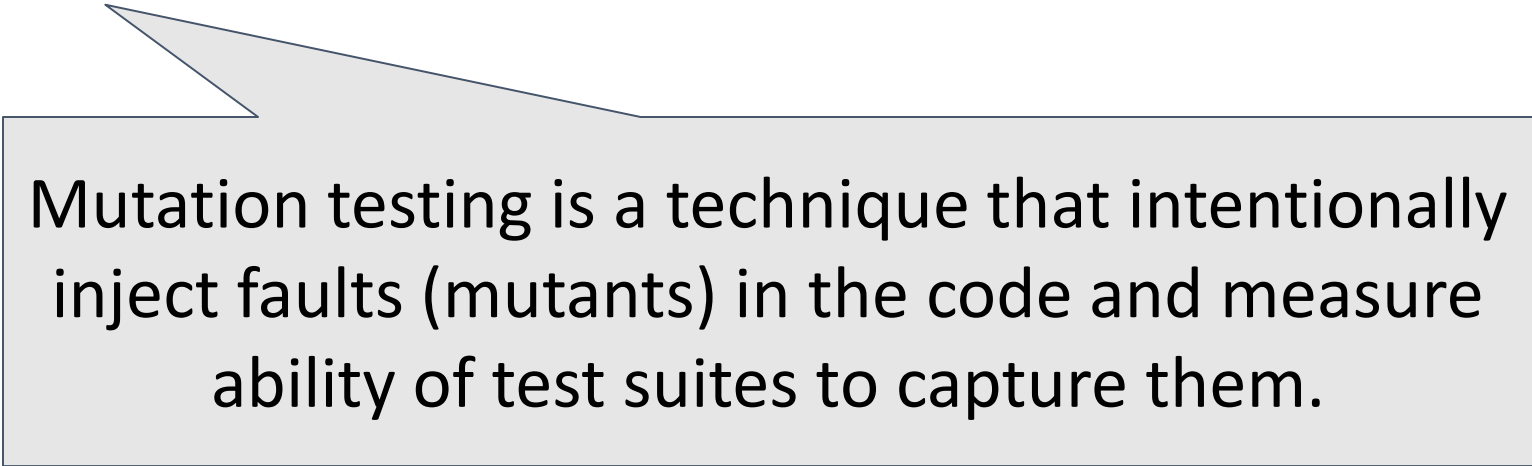


Examples: definitions, uses, def-uses (aka all-pairs), and def-use paths.

- Various approaches to measure coverage
  - **Structural coverage** measures amount of code elements covered by the test suite
  - **Logical coverage**: measures extent to which conditionals are covered
  - **Dataflow coverage**: measures extent to which data flows are covered
  - **Mutation coverage** measures amount of (injected) faults covered by the test suite
  - ...

# Test Adequacy Criterion

- Various approaches to measure coverage
  - **Structural coverage** measures amount of code elements covered by the test suite
  - **Logical coverage**: measures extent to which conditionals are covered
  - **Dataflow coverage**: measures extent to which data flows are covered
  - **Mutation coverage** measures amount of (injected) faults covered by the test suite
  - ...



Mutation testing is a technique that intentionally inject faults (mutants) in the code and measure ability of test suites to capture them.

# Test Adequacy Criterion

- Various approaches to measure coverage
  - **Structural coverage** measures amount of code elements covered by the test suite
  - **Logical coverage:** measures extent to which conditionals are covered
  - Practice – Sections 3 and 4 in our repo
  - ...

# Exercise 1

Try to maximize branch coverage of the test suite for `doublelinkedlist`

Please, add coherent test oracles to  
your test cases

## Exercise 2

Try to maximize mutation coverage of the test suite for triangle

Please, add coherent test oracles to  
your test cases



# System Testing

# System Testing

- The entire system is the object tested  
(not the units or their integration)
- Critical to validate requirements with stakeholders
  - Acceptance tests

# Functional Testing

- Models entire system as a function
- Simplistic Example: Testing configurations of a mobile App

Orientation	landscape, portrait
Size	1080x1920, 750x1334, 720x1280
OS	iPhone, Android

# Functional Testing

- Models entire system as a function
- Simplistic Example: Testing configurations of a mobile App

Orientation	landscape, portrait
Size	1080x1920, 750x1334, 720x1280
OS	iPhone, Android

How many configurations would be required to test the phone exhaustively?

# Combinatorial Interaction Testing (CIT) \*

- Combining all values can be daunting for large number of params/values!



34 switches =  
 $2^{34}$  combinations of inputs =  
 $1.7 \times 10^{10}$  inputs/tests

# Combinatorial Interaction Testing (CIT) \*

- Combining all values can be daunting for large number of params/values!
- CIT is a technique to mitigate combinatorial explosion from exhaustive/combinatorial testing

# CIT – Pairwise Testing

- All combinations of values across pairs of parameters are tested
- Reveals most (50-90%) of the faults in configurable software

# CIT – Pairwise Testing

Orientation	landscape, portrait
Size	1080x1920, 750x1334, 720x1280
OS	iPhone, Android

Orientation	Size	OS
landscape	1080x1920	iPhone
landscape	750x1334	Android
landscape	720x1280	Android
portrait	1080x1920	Android
portrait	750x1334	iPhone
portrait	720x1280	iPhone



# CIT – Pairwise Testing

Orientation	landscape
Size	1080x1920
OS	iPhone

6 combinations as opposed to  
12 combinations for  
exhaustive testing

Orientation	Size	OS
landscape	1080x1920	iPhone
landscape	750x1334	Android
landscape	720x1280	Android
portrait	1080x1920	Android
portrait	750x1334	iPhone
portrait	720x1280	iPhone

# CIT – Pairwise Testing

Orientation	landscape
Size	1080x1920
OS	iPhone

6 combinations as opposed to  
12 combinations for  
exhaustive testing

Practice – Section 5 in our repo

landscape	1080x1920	iPhone
landscape	750x1334	Android
landscape	720x1280	Android
portrait	1080x1920	Android
portrait	750x1334	iPhone
portrait	720x1280	iPhone

# Exercise 1

Use the ACTS tool to produce test inputs for the problem presented previously of testing mobile apps. Are the outputs identical? Why?

Hint: Use the GUI  
`$> java -jar acts_cmd.jar`

# (Take Home) Exercise

Model a problem of your interest with various parameters/values and use the ACTS tool (with  $t=2$ ) to generate test inputs.

# UI Testing

Approach to test software through the User Interface

- + Requires low technical background
- + Very intuitive
- Can be difficult to maintain
- Can be slow to execute
- Requires special environment (e.g., cloud)

# UI Testing

Approach to test software through the User Interface

- + Requires low technical background

- + Very

Practice – Section 6 in our repo

- Can be difficult to maintain

- Can be slow to execute

- Requires special environment (e.g., cloud)

# Exercise 1

Create a test on the Mercury tours website using Selenium  
(<http://demo.guru99.com/test/newtours/>)

# Exercise 2

Create a test on the Address book website using watir  
(<http://a.testaddressbook.com>)



# Test Design

# Test Design

- Activity of creating test cases from specifications. Typically requires some domain knowledge.
- Formal models, when exist (it is rare), could be used to derive test cases (Model-based Testing)

# Behavior-Driven Development

- Way of writing test cases that focuses on collaboration to clarify requirements across teams/stakeholders
  - Uses natural language to describe test cases
  - Typical focus is acceptance (system) tests

# Behavior-Driven Development

- Way of writing test cases that focuses on collaboration to clarify requirements across teams/stakeholders
  - Uses natural language to describe test cases
  - Typical focus is acceptance (system) tests

Practice – Section 7 in our repo

# Exercise 1

Implement the skipped scenario

# Exercise 2

Create and implement a new scenario (after login)

# Fuzzing

- Technique to automatically generate test inputs to find crashes
  - Modifies given input of a program
- Various approaches
  - Black-box (e.g., Radamsa), White-box (e.g., KLEE), Grey-box (e.g., AFL)
  - Grammar-based or not

# Fuzzing

- Technique to automatically generate test inputs to find crashes
  - Modifies given input of a program
- Various approaches
  - Black-box (e.g. American Fuzz Tester, AFL)
  - Grammar-based

Practice – Section 8 in our repo



# (Take home) Exercise

Run AFL on an executable of your choice

# Final Quiz

- Cite one benefit of using automated tests?
- Why developers use build systems?
- Why test adequacy is important?
- Is it correct to say that no more testing is necessary for a test suite with 100% branch coverage?
- What is a mutant?

# Final Quiz

- When pairwise testing produces the same number of inputs as exhaustive testing?
- Cite one benefit of using automated tests?
- Why UI tests can be problematic to maintain?
- Why BDD-style tests are easy to explain to non-technical personnel?
- What kinds of bugs gray-box fuzzing are good to capture?

# Recent Research Projects

# Detection of Plugin Conflicts in CMS

[Information and Software Technology, Feb. 2020]



(a) The WEN-LOGO-SLIDER plugin adds a slider for the images uploaded by the user. The slider animation is implemented in JavaScript.

Lorem ipsum dolor sit amet,  
consectetur adipiscing elit, sed  
do eiusmod tempor incididunt ut

Excepteur sint occaecat  
cupidatat non proident, sunt in  
culpa qui officia deserunt mollit

(b) The WEN-RESPONSIVE-COLUMNS plugin renders input text in multiple columns.



(c) The slider disappears when both plugins are active.

```
<p>  
<script type="text/javascript">  
  jQuery( document ).ready(function($) {  
    jQuery.fn.randomize = function(selector) {  
      var $elems = selector ? $(this).find(selector) :  
      $parents = $elems.parent();  
      $parents.each(function() {  
        $(this).children(selector).sort(function(a, b) {  
          return Math.random() - 0.5;  
        });  
      });  
    }  
  });  
</script>  
</p>
```

(d) Snippet of JavaScript code for the image slider animation. The code becomes not parsable when both plugins are active.

Figure 1: Example plugin conflict.

# Using Docker to Reproduce StackOverflow Posts

[IEEE Transactions on Software Engineering, Dec. 2019]



Frisk Home Faq Tutorial

## Frisk!

Prototyping your server-side project made easy

Frisk is a tool for prototyping and sharing server-side projects compatible with Docker technology! Frisk is an on-going project. Hence, there might be interesting features that have not yet been implemented, or even unexpected behavior.

## Frameworks

Filter...

### Empty

- Language: N/A
- Database: None
- Template Engine: None

### Express.js

- Node.js: 6.9.5
- Database: None
- Template Engine: None

### Express.js


- Node.js: 6.9.5
- Database: None
- Template Engine: Jade

### Rails 5

- Ruby: 2.5.3
- Database: SQLite 3
- Template Engine: ERB

# Using Docker to Reproduce StackOverflow Posts

[IEEE Transactions on Software Engineering, Dec. 2019]


 Frisk

Home

Faq

Tutorial


CLOSE SESSION

VMs 

+ ADD NEW MACHINE

10.0.108.4  
node1


SSH

ssh ip10-0-108-4-bp57j936m1ugccp1gf2g@direct.pg.frisk.ic 

DELETE


EDITOR


SHARE

 BUILD BUILD DOCKER COMPOSE RUN

STOP DELETE LIST

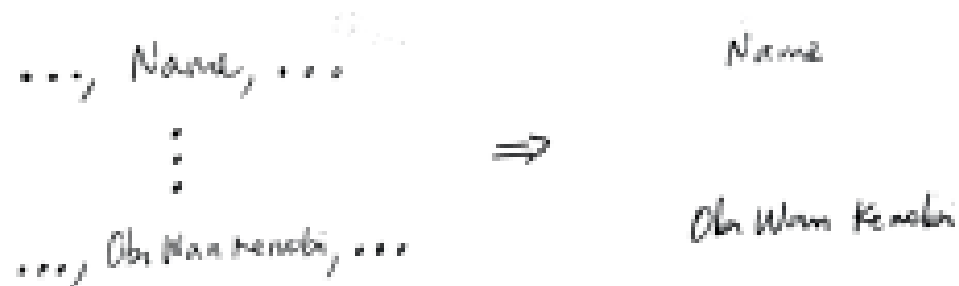
```
$ undefined#####  
##  
#  
# WARNING!!!!  
# This is a sandbox environment. Using personal crede  
# is HIGHLY! discouraged. Any consequences of doing s  
# completely the user's responsibilites. No changes i  
# environment will be persisted.  
#  
# The FRISK team.  
#####  
[node1] (local) root@10.0.108.4 ~  
$
```

Create or upload files in the session terminal and then refresh 

  
/root  
├── Dockerfile  
├── Gemfile  
├── Gemfile.lock  
├── README.md  
├── Rakefile  
├── app  
├── bin  
├── config  
├── config.ru  
├── lib  
├── log  
├── package.json  
├── public  
├── test  
├── tmp  
└── vendor

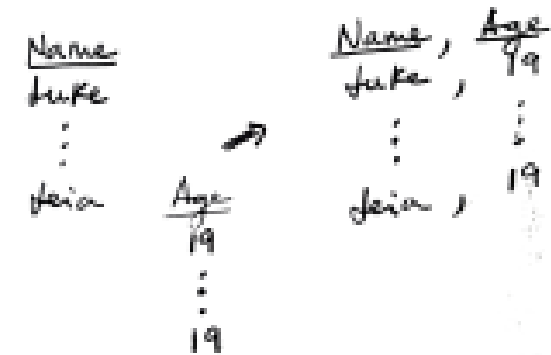
# Visual Sketching

[ICSE-NIER 2020]



```
import pandas as pd
fileName = ? #fill hole (in)
df1 = pd.read_csv(fileName)
colName = ? #fill hole (in)
df2 = df1[[colName]]
file = ? #fill hole (out)
df2.to_csv(file, index=None)
```

Figure 1: Column projection from csv file.



```
import pandas as pd
file1 = ? #fill hole (in)
df1 = pd.read_csv(file1)
file2 = ? #fill hole (in)
df2 = pd.read_csv(file2)
df3 = pd.concat([df1, df2], axis=1, sort=False)
filename = ? #fill hole (out)
df3.to_csv(filename, index=None)
```

Figure 2: Merging columns from two files with aligned data.



# Test Diversity to Find Bugs in JavaScript Engines

[under review]

- Two techniques:
  - Transplantation—Used tests from one engine into another
  - Differential testing—Generated JS inputs and run them on multiple engines
- Found over 50 bugs across 4 main JS engines

# Detecting Flaky Tests with Machine Learning

[under review]

- Flaky tests are tests that non-deterministically pass/fail based on the environment. E.g., tests with `Thread.sleep`
  - Very common and problematic
- Approach: use ML to learn patterns from large base of previously found flaky tests

# Willow--a visualization tool to teach programming

<https://github.com/pedro00dk/willow/>

<http://34.73.147.9/>

Language python +

1- class Node:  
2- def \_\_init\_\_(self, v):  
3- self.v = v  
4- self.prev = None  
5- self.next = None  
6-  
7- class LList:  
8- def \_\_init\_\_(self):  
9- self.head = None  
10- self.tail = None  
11- self.size = 0  
12-  
13- def append(self, v):  
14- temp = Node(v)  
15- if not self.head:  
16- self.head = temp  
17- self.tail = temp  
18- else:  
19- self.tail.next = temp  
20- temp.prev = self.tail  
21- self.tail = self.tail.next  
22- self.size += 1  
23-  
24-  
25- ll = LList()  
26- ll.append(input())  
27- ll.append(input())  
28- ll.append(input())  
29- ll.append(input())  
30- ll.append(input())  
31- print('done')

1  
2  
3

(1)  
(2)

module

Node LList \_\_init\_\_ append \_\_init\_\_ append \_\_init\_\_ append \_\_init\_\_ append

(4)

(5)

LList

head	2
tail	3
size	5

Node

1

Node

2

Node

3

Node

4

Node

5

# Other projects

- Synthesis of Network Intrusion Detection Rules
- Grey-box Combinatorial Interaction Testing
- Black-box Differential Generation of Adversarial Inputs for ML Classifiers

# Software Testing: From Practice to Research

Marcelo d'Amorim  
damorim@cin.ufpe.br



UNIVERSIDADE  
FEDERAL  
DE PERNAMBUCO

Rio Cuarto, Argentina, February 2020

<https://star.cin.ufpe.br/>

Considering to Visit (or do a MS/PhD with us)?

# STAR (Software Testing and Analysis Research)

Interested in preventing, discovering, diagnosing, and repairing software bugs

Faculty: Breno, Leopoldo, and Marcelo

Web: <http://star.cin.ufpe.br>



Leopoldo Teixeira



Breno Miranda

# Center of Informatics at UFPE





# Center of Informatics at UFPE



With 5 other programs, CIn-UFPE holds the highest classification of a PhD program in Brazil (7/7). According to [csindexbr.org](http://csindexbr.org) CIn-UFPE is 1<sup>st</sup> in number of \*selected\* publications in the area of SE. ~90 faculty, ~10 faculty in SE, 6 SE faculty in SPG+STAR.



# Informatics at UFPE



2x4-story buildings like this.  
Grad student's offices,  
classrooms, and admin offices.

Faculty offices, undergrad labs,  
and facilities of partners (Apple,  
Samsung, Motorola, and OKI).

# If you are curious...

- Short video about our department -> <https://tinyurl.com/wl68cvx>
- Short video about our group -> <https://tinyurl.com/u3ytuls>
- More info:
  - Email: damorim@cin.ufpe.br
  - Web: <http://www.cin.ufpe.br/~damorim>

# Considering working with us?



- About fellowship
  - Sources: Program pays (student ranks in the top-20s) XOR state agency pays (advisor needs to prepare proposal to state agency) XOR project pays
  - Stipend ~R\$2,600 free of taxes. OKish for living close to the University.
- Complements of funding
  - Lots (really) of different opportunities for summer internships abroad
  - Being a Teaching Assistant in a Professional Masters course (Once a year it pays ~2 salaries for a two-week job of work)

If you are still undecided... 😊



# Recife



~2 million people (metropolitan area)  
International flights from/to big cities worldwide  
straight flight Cordoba <-> Recife (once a week)  
Temperature: 22-32C





# Porto de Galinhas



- 1h drive by car
- shuttle options from airport
- venue of ICST 2021

<https://icst2021.icmc.usp.br/>



# Muro Alto



- Natural Pool
- 1Km to the south of Porto de Galinhas



# Maracaípe

- More isolated
- 1Km to the north of Porto de Galinhas





# Informatics at UFPE

An aerial photograph of a university campus. On the left is a modern white building with large glass windows. To its right is a large green tree. Further right is a red building with a blue roof. In the foreground, there is a paved area with several cars parked, including a red one and a blue one. A grey rectangular box with a black border is centered over the image, containing the text "Hope to see you soon!".

Hope to see you soon!