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

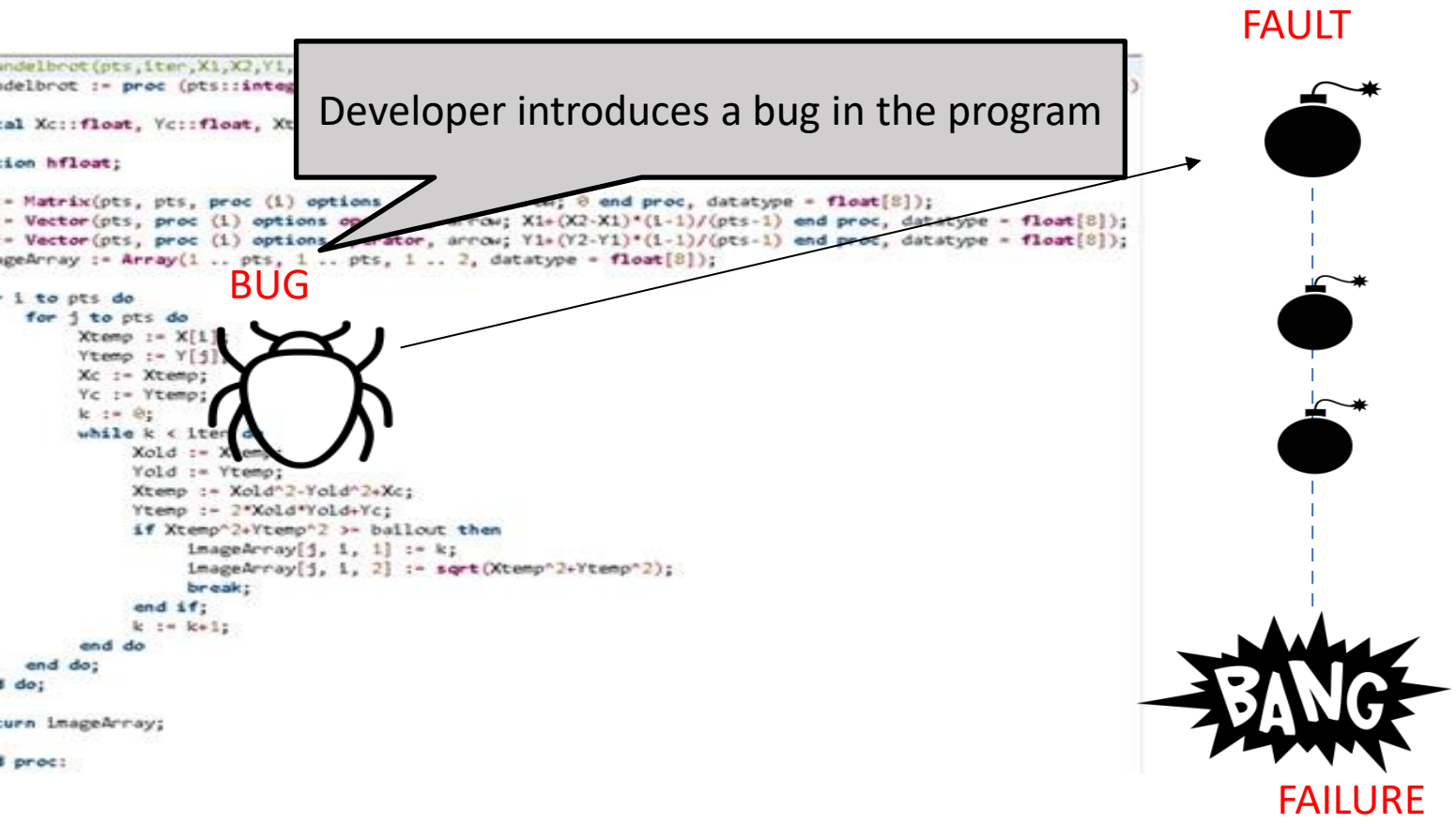
We use multiple languages to demonstrate concepts.

GitHub Repository

https://tinyurl.com/s4emvrg

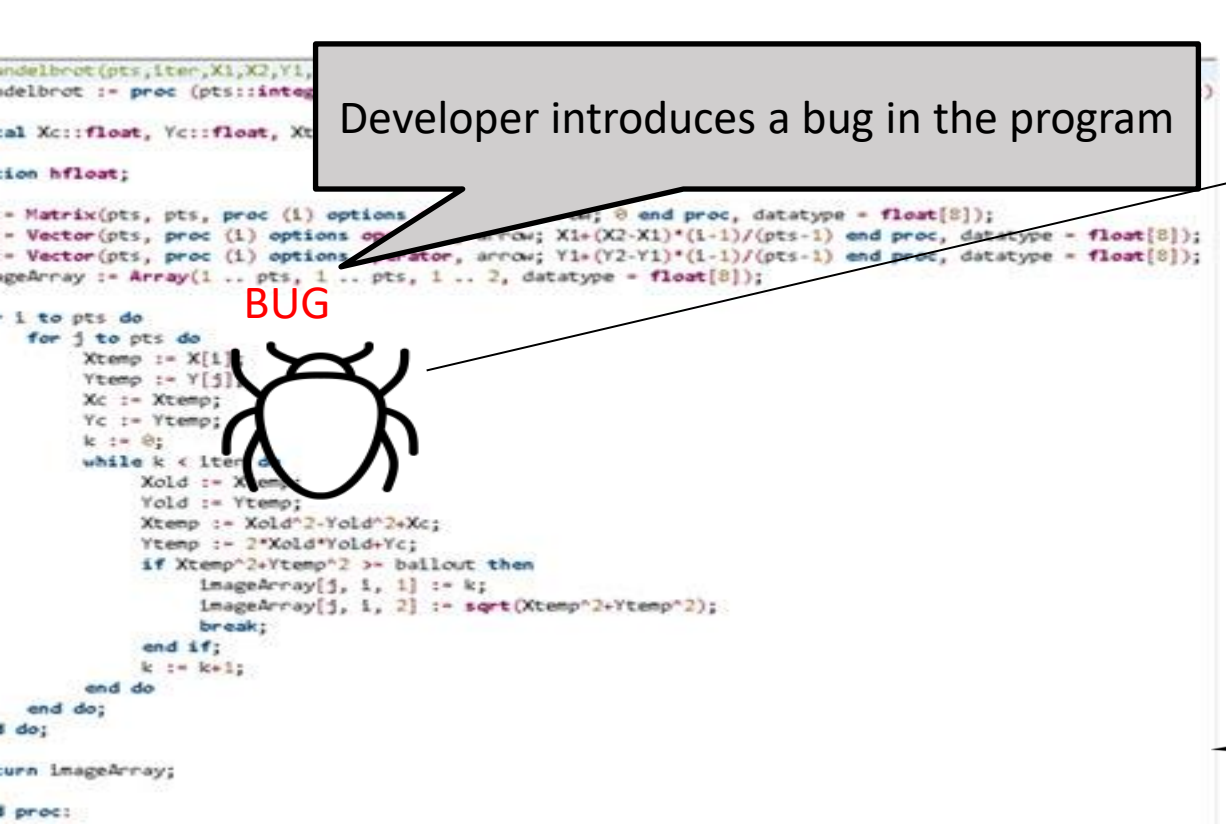
Preliminaries

The lifecycle of a bug



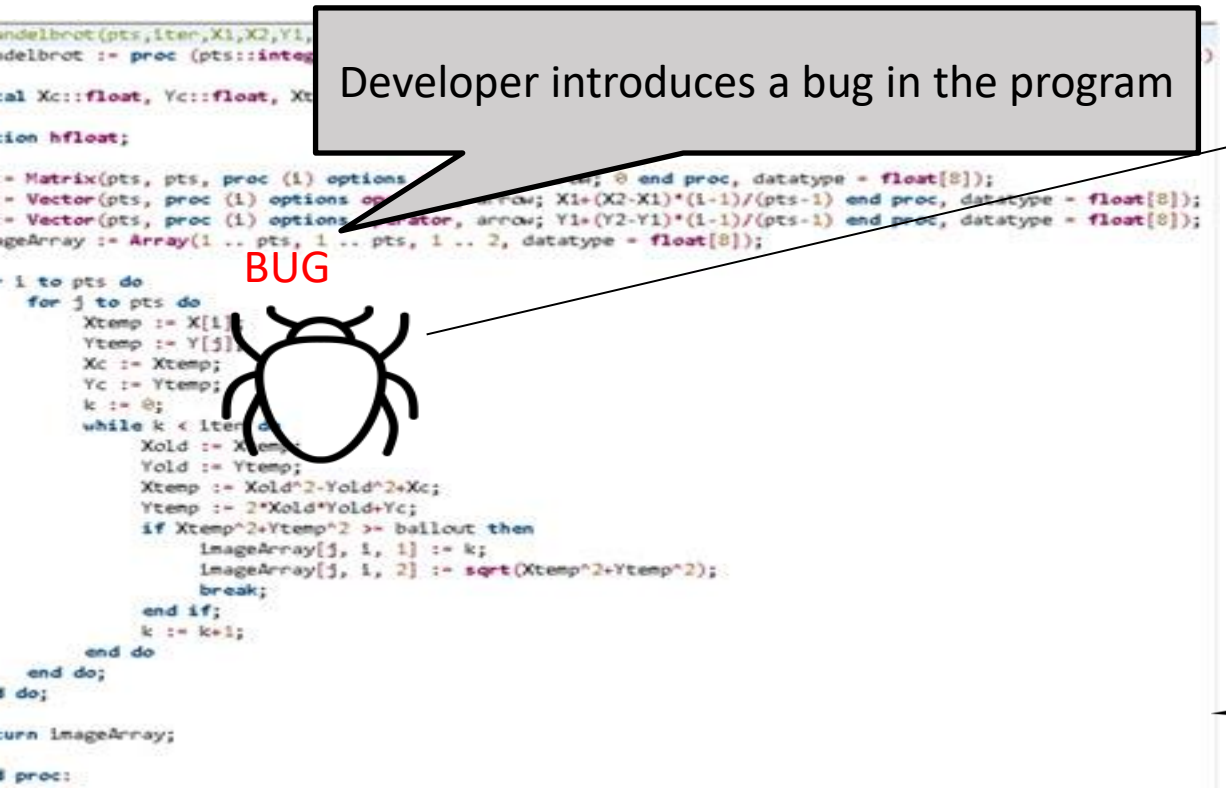
J. Voas, PIE: A Dynamic Failure-Based Technique, IEEE Transactions on Software Engineering. 1992

The lifecycle of a bug



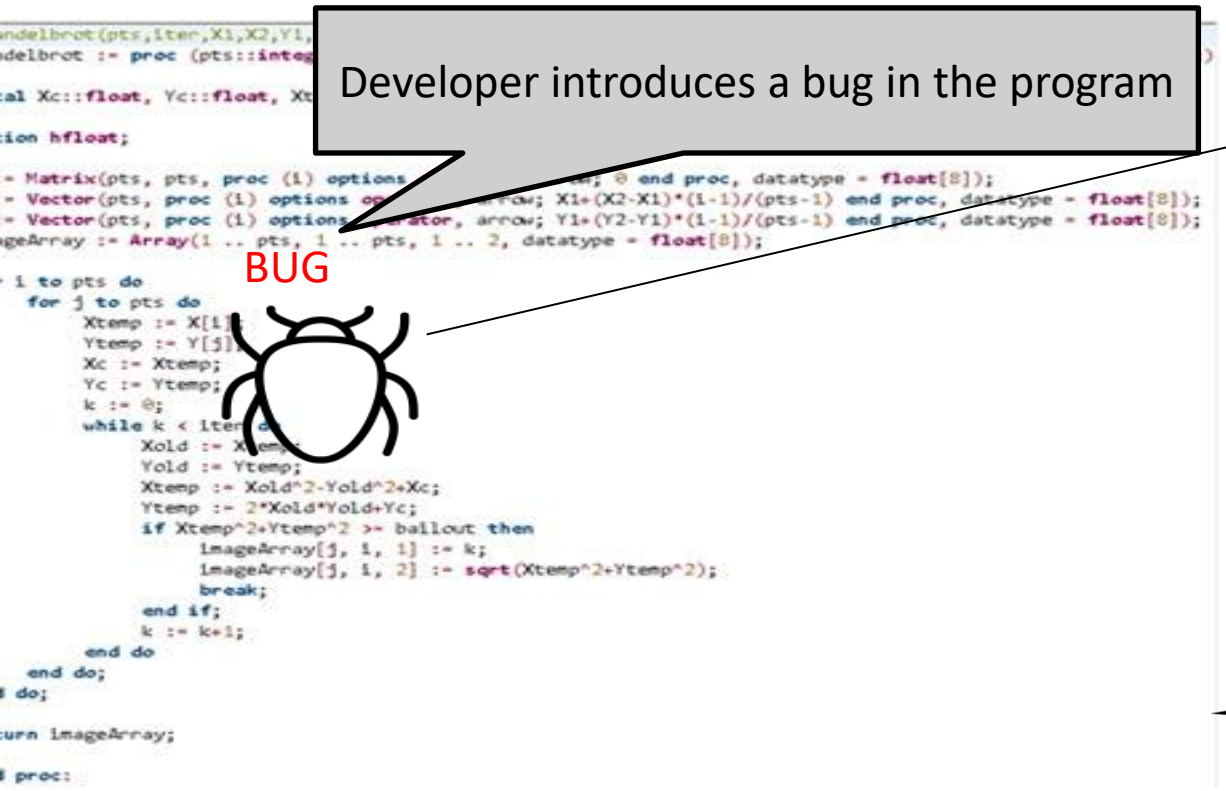
J. Voas, PIE: A Dynamic Failure-Based Technique, IEEE Transactions on Software Engineering. 1992

The lifecycle of a bug



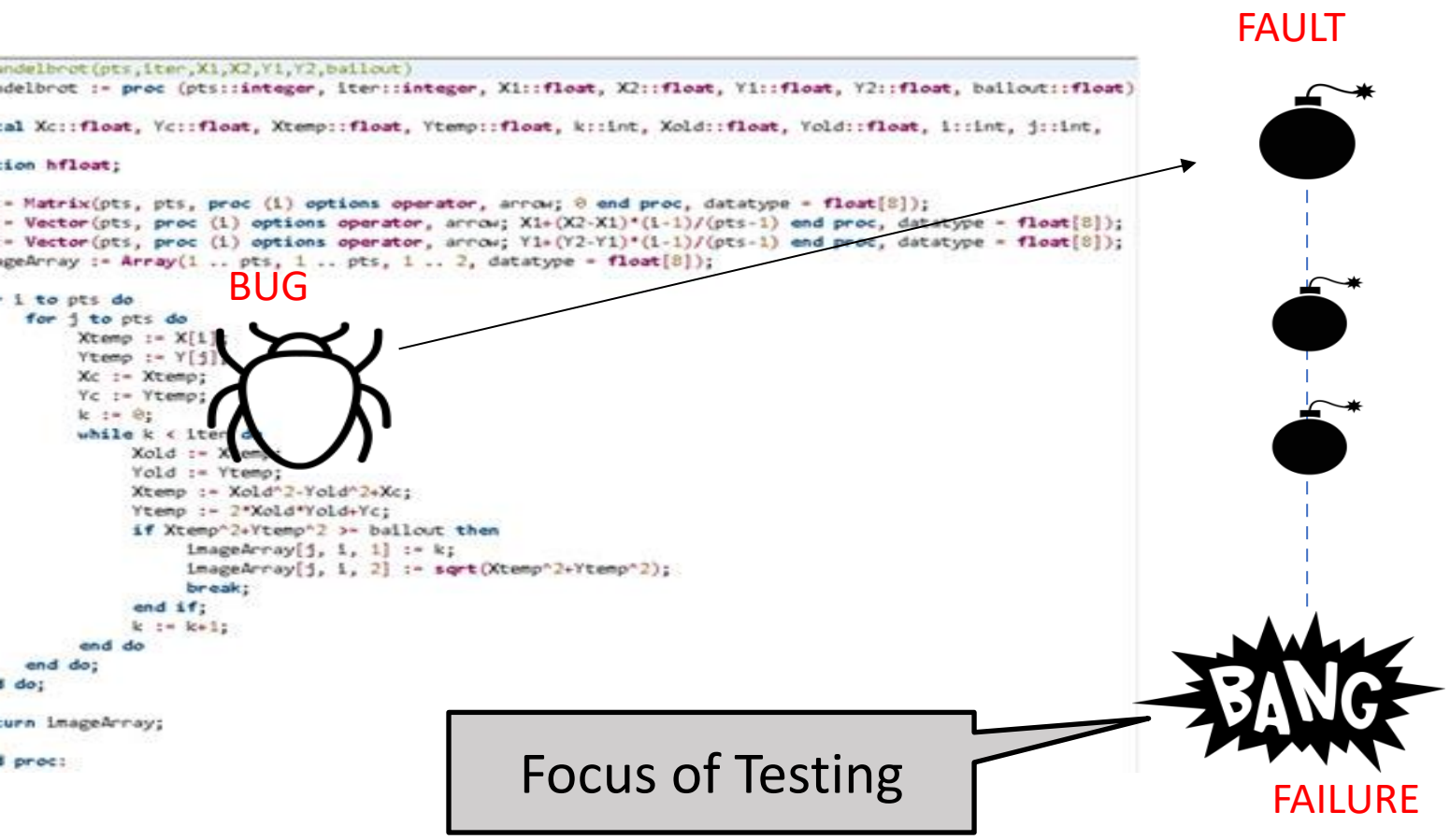
J. Voas, PIE: A Dynamic Failure-Based Technique, IEEE Transactions on Software Engineering. 1992

The lifecycle of a bug



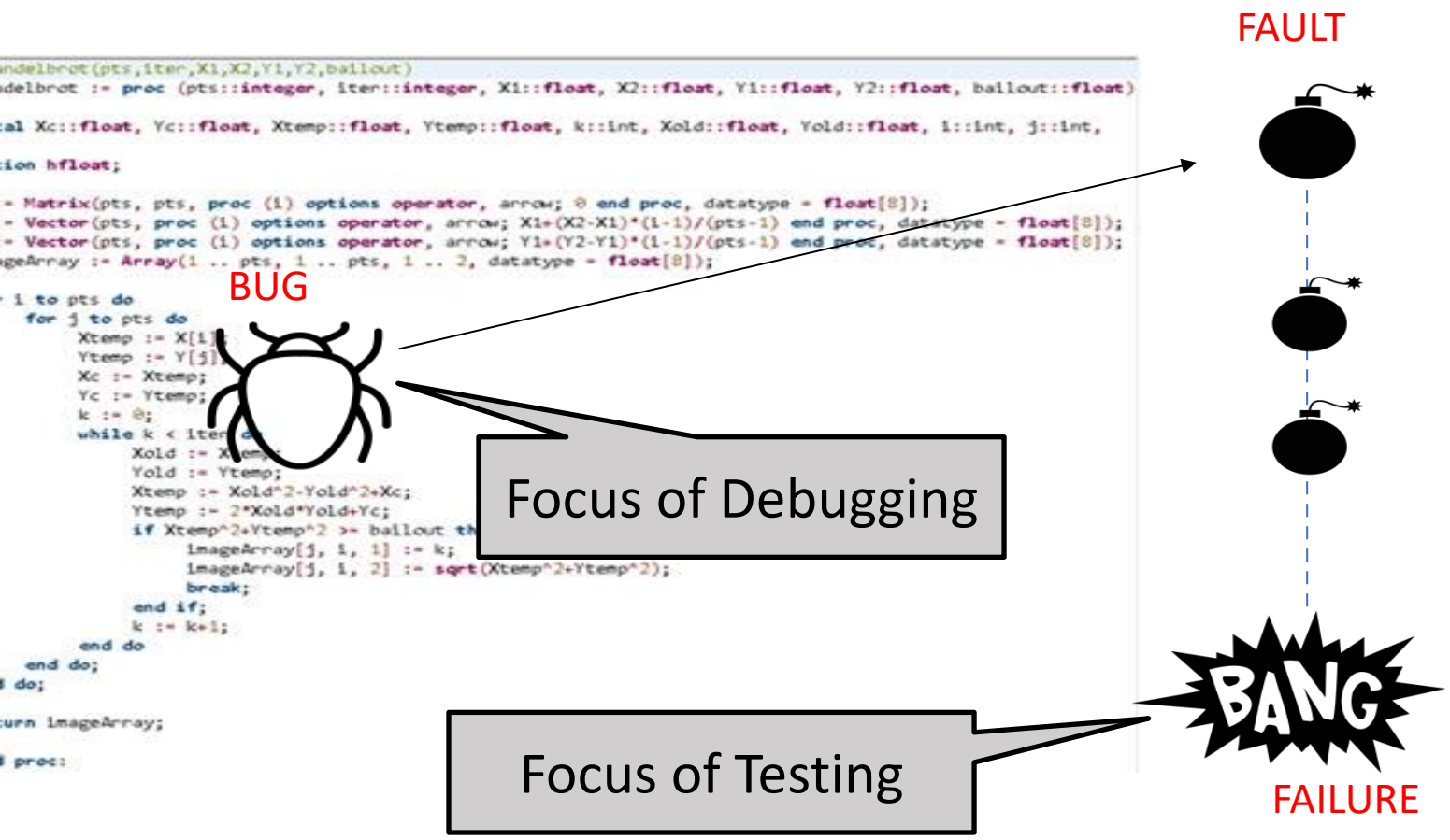
J. Voas, PIE: A Dynamic Failure-Based Technique, IEEE Transactions on Software Engineering. 1992

The lifecycle of a bug



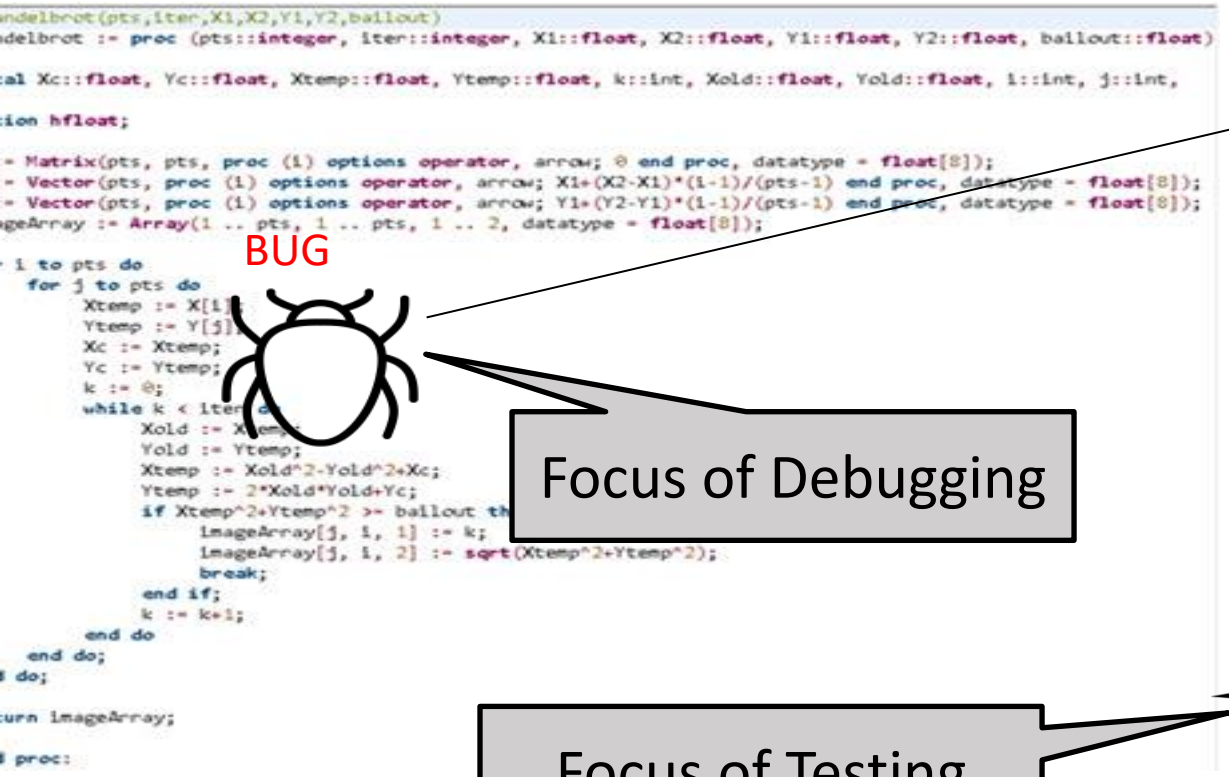
J. Voas, PIE: A Dynamic Failure-Based Technique, IEEE Transactions on Software Engineering. 1992

The lifecycle of a bug



J. Voas, PIE: A Dynamic Failure-Based Technique, IEEE Transactions on Software Engineering. 1992

The lifecycle of a bug



BUG

Focus of Debugging

Focus of Testing

FAULT

BANG

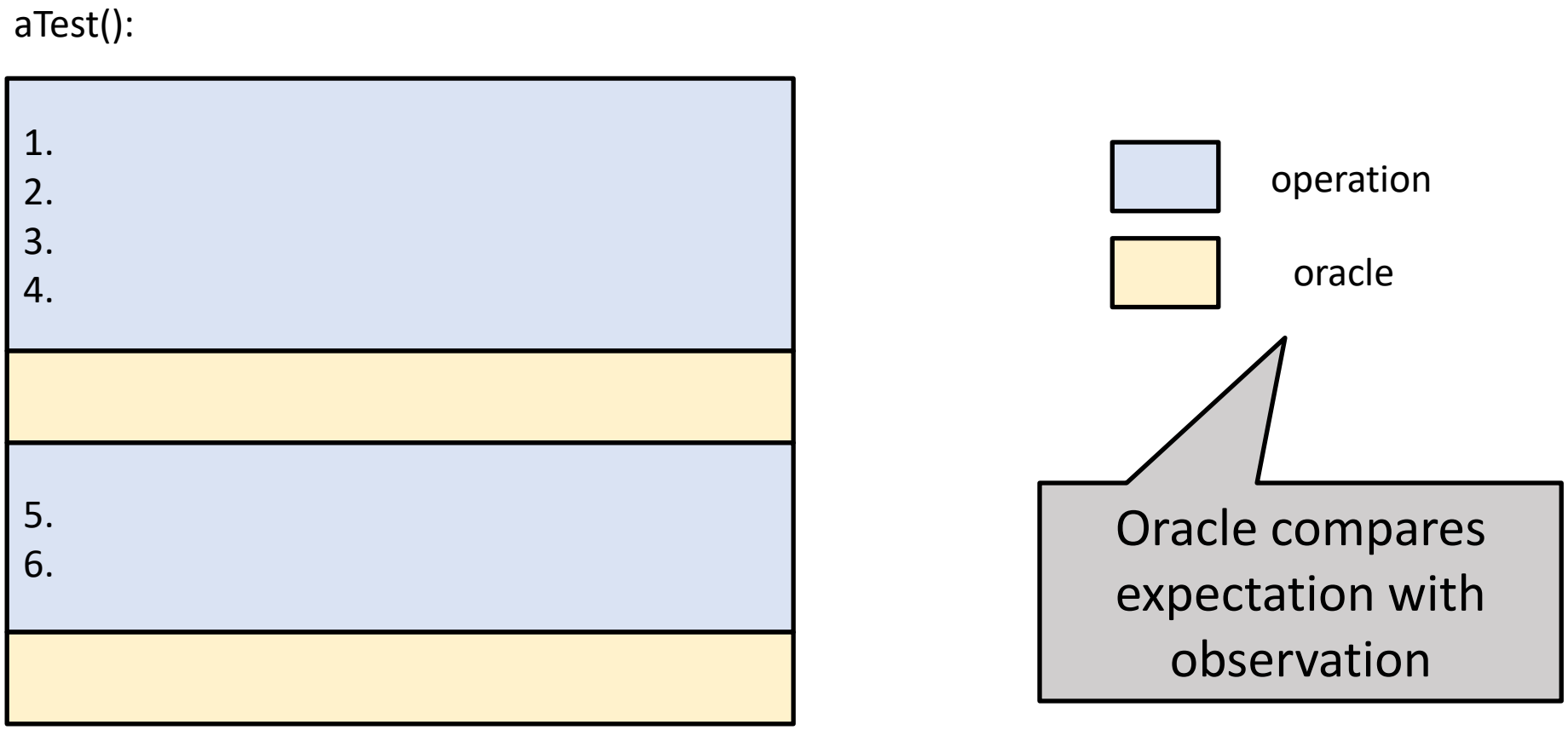
FAILURE

Scientific Debugging:

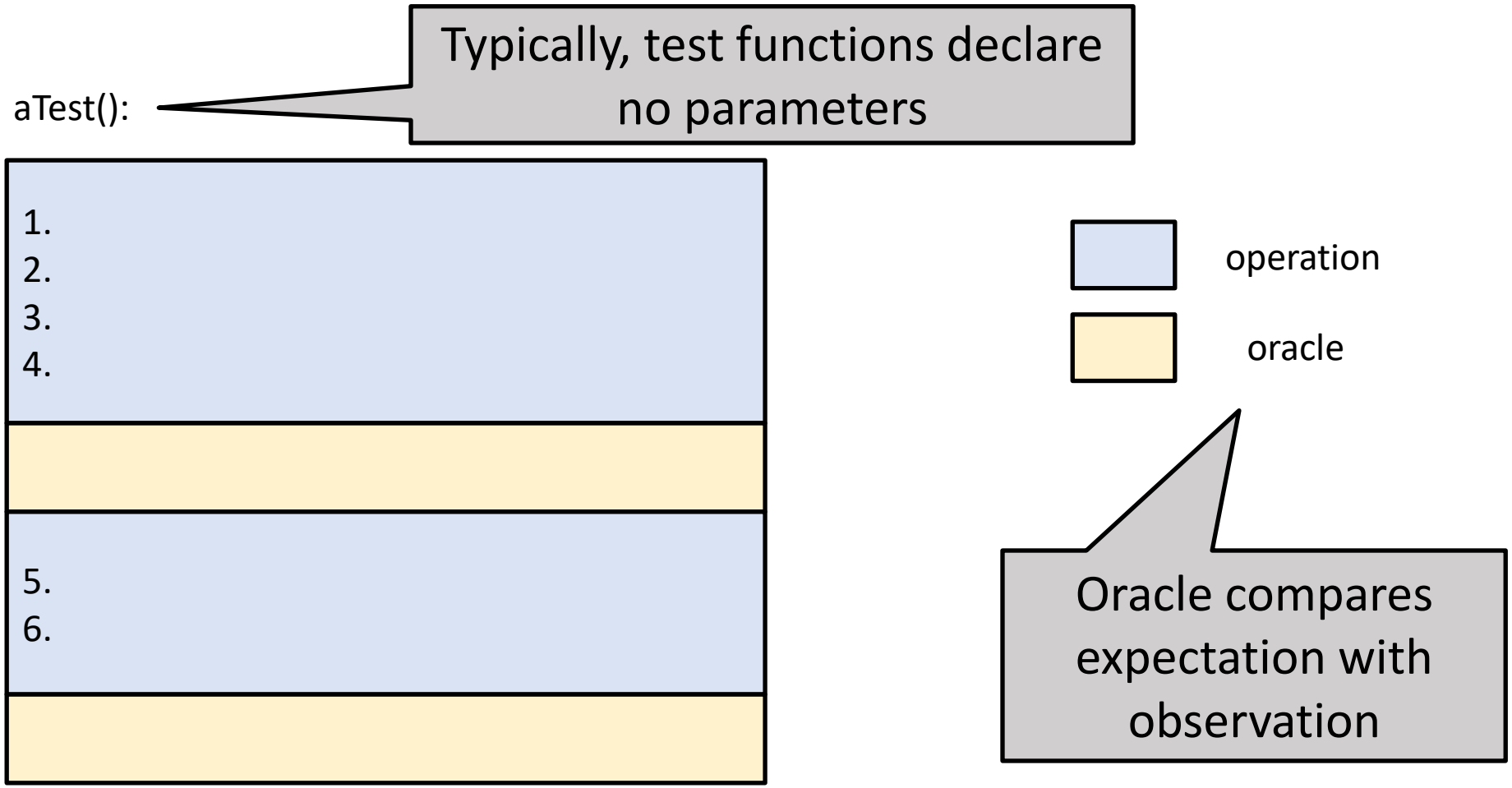
1. Observation (e.g., failure)
2. Build Hypothesis
3. Predict behavior based on 2
4. Make an experiment
5. Observation
6. If hypothesis inconsistent
   then goto 2 (revise)
   else fix code

J. Voas, PIE: A Dynamic Failure-Based Technique, IEEE Transactions on Software Engineering. 1992

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



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



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 Ca

- Top-down: As soon as you know your requi
  - 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 Ca

- Top-down: As soon as you know your requi
  - 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.

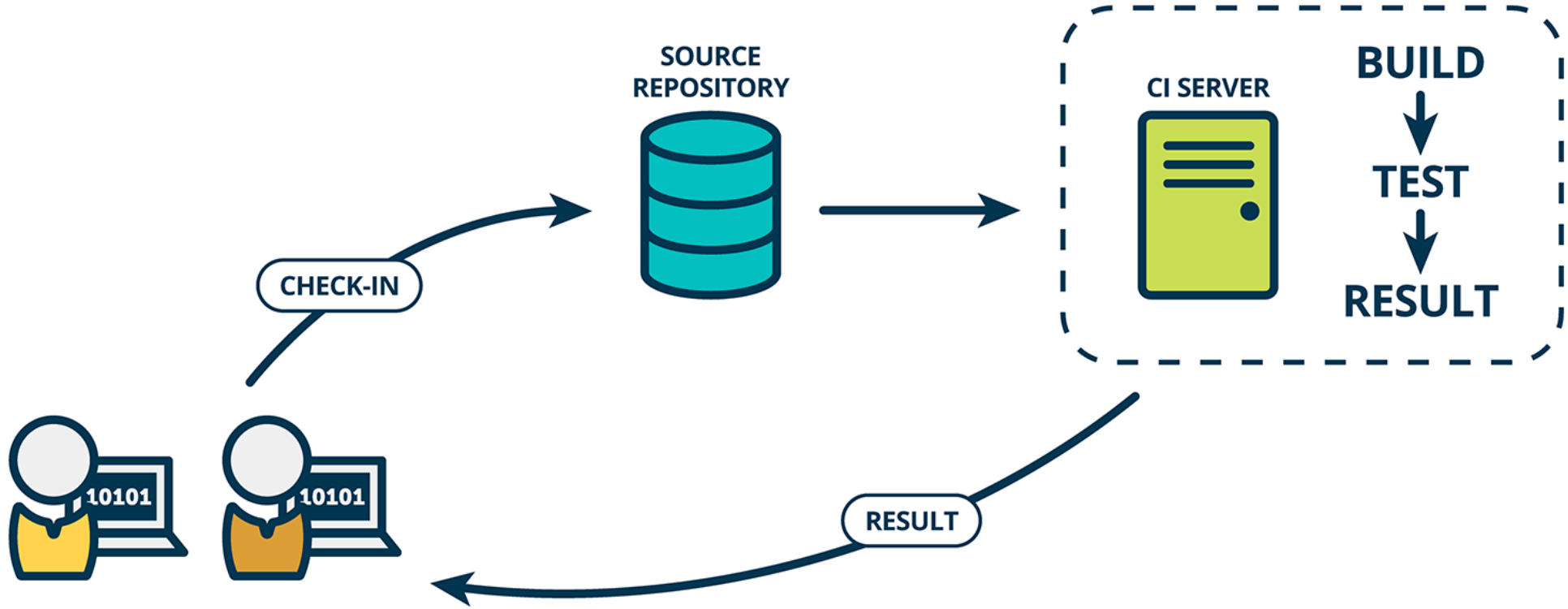
Regression Testing

- 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" you 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

```
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;
}
```

What can happen if pow is called with negative y?

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 = y = 2

2^-2 = 1/(2^2) = 1/4

So, pow should return 1 (note integer arithmetic)

Instead, it returns 1 * 2 = 2

Example: Power

```
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;
}
```

What can happen if pow is called with negative y?

Consider x = y = 2

2^-2 = 1/(2^2) = 1/4

So, pow should return 1 (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 ==
      r = x
    }
  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 \old(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

boolean PERM(int[] ar1, int[] ar2) // returns true iff ar1 and ar2 are
                                   // permutations of the same set
```

(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
- Build systems

Test Infrastructure (to create and run tests)

- Coverage
- Mutation testing

Test adequacy

- Combinatorial testing
- UI testing

System 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
- Variou
  - E.g., make, ant, maven, gradle

Practice – Section 2 in our repo

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

- 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
  - …

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

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
  -
  - e test suite
  - …

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?

* https://personal.utdallas.edu/~ewong/SE6367/03-Lecture/29-A-Combinatorial-Testing-by-Kuhn.pdf

Combinatorial Interaction Testing (CIT) *

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

34 switches =
2^32 combinations of inputs =
1.7 x 10^10 inputs/tests

* https://personal.utdallas.edu/~ewong/SE6367/03-Lecture/29-A-Combinatorial-Testing-by-Kuhn.pdf

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

* https://personal.utdallas.edu/~ewong/SE6367/03-Lecture/29-A-Combinatorial-Testing-by-Kuhn.pdf

CIT – Pairwise Testing

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

* https://personal.utdallas.edu/~ewong/SE6367/03-Lecture/29-A-Combinatorial-Testing-by-Kuhn.pdf

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

* https://personal.utdallas.edu/~ewong/SE6367/03-Lecture/29-A-Combinatorial-Testing-by-Kuhn.pdf

CIT – Pairwise Testing

Orientation	landscap
Size	1080x19
OS	iPhon

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

* https://personal.utdallas.edu/~ewong/SE6367/03-Lecture/29-A-Combinatorial-Testing-by-Kuhn.pdf

CIT – Pairwise Testing

Orientation	landscap
Size	1080x19
OS	iPhon

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

Exercise 2

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

+ Req
+ Ver
- Can be difficult to maintain
- Can be slow to execute
- Requires special environment (e.g., cloud)

Practice – Section 6 in our repo

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
  - Gram

Practice – Section 8 in our repo

AFL)

(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?

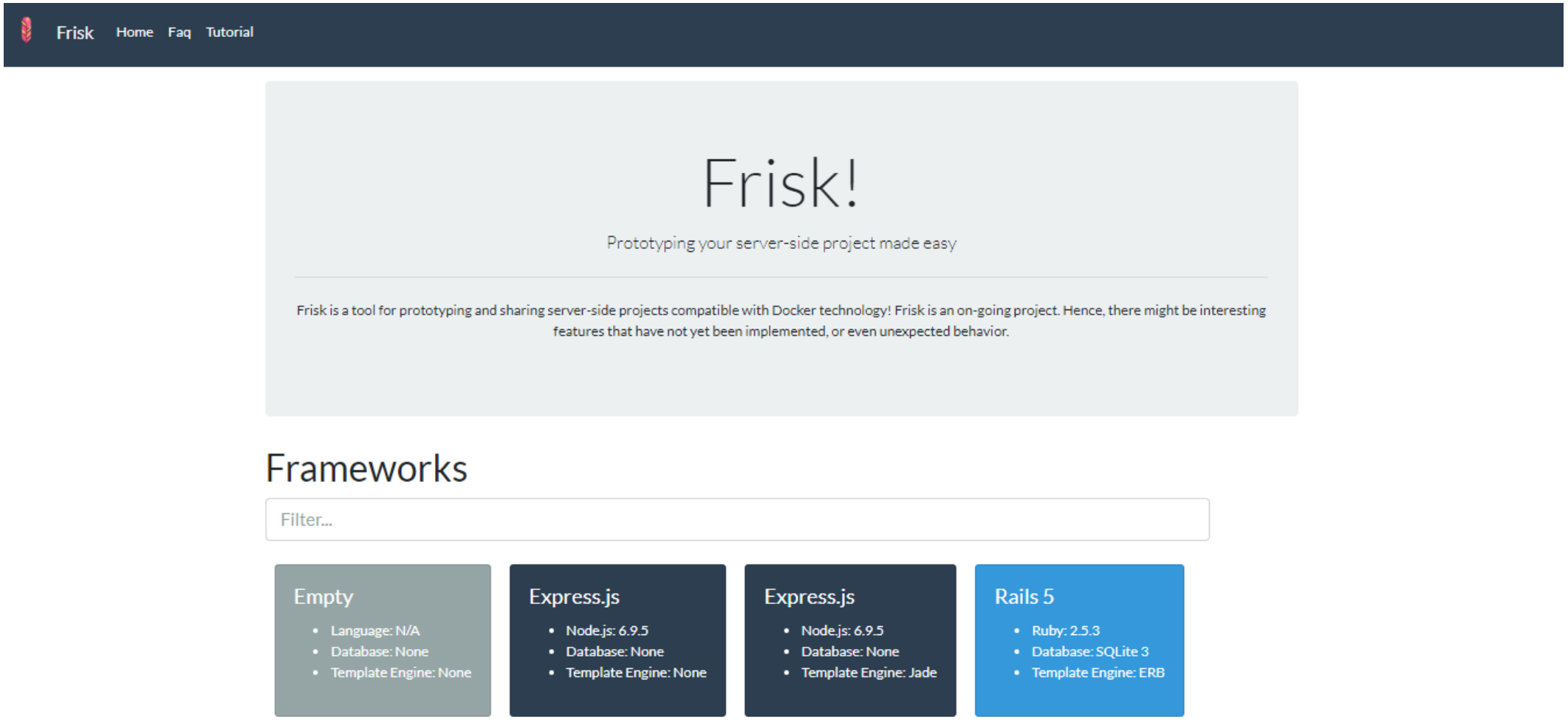
Recent Research Projects

Detection of plugin conflicts in CMS

[Information and Software Technology, Feb. 2020]

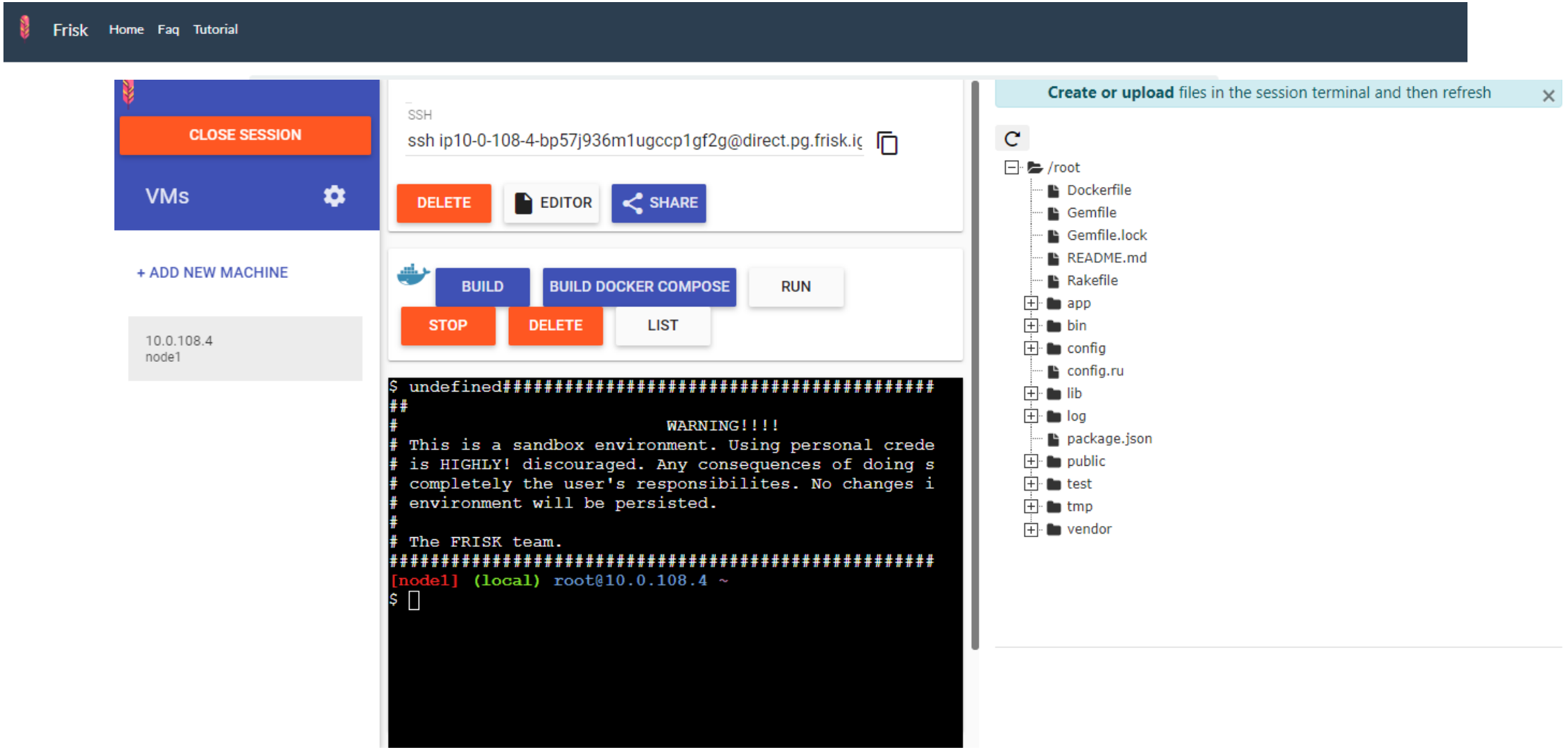
Using Docker to reproduce StackOverflow posts

[IEEE Transactions on Software Engineering, Dec. 2019]



Using Docker to reproduce StackOverflow posts

[IEEE Transactions on Software Engineering, Dec. 2019]



Visual Sketching

[ICSE-NIER 2020]

- ..

Test diversity to find bugs in JavaScript engines

[under review]

- ..

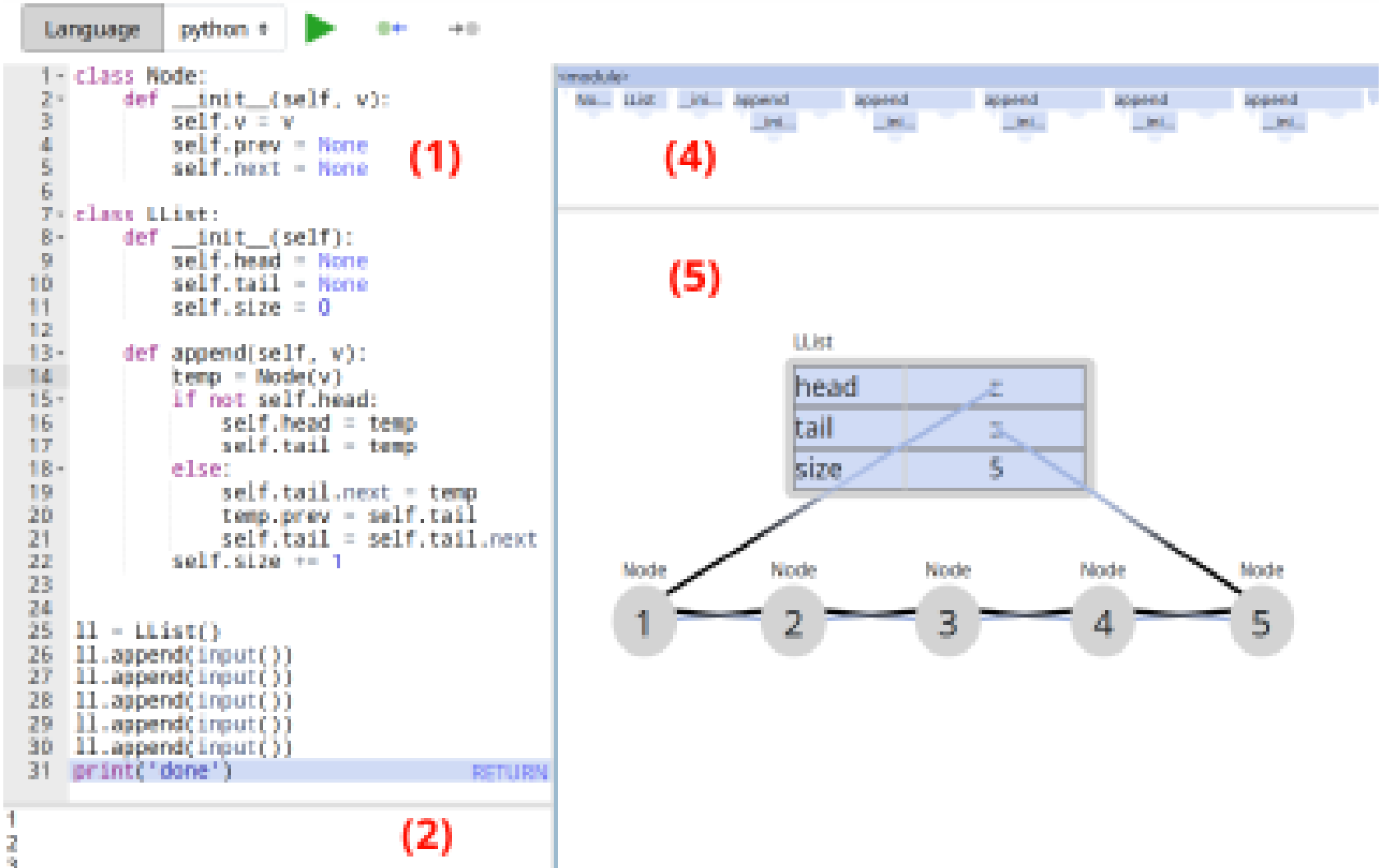
Detecting Flaky Tests with Machine Learning

[under review]

- ..

Willow--a visualization tool to teach programming

https://github.com/pedro00dk/willow/ http://34.73.147.9/



Synthesis of Rules for Network Intrusion Detectors

- ..

Grey-box Combinatorial Interaction Testing

- ..

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 CIn-UFPE is 1st 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?

1st entry	NOV	MAR	application
2nd entry	MAY	JUL	enrollment

- 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)

Porto de Galinhas

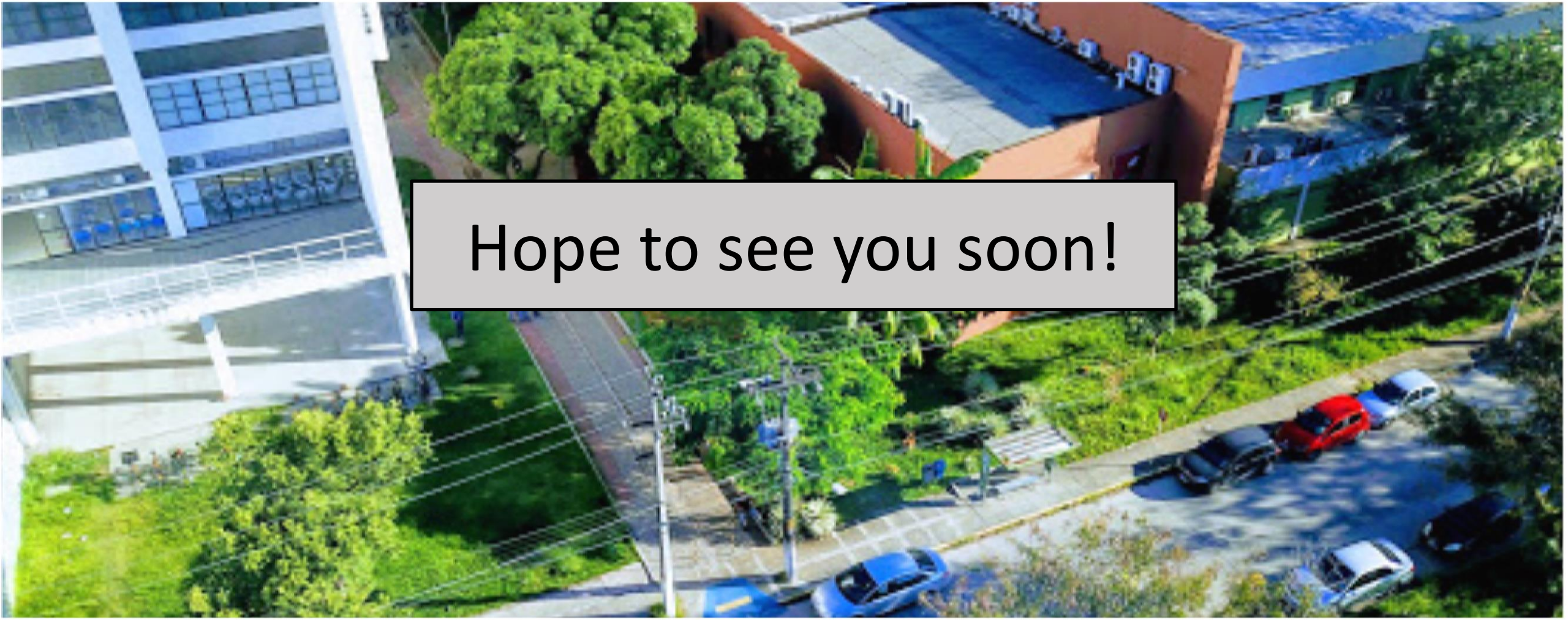
- 1h drive by car
- shuttle options from airport
- venue of ICST 2021

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