CS 5154: Software Testing

Foundations

Check-in & Announcements

- Did you get access to the textbook?
- Reading 1 will be assigned after class (Canvas & Gradescope)
- Next ~2 classes: Hands-on lessons on Test Automation

Earlier in this course…

Testing is usually the last line of defense against bugs

But what exactly is a “bug”?

What is a “bug” in this program?

```
// count no. of “0” elements in x
public static int numZero (int[] x) {
  int count = 0;
  for (int i = 1; i < x.length; i++) {
    if (x[i] == 0) count++;
  }
  return count;
}
```

In this program, “bug” could mean…

Should start searching at 0, not 1

```
public static int numZero (int[] x) {
 int count = 0;
 for (int i = 1; i < x.length; i++) {
  if (x[i] == 0) count++;
 }
 return count;
}
```

i is 1, not 0, on the first iteration

Test 1
[2, 7, 0]
Expected: 1 Actual: 1

count is 0, instead of 1, at the return statement

Test 2
[0, 2, 7]
Expected: 1 Actual: 0

Building shared terminology in CS 5154

- **Fault** : static defect in the code
- **Error :** incorrect internal state caused by a fault
- **Failure** : observed behavior ≠ expected behavior

Faults of commission vs. Faults of omission

```
// count no. of “0” elements in x
public static int numZero (int[] x) {
  int count = 0;
  for (int i = 1; i < x.length; i++) {
    if (x[i] == 0) count++;
  }
  return count;
}
```

Why is this shared terminology important?

- Show off the knowledge you gained in CS 5154 ☺
- Be on the same page in CS 5154
- We will build on these terminologies
- Software testing industry standard terminologies

Example: identify a fault, error, failure

```
// compute arithmetic mean of elements in array
double avg(double[] nums) {
  int n = nums.length; double sum = 0; int i = 0;
  while (i<n)
    sum = sum + nums[i];
    i = i + 1;
  double avg = sum / n;
  return avg;
}
```

The faults that caused major failures

Failure	Impact	Fault
NASA's Mars lander	$125,000,000 satellite lost	No Pound/Newton conversion
THERAC-25	6 patients died	Several: see link
Ariane 5 explosion	$7,500,000,000 lost	Exception-handling fault (64-bit to 16-bit conversion)
Northeast blackout	50 million people lost power in US and Canada, $6,000,000,000 lost	Buffer overflow in monitoring system

Questions about Faults, Errors, Failures

?

In software testing, we write tests to find faults before those faults find the users

Recall: what is a test?

```
public static int numZero (int[] x) {
  int count = 0;
  for (int i = 1; i < x.length; i++) {
    if (x[i] == 0) count++;
  }
  return count;
}
```

Test 1
[2, 7, 0]
Expected: 1 Actual: 1

Test 2
[0, 2, 7]
Expected: 1 Actual: 0

Some components of a test

- **Test Case Values**: input data needed to execute the code under test
- **Expected Results:** output that is produced if the code is correct
- **Test Oracle:** decides if observed output match expected output

Last lecture: why “well-tested” software fails?

Why does “well-tested” software fail?

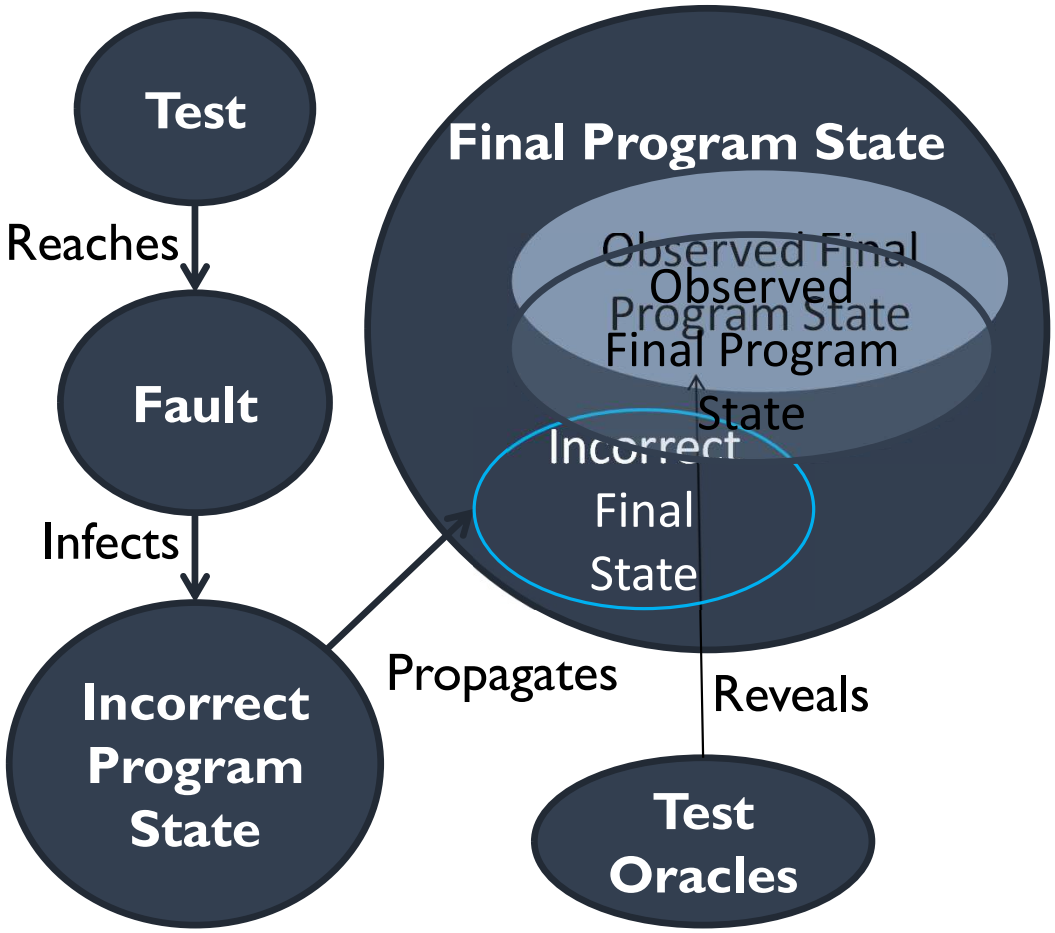
- Are the tests effective for finding faults?
- Can testing guarantee the absence of failures?
- Is the software really “well tested”?
- Has the testing been done with the right goals?

A test is effective if it…

1. **R**eaches program location(s) that contain a fault
2. **I**nfects the program state after executing a faulty location
3. **P**ropagates the infected state into incorrect output
4. **R**eveals part of the incorrect output to the test oracle

RIPR fault/failure model of test effectiveness

- **R**eachability
- **I**nfection
- **P**ropagation
- **R**evealability

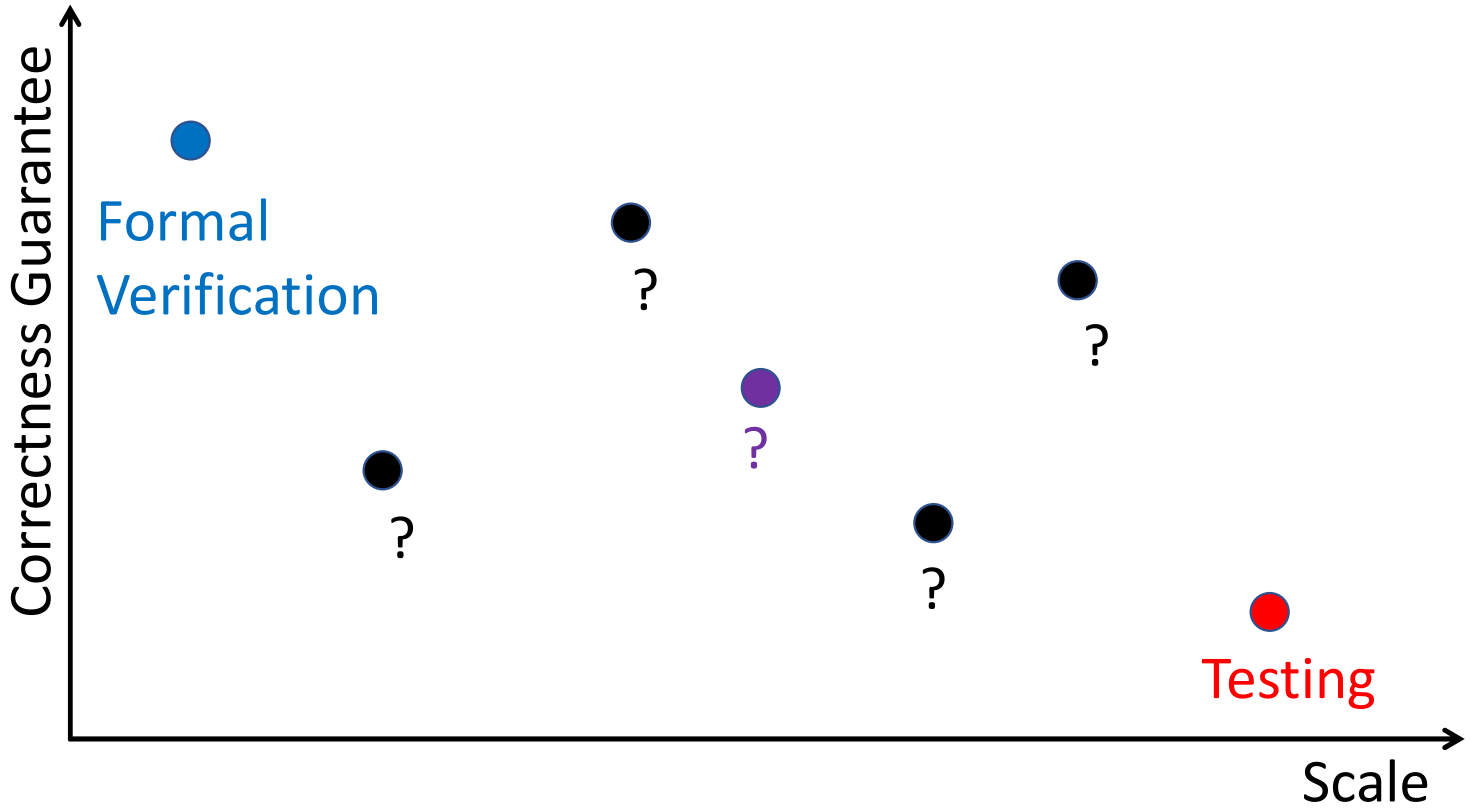


We will use the RIPR model to learn how to write effective tests

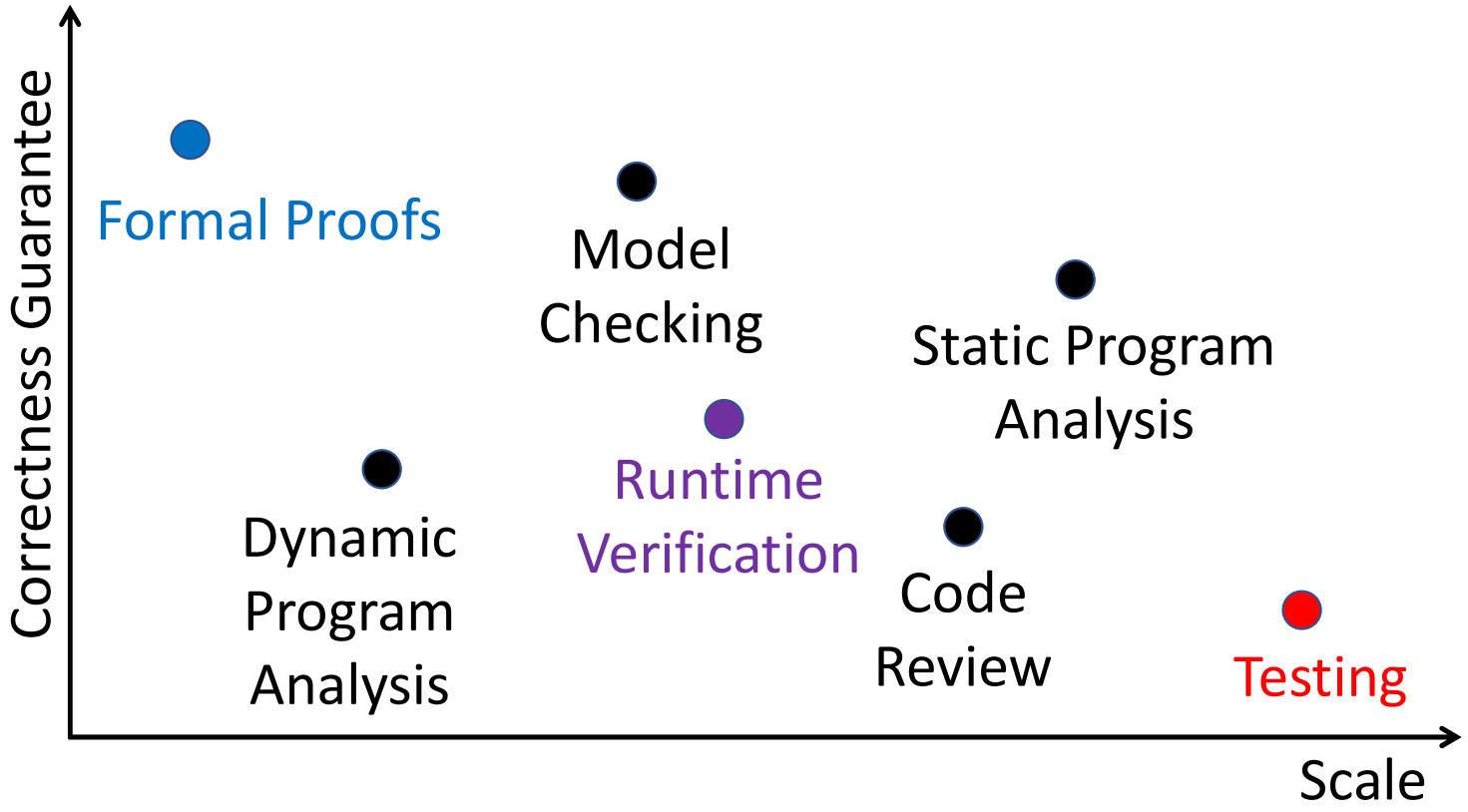
A fundamental limitation of software testing

- Claim: testing can only show the presence of failures, not their absence
- Is this claim true?
- Lesson: testing is one of many tools for improving software quality

Other software quality assurance techniques



Other software quality assurance techniques



Is software really “well-tested”?

- Testers use coverage criteria to measure how well-tested software is
- What are some coverage criteria that you know?

Coverage criteria: pros

- Provides a way to know when to stop testing
- Can be continuously measured during regression testing
- Maximize the “bang for the buck”
  - find the **fewest tests** that will find the **most faults**

Coverage criteria: cons

- Some criteria are “weaker” than others
- Strong criteria are harder to achieve or more expensive
- HUNDREDS of criteria have been proposed!
- Many developers are not trained in test design ☹

Discuss: how to create effective tests?

```
/** Return first index of Node n in path, or
 * -1 if n is not present in path */
public int indexOf (Node n, List<Node> path){
  for (int i=0; i < path.size(); i++){
    if (path.get(i).equals(n))
      return i;
  }
  return -1;
}
```

How would you go about producing effective test cases for this method?

“We cannot solve our problems with the same thinking that we used when we created them”

- Albert Einstein (?)
- Yogi Bera (?)

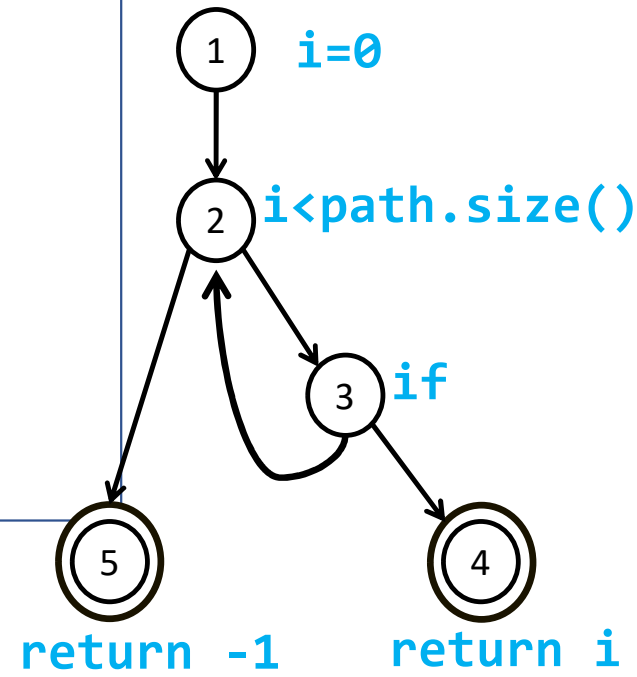
“It is difficult to create effective tests if we only look at code. We need a higher level of abstraction”

- Offutt and Ammann

Producing effective tests for indexOf (1)

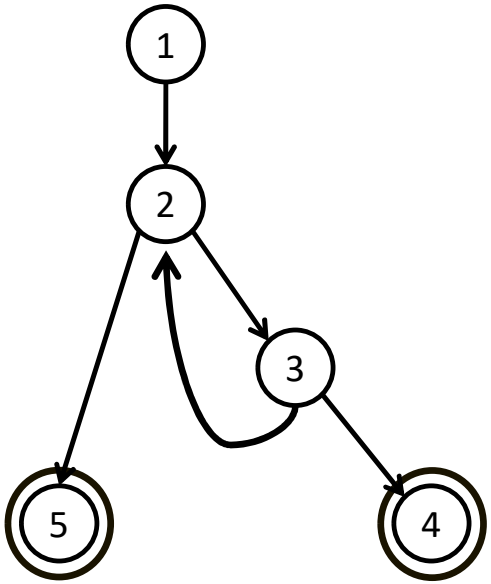
```
int indexOf (Integer n, List<Integer> path){
  for (int i=0; i < path.size(); i++){
    if (path.get(i).equals(n))
      return i;
  }
  return -1;
}
```

Control Flow Graph



Producing effective tests for indexOf (2)

Graph: abstract version



Edges
1 2
2 3
3 2
3 4
2 5
Initial Node: 1
Final Nodes: 4, 5

6 requirements for Edge-Pair Coverage
1. [1, 2, 3]
2. [1, 2, 5]
3. [2, 3, 4]
4. [2, 3, 2]
5. [3, 2, 3]
6. [3, 2, 5]

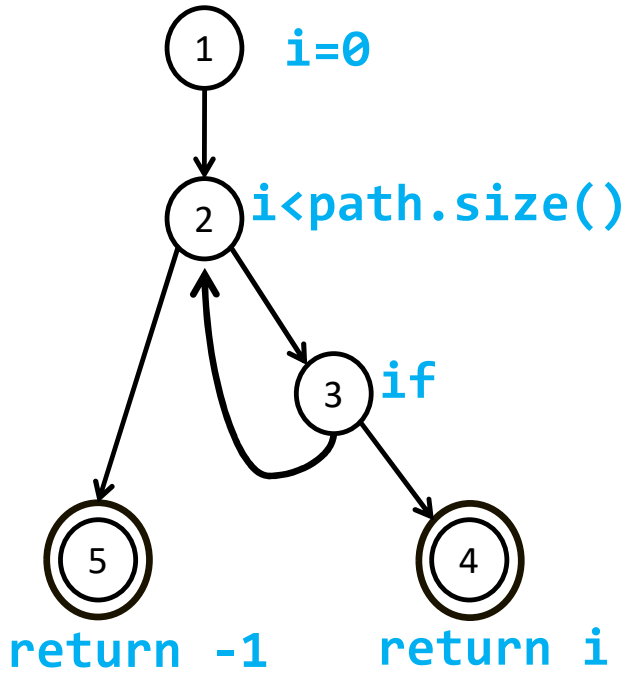
Test Paths
[1, 2, 5]
[1, 2, 3, 2, 5]
[1, 2, 3, 2, 3, 4]

Work with your neighbor

- Write input values that satisfy the Edge-Pair coverage requirements

```
/**Return first index of Integer n in path, or
 * -1 if n is not present in the path */
int indexOf (Integer n, List<Integer> path){
  for (int i=0; i < path.size(); i++){
    if (path.get(i).equals(n))
      return i;
  }
  return -1;
}
```

Test Paths
[1, 2, 5]
[1, 2, 3, 2, 5]
[1, 2, 3, 2, 3, 4]



Question: is indexOf now well-tested?

We just saw Test Design in action

- Test Design: a process for creating effective tests
- A major ingredient towards becoming a great tester
- The most **mathematical** and **technically challenging** testing activity
  - Requires knowledge of discrete math: graphs, sets, relations, etc.

The steps in test design

1. Do math or analysis to obtain test requirements
2. Find input values that satisfy the test requirements
3. Automate the tests
4. Run the tests
5. Evaluate the tests

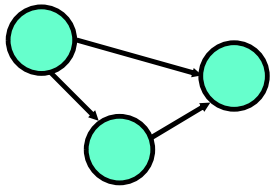
In CS5154: Model-Driven Test Design

- We will do test design w.r.t. four models of software

Input Domains

A: {0, 1, >1}
B: {600, 700, 800}
C: {cs, ece, is, sds}

Graphs



Logic Expressions

(not X or not Y) and A and B

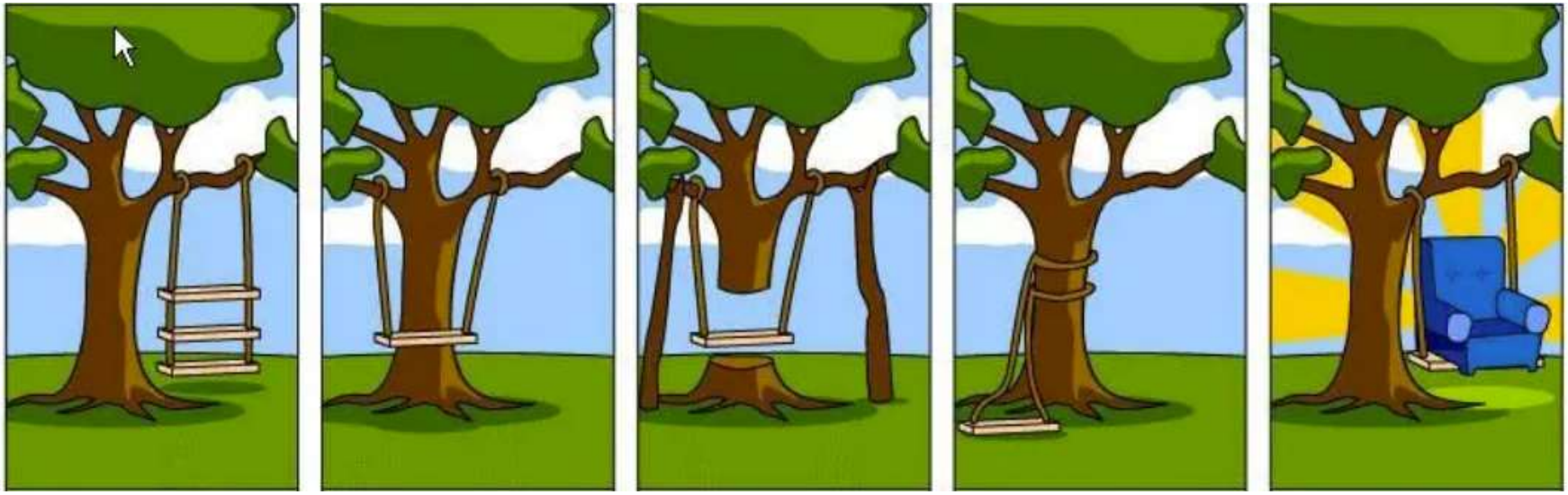
Syntax

```
if (x > y)
   z = x - y;
else
  z = 2 * x;
```

- The first part of the course and the textbook cover MDTD

MDTD is about DESIGN

- Multiple test designs may exist for the same code



- Considering cost/benefit tradeoffs in designs is an essential part of SE

Image Credits: https://blogs.perficient.com/2011/07/22/how-to-build-a-tire-swing-a-case-for-agile-development/

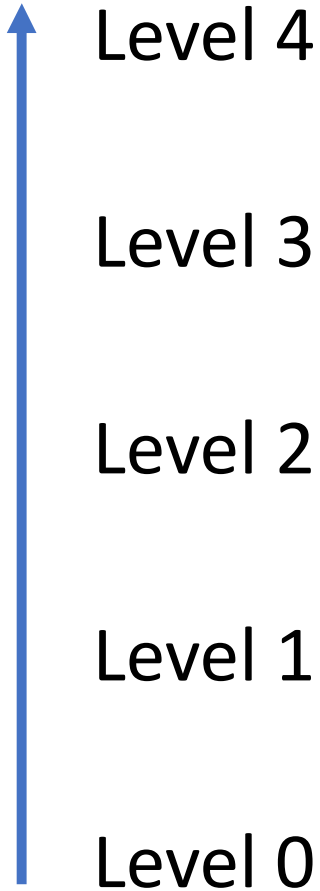
Why should you care about MDTD?

- Organize HUNDREDs of criteria around four models of software

- Develop a disciplined approach to engineering your tests
  - What's the difference btw a programmer and a software engineer?

- Develop "testing as a mental discipline" mindset (level 4)

Testing goals at different levels of maturity



Level 0 thinking

- Purpose: show that program runs on arbitrary/provided inputs
- Debug the program if it does not work on said inputs
- Problem: incorrect programmer behavior vs. programmer mistakes?

Level 1 thinking

- Purpose: use tests to show that a program is correct
- Problems:
  - If there are no failures, is software good or tests are not effective?
  - When to stop testing? (testing cannot prove programs correct)

Level 2 thinking

- Purpose: use tests to show that a program is incorrect

- Problems:
  - Can lead to adversarial relationship among developers ☹
  - What if the tests do not fail?

Level 3 thinking

- Purpose: team-based approach to reducing risk of software failures
- Problems:
  - Testing is the only way to improve software quality
  - Focuses on software, not on developers that write software

Level 4 thinking

- Purpose: testing as a mental discipline that improves software quality
- Effects:
  - Improve the ability of developers to write high-quality software
  - Invest in continued quality measurement and improvement
  - Make testers part of project leadership

Poll: what level of testing maturity are you at?

- Level 0: testing == debugging
- Level 1: testing is done to show program correctness
- Level 2: testing is done to show that software does not work
- Level 3: testing is done to reduce the risk of using software
- Level 4: testing is a mental discipline that helps build high-quality software

Some goals of CS 5154

- Moving you (and your organization) towards Level 4 thinking

- Teach you to be “change agents” who advocate for Level 4 thinking

What we learned

- Standard testing terminology (test, fault, error, failure)
- Conditions that effective tests must meet (the RIPR model)
- Fundamental limit of software testing
- Introduction to model-driven test design
- Levels of test maturity

Next

- Test Automation