CS/CE/SE 6367
Software Testing, Validation and Verification

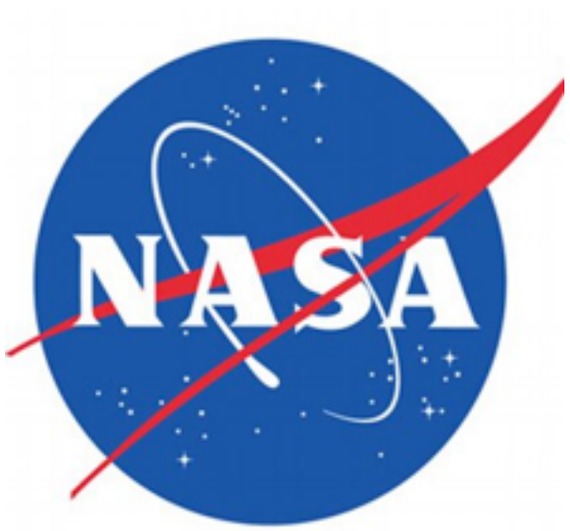
Lecture 01
Introduction

Who am I?

- Instructor Name: Lingming Zhang
- Office: ECSS 4.205
- Email: lingming.zhang@utdallas.edu
- Homepage: http://www.utdallas.edu/~lxz144130/

Education Background

Industry Collaboration



NEC Laboratories America

Relentless passion for innovation

Microsoft Research

What am I doing?

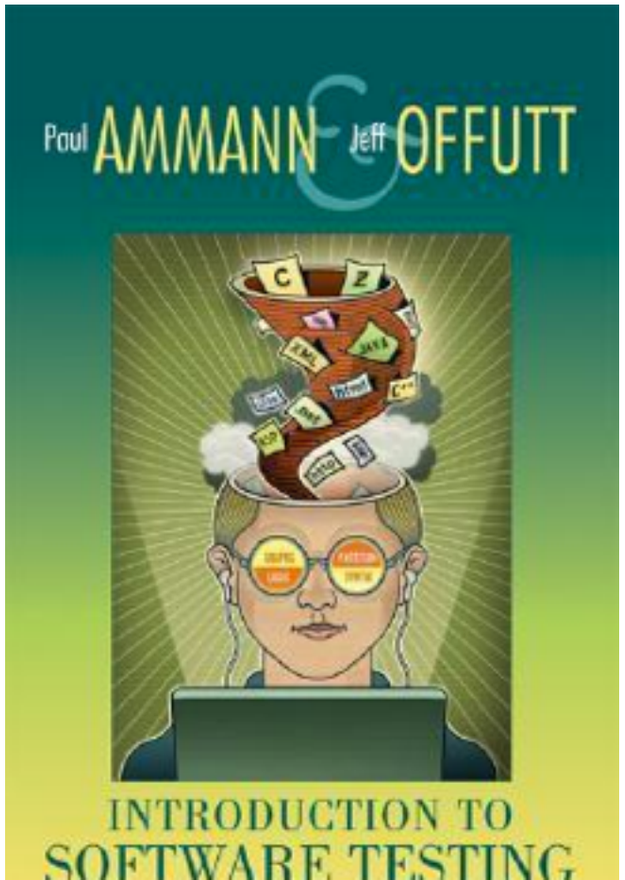
Hours and Resources

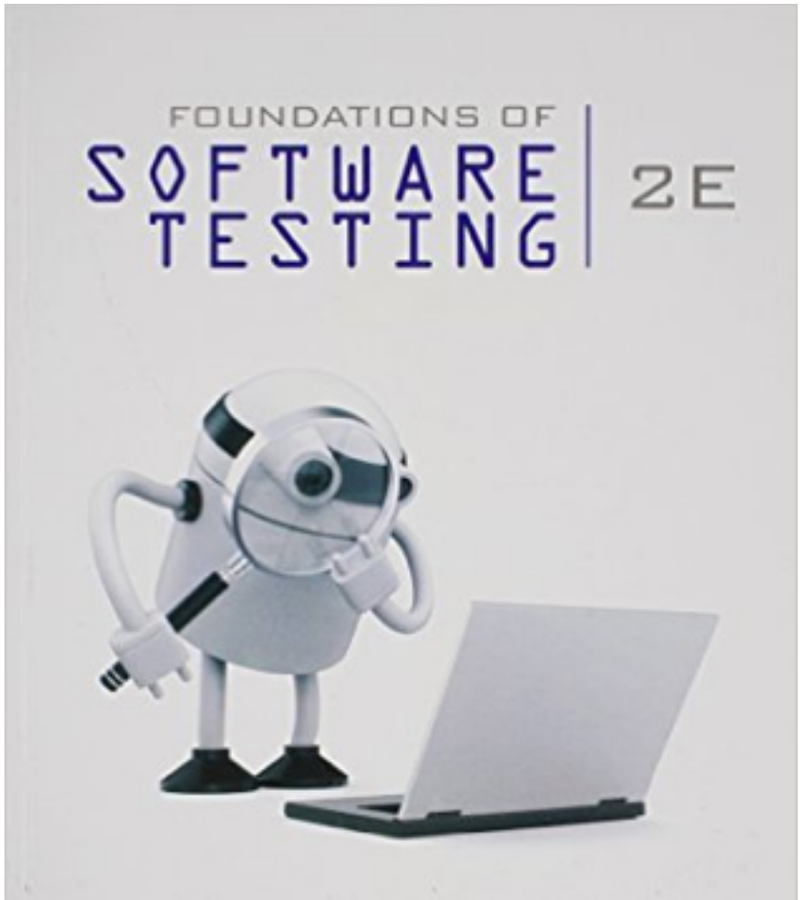
- Course Meetings:
  - Tues/Thur 4:00pm – 5:15pm
  - GR 2.530
- Office Hours:
  - Tues/Thur 2:30pm - 3:30pm
  - ECSS 4.205
- Course Web Page:
  - http://www.utdallas.edu/~lxz144130/cs6367.html

Required Textbook

Recommended Textbooks

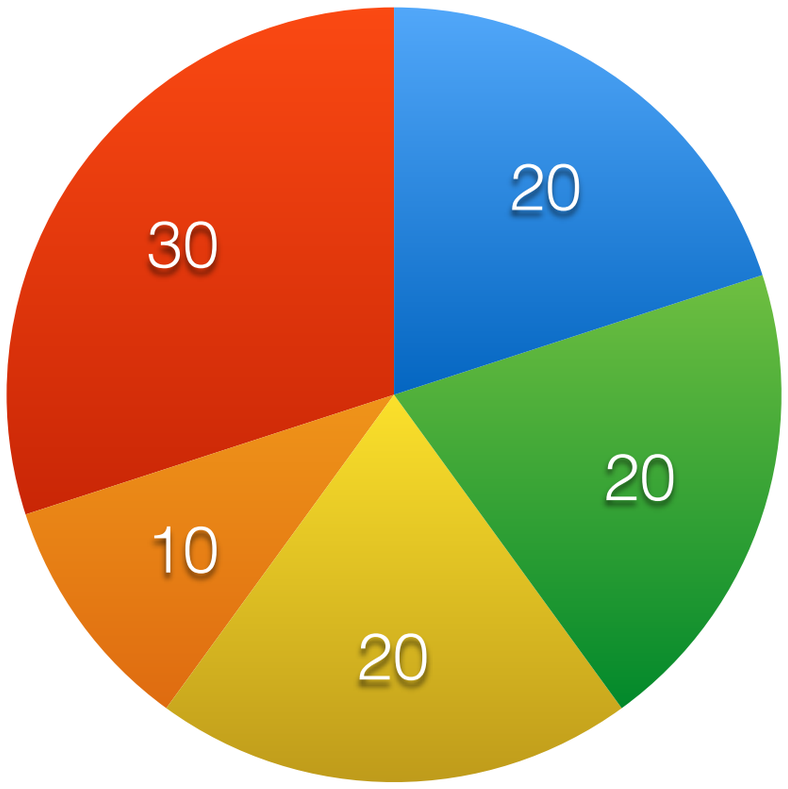
- Introduction to Software Testing (1st Edition)
  - ISBN: 978-0521880381
- Foundations of Software Testing (2nd Edition)
  - ISBN: 978-8131794760



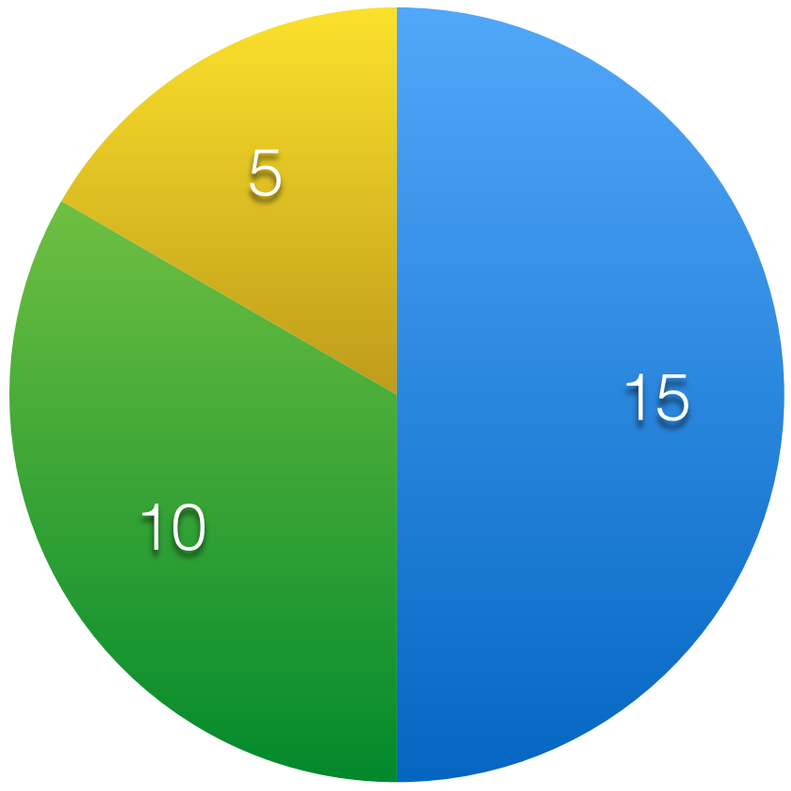


Grading Scheme

Overall



Course Project



Grading Scale

Score	Grade
93-100	A
90-92	A-
87-89	B+
83-86	B
80-82	B-
77-79	C+
70-76	C
<70	F

I may choose to curve the grades at the end of the term

More on the Course Project

> The way to learn software engineering is to go out there and do software engineering.
>
> -- Fred Brooks

- A research project: chosen from a set of topics (posted later), or proposed by the students
  - Analyzing/testing real-world Java code
- Work individually or in pair (1-2 people)
- Go through the whole research project process
  - Proposal
  - Implementation&Experimentation
  - Report writing
  - Project presentation

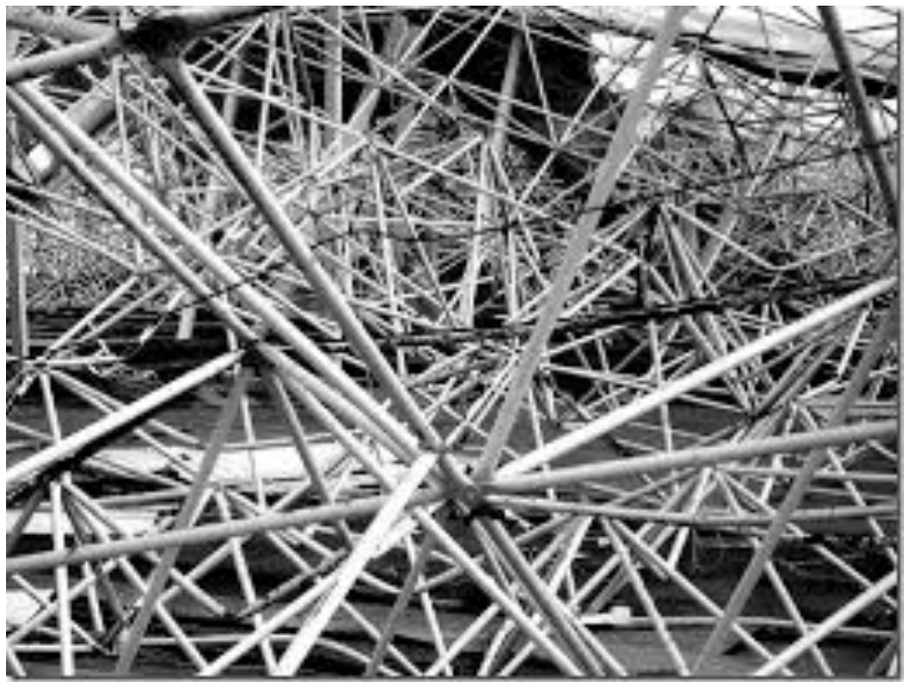
Project Outcomes

- 1-2 page project proposal [Due by Feb 29th midnight]
- Source code [Due by April 30th midnight]
- Project report in ACM SIGPLAN conference format (5-10 pages, double column) [Due by April 30th midnight]
- Final project presentation&Demo

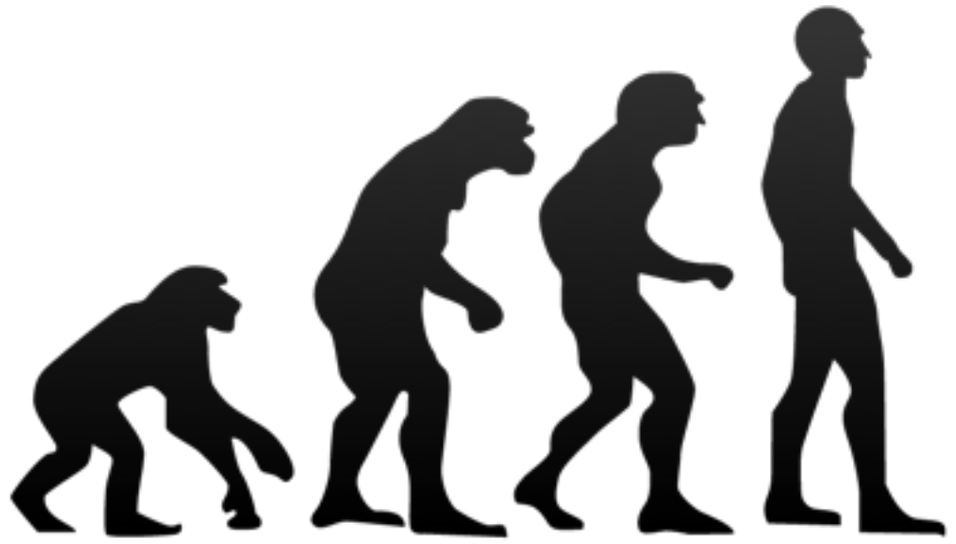
Now, let’s start!

Software is Everywhere

Software Characteristics



Complexity

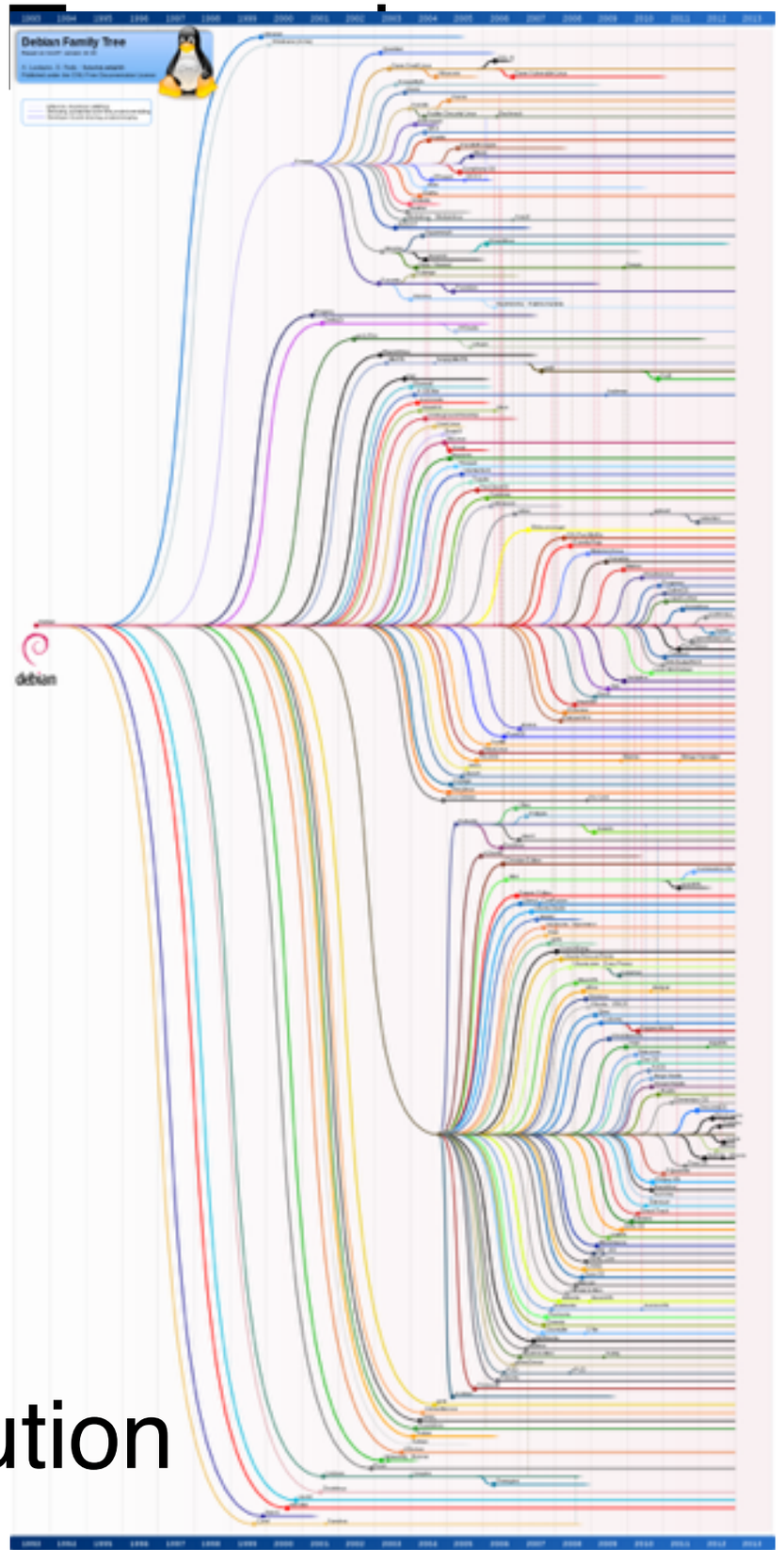


Evolution

The Debian OS

Year	OS	LoC(Million)
2000	Debian2.2	55-59
2002	Debian3.0	104
2005	Debian3.1	215
2007	Debian4.0	283
2009	Debian5.0	324
2012	Debian7.0	419

Complexity



Evolution

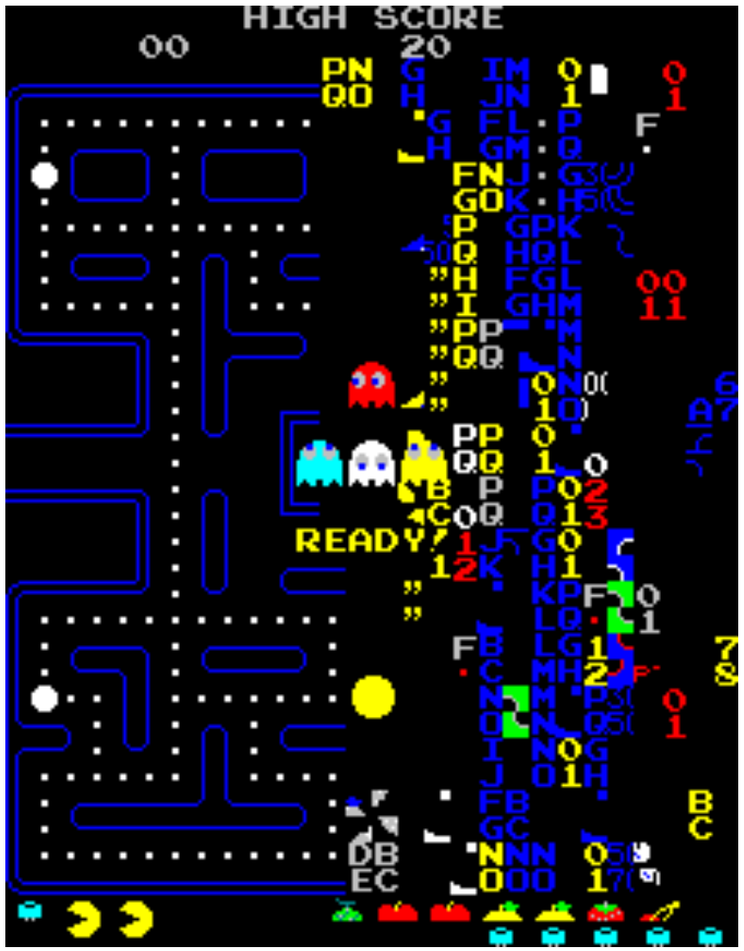
The Facts

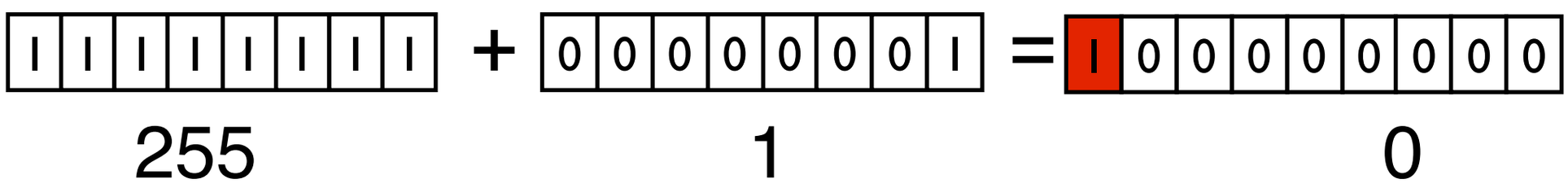
- Only 32% of software projects are considered successful (full featured, on time, on budget)
- Software failures cost the US economy $59.5 billion dollars every year [NIST 2002 Report]
- On average, 1-5 bugs per KLOC (thousand lines of code)
  - In mature software (more than 10 bugs in prototypes)

- 35MLOC
- 63K known bugs at the time of release
- 2 bugs per KLOC

Software Fault Examples

- Pac-Man (1980)
  - Should always have no ending
  - Has “Split Screen” at level 256
- Cause: Integer overflow
  - 8 bits: maximum representable value





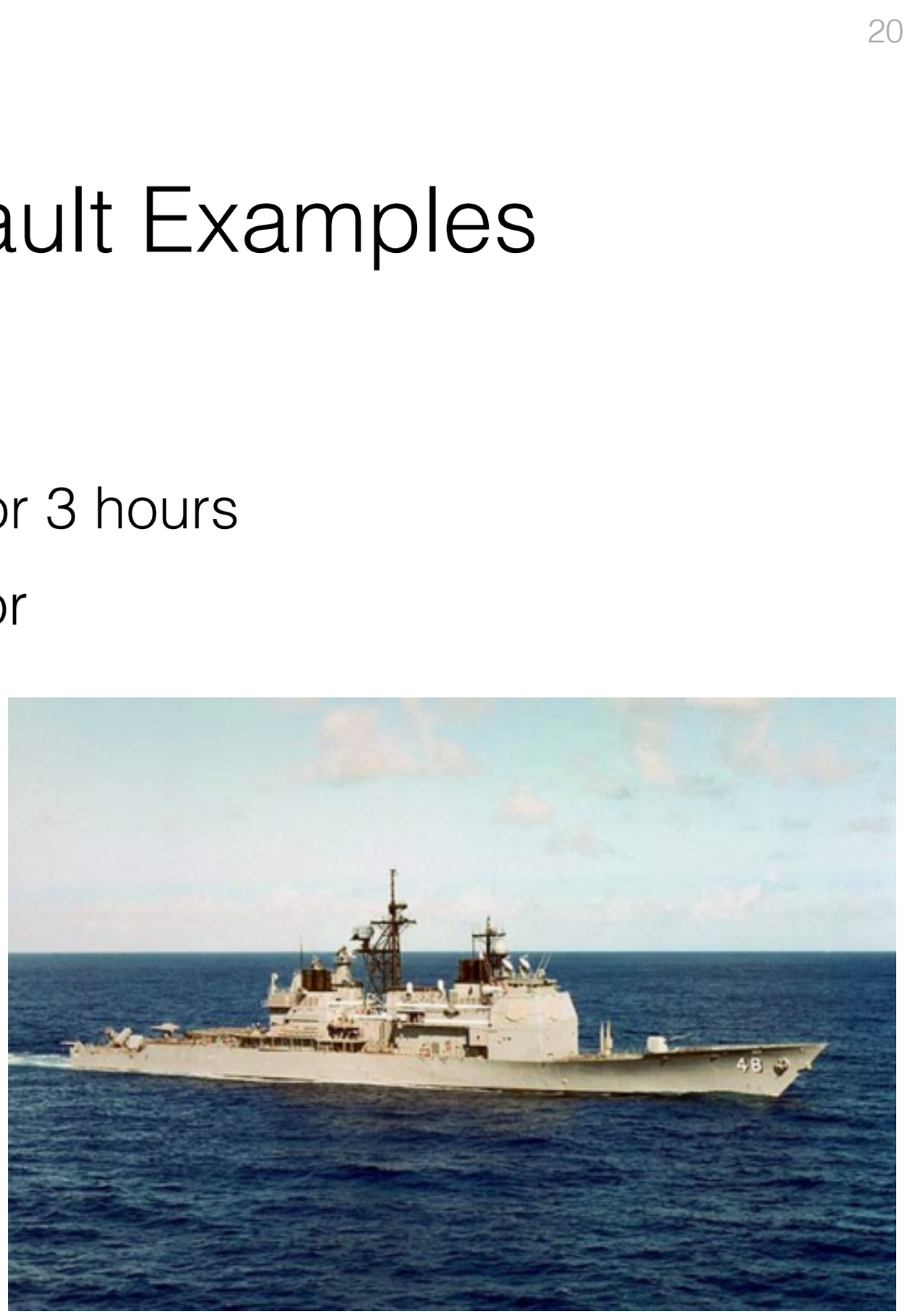
Software Fault Examples

- Mars Climate Orbiter (1998)
  - Sent to Mars to relay signal from Mars Lander
  - Smashed to the planet
- Cause: Failing to convert between different metric standards
  - Software that calculated the total impulse presented results in pound-seconds
  - The system using these results expected its inputs to be in newton-seconds

Software Fault Examples

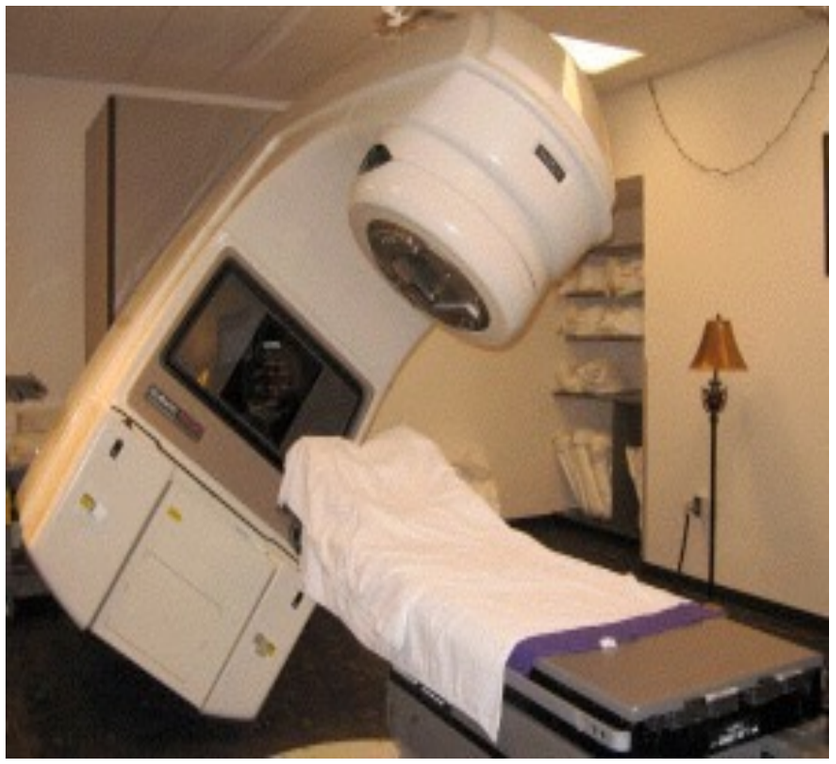
- USS Yorktown (1997)
  - Left dead in the water for 3 hours
- Cause: Divide by zero error

$$\frac{\text{Number}}{0} =$$



Software Fault Examples

- THERAC-25 Radiation Therapy (1985)
  - 2 cancer patients received fatal overdoses
- Cause:
  - Miss-handling of race condition of the software in the equipment



Software Fault Examples

- ATT (1990)
  - One switching system in New York City experienced an intermittent failure that caused a major service outage
  - The first major network problem in AT&T's 114-year history
- Cause: Wrong BREAK statement in C Code
  - Complete code coverage could have revealed this bug during testing

```
1. network code()
2. {
3. switch (line) {
4.     case THING1:
5.          doit1();
6.          break;
7.  case THING2:
8.           if (x == STUFF) {
9.              do_first_stuff();
10.             if (y == OTHER_STUFF)
11.                  break;
12.             do_later_stuff();}
13. /* coder meant to break to here... */
14.           initialize_modes_pointer();
15.           break;
16.    default:
17.         processing(); }
18. /* ...but actually broke to here! */
19.        use_modes_pointer();/
   * leaving the modes_pointer
20.    uninitialized */
21.}
```

Software Fault Examples

- Ariane 5 flight 501 (1996)
  - Destroyed 37 seconds after launch (cost: $370M)
- Cause: Arithmetic overflow
  - Data conversion from a 64-bit floating point to 16-bit signed integer value caused an exception
  - The software from Ariane 4 was re-used for Ariane 5 without re-testing

Software Failure, Fault & Error

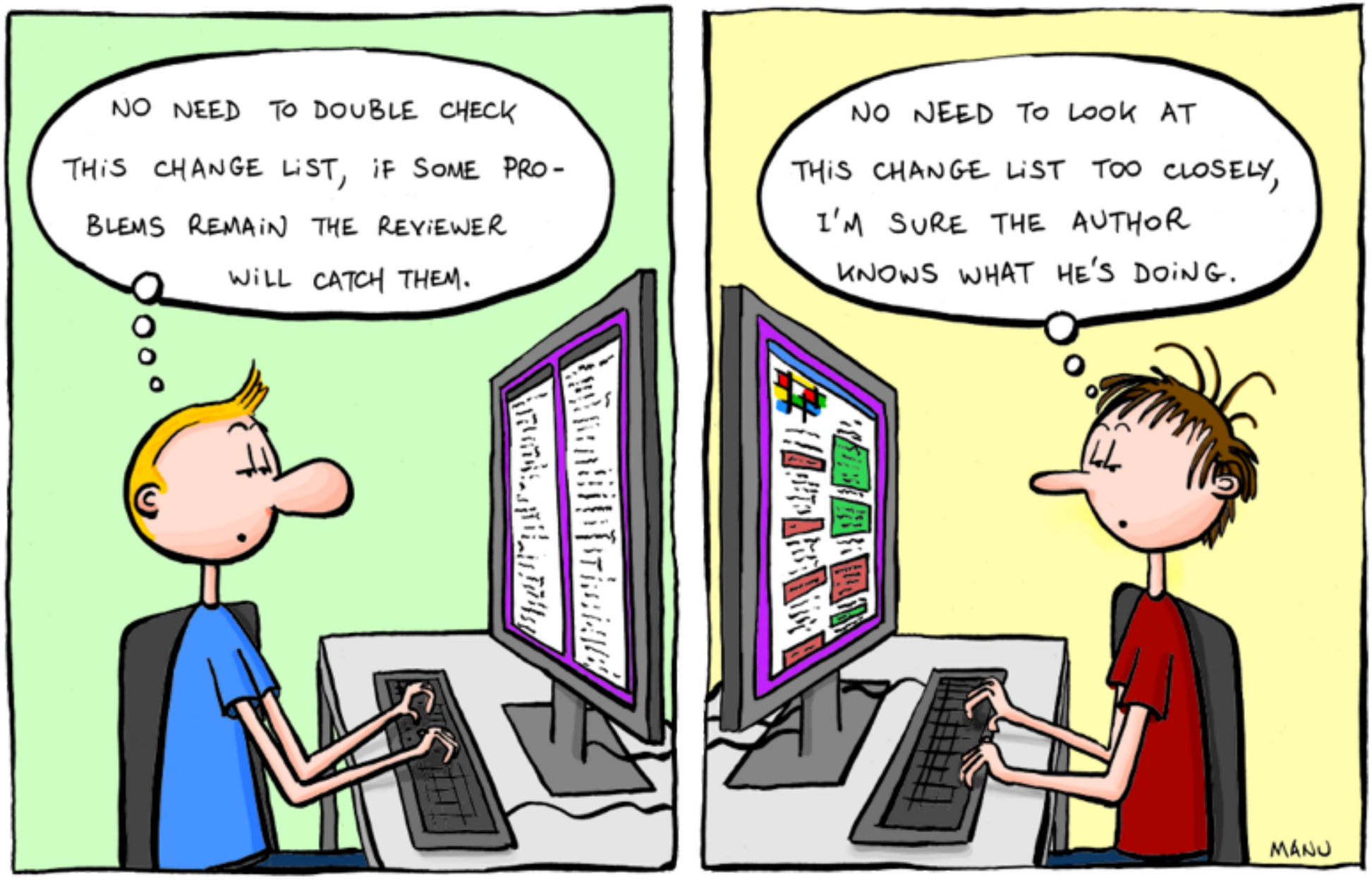
- Fault
  - Incorrect portions of code (may involve missing code as well as incorrect code)
  - Necessary (not sufficient) condition for the occurrence of a failure
- Failure
  - Observable incorrect behavior of a program.
- Error
  - Cause of a fault. something bad a programmer did (conceptual, typo, etc)
- Bug: informal term for fault/failure

Approaches to reduce faults

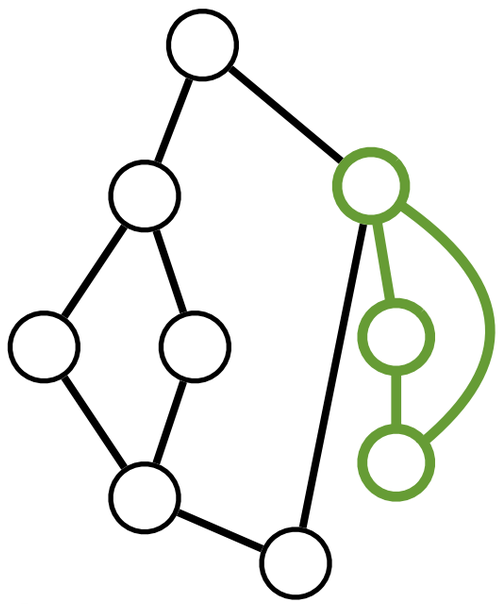
- Manual code review
  - Manually review the code to detect faults
  - Limitations:
    - Hard to evaluate your progress
    - Can miss many faults/bugs

Approaches to reduce faults

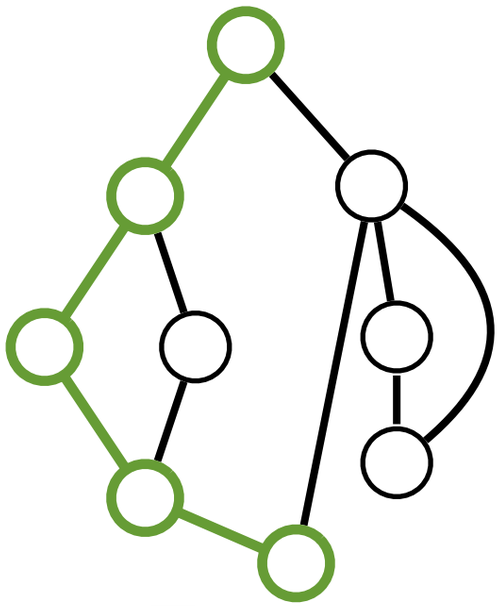
- Manual code review



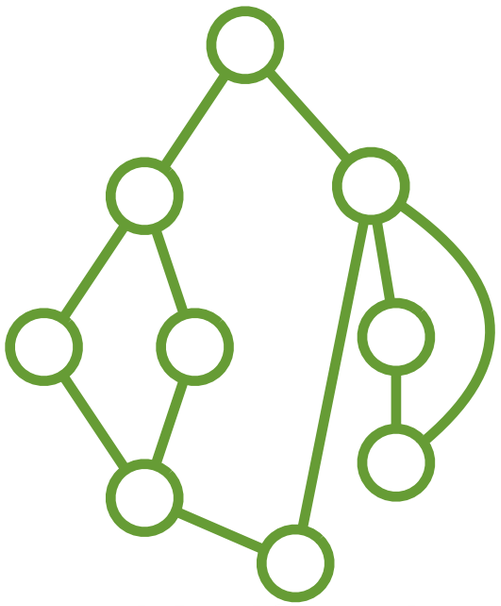
Automated approaches to reduce faults



Static Analysis

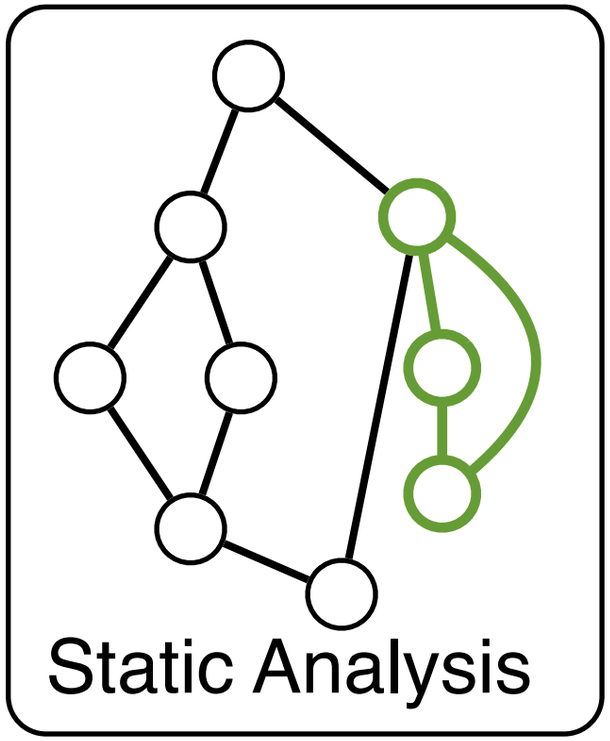


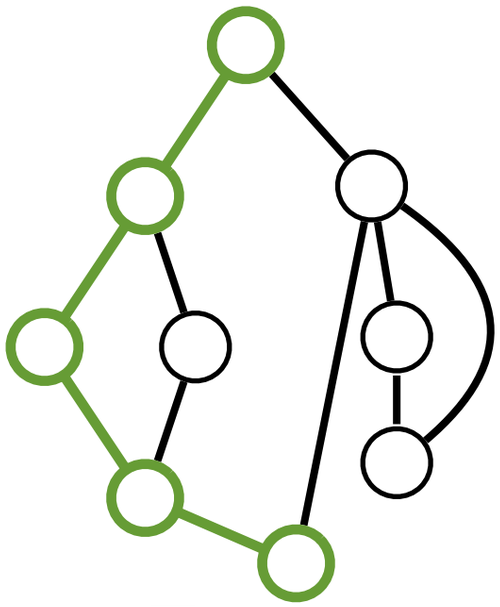
Testing

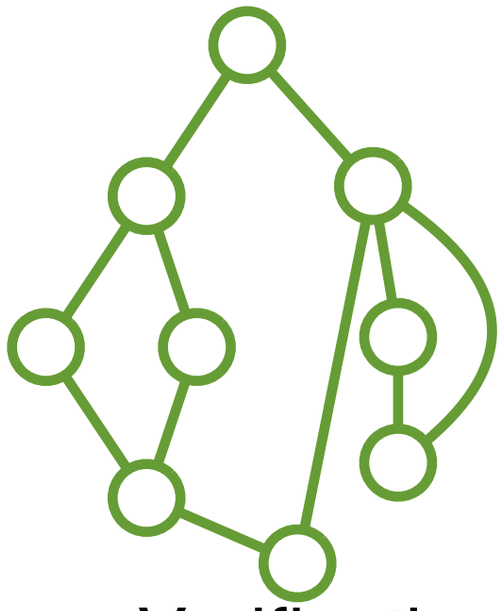


Verification

Automated approaches to reduce faults

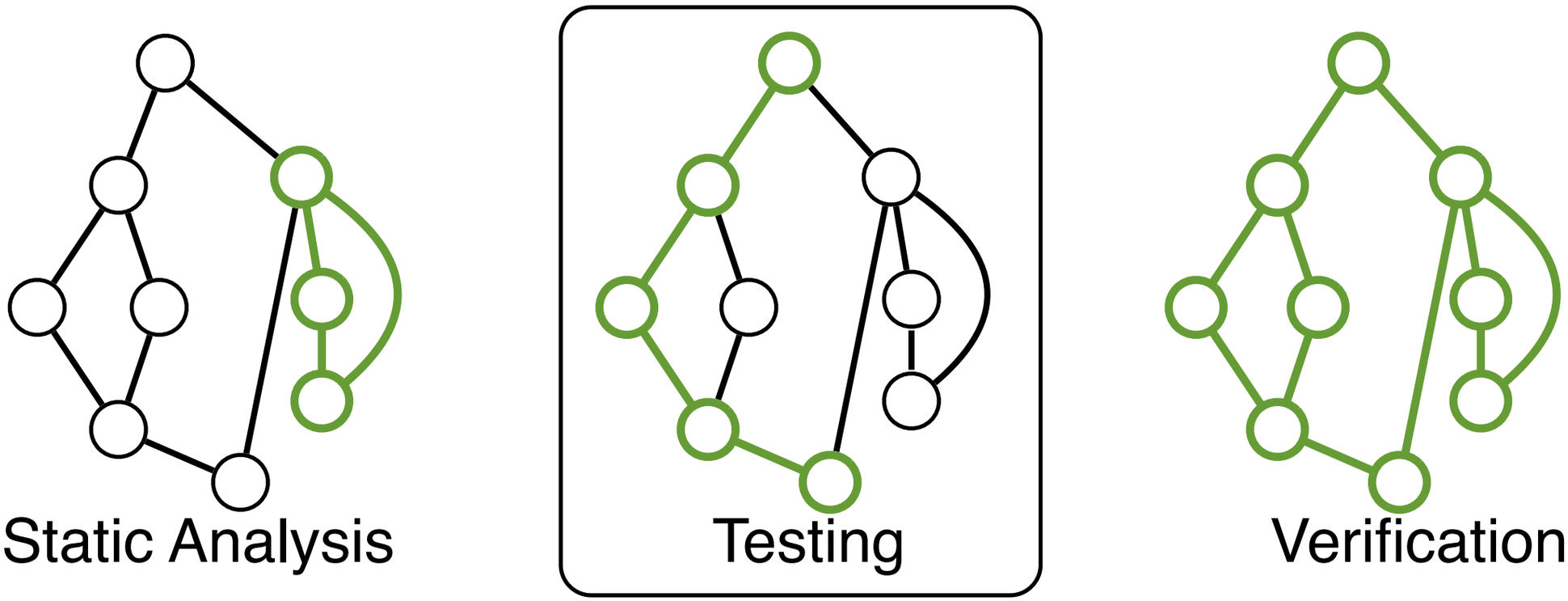






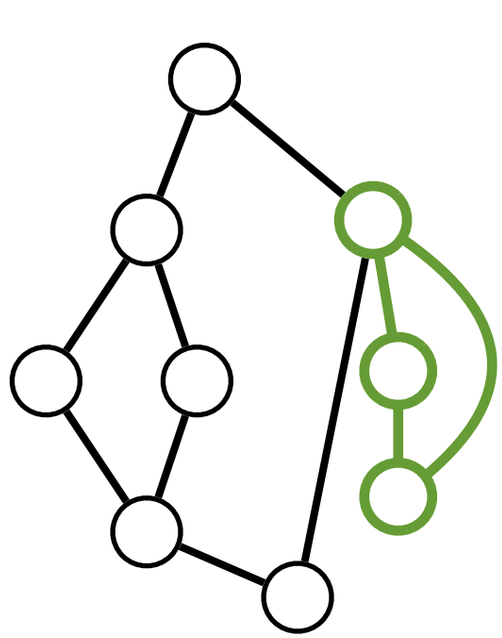
- Static analysis: Identify specific problems (e.g., memory leak) in the software by scanning suspicious patterns from the code
  - Limitations: (1) Limited problem types, (2) False positives

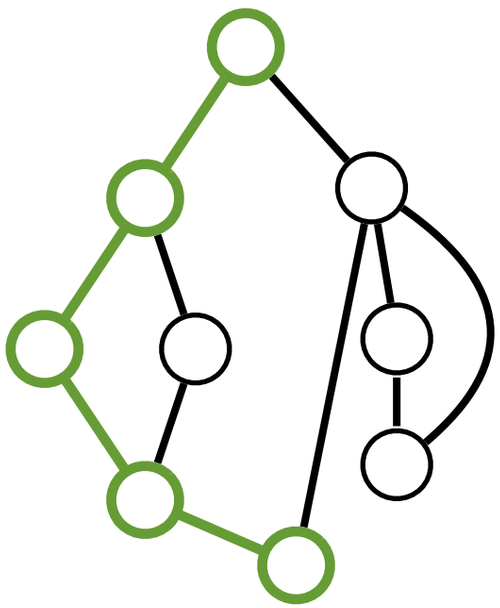
Automated approaches to reduce faults

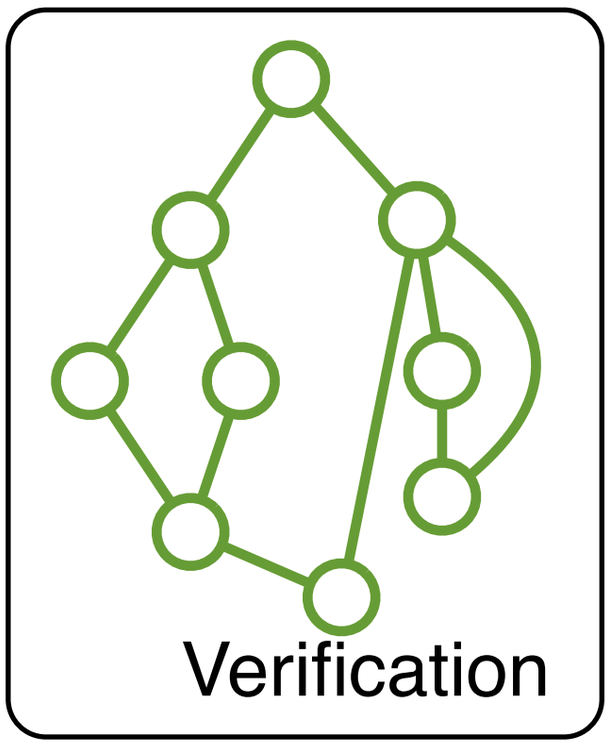


- Testing: Feed input to software and run it to see whether its behavior is as expected
- Limitations: (1) Impossible to cover all possible execution, (2) Need test oracles

Automated approaches to reduce faults







- Formal Verification: Consider all the possible program executions, and formally prove that the program is correct or not
- Limitations: (1) Difficult to have a formal specification, (2) Most real-world programs are too expensive to prove

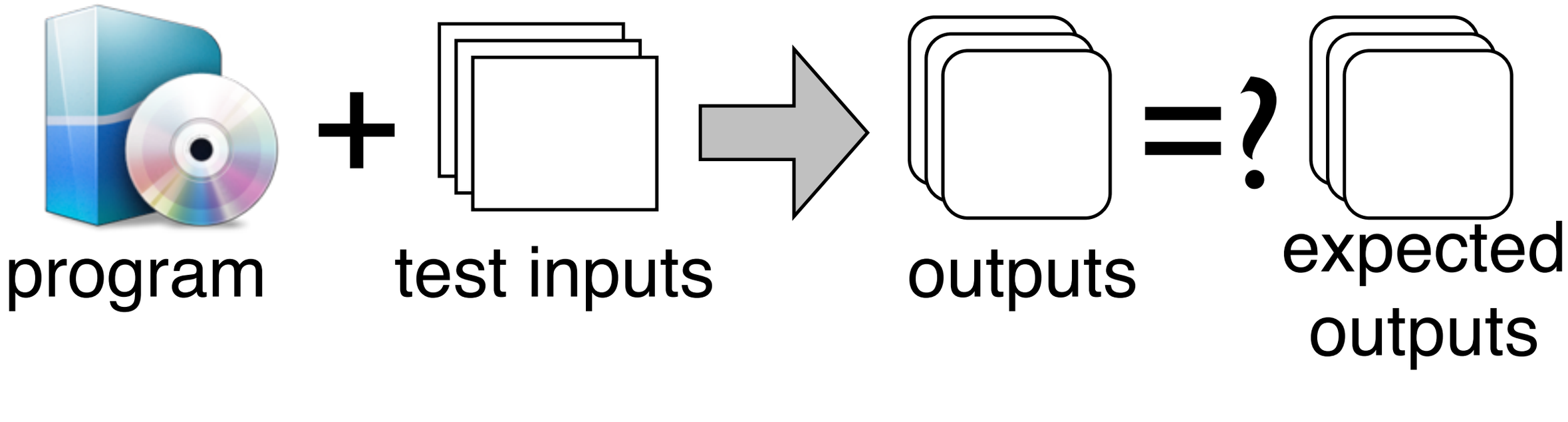
The Most Widely Used Approach

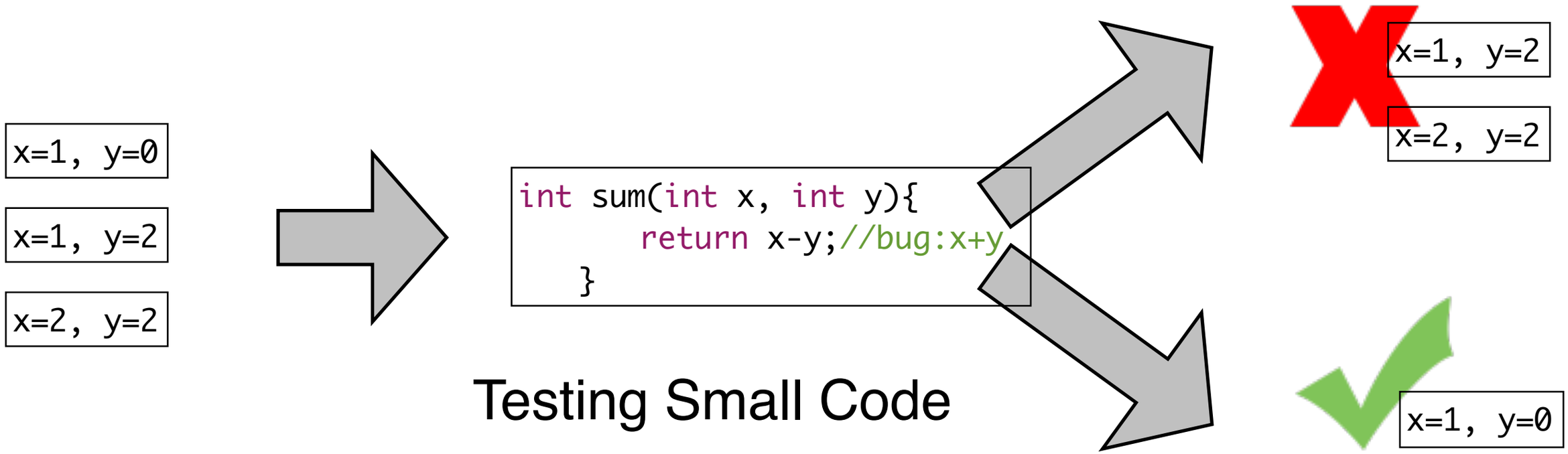
"50% of my employees are testers, and the rest spends 50% of their time testing"

Why Testing?

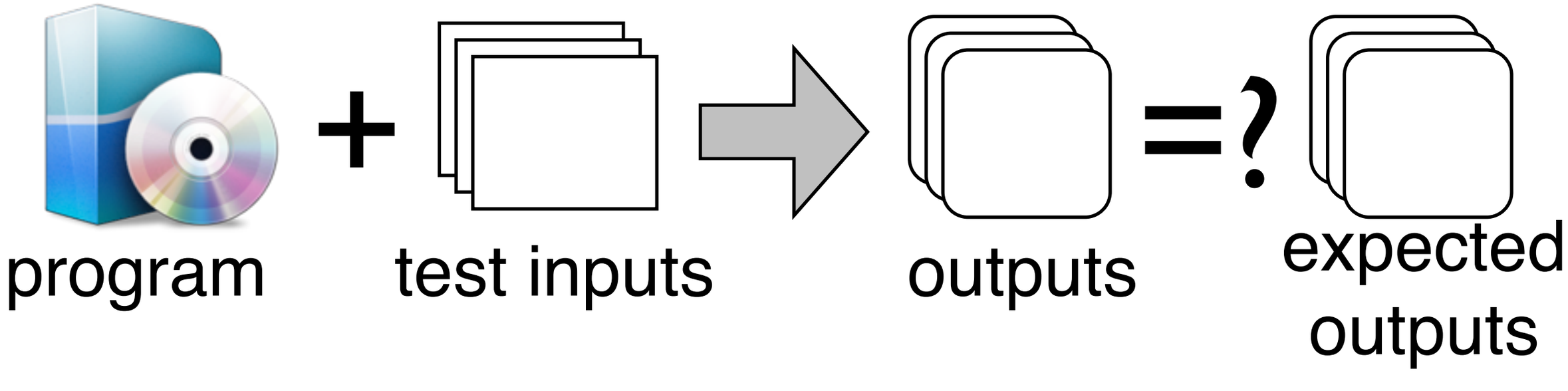
- Testing vs. code review:
  - More reliable than code review
- Testing vs. static checking:
  - Less false positive and applicable to more problems
- Testing vs. formal verification:
  - More scalable and applicable to more programs
- You get what you pay (linear rewards)
  - While the others are not!

What will this course cover?

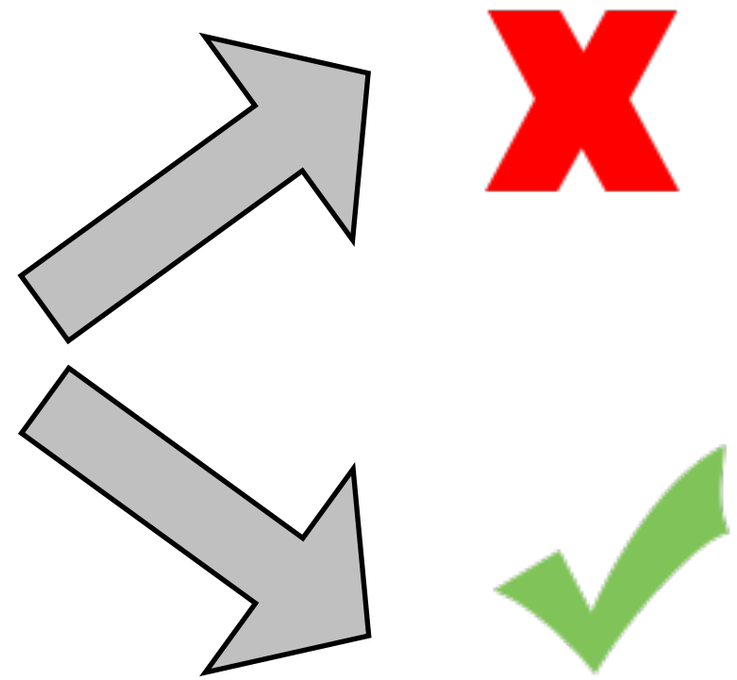




What will this course cover?

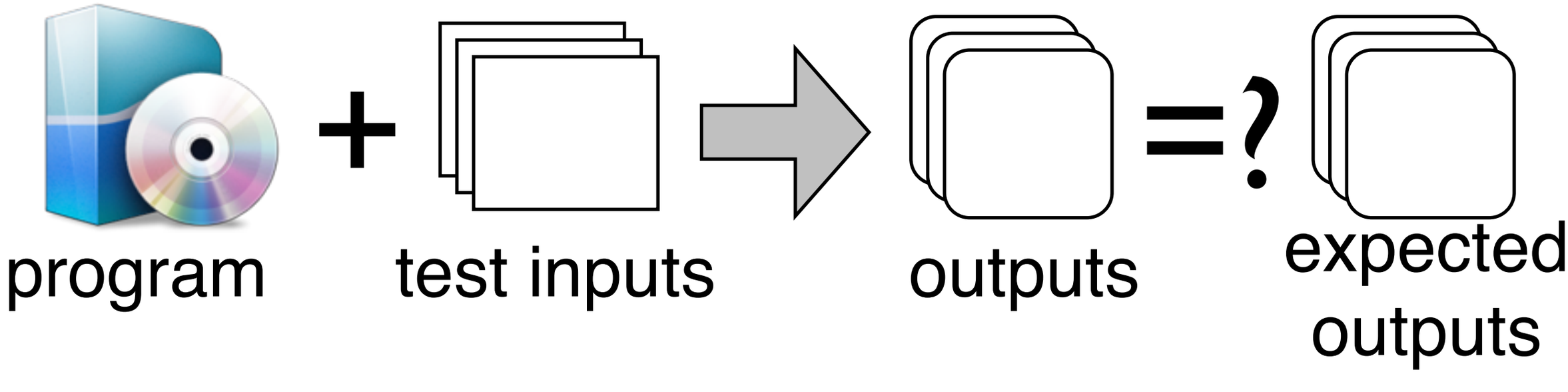


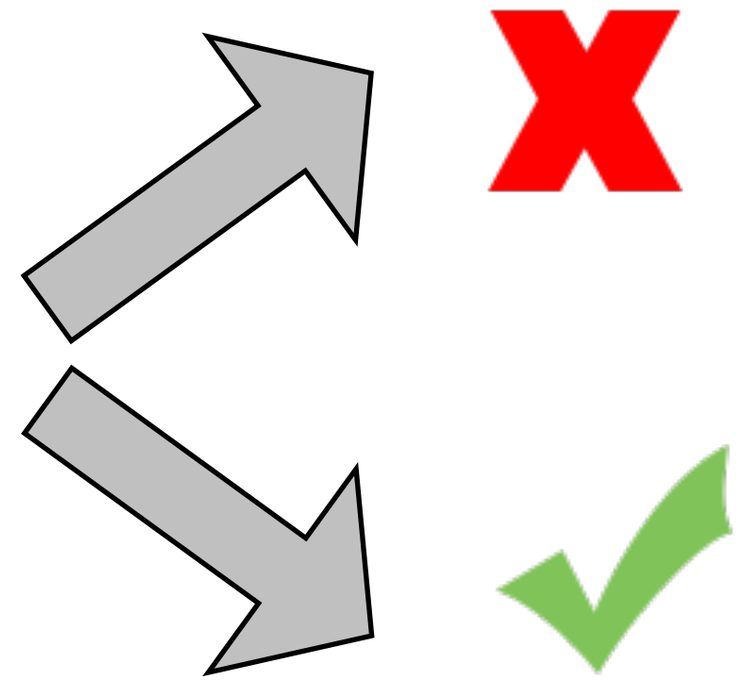
Testing Browsers



layout?=expected layout

What will this course cover?

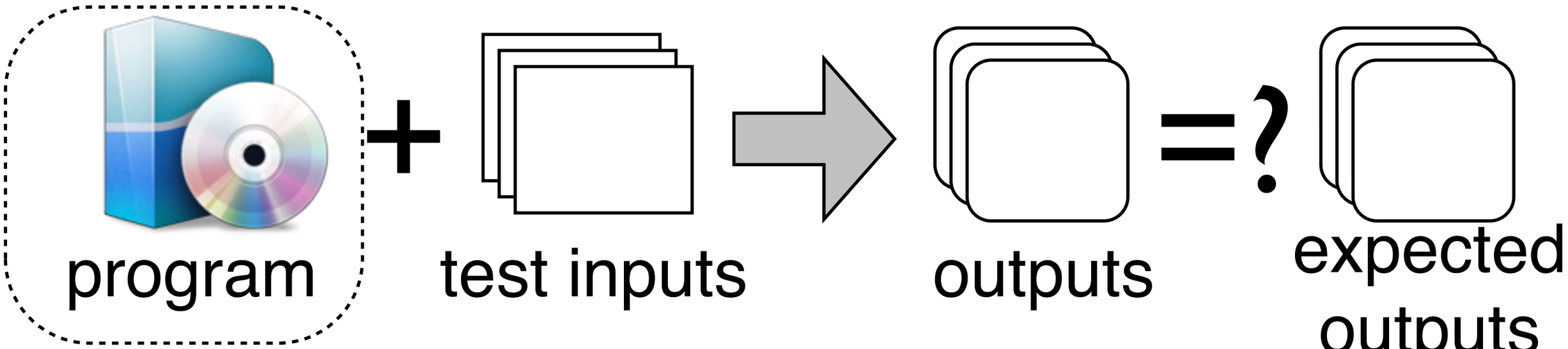




Testing Compilers

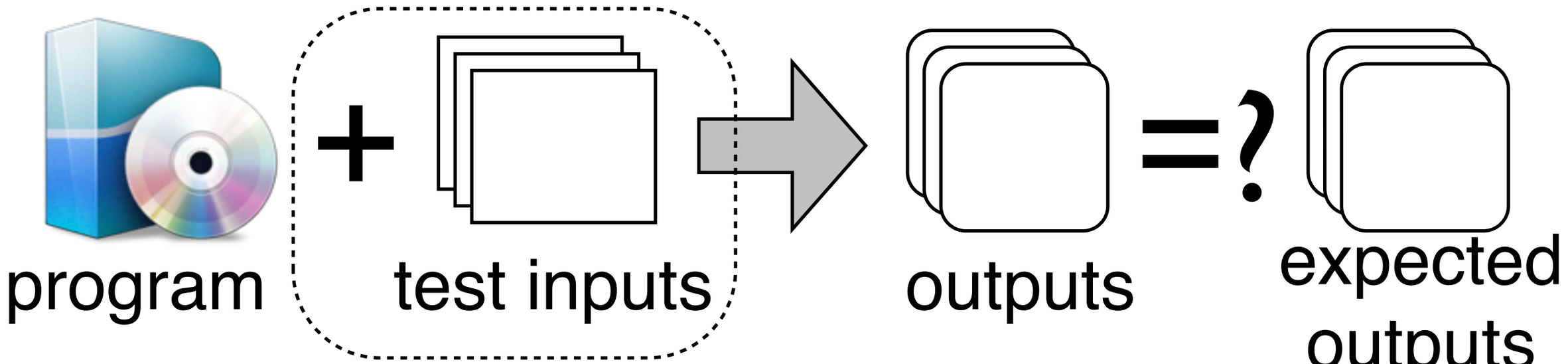
compiled code?=expected code

What will this course cover?



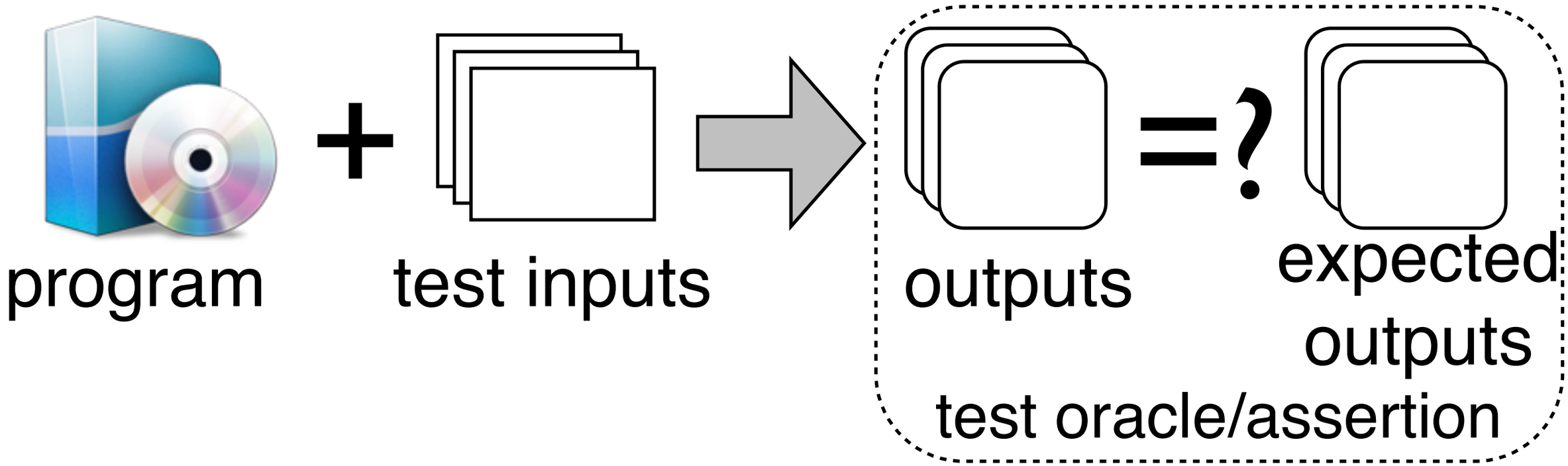
- How to analyze program source code to find potential software faults? [Software Analysis]
- How to formally verify program to find all the possible software faults? [Software Verification/Formal Methods]

What will this course cover?



- How to evaluate the quality of generated tests? [Structural Code Coverage, Mutation Testing]
- How to generate high-quality tests automatically? [Automated Test Generation]
- How to run tests faster when program evolves? [Regression Testing]

What will this course cover?



- How to automatically generate test oracle/assertions? [Oracle Generation]
- How to automatically localize program faults? [Automated Debugging]
- How to automatically fix program faults? [Automated Program Repair]

What else will this course cover?

- How to analyze Java source code?
  - E.g., using Eclipse JDT
- How to analyze and modify Java bytecode?
  - E.g., using ASM Java bytecode analysis framework
- How to analyze and modify Java intermediate code representation?
  - E.g., using WALA/Soot

Course Topics

- Basic concepts
- Test adequacy
- Automated test generation
- Mutation Testing
- Regression testing
- Automated oracle generation
- Automated debugging
- Automated program repair
- Software analysis
- Formal methods in software testing

Why This Course?

- In academia
  - Advance your current research
  - Find your future research interests
- In industry
  - QAs/Software Testers/Test Engineers are in high demand
  - Software engineers are also strongly related to software testing/verification

Thanks!

Hope you will enjoy the course!