Software Testing

Manar Elkady

Some of the material are retrieved from a previous course offering by Dr.Soha Makady and Prof. Amr Kamel

Code Example 2: Fault, Error, Failure

```
public static void isLeap(int year)
{
    if (year % 4 != 0) return false;
    if (year % 400 == 0) return true;
    if (year % 100 < 0) return false;
    return true;
}
```

Exercise

Read this faulty program, which includes a test case that results in failure.

Answer the following questions.

a) Identify the fault.

b) If possible, identify a test case that does not execute the fault.

c) If possible, identify a test case that executes the fault, but does not result in an error state.

d) If possible, identify a test case that results in an error, but not a failure. Hint: Don't forget about the program counter.

e) For the given test case, identify the first error state. Be sure to describe the complete state.

Leap Year

- Any year that is evenly divisible by 4 is a leap year.
  - for example, 1988, 1992, and 1996 are leap years.

```
public static bool isLeap(int year)
{
    if (year % 4 != 0) return false;
        return true;
}
```

Leap Year

- A year that is evenly divisible by 100 (for example, 1900) is a leap year only if it is also evenly divisible by 400.

```
public static bool isLeap(int year)
{
    if (year % 4 != 0) return false;
        return true;
}
```

Leap Year

- The rule is that if the year is divisible by 100 and not divisible by 400, leap year is skipped.
  - The year 2000 was a leap year, for example, but the years 1700, 1800, and 1900 were not. The next time a leap year will be skipped is the year 2100

```
public static bool isLeap(int year)
{
    if (year % 4 != 0) return false;
    if (year % 400 == 0) return true;
    if (year % 100 == 0) return false;      //skipped
    return true;
}
```

Code Example 2: Fault, Error, Failure

```
public static bool isLeap(int year)
{
    if (year % 4 != 0) return false;
    if (year % 400 == 0) return true;
    if (year % 100 < 0) return false;
    return true;
}
```

Did we reach the fault?
Did we infect?

Execution states:

- Year = 2000, PC = isLeap()
- Year = 2000, PC = if (year % 4 != 0)
- Year = 2000, PC = if (year % 400 == 0)
- Year = 2000, PC = return true;

Code Example 2: Fault, Error, Failure

```
public static void isLeap(int year)
{
    if (year % 4 != 0) return false;
    if (year % 400 == 0) return true;
    if (year % 100 < 0) return false;
    return true;
}
```

Execution states:

- Year = 2001, PC = isLeap()
- Year = 2001, PC = if (year % 4 != 0)
- Year = 2001, PC = return false;

Did we reach the fault?
Did we infect?

Code Example 2: Fault, Error, Failure

```
public static void isLeap(int year)
{
    if (year % 4 != 0) return false;
    if (year % 400 == 0) return true;
    if (year % 100 < 0) return false;
    return true;
}
```

Did we reach the fault?
Did we infect?

Execution states:

- Year = 2004, PC = isLeap()
- Year = 2004, PC = if (year % 4 != 0)
- Year = 2004, PC = if (year % 400 == 0)
- Year = 2004, PC = if (year % 100 < 0)
- Year = 2004, PC = return true;

Code Example 2: Fault, Error, Failure

```
public static void isLeap(int year)
{
    if (year % 4 != 0) return false;
    if (year % 400 == 0) return true;
    if (year % 100 < 0) return false;
    return true;
}
```

Did we reach the fault?
Did we infect?

Execution states:

- Year = 2100, PC = isLeap()
- Year = 2100, PC = if (year % 4 != 0)
- Year = 2100, PC = if (year % 400 == 0)
- Year = 2100, PC = if (year % 100 < 0)
- Year = 2100, PC = return true;

The year 2100 is not a leap year in the Gregorian calendar (because while it is divisible by 4, it is also divisible by 100 but not 400)

Testing & Debugging

- Testing : Evaluating software by observing its execution
- Test Failure : Execution of a test that results in a software failure
- Debugging : The process of finding a fault given a failure

Not all inputs will "trigger" a fault into causing a failure

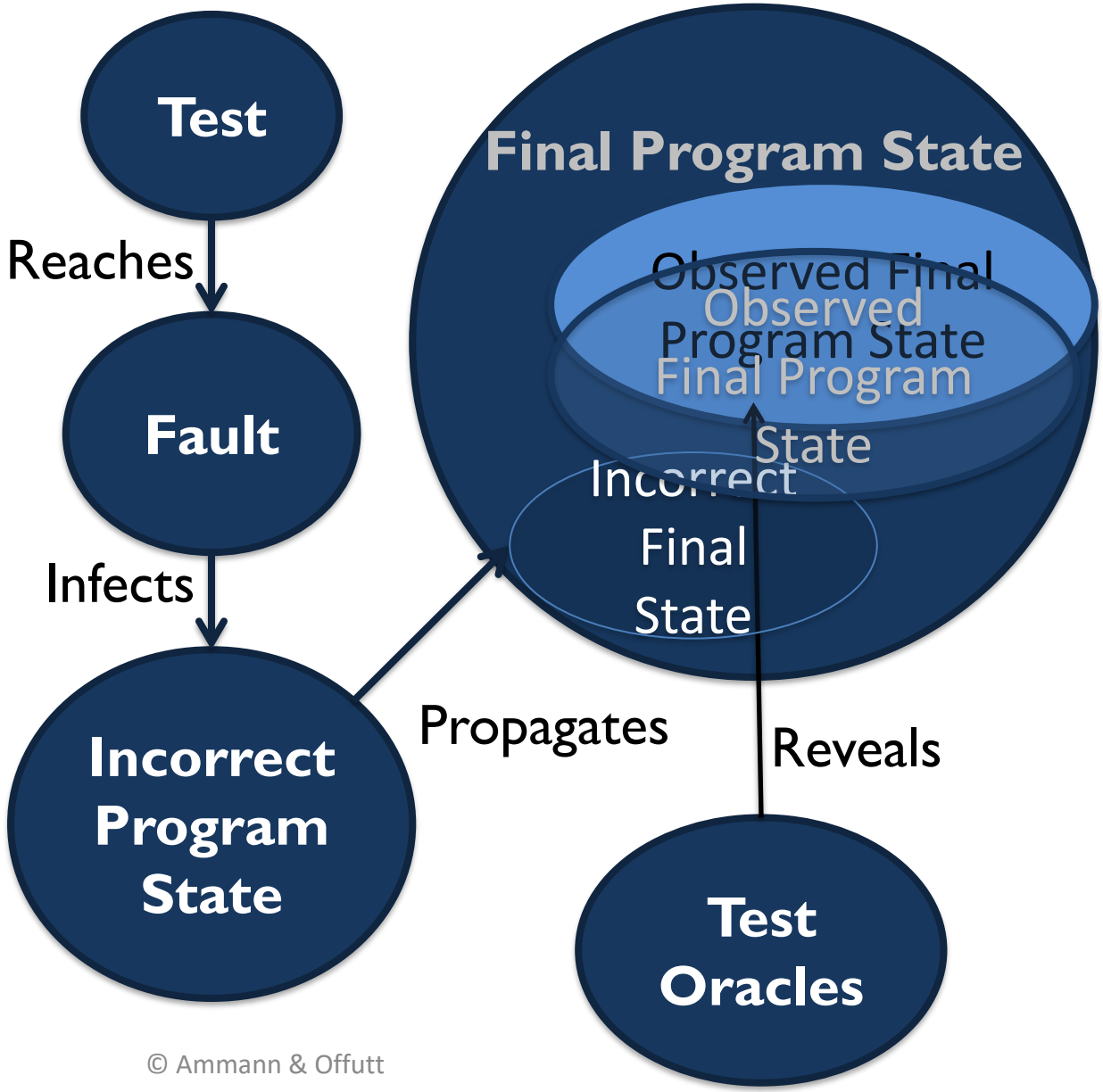
Fault & Failure Model (RIP-R Model)

Four conditions necessary for a failure to be observed

1. Reachability : The location or locations in the program that contain the fault must be reached
2. Infection : The state of the program must be incorrect
3. Propagation : The infected state must cause some output or final state of the program to be incorrect
4. Reveal : The tester must observe part of the incorrect portion of the program state

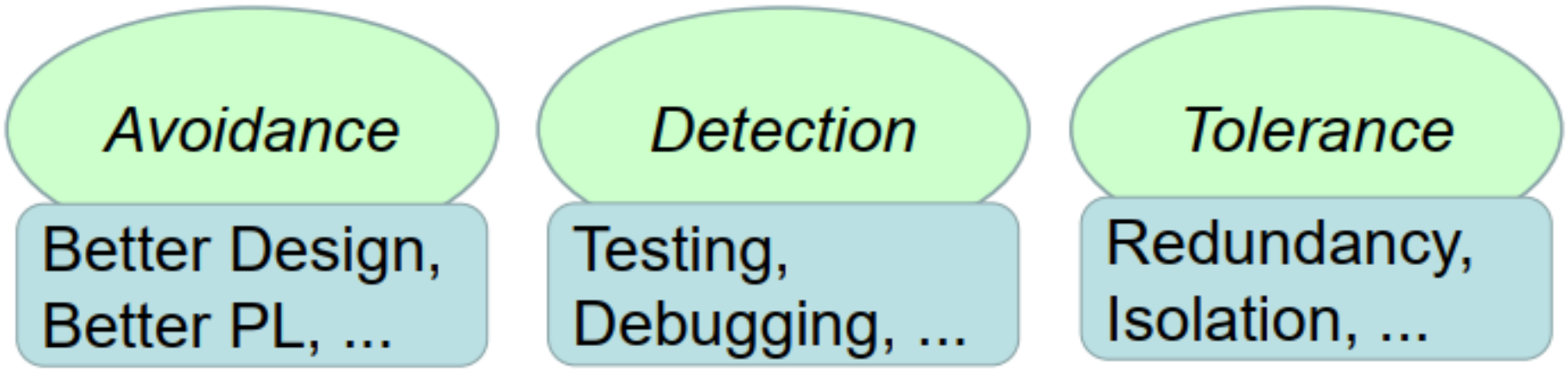
RIP-R Model

- Reachability
- Infection
- Propagation
- Revealability

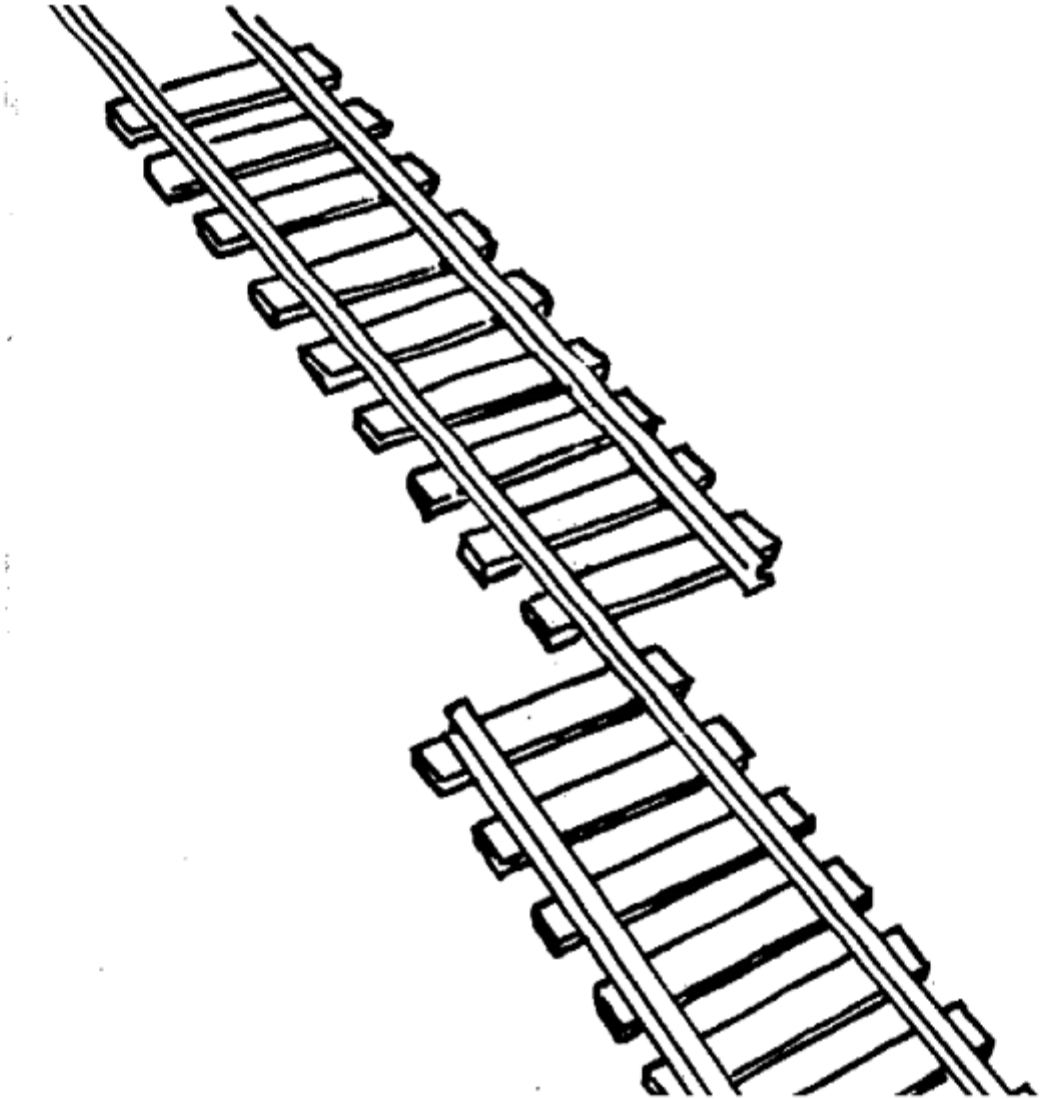



How do we deal with
Faults, Errors,
and Failures?

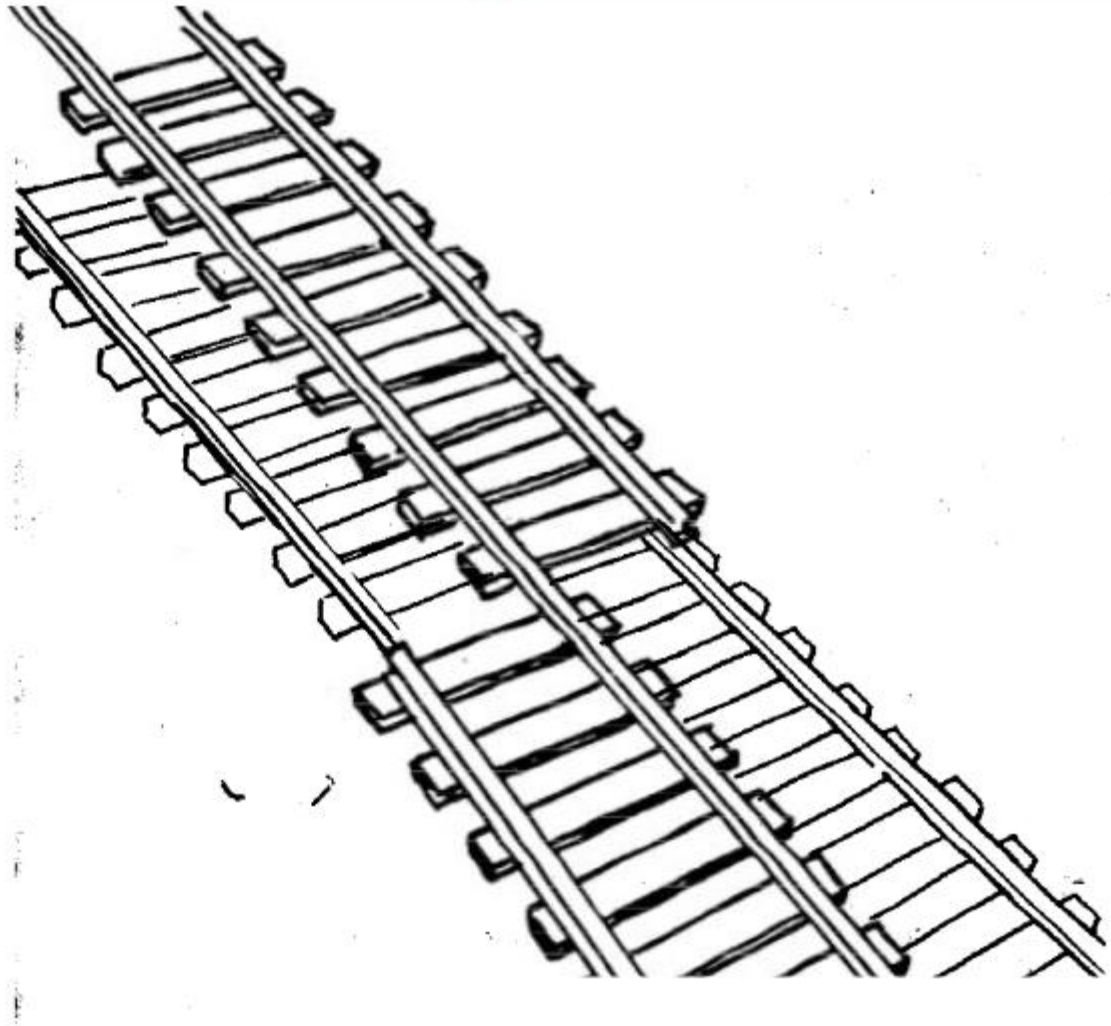
Addressing Faults at Different Stages



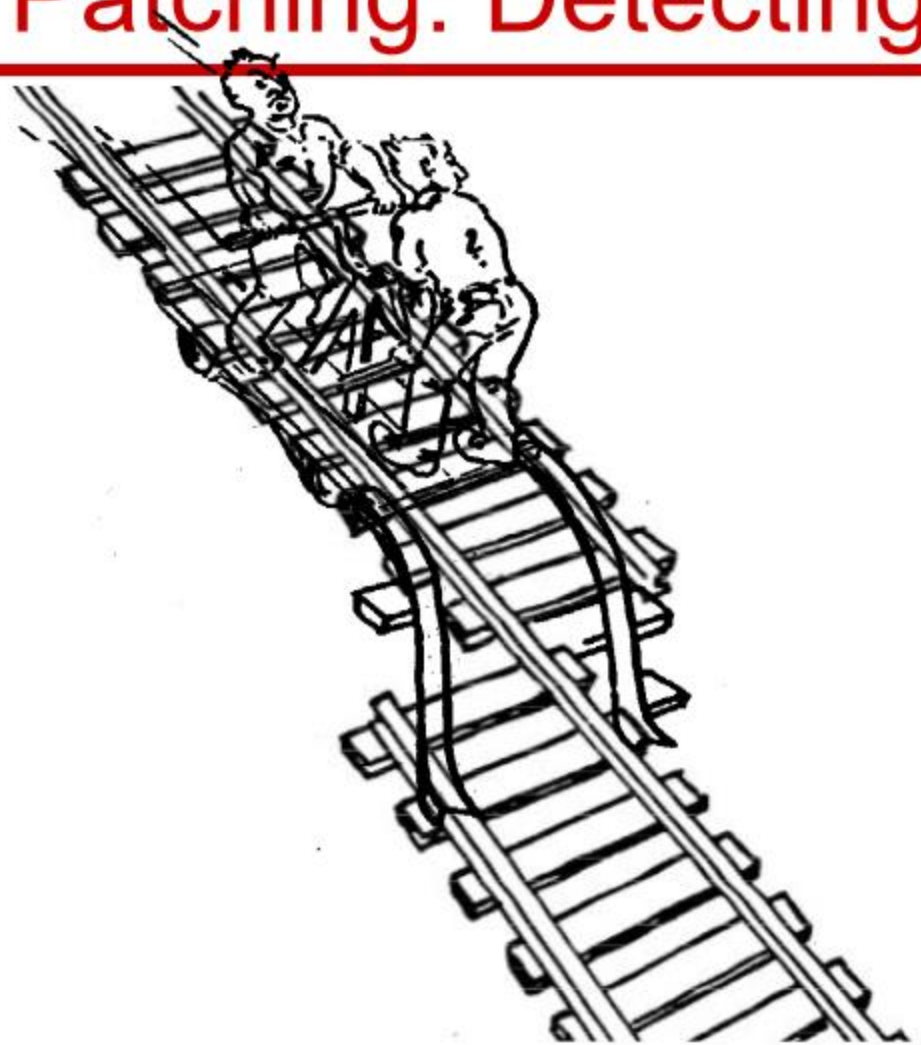
Declaring the Bug as a Feature



Modular Redundancy: Fault-Tolerance



Testing & Patching: Detecting & Fixing



Testing is hard

```
if ( x - 100 <= 0 )
  if ( y - 100 <= 0 )
    if ( x + y - 200 == 0 )
      crash();
```

- Only input x=100 & y =100 triggers the crash.
  - Probability to trigger the crash: 1 over 2^{64}
  - Assuming x and y are 32-bit integers

a random search over all 32-bit integers,
will never find the crash

Software Testing – Basic Definitions (Cont'd)

- Test case: A test case is a test-related item which contains the following information:
  - A set of test inputs
  - Execution conditions
  - Expected outputs
- Test suite: A test suite is a group of related test cases.
- Test oracle: A program, a document, or a formula that produces or specifies the expected outcome of a test, can serve as an oracle.
  - Are oracles easy to construct? E.g., GUI-testing, usability testing.

Software Testing – Basic Definitions (Cont'd)

```
public void simpleAdd() {
    Money m12CHF= new Money(12, "CHF");
    Money m14CHF= new Money(14, "CHF");
    Money expected= new Money(12+14, "CHF");
    Money result= m12CHF.add(m14CHF);
    assertTrue(expected.equals(result));
}
```

What elements do you see in the above example?

Software Testing – Basic Definitions (Cont'd)

- Verification vs. Validation?
  - Verifying that the product has been developed right
    - According to what?
  - Verifying that the right product has been developed
    - According to what?
- Verification vs. Validation?*
  - Verification is the process confirming that the software meets its specification.
  - Validation is the process confirming that the software meets the user's requirements.

*Definition retrieved from Dietmar Pfahl 2015

Verification vs. Validation (Example)

1. The Hubble space telescope (1990) used a large mirror to magnify the objects it's aiming at.
2. The telescope was built to orbit around the earth.
3. The telescope was designed for use in space, and could not be viewed through on Earth.
4. The only way to test its mirror was to measure its attributes, and compare them against its specification.
   - The measurements were the same.

Verification vs. Validation (Example)

- However, after being launched, its returned images that were out of focus.
- Was this a verification or a validation issue?
- Which of those two should software testing target?

Software Testing Principles

- A necessary part of a test case is a definition of the expected output.

- A programmer should avoid attempting to test his own program

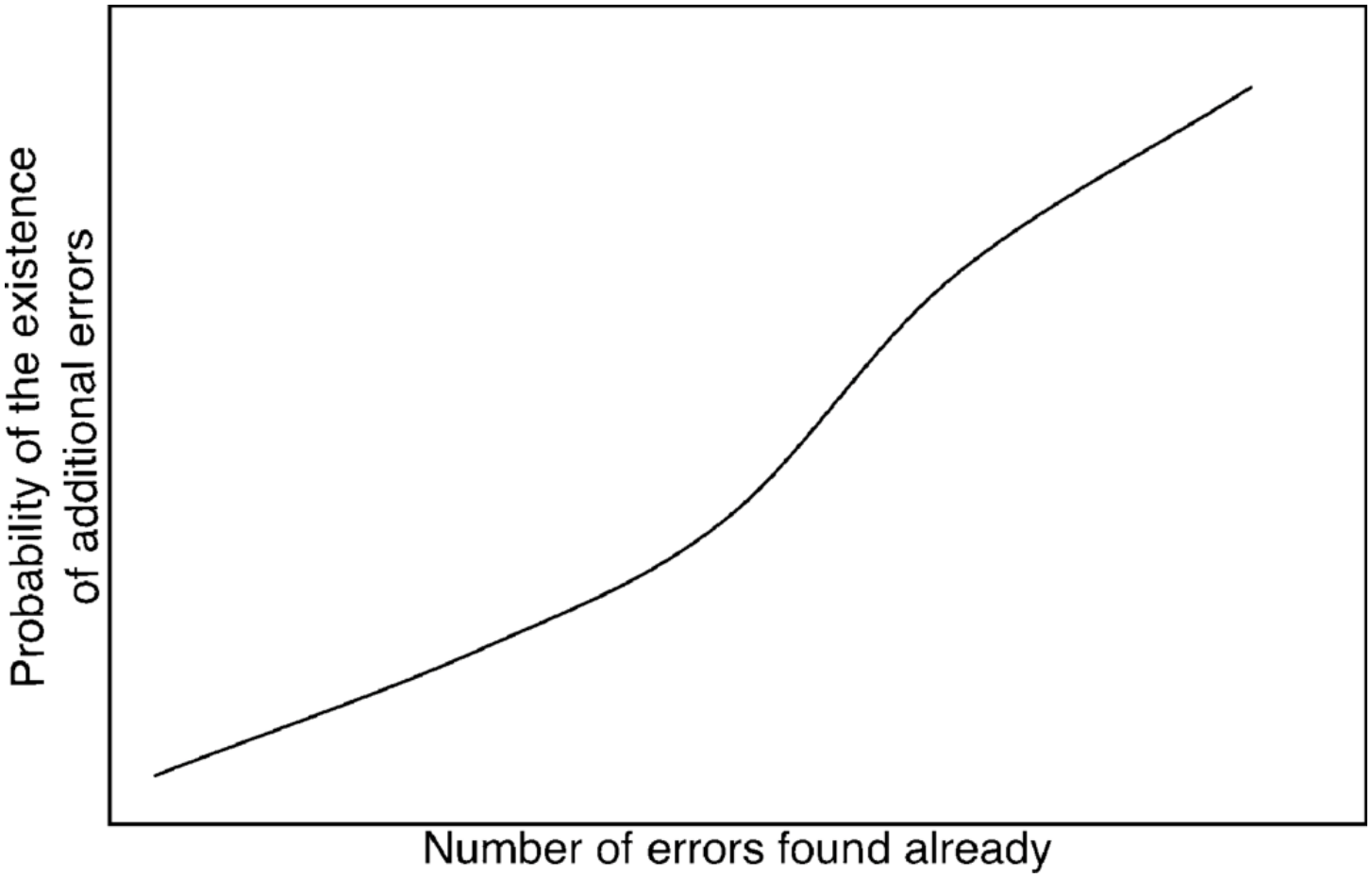
Software Testing Principles

- Any testing process should include a thorough inspection of the results of each test.

- Test cases must be written for invalid and unexpected inputs, as well as valid and expected inputs
  - Test-to-pass vs. test-to-fail
- Examining a program to see if it does what it is supposed to do is only half the battle.
  - How?

Software Testing Principles

- The probability of the existence of more errors is proportional to the number of errors already found.



Software Testing Principles

- Tests must be repeatable and reusable.

- Do NOT plan a testing effort under the assumption that no errors will be found.

Required Readings

- Practical Software Testing
  - Chapter 2: Testing Fundamentals
- An Introduction to Software Testing
  - Chapter 1, Section 2.1