

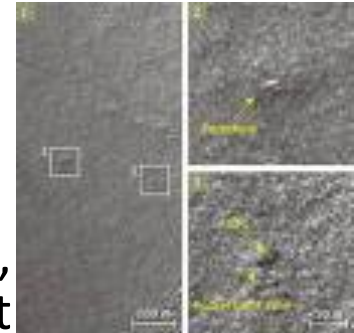
# Software Testing

Software Engineering 2  
(3103313-1)

Amirkabir University of Technology  
Fall 1399-1400

# Spectacular Software Failures

NASA's Mars lander: September 1999, crashed due to a units integration fault



# Ariane 5 explosion

## Millions of \$\$



THERAC-25 radiation machine : Poor testing of safety-critical software can *cost lives* : 3 patients were killed

# Northeast Blackout of 2003



508 generating units  
and 256 power plants  
shut down

Affected 10 million  
people in Ontario,  
Canada

Affected 40 million  
people in 8 US states

Financial losses of  
\$6 Billion USD

The [alarm system](#) in the energy management system [failed due to a software error](#) and operators were not informed of the power overload in the system

# Software Testing

Why testing?

What does testing demonstrate?

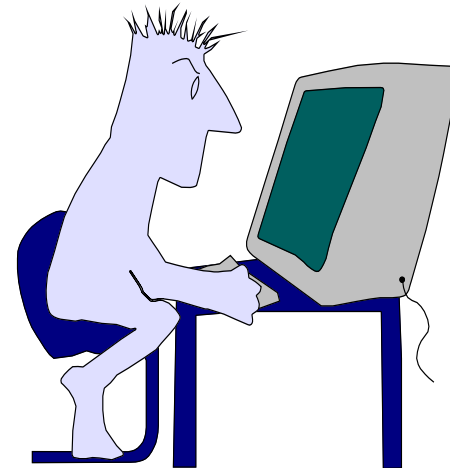
Challenges

# Who Tests the Software?



*developer*

Understands the system but, will test  
"gently" and, is driven by "delivery"



*independent tester*

Must learn about the system, but, will  
attempt to break it and, is driven by  
quality

# Types of Testing

- Functional Testing
- User Testing
- Performance & Load Testing
- Security Testing

- Unit Testing
- Feature Testing
- System Testing
- Release Testing

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing
- Regression Testing
- ?

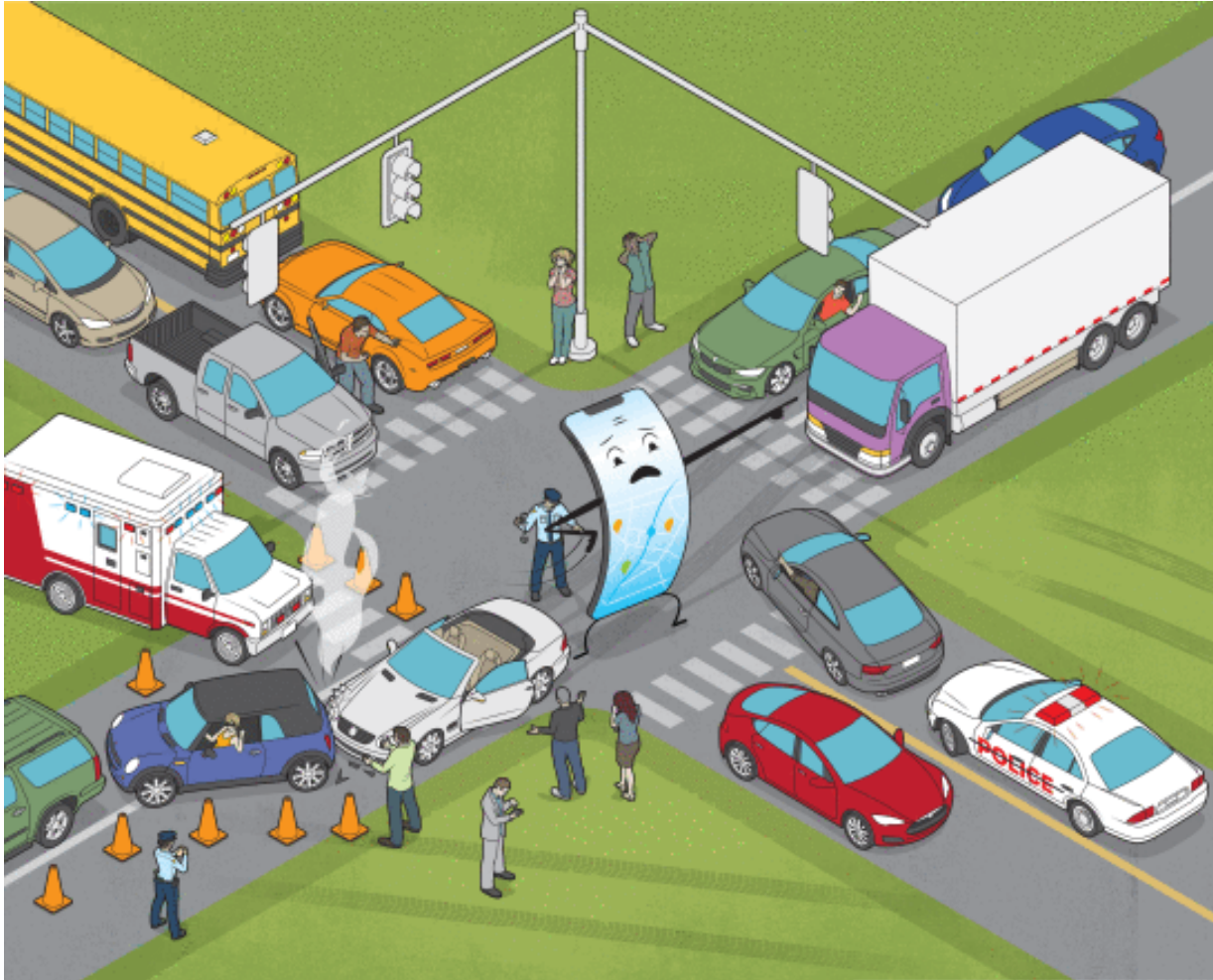
# Regression testing

- You should **preserve** all the test cases for a program
- Regression testing is crucial during **maintenance**
  - It is a good idea to automate regression testing so that all test cases are run after each modification to the software
- When you find a bug in your program you should write a test case that exhibits the bug
  - Then using regression testing you can make sure that the old bugs do not reappear

break







# WHEN APPS RULE THE ROAD

THE PROLIFERATION OF NAVIGATION  
APPS IS CAUSING TRAFFIC CHAOS.  
IT'S TIME TO RESTORE ORDER

By Jane Macfarlane

J. Macfarlane, "[When apps rule the road: The proliferation of navigation apps is causing traffic chaos. It's time to restore order](#)," in IEEE Spectrum, vol. 56, no. 10, pp. 22-27, Oct. 2019.

# martinFowler.com

A website on building software effectively



Software development is a young profession, and we are still learning the techniques and building the tools to do it effectively. I've been involved in this activity for over three decades and in the last two I've been writing on this website about patterns and practices that make it easier to build useful software. The site began as a place to put my own writing, but I also use it to publish articles by my colleagues.

In 2000, I joined [ThoughtWorks](#), where my role is to learn about the techniques that we've learned to deliver software for our clients, and pass these techniques on to the wider software industry. As this site has developed into a respected platform on software development, I've edited and published

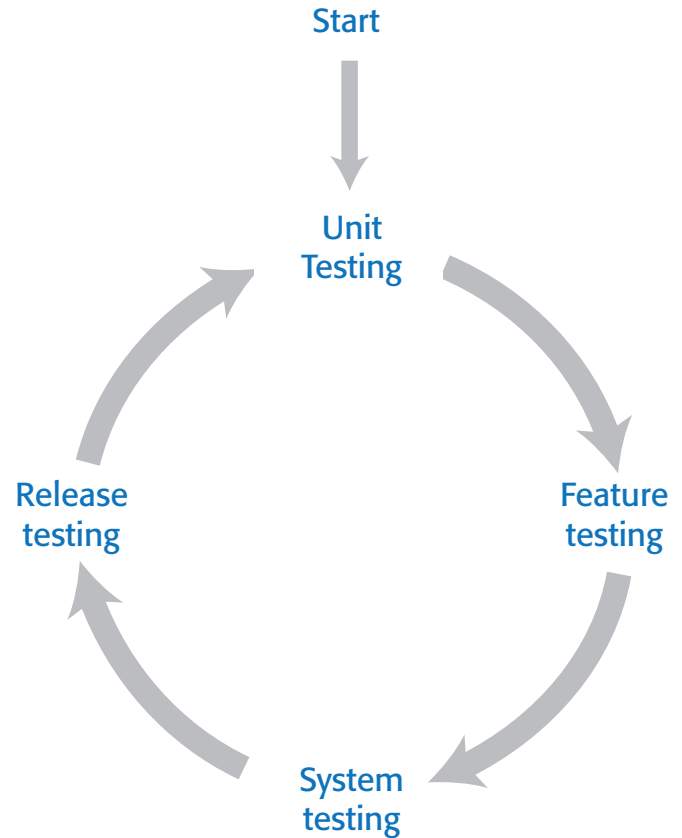


photo: Christopher Ferguson

**Martin Fowler**

# Functional Testing

- Test the functionality of the system
- Discover as many bugs as possible in the **implementation**
- Demonstrate the system is **fit** for its intended **purpose**



# Unit Testing

*your first chance to catch bugs*

- Test a specific piece of code, in **isolation**
  - What about Dependencies?
- Test Double
  - Any kind of **pretend** object used in place of a real object for testing purposes
  - Mocks, Stubs, ....
- Sociable and Solitary Unit Tests

```
public class OrderStateTester extends TestCase {  
    private static String TALISKER = "Talisker";  
    private static String HIGHLAND_PARK = "Highland Park";  
    private Warehouse warehouse = new WarehouseImpl();  
  
    protected void setUp() throws Exception {  
        warehouse.add(TALISKER, 50);  
        warehouse.add(HIGHLAND_PARK, 25);  
    }  
  
    public void testOrderIsFilledIfEnoughInWarehouse() {  
        Order order = new Order(TALISKER, 50);  
        order.fill(warehouse);  
        assertTrue(order.isFilled());  
        assertEquals(0, warehouse.getInventory(TALISKER));  
    }  
}
```

Regular Test

```
public void testFillingDoesNotRemoveIfNotEnoughInStock() {  
    Order order = new Order(TALISKER, 51);  
    Mock warehouse = mock(Warehouse.class);  
  
    warehouse.expects(once()).method("hasInventory")  
        .withAnyArguments()  
        .will(returnValue(false));  
  
    order.fill((Warehouse) warehouse.proxy());  
  
    assertFalse(order.isFilled());  
}
```

Tests with  
Mock Objects

## Tests with Stubs

```
public interface MailService {
    public void send (Message msg);
}

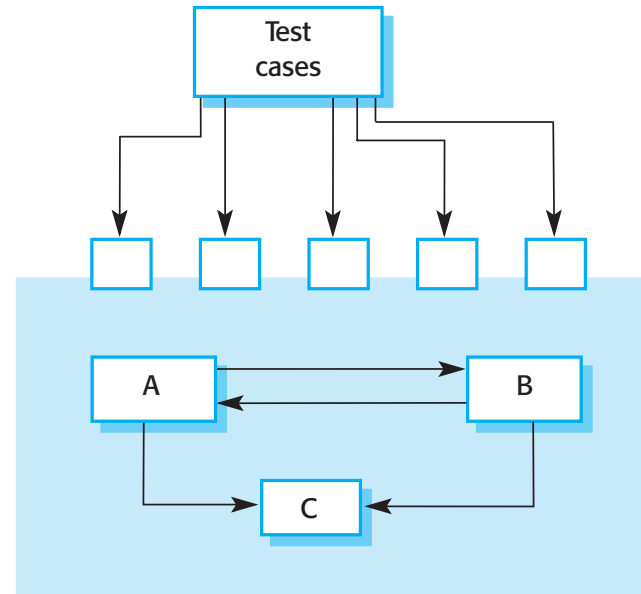
public class MailServiceStub implements MailService {
    private List<Message> messages = new ArrayList<Message>();
    public void send (Message msg) {
        messages.add(msg);
    }
    public int numberSent() {
        return messages.size();
    }
}

public void testOrderSendsMailIfUnfilled() {
    Order order = new Order(TALISKER, 51);
    MailServiceStub mailer = new MailServiceStub();
    order.setMailer(mailer);
    order.fill(warehouse);
    assertEquals(1, mailer.numberSent());
}
```

# Feature Testing

*Component Testing, Integration Testing*

- A functionality is **implemented** as expected and **meets** the real needs of users.
  1. Interaction Tests
    - Interfaces behave according to the spec.
  2. Usefulness Tests
    - feature implements what users are likely to want
- Incremental Integration
  - Top-Down/Bottom-Up
  - Regression Testing
  - Smoke Testing





# System Testing

*end-to-end run-through of the whole system*


- Testing the **system as a whole**, rather than the individual system features.
  - unexpected and unwanted interactions
  - features work together effectively
  - operates in the expected way in the different environments
  - quality attributes: recovery, security, ...
- Scenario-Based Testing
  - **end-to-end** pathways that users might follow when using the system
- Requirements-Based Testing
  - is **validation** rather than defect testing—you are trying to demonstrate that the system has **properly implemented** its requirements.

## Other Testing Categories

Test categories based on scope, focus, or point of view:

Tests based on their scale

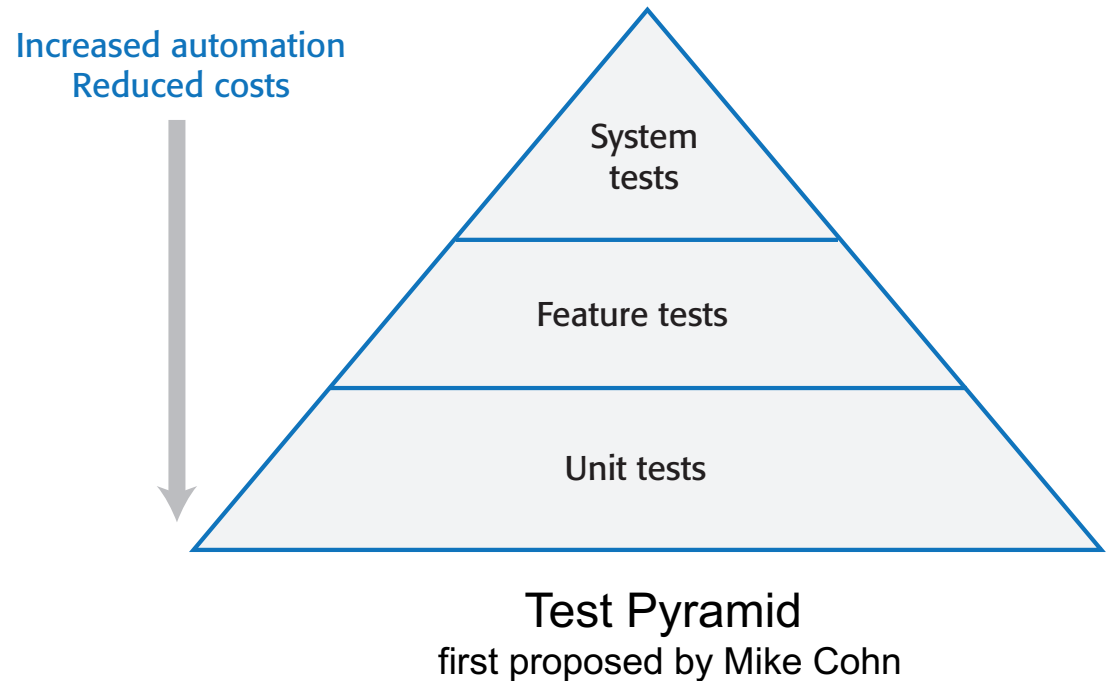
- Unit
- Integration
- System

- 
- Accessibility
  - Alpha
  - Beta
  - Compatibility
  - Destructive
  - Functional
  - Installation
  - Internationalisation
  - Nonfunctional
  - Performance
  - Security
  - Usability
  - ...

# Test Automation

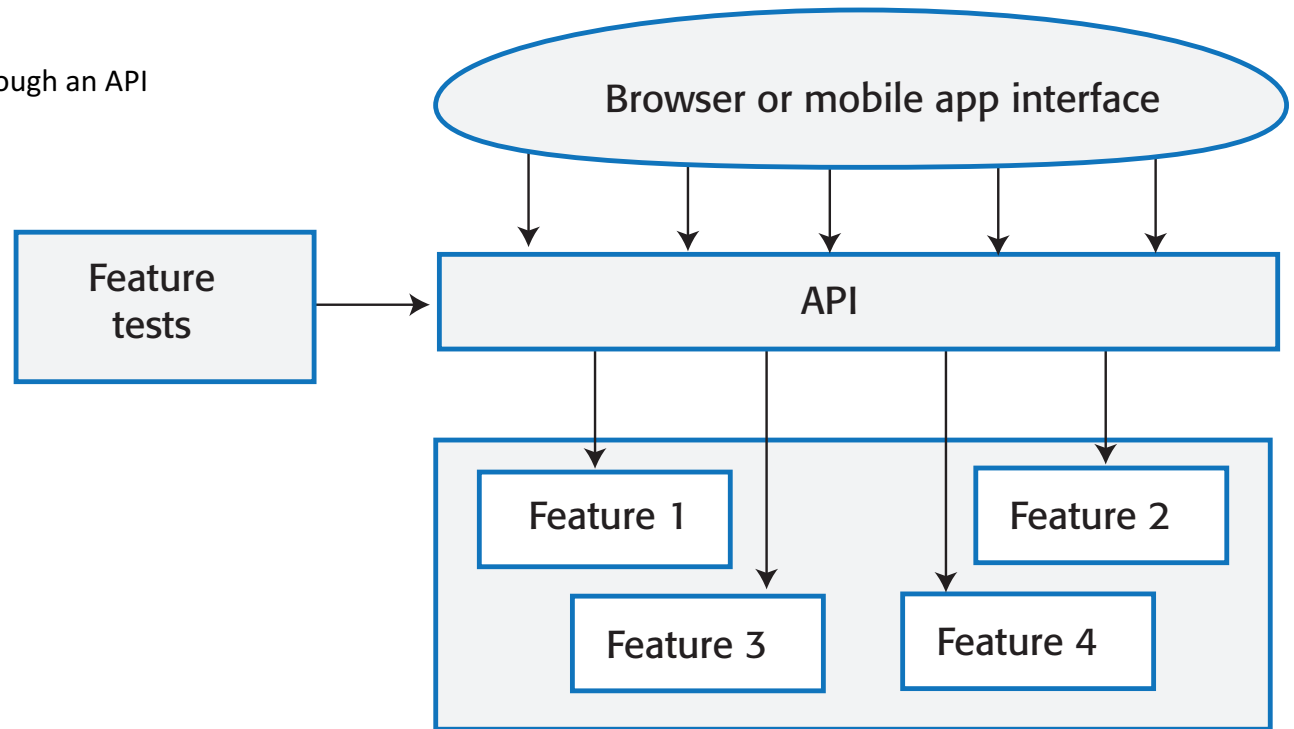
- Executable tests
  - Input
  - Expected output
  - Check the result

- Arrange, Action, Assert
- Setup, Call, Assert



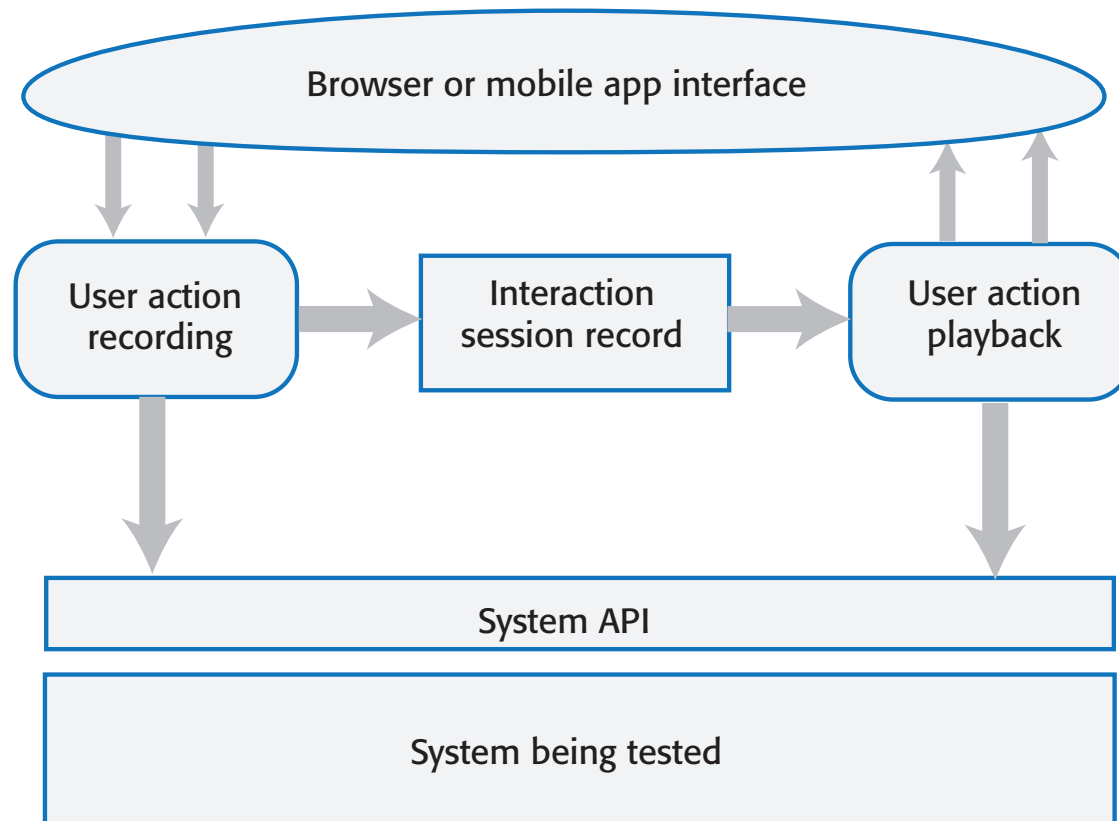
# Automated Feature Testing

- GUI-Based Testing
- Accessing features through an API



# Automated System Testing

*E.g., Record/Playback*



# Effective Tests

Testing is expensive and time consuming

Good Tests?

Effectiveness?

# Unit Testing Guidelines

What are the inputs from each partition that are **most likely to uncover bugs**?

- Test edge cases
- Force errors
- Fill buffers
- Repeat yourself
- Overflow and underflow
- Don't forget null and zero
- Keep count
- One is different
- ....

James A. Whittaker, How to Break Software:  
A Practical Guide to Testing (Boston:  
Addison-Wesley, 2002)

# Choosing Test Cases

## Effective Tests (Test Criteria)

- cover all the code
- cover the behaviour
- cover all the possible state changes
- examine all normal operations
- examine exceptions (abnormal inputs, ....)
- more likely to break the system
- ...

## Test Data

- Random Testing
- Partition Testing
- Guideline-based Testing
  - use testing guidelines to choose test cases, e.g., previous experience of the kinds of errors
- ...



break



# THE 7 HABITS OF HIGHLY EFFECTIVE PEOPLE

Infographics Edition

**Stephen R. Covey**

FranklinCovey.

Over  
30 million  
sold!

## The **7** HABITS Of Highly Effective People

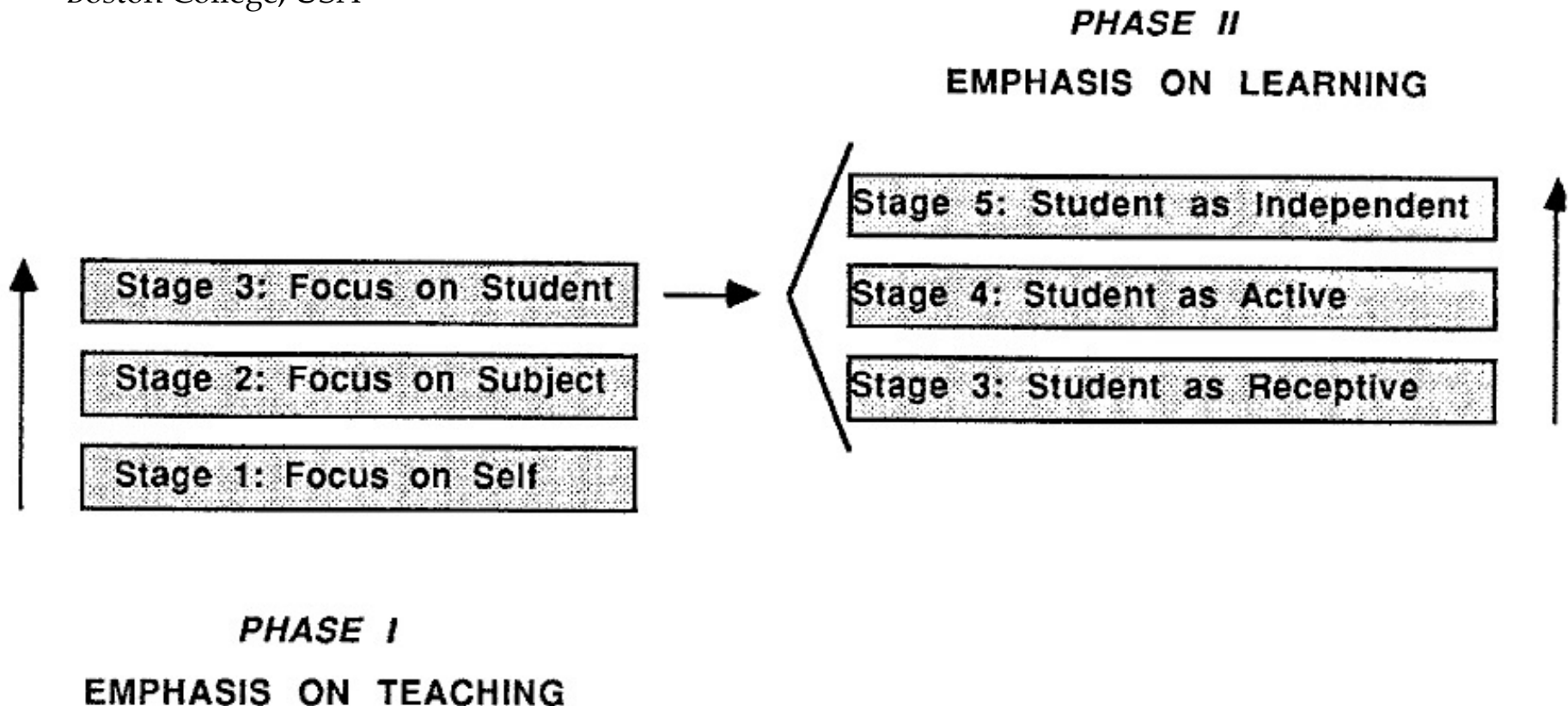
- 1 Be Proactive
- 2 Begin With The End In Mind
- 3 Put First Things First
- 4 Think Win-Win
- 5 Seek First To Understand,  
Then To Be Understood
- 6 Synergize
- 7 Sharpen The Saw



<https://www.youtube.com/watch?v=ktlTxC4QG8g>

# How Professors Develop as Teachers

PETER KUGEL  
Boston College, USA



# Testing Fundamentals

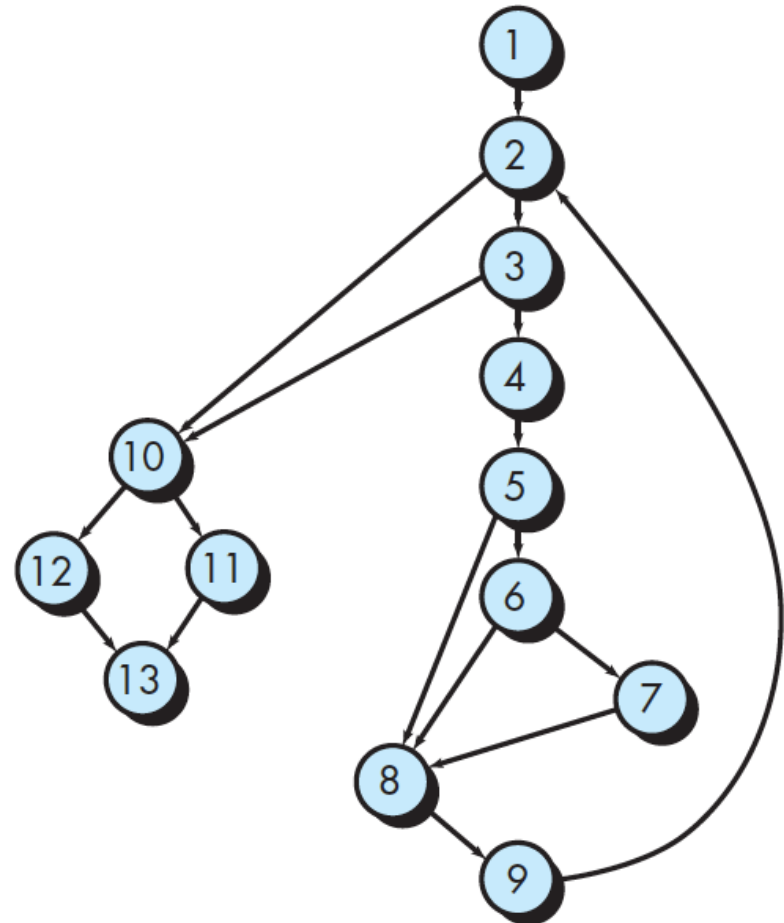
# Testing Techniques

*Internal vs. External View*

- **Black-Box**
  - pretend the method is a black box that you can't peek inside
- **White-Box**
  - know how the method does its work, then use the extra knowledge to design tests to try to make the method crash and burn.
- **Gray-Box**
  - combination of white-box and black-box testing; partial knowledge of the method lets you design specific tests to attack it.

## Path-Based Testing

- Flow Graph Notation
- Independent/Simple Paths



Path 1: 1-2-10-11-13

Path 2: 1-2-10-12-13

Path 3: 1-2-3-10-11-13

Path 4: 1-2-3-4-5-8-9-2- . . .

Path 5: 1-2-3-4-5-6-8-9-2- . . .

Path 6: 1-2-3-4-5-6-7-8-9-2- . . .

## Path-Based Testing

- Flow Graph Notation
- Independent/Simple Paths

Path 1: 1-2-10-11-13

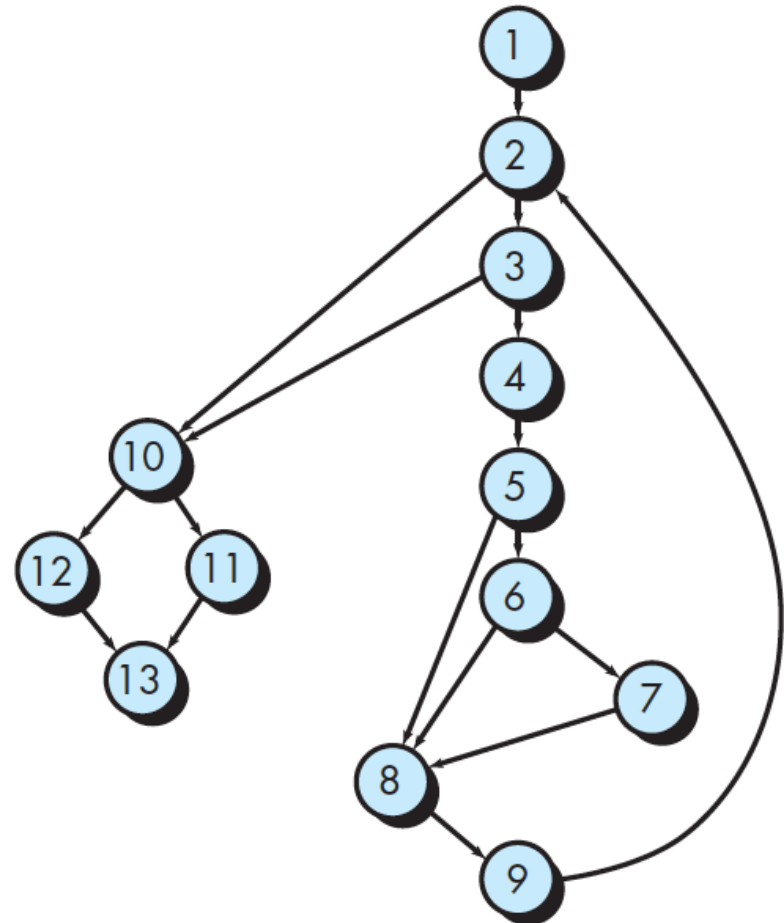
Path 2: 1-2-10-12-13

Path 3: 1-2-3-10-11-13

Path 4: 1-2-3-4-5-8-9-2- . . .

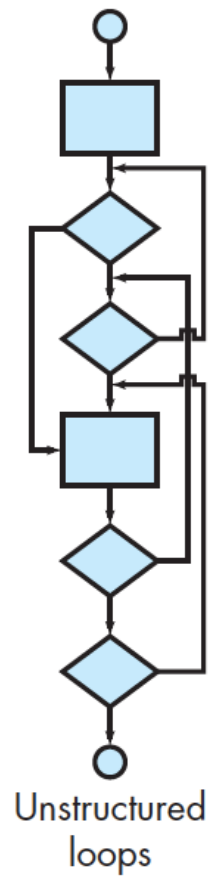
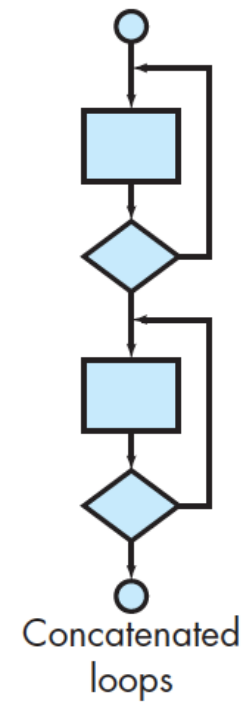
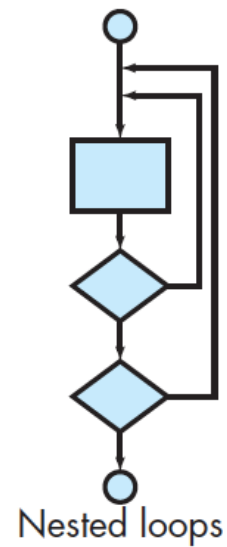
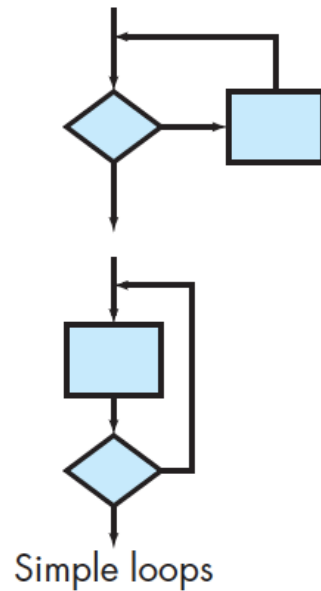
Path 5: 1-2-3-4-5-6-8-9-2- . . .

Path 6: 1-2-3-4-5-6-7-8-9-2- . . .



Test Data?

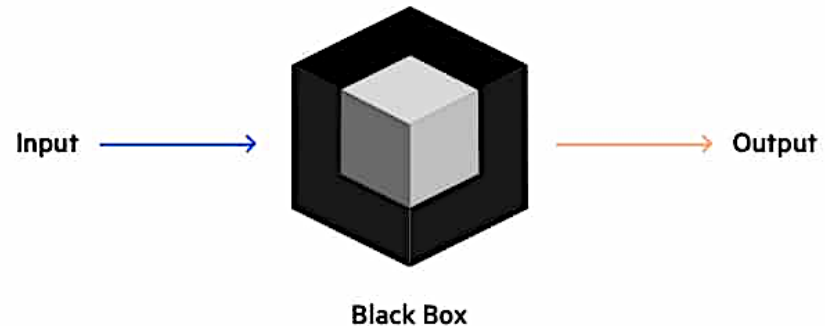
## Control Structure Testing



- Loop Testing
- Condition Testing
  - Exercises the logical conditions contained in a program module
- Data Flow Testing
  - Test paths according to the locations of def. and uses of variables



## Black Box Testing

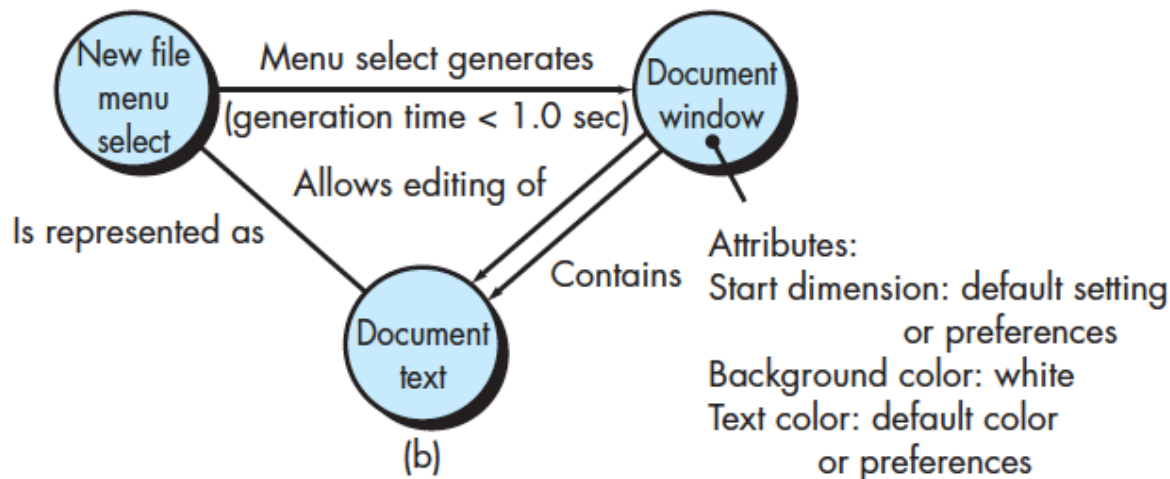
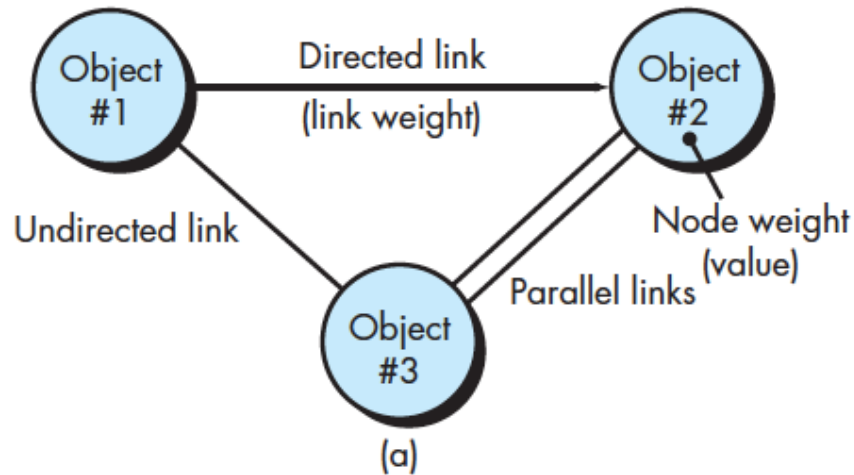


# Black-Box Testing Techniques

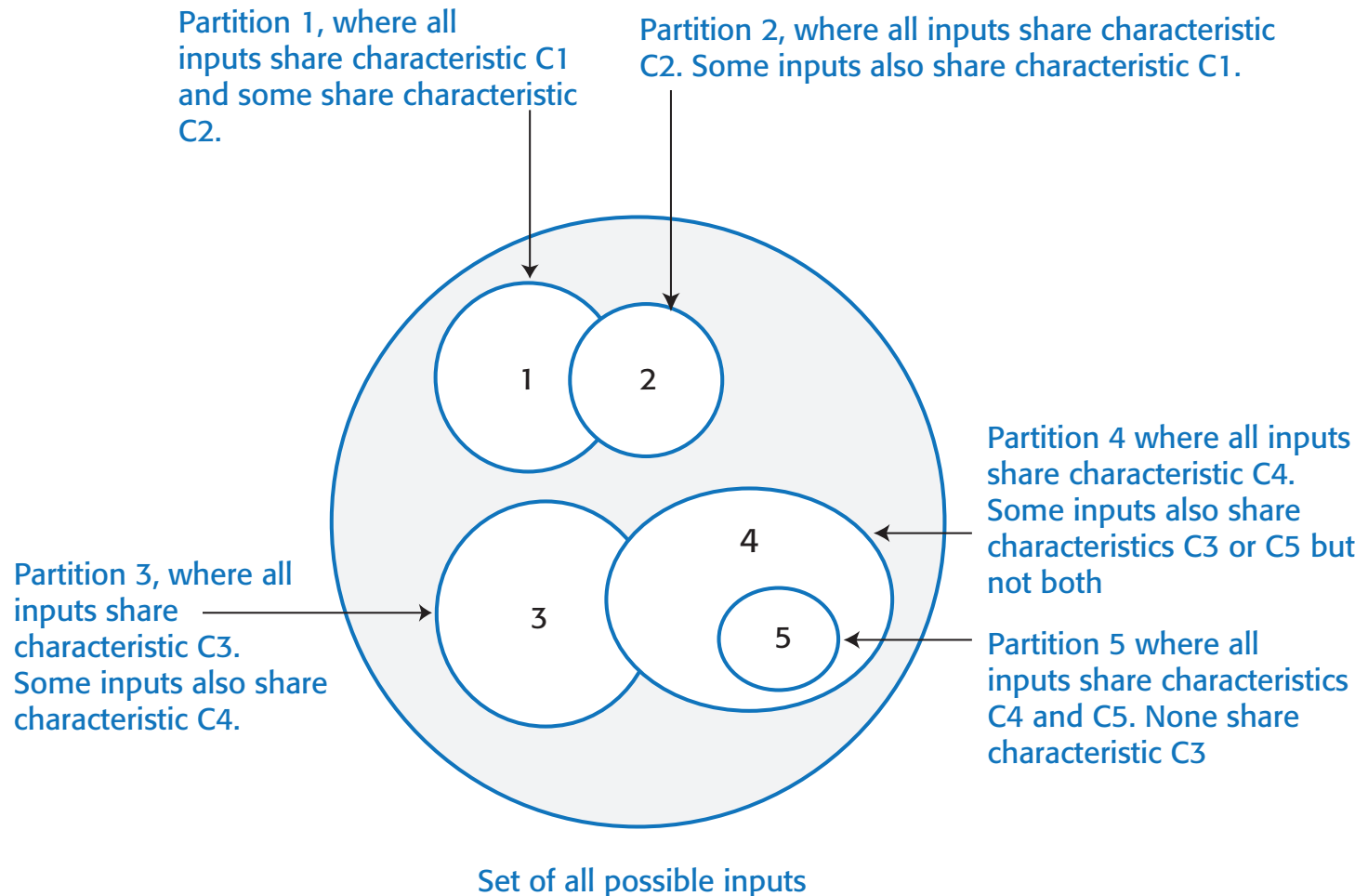
- Graph-based Testing, State Transition Testing
- Equivalence Partitioning
- Boundary Value Analysis
- Decision Table Testing, Orthogonal Array Testing
- Error Guessing!

# Graph-Based Testing

A model of the software system under test (SUT)



# Equivalence Partitioning



# Equivalence Partitioning

- Find **characteristics** in the inputs

Consider the “order of file F”

- **Partition** each characteristic

$\left\{ \begin{array}{l} P_1 = \text{sorted in ascending order} \\ P_2 = \text{sorted in descending order} \\ P_3 = \text{arbitrary order} \end{array} \right.$

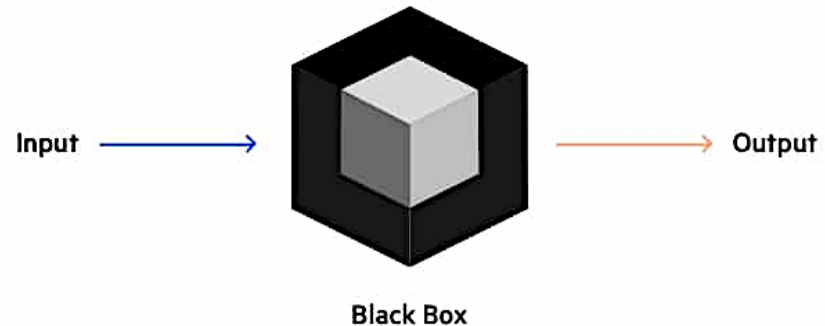
- Choosing (or defining) partitions seems easy, but is **easy to get wrong!**

# Decision Table Testing

*Orthogonal Array Testing*

	Rules							
Conditions	R1	R2	R3	R4	R5	R6	R7	R8
C1:Employment for more than 1 year?	YES	NO	YES	NO	YES	NO	YES	NO
C2:Agreed target?	NO	NO	YES	YES	NO	NO	YES	YES
C3:Achieved target?	NO	NO	NO	NO	YES	YES	YES	YES
Actions								
A1:Bonus payment?	NO	NO	NO	NO	NO	NO	YES	NO

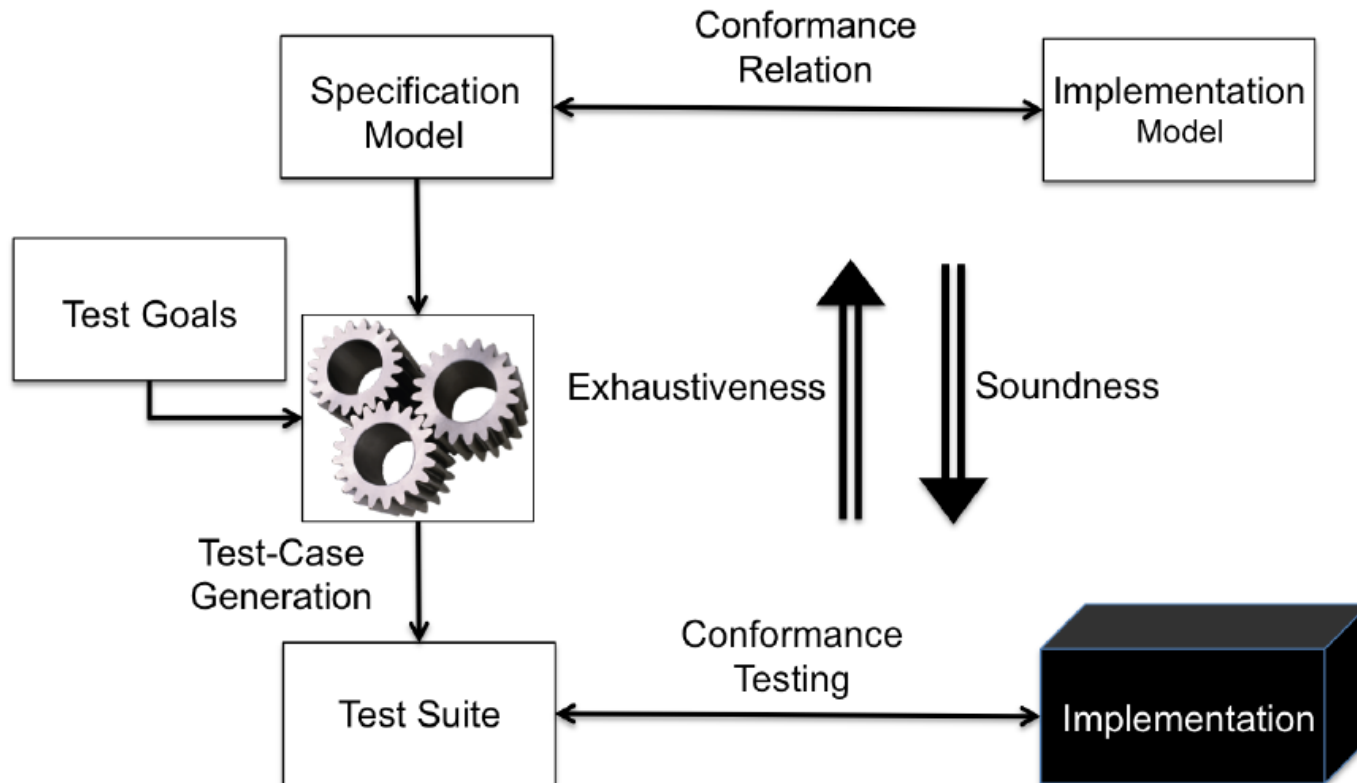
## Black Box Testing



# Black-Box Testing Techniques

- Graph-based Testing, State Transition Testing
- Equivalence Partitioning
- **Boundary Value Analysis**
- Decision Table Testing, Orthogonal Array Testing
- **Error Guessing!**

# Model-Based Testing



# Software Faults, Errors & Failures

**Fault** : A **static** defect in the software

**Failure** : **External**, **incorrect behavior** with respect to the requirements or other description of the expected behavior

**Error** : An **incorrect internal state** that is the manifestation of some fault



# A Concrete Example

```
public static int numZero (int [ ] arr)
{ // Effects: If arr is null throw NullPointerException
  // else return the number of occurrences of 0 in arr
  int count = 0;
  for (int i = 1; i < arr.length; i++)
  {
    if (arr [ i ] == 0)
    {
      count++;
    }
  }
  return count;
}
```

# A Concrete Example

**Fault:** Should start searching at 0, not 1

```
public static int numZero (int [ ] arr)
{ // Effects: If arr is null throw NullPointerException
  // else return the number of occurrences of 0 in arr
  int count = 0;
  for (int i = 1; i < arr.length; i++)
  {
    if (arr [ i ] == 0)
    {
      count++;
    }
  }
  return count;
}
```

# A Concrete Example

**Fault:** Should start searching at 0, not 1

```
public static int numZero (int [ ] arr)
{ // Effects: If arr is null throw NullPointerException
  // else return the number of occurrences of 0 in arr
  int count = 0;
  for (int i = 1; i < arr.length; i++)
  {
    if (arr [ i ] == 0)
    {
      count++;
    }
  }
  return count;
}
```

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

# A Concrete Example

**Fault:** Should start searching at 0, not 1

```
public static int numZero (int [ ] arr)
{ // Effects: If arr is null throw NullPointerException
  // else return the number of occurrences of 0 in arr
  int count = 0;
  for (int i = 1; i < arr.length; i++)
  {
    if (arr [ i ] == 0)
    {
      count++;
    }
  }
  return count;
}
```

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

**Error:** i is 1, not 0, on the first iteration  
**Failure:** none

# A Concrete Example

**Fault:** Should start searching at 0, not 1

```
public static int numZero (int [ ] arr)
{ // Effects: If arr is null throw NullPointerException
  // else return the number of occurrences of 0 in arr
  int count = 0;
  for (int i = 1; i < arr.length; i++)
  {
    if (arr [ i ] == 0)
    {
      count++;
    }
  }
  return count;
}
```

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

# A Concrete Example

**Fault:** Should start searching at 0, not 1

```
public static int numZero (int [ ] arr)
{ // Effects: If arr is null throw NullPointerException
  // else return the number of occurrences of 0 in arr
  int count = 0;
  for (int i = 1; i < arr.length; i++)
  {
    if (arr [ i ] == 0)
    {
      count++;
    }
  }
  return count;
}
```

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

**Error:** i is 1, not 0

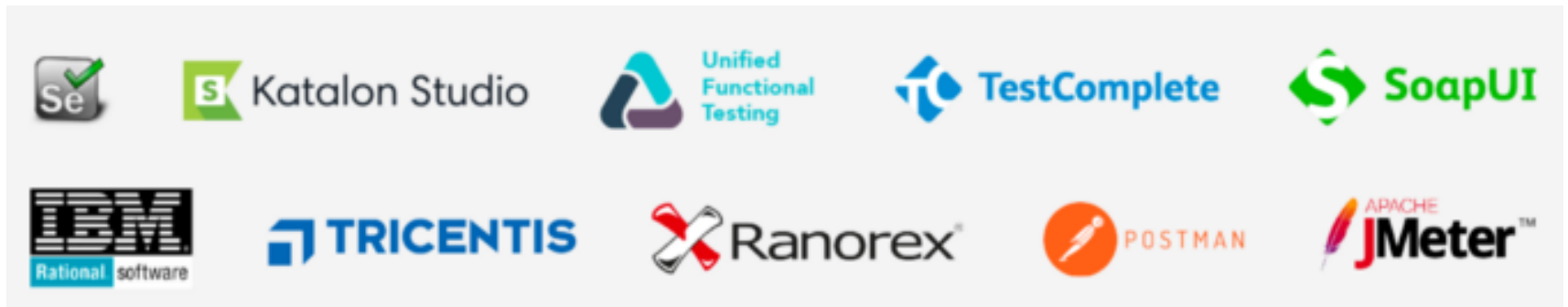
Error propagates to the variable count

**Failure:** count is 0 at the return statement

# Patterns for Software Testing

**Testing patterns** describe common testing problems and solutions that can assist you in dealing with them.

# Testing Tools



- Selenium
- Jenkins
- Postman
- Ranorex
- HP UFT
- GitHub
- Cucumber
- ?