TIC4302 - Information Security Practicum II Course

Introduction to Software Testing

Adopted from “Software Testing and Automation Specialization” Course
by **University of Minnesota**

Software
Engineering

Verification and Validation

Verification

- "Are we building the product **right**?"
- The software should conform to its specification

Validation

- "Are we building the **right** product?"
- The software should do what the user really requires

Goal: Assure the software meets user's needs

Techniques for Getting Software "Right"

Need to **understand** and **validate** software requirements

Need to apply multiple V&V techniques throughout the development cycle

- Inspections
- Design discussions
- Static analysis
- Testing
- Runtime verification

Why focus on Software Testing?

The only software defect detection technique that can check the **whole system**

Compiler, Processor, Devices, Network, Linker, Loader, etc…

Currently, the best way to accurately assess some system behaviors (e.g., performance)

For contract software, necessary for customer to “accept” the system

A reasonably good way of documenting expected system behavior

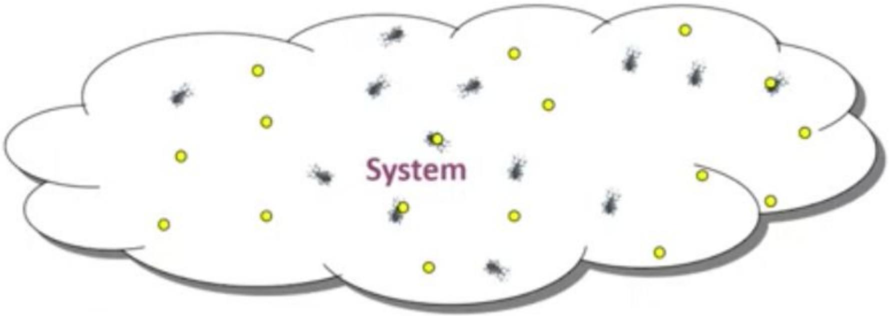
But testing is <u>always incomplete</u>

Our goal is to make it *effective* despite incompleteness

Why is Testing Hard?

Software Engineering

The Problem with Software Testing



…is that it only samples a set of possible behaviors

Unlike physical systems (where many engineers gained their experience), most software systems are discontinuous

There is no sound basis for extrapolating from tested to untested cases

So we need to consider all possible states of the system

Even small systems have trillions and trillions of possible states

Testing from 10,000 feet

Scale

- **Unit tests:** testing individual classes / functions
- **Integration tests:** testing packages / subsystems
- **System tests:** testing the entire system
  - In web & service world, the concept of "system" is fluid!

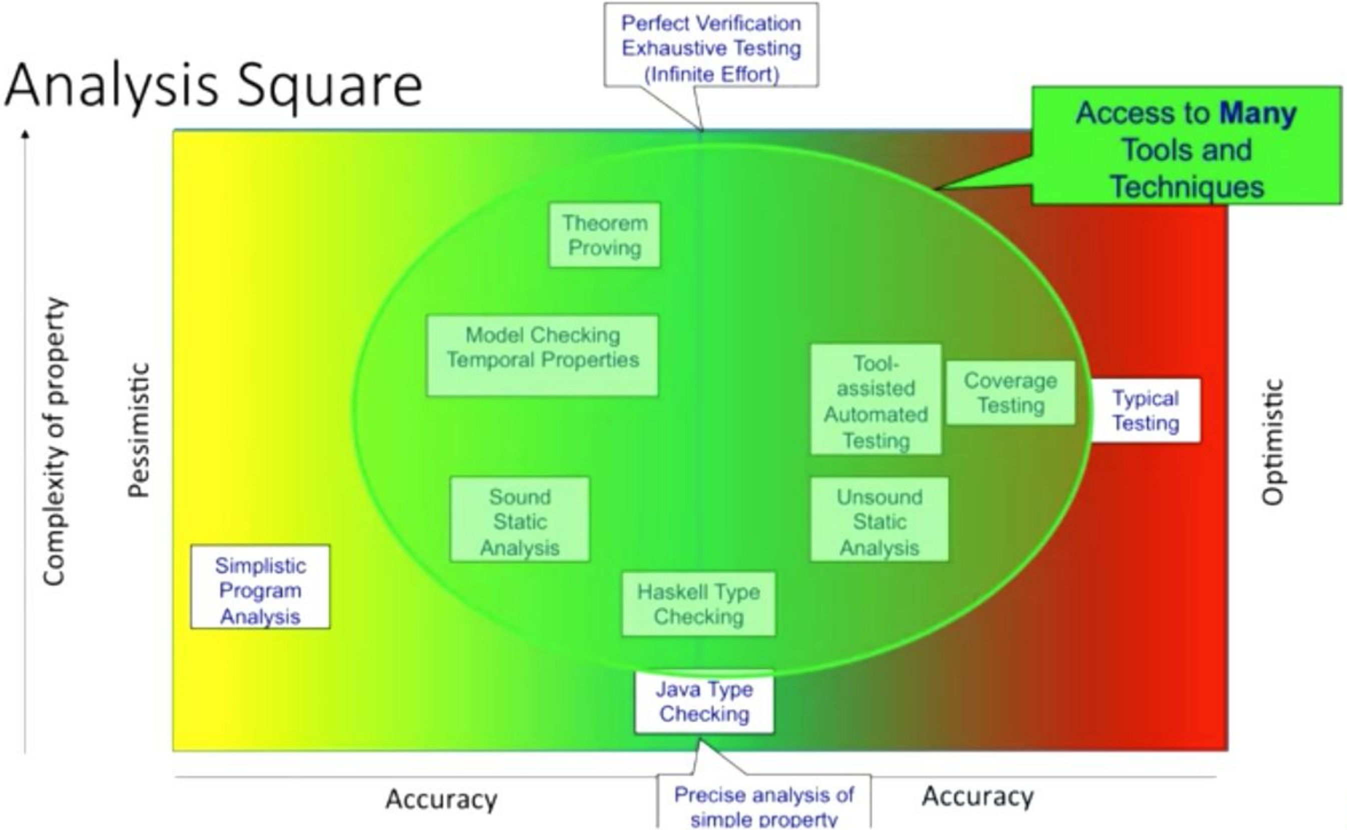
Process

- Test first: test-driven development (TDD)
  - Write the tests **before the code.**
  - Write code to pass the tests
- Test after
  - Check whether existing code passes tests
  - Even TDD is "test after" most of the time (refactoring)
- Iteration
  - Either way, you will spend most of your time re-testing

Purpose

- Functional Testing
- Performance Testing
- Security Testing
- Usability Testing
- Availability Testing

Analysis Square



Perfect Verification
Exhaustive Testing
(Infinite Effort)

Access to **Many** Tools and Techniques

Complexity of property

Pessimistic

Optimistic

Theorem Proving

Model Checking Temporal Properties

Tool-assisted Automated Testing

Coverage Testing

Typical Testing

Sound Static Analysis

Unsound Static Analysis

Haskell Type Checking

Simplistic Program Analysis

Java Type Checking

Accuracy

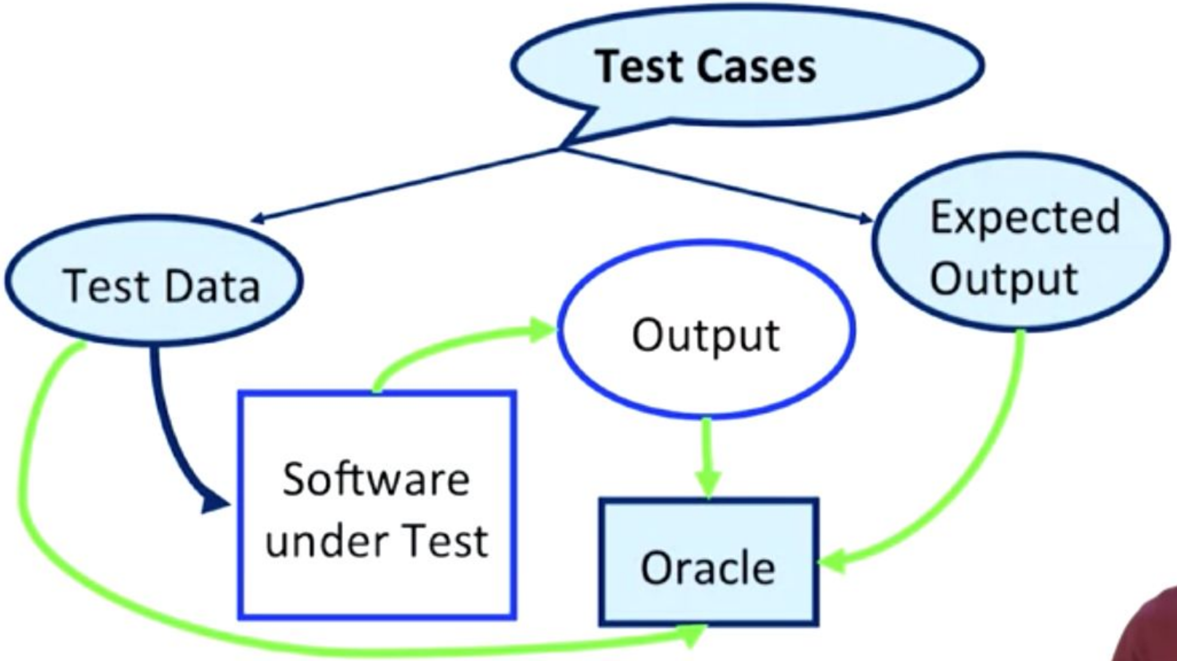
Precise analysis of simple property

Accuracy

What is a test?

Software Engineering

What is a Test?



Test Cases

Test Data

Expected Output

Output

Software under Test

Oracle

Dissecting the anatomy of a test

Setup

Invocation

Assessment

Teardown

A test is made up of test data (input) and expected output

Test Cases

- Test Data
- Expected Output

Software Under Test

- Actual Output

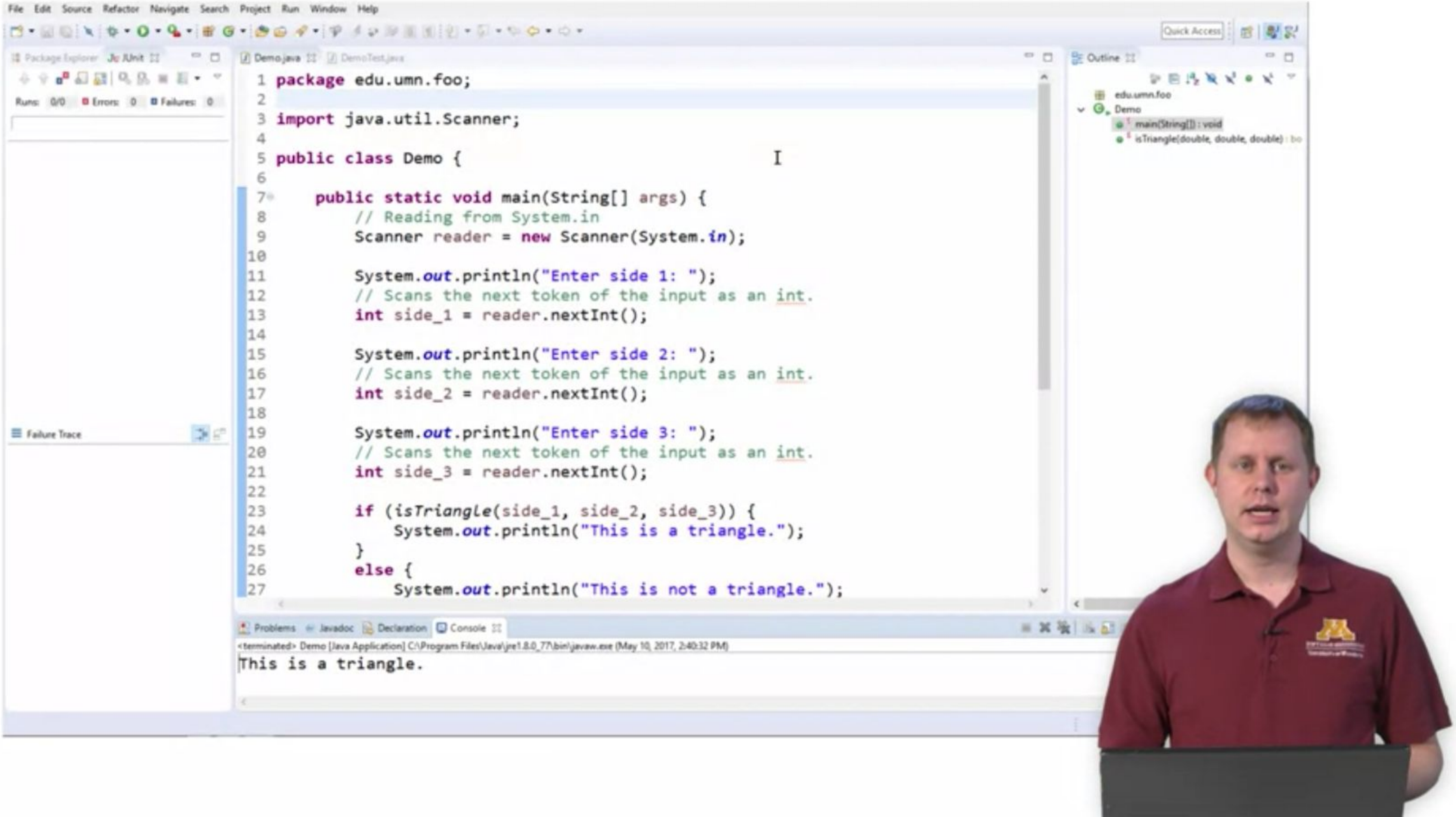
Oracle

Setup

Invocation

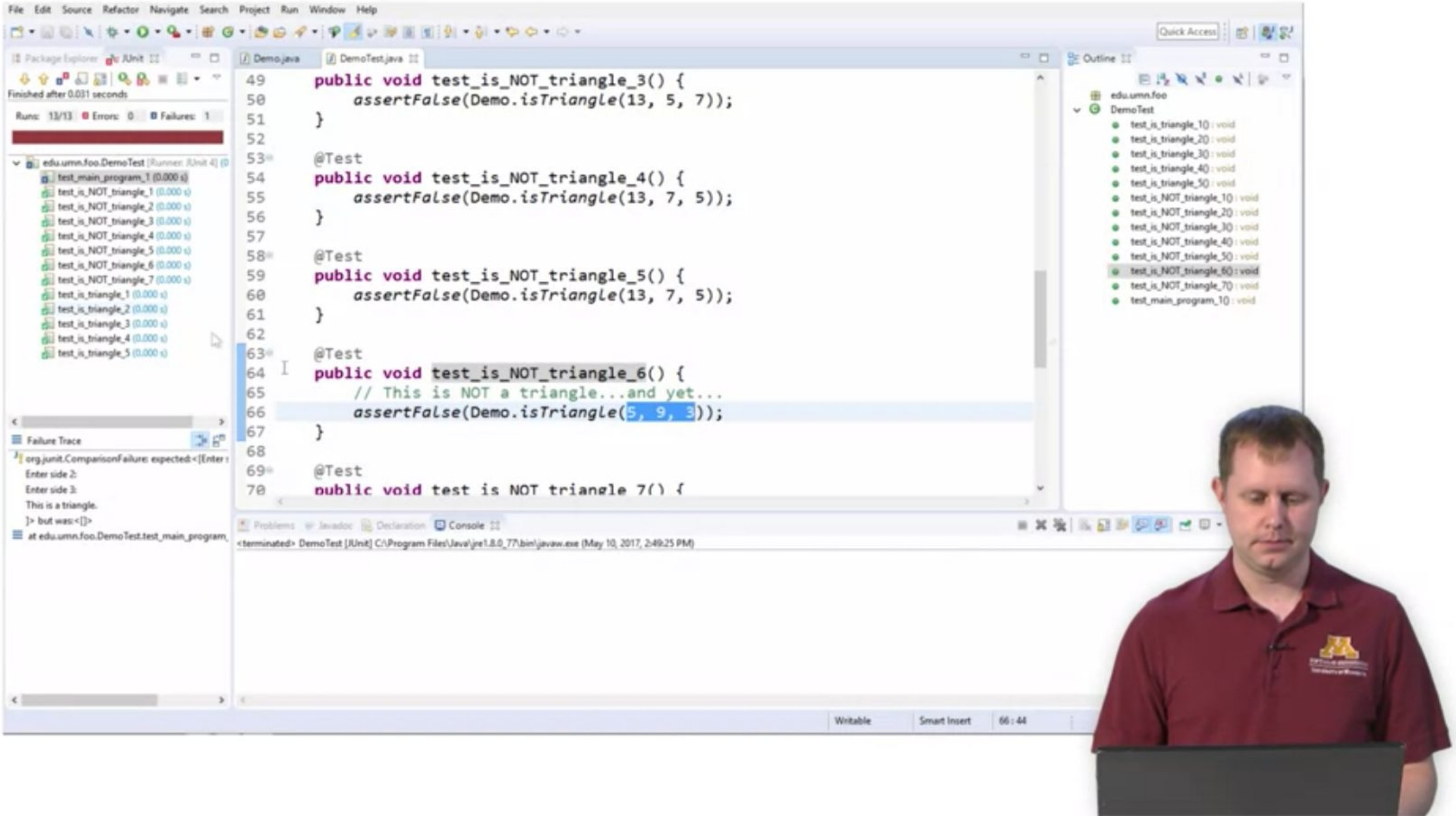
Assessment

Teardown



```java
package edu.umn.foo;

import java.util.Scanner;

public class Demo {

    public static void main(String[] args) {
        // Reading from System.in
        Scanner reader = new Scanner(System.in);

        System.out.println("Enter side 1: ");
        // Scans the next token of the input as an int.
        int side_1 = reader.nextInt();

        System.out.println("Enter side 2: ");
        // Scans the next token of the input as an int.
        int side_2 = reader.nextInt();

        System.out.println("Enter side 3: ");
        // Scans the next token of the input as an int.
        int side_3 = reader.nextInt();

        if (isTriangle(side_1, side_2, side_3)) {
            System.out.println("This is a triangle.");
        }
        else {
            System.out.println("This is not a triangle.");
```

```
<terminated> Demo [Java Application] C:\Program Files\Java\jre1.8.0_77\bin\javaw.exe (May 10, 2017, 2:40:32 PM)
This is a triangle.
```



```
Finished after 0.031 seconds

Runs: 13/13   Errors: 0   Failures: 1

edu.umn.foo.DemoTest [Runner: JUnit 4] (0
    test_main_program_1 (0.000 s)
    test_is_NOT_triangle_1 (0.000 s)
    test_is_NOT_triangle_2 (0.000 s)
    test_is_NOT_triangle_3 (0.000 s)
    test_is_NOT_triangle_4 (0.000 s)
    test_is_NOT_triangle_5 (0.000 s)
    test_is_NOT_triangle_6 (0.000 s)
    test_is_NOT_triangle_7 (0.000 s)
    test_is_triangle_1 (0.000 s)
    test_is_triangle_2 (0.000 s)
    test_is_triangle_3 (0.000 s)
    test_is_triangle_4 (0.000 s)
    test_is_triangle_5 (0.000 s)
```

Failure Trace

```
org.junit.ComparisonFailure: expected:<[Enter s
  Enter side 2:
  Enter side 3:
  This is a triangle.
  ]> but was:<[]>
at edu.umn.foo.DemoTest.test_main_program_
```

```java
    public void test_is_NOT_triangle_3() {
        assertFalse(Demo.isTriangle(13, 5, 7));
    }

    @Test
    public void test_is_NOT_triangle_4() {
        assertFalse(Demo.isTriangle(13, 7, 5));
    }

    @Test
    public void test_is_NOT_triangle_5() {
        assertFalse(Demo.isTriangle(13, 7, 5));
    }

    @Test
    public void test_is_NOT_triangle_6() {
        // This is NOT a triangle...and yet...
        assertFalse(Demo.isTriangle(5, 9, 3));
    }

    @Test
    public void test is NOT triangle 7() {
```

```
edu.umn.foo
DemoTest
    test_is_triangle_1() : void
    test_is_triangle_2() : void
    test_is_triangle_3() : void
    test_is_triangle_4() : void
    test_is_triangle_5() : void
    test_is_NOT_triangle_1() : void
    test_is_NOT_triangle_2() : void
    test_is_NOT_triangle_3() : void
    test_is_NOT_triangle_4() : void
    test_is_NOT_triangle_5() : void
    test_is_NOT_triangle_6() : void
    test_is_NOT_triangle_7() : void
    test_main_program_1() : void
```

```
<terminated> DemoTest [JUnit] C:\Program Files\Java\jre1.8.0_77\bin\javaw.exe (May 10, 2017, 2:49:25 PM)
```

What is a test plan?

Software Engineering

Stages of Software Testing Process

Unit Test

Unit Test Plan

Design Verification Test

-- (Integration Test)
-- Functional Testing

DVT Test Plan/Test Cases

System Validation Test

-- System Test
-- Non-functional Testing
-- Test Report

SVT Test Plan/Test Cases

Customer Acceptance Test

Customer Acceptance Test Plan

Components of a Test Plan

Testing approach/strategy

Scope

Schedule

Resources/Test Environment

Entry and Exit criteria

Requirements Matrix (for Traceability)

What is NOT tested

Test cases and scripts [separate document(s)]

Test Plan Activities

Use a Test Plan template, or design one

List what cannot be tested

Write only what you need

Have the Test Plan reviewed

Make it a “living” document

Why we need a good test plan

1 2

Organize, schedule, and manage testing effort

Helps in writing test cases

Improves communication between developers and management

Why we need a good test plan 1 2

Measuring software quality is the intent (and must be planned)

Developing good test sets takes planning

Knowing when to stop

More effective arguments when you have the facts