

Quality Assurance Activities

- **Verification**
 - Check product against specification
- **Validation**
 - Check product against world (stakeholder expectations)
- **van Vliet considers all quality assurance activities as testing**
 - Even before you have code!

Testing

- Software lifecycle diagrams show testing as an activity or box
 - In practice, testing is performed constantly
- There has never been a project where there was too much testing.
 - Products always ship with some defects
- Test cases are a valuable resource
 - Should be managed like code
 - Used for testing as you add code

Testing in Lifecycle Phases

Phase	Activities
Requirements	<ul style="list-style-type: none">-Determine test strategy-Test requirements specification-Generate functional test data
Design	<ul style="list-style-type: none">-Check consistency between design and requirements specification-Evaluate the software architecture-Test the design-Generate structural and functional test data
Implementation	<ul style="list-style-type: none">-Check consistency between design and implementation-Test implementation-Generate structural and functional test data-Execute tests
Maintenance	<ul style="list-style-type: none">-Repeat the above tests in accordance with the degree of redevelopment

Review

- Name and describe four types of testing.

Unit testing

Error-based testing

Structural testing

Integration testing

Fault-based testing

Functional testing

System testing

Coverage-based testing

Acceptance testing

- What is the difference between black box and white box testing?

Test Stages

- Unit Testing
 - Same as Module Testing
- Integration Testing
- System Testing
- Acceptance Testing
- Installation Testing

Testing Techniques

- Manual Test Techniques
 - Reading
 - Walkthroughs and Inspections
- Scenario-Based Evaluation
- Correctness Proofs
- Stepwise Abstraction
- Coverage-Based Techniques
- Fault-Based Techniques
- Error-Based Techniques

Properties of Good Tests

- Automatic
- Thorough
- Accurate
- Repeatable
- Independent
- Professional

Benefits of Unit Tests

- Specify software
- Reconcile conflicting concepts
- Clarify assumptions
- Starting point for QA
- Code is written to be testable

Junit

- Widely accepted standard for unit testing
- Simple
- Automatically compares expected and actual results
- Easy to run (and rerun!) many tests
- Oriented towards developers. Unit tests
- Free, open source

- It's a tool, not a panacea
 - Still need to design good test cases
 - Expose assumptions!

To test: Add.java

```
package add;

public class Add {
    public int addInts(int int1, int int2)
    {
        return int1 + int2;
    }
}
```

AddTest1.java – as generated

```
package addUnitTest;
import junit.framework.TestCase;

public class AddTest1 extends TestCase {
    protected void setUp() throws Exception {
        super.setUp();
    }
    protected void tearDown() throws Exception {
        super.tearDown();
    }
    /*
     * Test method for 'add.Add.addInts(int, int)'
     */
    public void testAddInts() {
    }
}
```

AddTest1.java – as edited, part 1

```
import add.Add;

public class AddTest1 extends TestCase {
    Add myAdd;
    protected void setUp() throws Exception {
        myAdd = new Add();
        super.setUp();
    }
    protected void tearDown() throws Exception {
        myAdd = null;
        super.tearDown();
    }
    /*
     * Test method for 'add.Add.addInts(int, int)'
     */
    public void testAddInts() {
    }
}
```

AddTest1.java - as edited, part 2

```
public class AddTest1 extends TestCase {
    /*
     * Test method for 'add.Add.addInts(int, int)'
     */
    public void testAddInts() {
        Assert.assertEquals(4, myAdd.addInts(2, 2));
        Assert.assertEquals(-4, myAdd.addInts(-2, -2));
        Assert.assertEquals(27, myAdd.addInts(28, -2));
    }
}
```

Run the JUnit – results:

Package Explorer Hierarchy JUnit

Finished after 0.016 seconds

Runs: 1/1 Errors: 0 Failures: 1

Failures Hierarchy

testAddInts - addUnitTest.AddTest1

Failure Trace

```
junit.framework.AssertionFailedError: expected: <27> but was: <26>
at addUnitTest.AddTest1.testAddInts(AddTest1.java:26)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
```