Summary

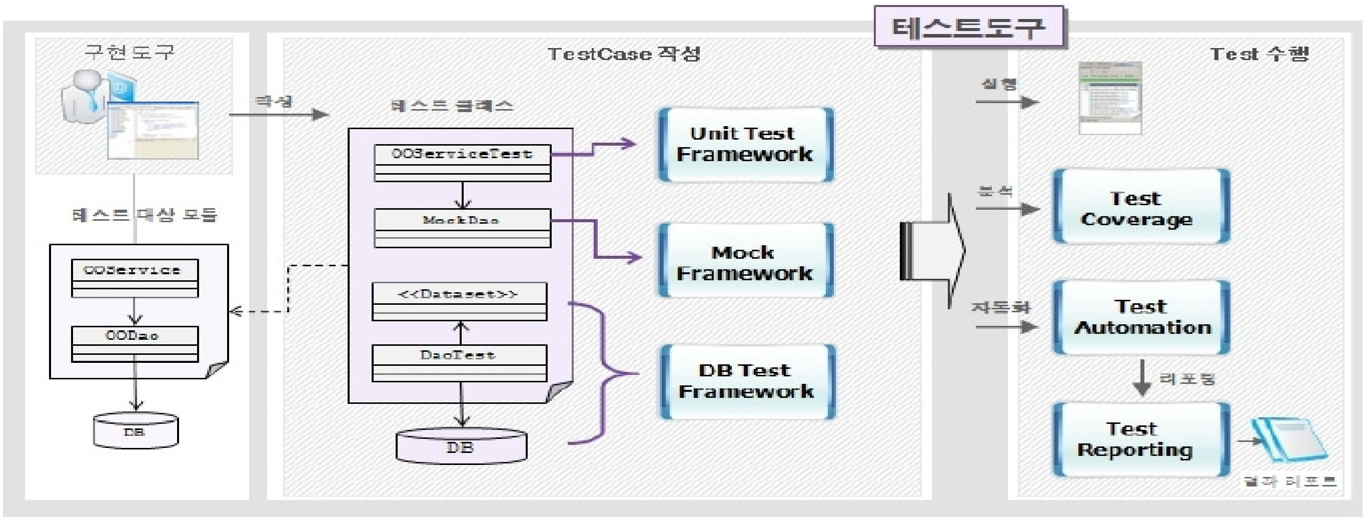
This guide will discuss Unit Tests that the developers can run tests with in order to test their own developed codes.

Description

Unit test is a test that developers create in order to test their own developed codes, and usually has a format of running certain methods to see if it returns expected values; these Unit Tests need to be ran independently of each other.

- Benefits
  - Enhances code quality while codes are being developed
  - Facilitates discovering bugs while editing code
  - Enables automated regression tests
- Coverage
  - Main code path ("The happy path")
  - The main alternative path
  - Boundary test cases (i.e. null checking)
  - Exception testing

The tool provided with the IDE supports Unit Test, Mock, DB Test Framework, and also provides Test Automation, Test Coverage, and Test Reporting.



Writing Test Cases

Any code being developed by developers can be targeted for test cases. For example, Unit Test Framework that can test against OOService class is called OOServiceTest. ("Test Case")

Libraries for creating MockDao for OODao that is related to OOService the developer wants to test are provided. ("Mock Support")

DB Test Framework for testing Persistence Layer across Dao and DB is provided for creating test cases to test DB connection, data initialization, transaction handling and so on. ("DB Support")

Running tests

Runs test cases and returns fail/success results.
Provides numerical feedback on test coverage (i.e. code lines, percentage) to indicate what the test was not able to cover. ("Test Coverage")

Test cases can be run automated repeatedly. - Test Automation

Test Reporting

- Coverage Analysis: Analyzes test coverage for a given test, and displays the results - Test Coverage
- Test Reporting: Test results can be converted into reports in text, HTML, XML, Excel and other formats - Test Reporting

Open source components

Name	Version	Description
JUnit [http://junit.org/]	4.4, 4.3	Provides basic features for Unit test class creation and execution.
EasyMock [http://www.e asym ock .org/]	2.4	Used to perform Mocking when creating Unit test classes.
DbUnit [http://dbunit.source forge .ne t/]	2.4.2	Used for DB Support such as synthesizing database-related Test Fixture.
EMMA [http://e m m a.source forge .ne t/]	2.0	Test C overage Analysis.
Spring Test [http://www.springsource .org/]	2.5.6	Exploits Test related features of Springframework
Unitils [http://unitils.source forge .ne t/]	2.2	Provides a base structure and utilities to flexibly combine JUnit, EasyMock, DbUnit, Spring Test, Ibatis, Hibernate to test.
Ant [http://ant.apache .org/]	1.6.5 or higher	Java-based build tool
Maven [http://unitils.source forge .ne t/sum m ary.html]	2.0	Project Management Tool
EclEmma [http://www.e cle m m a.org]	1.3.2	EMMA Eclipse Plug-in

Environmental settings

For Maven projects, set dependencies as below.

```
<dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.4</version>
      <scope>test</scope>
      </dependency>
<dependency>
      <groupId>org.easymock</groupId>
      <artifactId>easymock</artifactId>
      <version>2.4</version>
      <scope>test</scope>
</dependency>

<dependency>
      <groupId>org.easymock</groupId>
      <artifactId>easymockclassextension</artifactId>
      <version>2.4</version>
      <scope>test</scope>
</dependency>

<dependency>
      <groupId>org.dbunit</groupId>
      <artifactId>dbunit</artifactId>
      <version>2.4.3</version>
      <scope>test</scope>
</dependency>

<dependency>
      <groupId>org.unitils</groupId>
      <artifactId>unitils</artifactId>
      <version>2.2</version>
      <scope>test</scope>
</dependency>
```

✔ Note: If you are using Springframework 2.5, use JUnit 4.4 instead of 4.5.
✔ Note: To find out more about Test Automation / Test Reporting / Test Coverage, see Test Automation , Test Reporting , Test Coverage.
✔ You can run Maven > Update project configuration if needed.

Manual

Unit test cases organized as below.

- Class that uses the test framework
- Test fixture commonly used for testing
- Setup of test data
- Test methods
    - Individual preparations
    - Run test methods
    - User asserts for result checking (assertTrue, assertEquals etc.)

See the Test Case for basic explanations of JUnit, and also refer to Mock Support, DB Support if needed.

Examples

- Basic JUnit Test Cases: Test Case
- Mocking-based Test Cases: Mock Support
- DAO-based Test Cases: DB Support

References

- Unitils Guildelines http://unitils.sourceforge.net/guidelines.html [http://unitils.sourceforge.net/guidelines.html]
- Using Mock Object for easy testing http://www.ibm.com/developerworks/kr/event/screencast/final/01/[http://www.ibm.com/developerworks/kr/event/screencast/final/01/]
- Effective Unit Testing with DbUnit http://www.onjava.com/pub/a/onjava/2004/01/21/dbunit.html [http://www.onjava.com/pub/a/onjava/2004/01/21/dbunit.html]
- An early look at JUnit4 http://www.ibm.com/developerworks/java/library/j-junit4.html [http://www.ibm.com/developerworks/java/library/j-junit4.html]