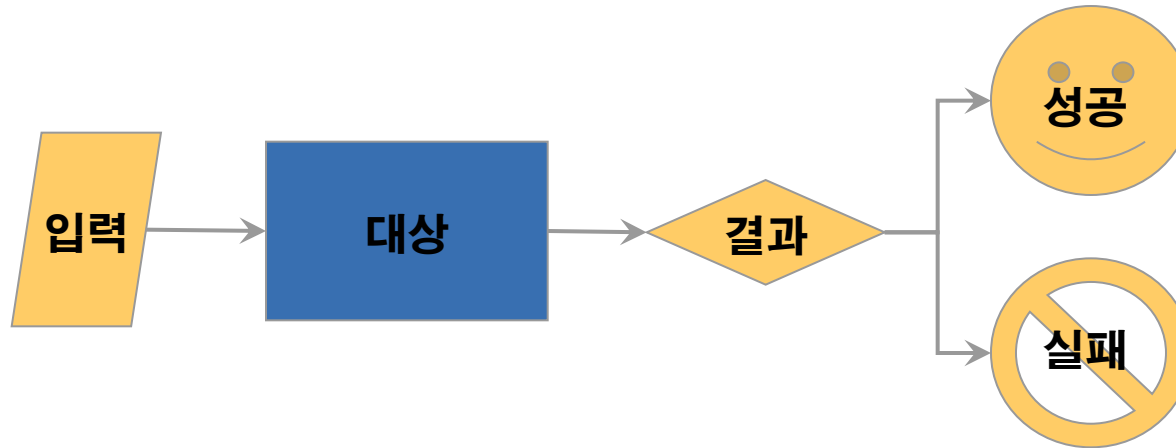


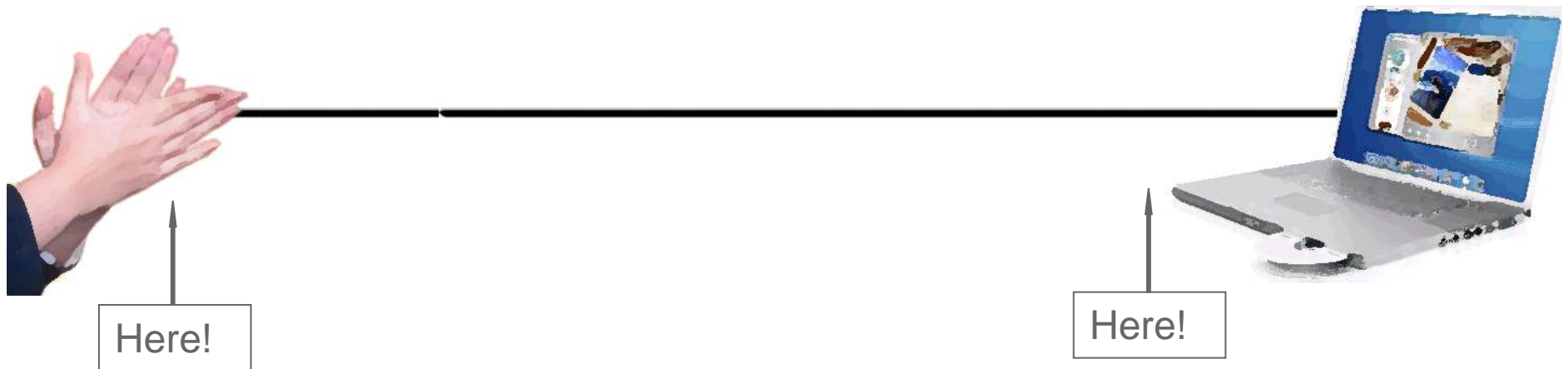
1. 테스트
2. 개요
3. 오픈소스
4. 테스트 종류
5. Unit Test
6. Mock
7. Batch Job Test
8. DB Test
9. Test Automation
10. Test Reporting
11. Test Coverage
12. 참고자료

□ 테스트(Test)

- 테스트 대상에 입력값을 넣었을 때 그 결과가 성공 혹은 실패의 결과를 내는 것이다.

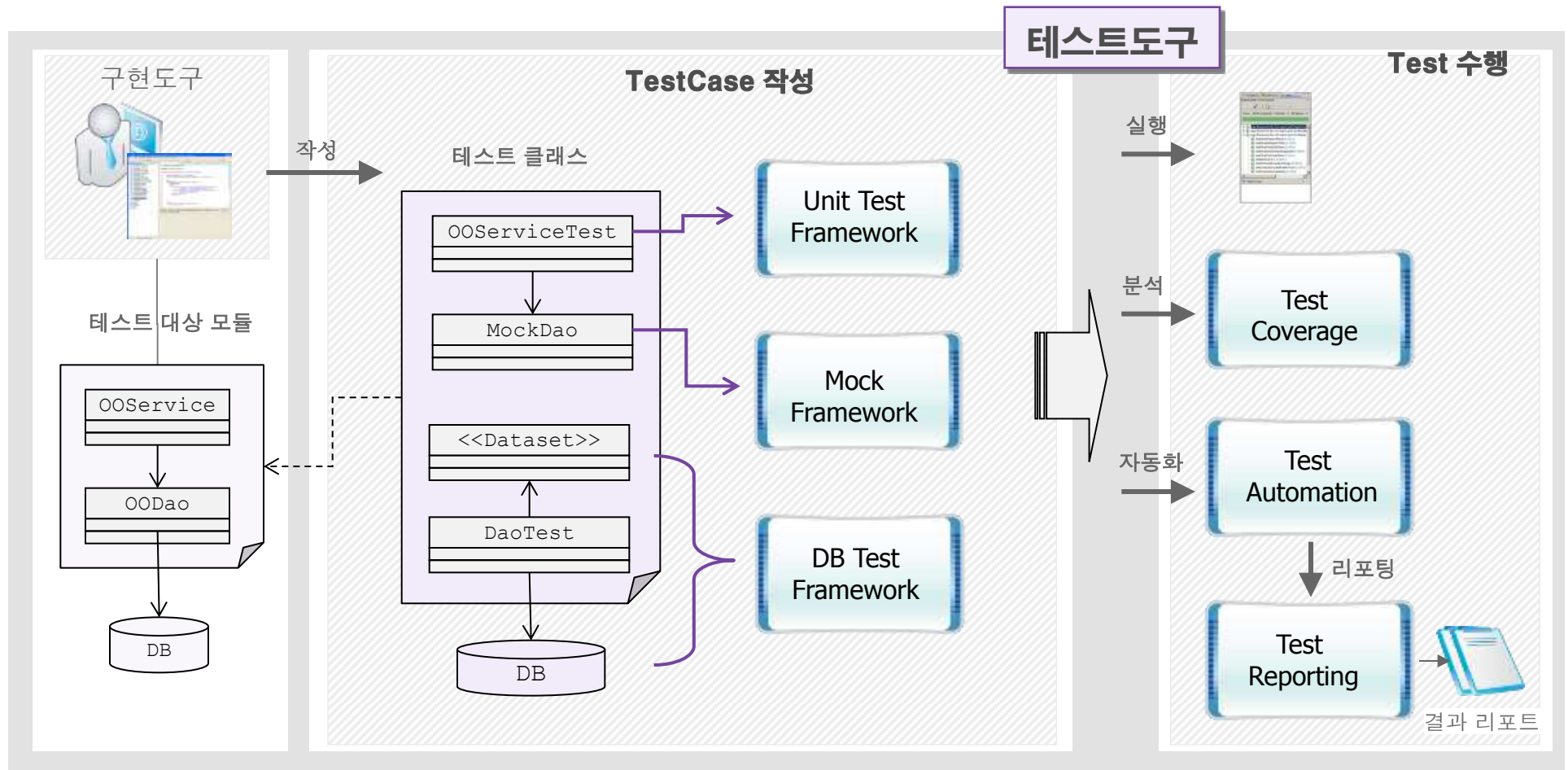


□ 수동 테스트 vs. 자동 테스트

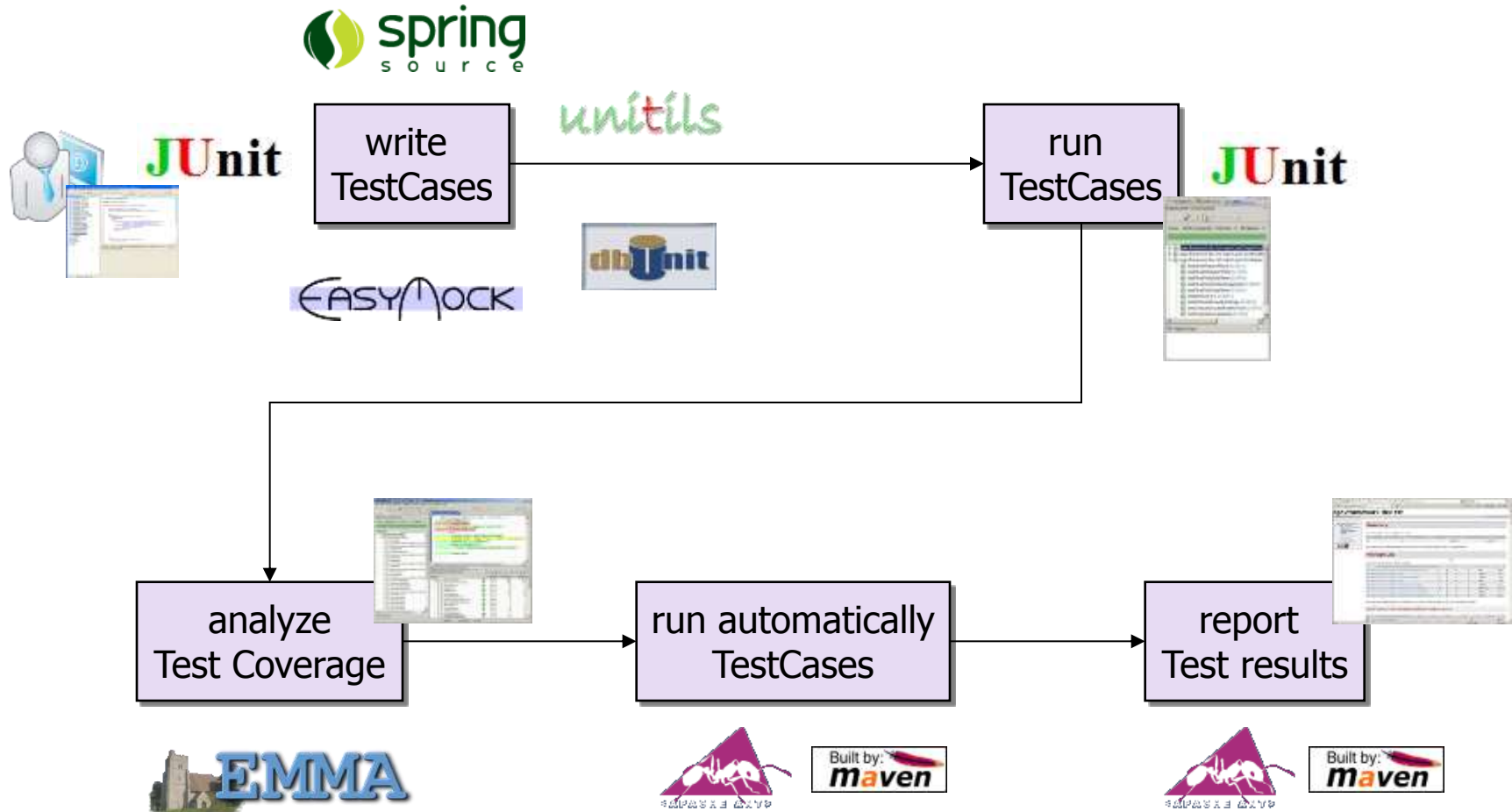


	수동 테스트	자동 테스트
장점	<div> <div>□ 쉽다. 간편하다.</div> <div>□ 테스트 불가능한 상황이 별로 없다.</div> </div>	<div> <div>□ 언제든지 같은 테스트를 여러 번 수행 가능</div> <div>□ 기존 테스트는 새 테스트를 작성하는 발판이 된다. (지식축적)</div> <div>□ 빠르므로 자주 돌려볼 수 있다.</div> <div>□ 개발자가 코드 개선 활동을 수행할 때에도 회귀테스트(Regression Test)를 자주 할 수 있다.</div> </div>
단점	<div> <div>□ 휘발성</div> <div>□ 테스트 항목이 늘어난다면?</div> <div>□ 다른 사람에게 테스트 내용을 설명하기 어려움</div> <div>□ 재현하기 어려움</div> </div>	<div> <div>□ 코드로 작성해야 한다. <div> -기술, 노하우가 필요하다</div> <div> -코드로 테스트를 작성할 수 없는 상황이 있다. (수동 테스트와 공조)</div> <div> -테스트도 관리 대상이다.</div> </div> </div> <div> <div>□ 모듈화가 잘 되어 있어야 한다. <div> -디자인의 개선 필요</div> <div> -Legacy Code의 어려움</div> </div> </div>

테스트 도구는 Unit Test, Mock, DB Test Framework을 통해 TestCase 작성을 지원하고, Test Automation, Test Coverage, Test Reporting 등의 기능을 제공함



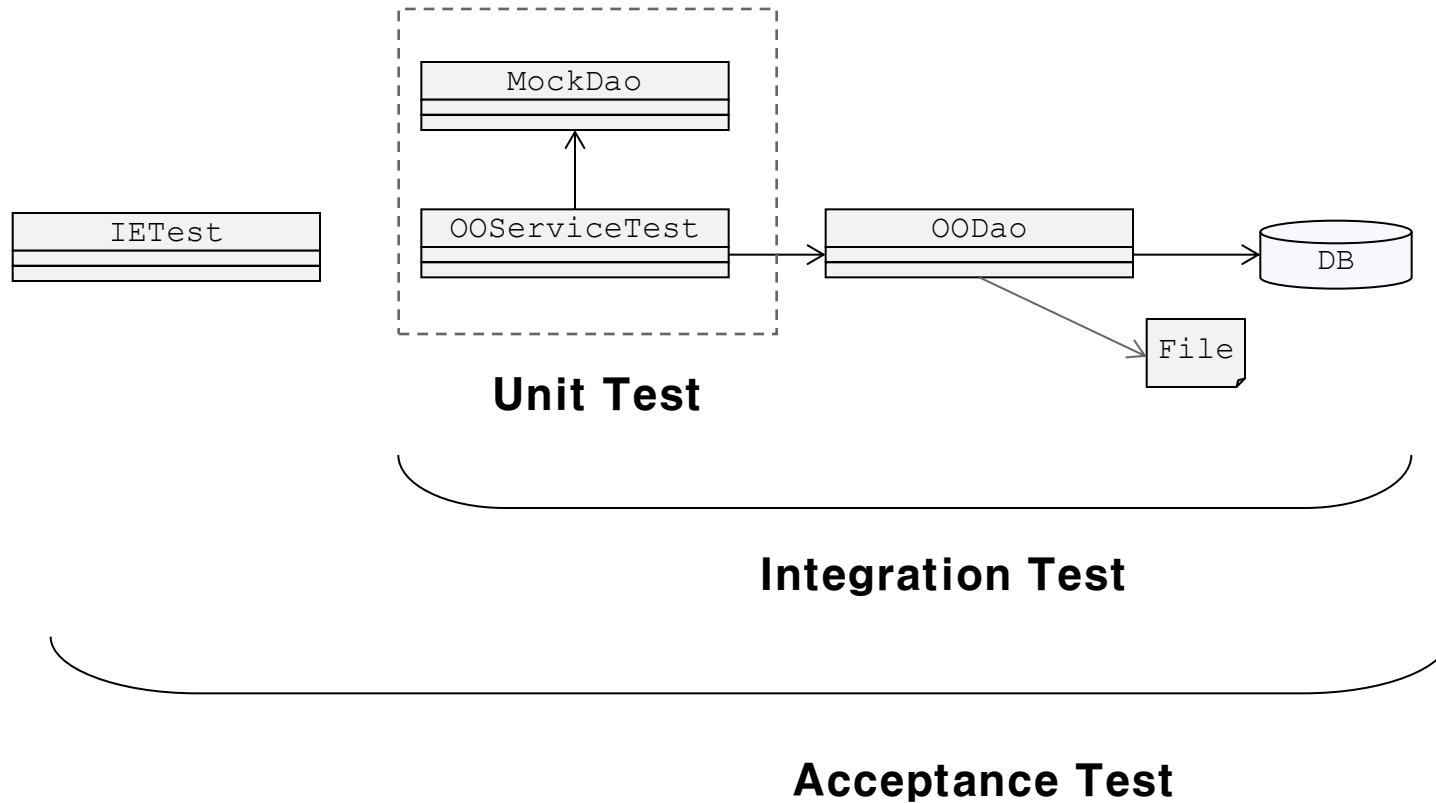
□ 테스트 도구 사용 프로세스



□테스트 도구에 채택한 오픈소스

구분	오픈소스	버전	비고
write & run TestCases	JUnit	4.4, 4.3	Test Framework
	EasyMock	2.4	Mock Framework
	Spring test	2.5.6	Spring Test Module
	DBUnit	2.4.2	DB Test Framework
	Unitils	2.2	JUnit, EasyMock, DbUnit, Spring Test, iBatis, Hibernate 등을 유연한 구조로 엮어서 테스트할 수 있는 기능 제공
Test Coverage	EMMA	2.0	Free Java code coverage tool
	EclEmma	1.3.2	EMMA Eclipse plugin
run automatically & report	Ant	1.6.5	Java-based build tool
	Maven	2.0	S/W project management & comprehension tool

테스트도구는 개발자가 코드로 작성하고 도구를 통해 자동화할 수 있는 테스트 중 단위테스트에 해당하며 이후 **Integration Test, Acceptance Test** 까지 범위를 확장할 수 있는 토대를 마련함



Unit Test란 대상 코드에 대해 테스트하고자 개발자가 작성한 코드로서, 주로 특정 메소드를 실행해서 그 결과가 기대값과 일치하는지 확인하는 형태이며, **Unit Test**는 서로 독립적으로 수행되어야 함.

□ 효과

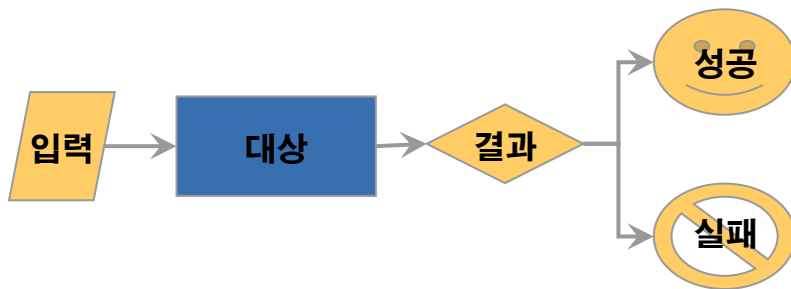
- 작성한 코드의 설계 개선 작업 시, 코드 품질에 대한 확신
- 코드 수정 시 버그를 쉽게 찾을 수 있게 해줌
- 자동화된 회귀 테스트 (Regression Test)를 가능하게 해주는 Source가 됨

□ 작성 범위

- 주요 흐름에 대한 테스트 (the happy path)
- 또 다른 주요 흐름에 대한 테스트 (the main alternative path)
- 경계 조건에 대한 테스트 (null 인자 체크 등)
- Exception 테스트 (Exception 발생하는 조건에 대한 테스트)

□ 구성

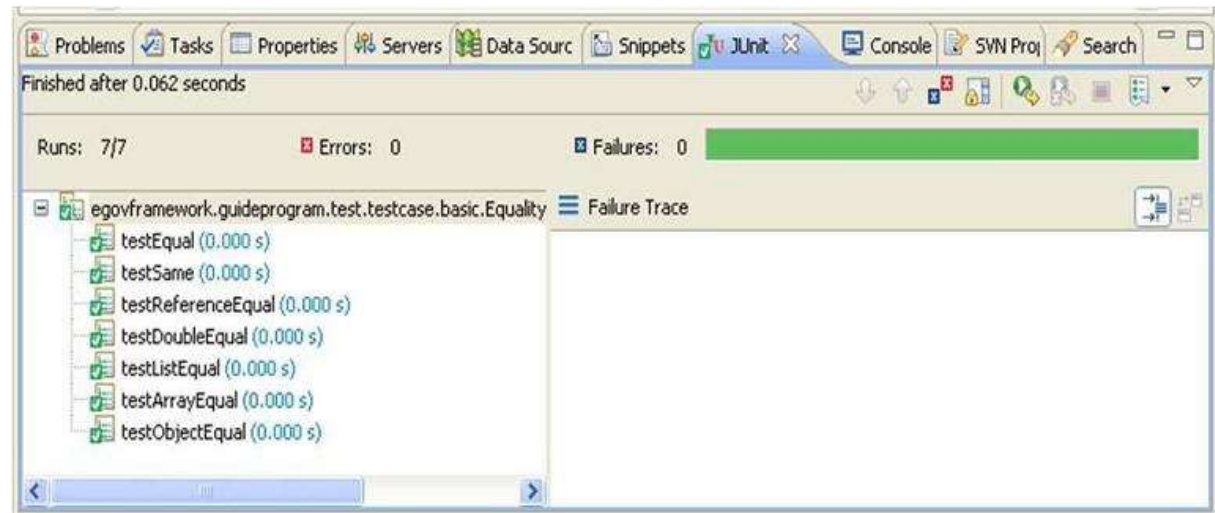
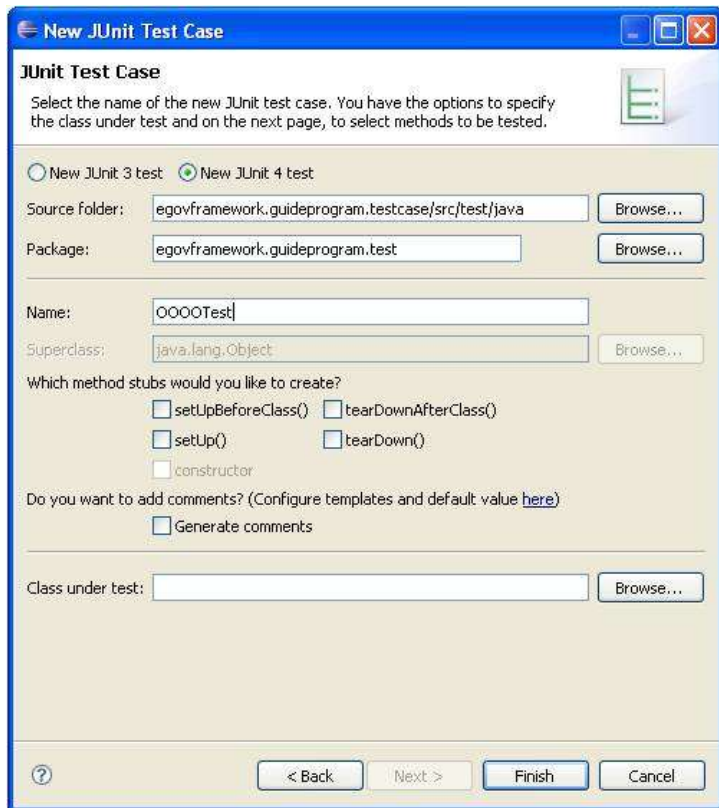
- 테스트 프레임워크를 사용하는 Class
- 공용으로 사용하는 테스트 데이터 (test fixture)
- 테스트 데이터 준비 (Setup of test data)
- 테스트 메소드 (testXXX)
 - (테스트 별 준비)
 - 테스트 대상 메소드 실행
 - assert 문을 이용한 결과 확인 (assertTrue, assertEquals etc.)
- (내부 메소드)



```

1 public class UserAdminTest {
2
3     /* Class under test */
4     private UserAdmin userAdmin;
5
6     /* A simple test user */
7     private User user; // Test Data (Fixture)
8     /* An administrator role */ //
9     private Role adminRole; //
10
11     /**
12      * Initializes the test fixture.
13      */
14     @Before
15     public void setUp() throws Exception {
16         userAdmin = new UserAdmin();
17         user = new User("John", "Doe"); // Test Data Setup
18         adminRole = new Role("Administrator"); //
19     }
20
21     /**
22      * Test for method with ...
23      */
24     @Test
25     public void testAddUser() {
26         user.setAge(18); // Extra Test Setup
27         userAdmin.addUser("jdoe", user, adminRole); // Use
28         Test Data
29
30         User result = userAdmin.getUser("jdoe");
31         assertEquals("John", result.getFirstName());
32         assertEquals("Doe", result.getLastName());
33     }
34 }
  
```

JUnit은 자바 프로그래밍 언어를 위한 Unit Test Framework로, Unit Test 코드를 작성하고 자동화된 테스트를 수행할 수 있는 기능을 제공함



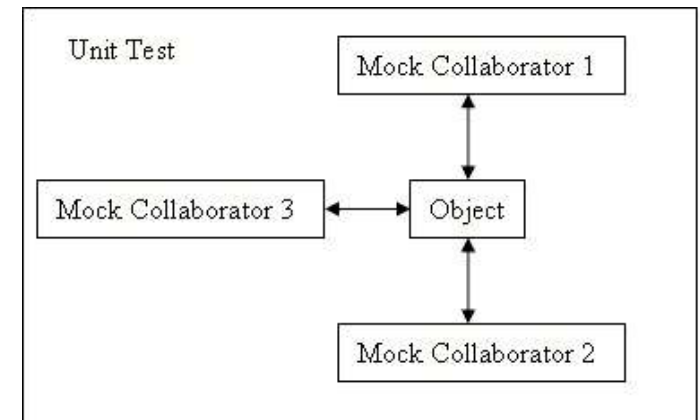
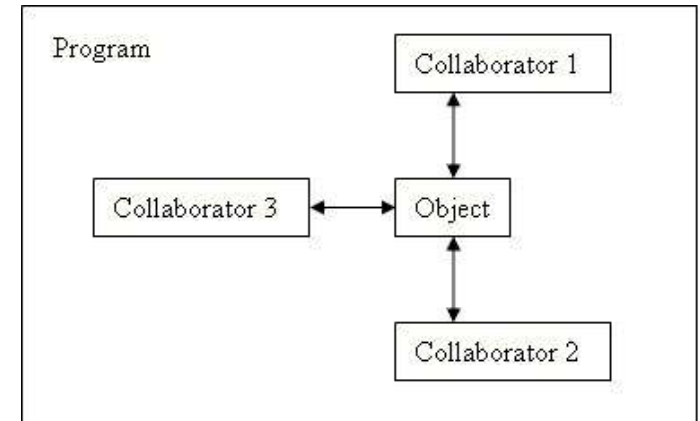
Mock 객체는 **Unit Test**의 독립성(isolation)을 높여주기 위해 사용하며, 테스트하고자 코드와 관련이 있는 객체(collaborator)를 흉내내어 **Unit Test**를 수행할 수 있도록 도와주는 객체임

□ Mock 객체를 사용하는 경우

- 진짜 객체를 준비, 설정하기 어렵다
- 진짜 객체가 느리다.
- 진짜 객체가 사용자 인터페이스를 갖거나 그 자체이다.
- 진짜 객체가 아직 없다.

□ Mock 사용 종류

- 이미 구현된 Mock 객체 사용 : Spring test (web, jndi), mockrunner
- EasyMock, JMock, Mockito 등의 Mock 라이브러리 사용
- Mock으로 사용할 가짜 클래스 직접 구현



참고 : <http://www.shinetech.com/thoughts/articles/30-unit-testing-with-mock-objects?start=1>

□ Spring Test web 사용 샘플

```

1  public class SessionMockTest {
2      /** mock object of the HttpServletRequest */
3      private MockHttpServletRequest request;
4
5      /** mock object of the HttpServletResponse */
6      private MockHttpServletResponse response;
7
8      /** mock object of the HttpSession */
9      private MockHttpSession session;
10
11     /** Target Class */
12     private Servlet servlet;
13
14     @Before
15     public void setUp() throws Exception {
16         servlet = new Servlet();
17         request = new MockHttpServletRequest();
18         session = new MockHttpSession();
19     }
20
21     @Test
22     public void testConfirmAdmin() throws Exception {
23         session.setAttribute("userid", "administrator");
24         session.setAttribute("password", "1234");
25
26         request.setSession(session);
27         servlet.confirmAdmin(request, response);
28
29         assertTrue(servlet.isAdmin());
30     }
31 }
32

```

Spring Test의
MockHttpServletRequest/Request
MockHttpSession 등 사용

테스트 대상 클래스

테스트 데이터 설정

실제 session 처럼 사용하여 테스트하고자 하는 메소드
확인

□ Easymock + Unitils 샘플

```

1  @RunWith(UnitilsJUnit4TestClassRunner.class)
2  public class EgovControllerTest {
3
4      @Mock
5      @InjectIntoByType
6      private EgovService mockService;
7
8      @TestedObject
9      private EgovController egovController = new EgovController();
10
11     @Test
12     public void testSelectList() throws Exception {
13         assertNotNull(egovController);
14         expect(mockService.selectList()).andReturn(
15             Arrays.asList(new Message(101), new Message(102)));
16         EasyMockUnitils.replay();
17
18         List<Message> resultList = egovController.selectList();
19         assertNotNull(resultList);
20         assertEquals(2, resultList.size());
21     }
22
23     @Test
24     public void testInsert() throws Exception {
25         mockService.insert(makeVO());
26         EasyMockUnitils.replay();
27
28         SessionStatus status = new SimpleSessionStatus();
29         String view = egovController.add(makeVO(), status);
30     }
31 }

```

Mock으로 생성할 인터페이스에 대한 정의한 뒤, 테스트 대상 클래스에 Mock으로 정의한 클래스를 세팅한다.

리턴값을 갖는 메소드를 Mock 클래스에 정의하고 리턴값을 설정한 뒤, 진짜 객체처럼 사용한다.

리턴하지 않는 메소드를 Mock 객체에 정의 후 진짜 객체처럼 사용한다.

배치 테스트 도구를 통해 기 개발된 일괄처리 모듈에 대한 Junit 테스트 파일생성 및 테스트 수행과 함께, 기존에 생성한 Junit 테스트 파일을 재실행할 수 있는 기능을 제공

The screenshot shows the 'eGovFrame Batch Job Test' dialog box. It has a title bar and a main area with the following components:

- Project Name:** file.sche.egov.temp
- TestJob List:** A list of test jobs with columns 'TestJob', 'delimitedToDelimited', 'fixedLengthToFixedLength', 'fixedLengthToBatisJob', and 'fixedLengthTojdbcJob'.
- Job Launcher Information:** A table with columns 'File Name' and 'File Location'. It contains two entries: 'context-batch-job-launcher.x...' and 'newJobLauncher.xml'.
- Job Parameter Information (선택사항):** A table with columns 'Name', 'Value', and 'Data Type'. It contains one entry: 'name' with value 'eGovFrame' and data type 'String'. There are 'Add' and 'Remove' buttons.
- Buttons:** 'Generate Batch Test File' and 'Test' buttons.
- Result:** A text area showing '<eGovFramework Batch Test>' and 'Batch Execution Status [OK]'.
- Footer:** 'Finish' and 'Cancel' buttons.

Numbered annotations (1-5) are placed over the dialog box:

- Job 정보 선택** (Job Information Selection): Points to the TestJob list.
- Job Launcher 정보 선택** (Job Launcher Information Selection): Points to the Job Launcher table.
- Job Parameter 정보 선택 (선택사항)** (Job Parameter Information Selection (Optional)): Points to the Job Parameter table.
- 배치 테스트 파일 생성** (Batch Test File Generation): Points to the 'Generate Batch Test File' button.
- 배치 테스트 수행** (Batch Test Execution): Points to the 'Test' button.

□ 배치 테스트 절차

- ① 배치 Job 정보를 선택한다.
- ② Job Launcher 정보를 선택한다.
- ③ 필요 시 Job Parameter 정보를 선택한다.
- ④ Generate Batch Test File 버튼을 클릭하여 배치 테스트 파일을 생성한다.
- ⑤ Test 버튼을 눌러 배치 테스트를 수행한다.

DB Test는 **DAO**와 **DB** 모두를 통틀어 **Persistence layer**를 테스트하는 것을 말하며, 테스트**DB**와 테스트 데이터를 준비하여 실제로 **DB**를 이용한 단위테스트를 수행함

□ DB Test Framework

- DbUnit

□ DbUnit 기능

- DB 데이터를 XML 파일 형태로 import/export
- DB 연결, DB 초기화
- DB의 데이터가 기대값과 같은지 확인
- 빌드 도구를 통한 테스트 자동화 기능 제공

□ 효율적인 DB Test를 위해

- DB 구조만 갖고 있는 단위테스트용 DB 준비
- 각 개발자마다 별도의 단위테스트용 DB 공간
- Unitils, Spring Test 활용

□ TestCase 작성

```

1  @RunWith(UnitilsJUnit4TestClassRunner.class)
2  @Transactional(TransactionMode.COMMIT)
3  @DataSet
4  @SpringApplicationContext( {
5      "/META-INF/persistence/connection/datasource-spring-with-unitils.xml",
6      "/META-INF/spring/context-common.xml",
7      "/META-INF/spring/context-sqlmap.xml" })
8  public class DaoOperationTest_noticeDao {
9      @TestDataSource
10     private DataSource dataSource;
11
12     /** Target Dao */
13     @SpringBean("noticeDao")
14     private NoticeDao noticeDao;
15
16     private NoticeVo noticeVo;
17
18     @Test
19     @ExpectedDataSet("/META-INF/**/AutoVerifyTestResultsTest_ExpectedDataSet.xml")
20     public void testInsert() {
21         assertNotNull(noticeVo);
22         noticeDao.insert(noticeVo);
23         int count = noticeDao.selectCount();
24         assertEquals(4, count);
25     }
26 }

```

3: 클래스 시작할 때 같은 위치에 있는 DaoOperationTest_noticeDao.xml 파일을 읽어 DB를 초기화

5: Datasource 연결 정보 (Unitils를 사용함)

9 : 5 라인에서 설정한 파일에 설정된 dataSource 객체를 설정함. Unitils의 Datasource 정보 사용

13 : Unitils의 Injection of Spring beans - noticeDao 라는 이름으로 정의된 Bean을 Spring Application Context로부터 가져옴

19 : 지정된 위치의 Dataset 파일을 읽어 들여, 메소드 처리 후반에서 결과를 비교함. assert 기능

❑ DataSet (AutoInsertionTestDataTest_DataSet.xml)

```
<?xml version="1.0" encoding="UTF-8"?>

<dataset>
  <NOTICE NOTICE_ID="101"
    NOTICE_TITLE="101번 공지"
    NOTICE_FILE_CNT="0"
    NOTICE_RETRIEVED_CNT="0"
  />
  <NOTICE NOTICE_ID="102"
    NOTICE_TITLE="102번 공지"
    NOTICE_FILE_CNT="0"
    NOTICE_RETRIEVED_CNT="0"
  />
  <NOTICE NOTICE_ID="103"
    NOTICE_TITLE="103번 공지"
    NOTICE_FILE_CNT="0"
    NOTICE_RETRIEVED_CNT="0"
  />
</dataset>
```

❑ DataSet 구조

```
<dataset>
  <TABLE_NAME COLUMN_NAME1="VALUE"
    COLUMN_NAME2="VALUE"
  />
</dataset>
```

❑ Expected DataSet (AutoVerifyTestResultsTest_ExpectedDataSet.xml)

```
<?xml version="1.0" encoding="UTF-8"?>

<dataset>
  <NOTICE NOTICE_ID="201"
    NOTICE_TITLE="201번 공지"
    NOTICE_CONTENTS="테스트용으로 자동 입력된 공지사항 201번입니다."
    NOTICE_LAST_MODIFIER="OracleDataSetTest.class"
  />
</dataset>
```

작성된 다수의 **TestCase**를 자동으로 수행하여 소스 단위의 품질을 높이하고자 함. **CI** 서버와 연계될 때, 주기적인 테스트 수행으로 인해 목표 시스템의 품질을 향상할 수 있음

□ Test Suite

- TestCase 모음

1. Test Suite Class 작성

```
@RunWith(Suite.class)
@SuiteClasses( { HttpRequestMockTest.class, SessionMockTest.class,
FileUploadMockTest.class, EmailMockTest.class, JDBCMockTest.class })
public class MockTestSuite {

}
```

□ Test Suite 설정 방법

- Test Suite Class 작성
- 빌드 도구의 batchtest 묶음

2. batchtest로 묶음

```
<junit . . .>
  <batchtest fork="yes" todir="${testreports.dir}">
    <fileset dir="${testbuild.dir}">
      <include name="**/*Test.class" />
      <exclude name="**/Abstract*Test.class" />
    </fileset>
  </batchtest>
</junit>
```

❑ Ant (build.xml)

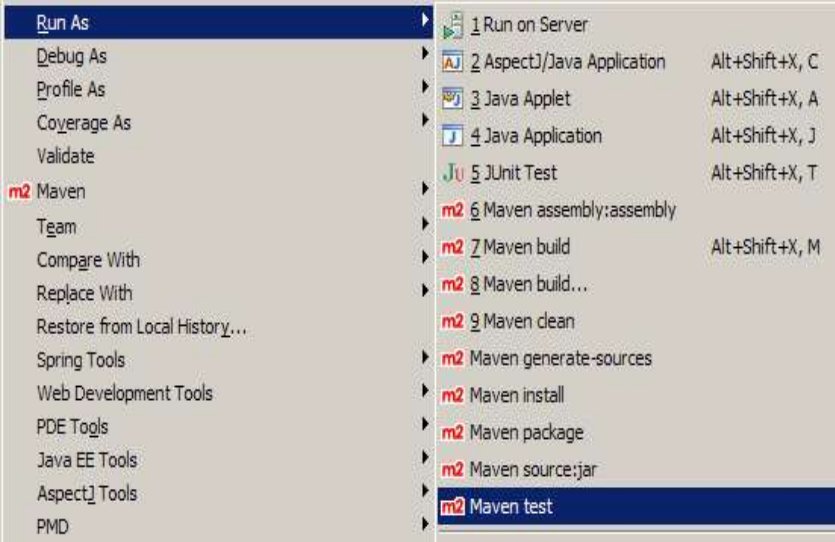
```
<junit forkmode="perBatch" printsummary="true" haltonfailure="yes" haltonerror="yes">
  <classpath refid="master-classpath" />
  <classpath refid="test-classpath" />
  <classpath path="${testbuild.dir}" />

  <formatter type="xml" />

  <batchtest fork="yes" todir="${testreports.dir}">
    <fileset dir="${testbuild.dir}">
      <include name="**/*Test.class" />
      <exclude name="**/Abstract*Test.class" />
    </fileset>
  </batchtest>
</junit>
```

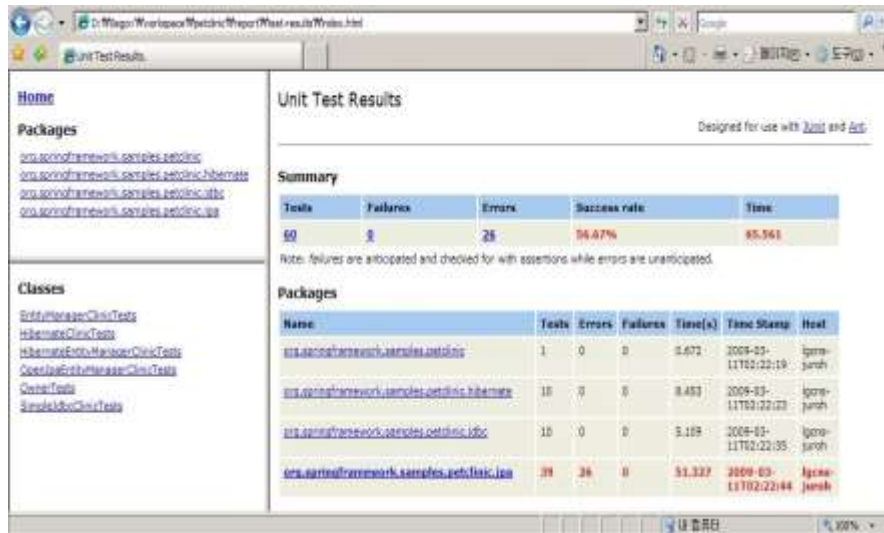
❑ Maven (pom.xml) goal : test

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <configuration>
        <reportFormat>xml</reportFormat>
        <excludes>
          <exclude>**/Abstract*.java</exclude>
        </excludes>
        <includes>
          <include>**/*Test.java</include>
        </includes>
      </configuration>
    </plugin>
  </plugins>
</build>
```



❑Ant (build.xml)

```
<!-- Junit Test Result Report -->
<target name="junitreport" depends="tests">
  <junitreport todir="${testhtml.dir}">
    <fileset dir="${testreports.dir}">
      <include name="TEST-*.xml" />
    </fileset>
    <report format="frames" todir="${testhtml.dir}" />
  </junitreport>
</target>
```



Unit Test Results

Summary

Tests	Failures	Errors	Success rate	Time
60	0	28	94.67%	85.561

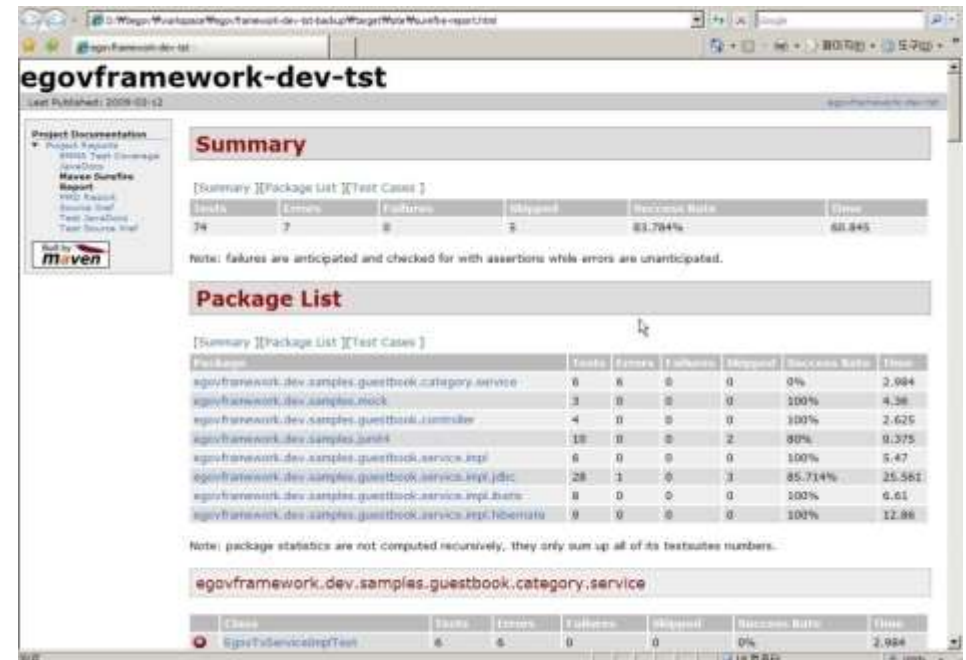
Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Packages

Name	Tests	Errors	Failures	Time(s)	Time Stamp	Host
org.springframework.samples.petclinic	1	0	0	0.672	2009-03-11T02:22:19	lgae-jungh
org.springframework.samples.petclinic.hibermate	10	0	0	0.453	2009-03-11T02:22:33	lgae-jungh
org.springframework.samples.petclinic.jdbc	10	0	0	0.189	2009-03-11T02:22:35	lgae-jungh
org.springframework.samples.petclinic.jpa	39	28	0	51.327	2009-03-11T02:22:44	lgae-jungh

❑Maven (pom.xml)

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-report-plugin</artifactId>
      <version>2.4.2</version>
    </plugin>
  </plugins>
</reporting>
```



egovframework-dev-tst

Summary

Tests	Errors	Failures	Skipped	Success Rate	Time
74	7	0	3	83.794%	60.845

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
egovframework.dev.samples.guestbook.category.service	6	6	0	0	0%	2.984
egovframework.dev.samples.mock	3	0	0	0	100%	4.38
egovframework.dev.samples.guestbook.controller	4	0	0	0	100%	2.625
egovframework.dev.samples.junit	18	0	0	2	80%	0.375
egovframework.dev.samples.guestbook.service.impl	6	0	0	0	100%	5.47
egovframework.dev.samples.guestbook.service.impl.jdbc	28	1	0	3	85.714%	25.561
egovframework.dev.samples.guestbook.service.impl.hbm	8	0	0	0	100%	6.61
egovframework.dev.samples.guestbook.service.impl.hibernate	0	0	0	0	100%	12.86

Note: package statistics are not computed recursively, they only sum up all of its testcases numbers.

egovframework.dev.samples.guestbook.category.service

Class	Tests	Errors	Failures	Skipped	Success Rate	Time
GuestServiceImpTest	6	6	0	0	0%	2.984

□Ant – Default로 사용한 예

```
<path id="egov.lib">
  <path refid="master-classpath" />
  <path location="${antlib.dir}/egovtest/egovframework-dev-tst-ant.jar" />
</path>

<taskdef resource="egovtest.properties" classpathref="egov.lib" />

<!-- JUnit Excel Report -->
<target name="egovtest" depends="tests">
  <egov-junitreport todir="${testxls.dir}">
    <fileset dir="${testreports.dir}" includes="**/TEST-*.xml" />
  </egov-junitreport>
</target>
```

Excel Reporting Ant Task 설정

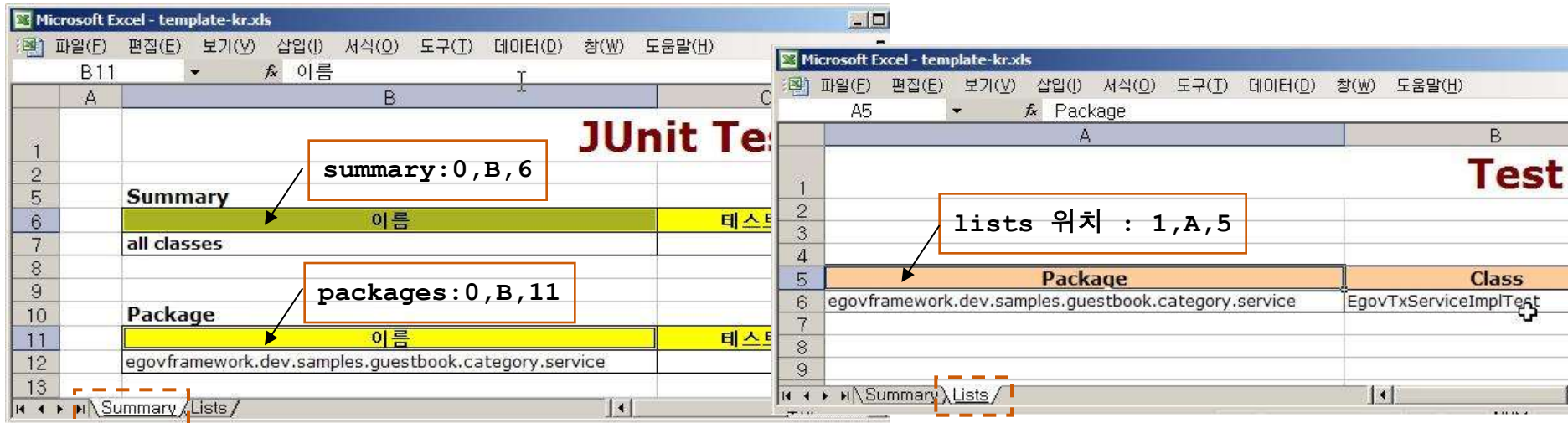
Default 사용예

□Ant – 템플릿 엑셀 파일을 별도로 사용한 예

```
<target name="egovtest-full" depends="tests">
  <egov-junitreport todir="${testxls.dir}"
    outputname="egovtest-junit-full.xls"
    templatepath="${basedir}/build/template-kr.xls"
    summary="0,B,6"
    packages="0,B,11"
    lists="1,A,5">
    <fileset dir="${testreports.dir}"
      includes="**/TEST-*.xml" />
  </egov-junitreport>
</target>
```

□Ant – 템플릿 엑셀 파일을 별도로 사용한 예

property	설명
todir	엑셀 파일 생성 디렉토리
outputname	엑셀 파일명
templatepath	템플릿 엑셀 파일 정보
summary, packages, lists	각 엑셀 테이블의 헤더 위치 정보 (sheet, column, row)
fileset	테스트 결과 XML 파일 위치

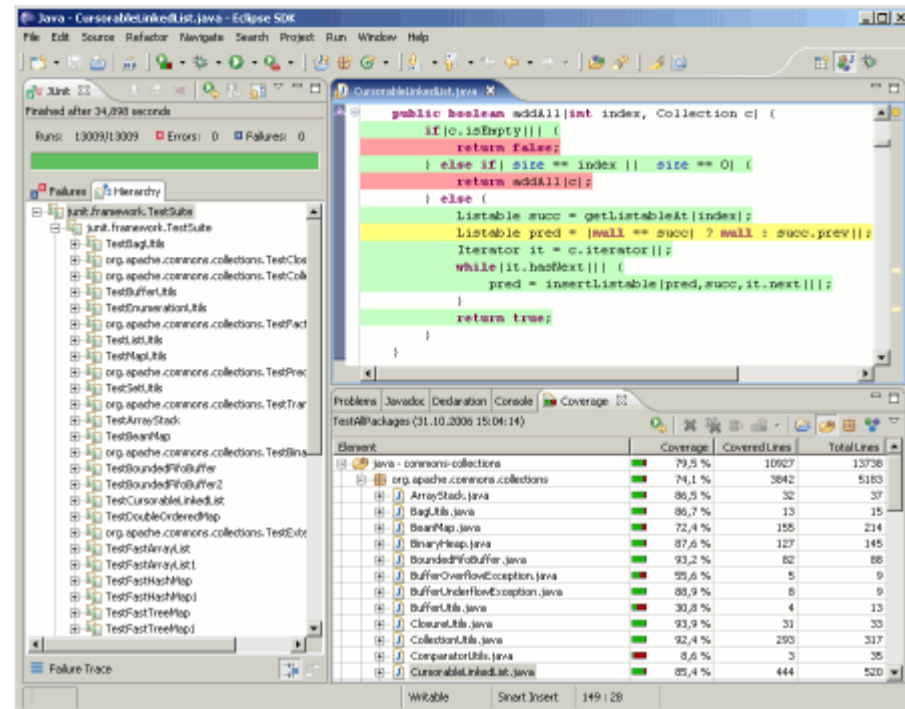


❑ Maven (goal : egovtest:junit-xls)

```
<build>
  <plugins>
    <!-- Egovframework JUnit Excel Reporting -->
    <plugin>
      <groupId>egovframework.dev</groupId>
      <artifactId>egovtest-maven-plugin</artifactId>
      <version>1.0.0-SNAPSHOT</version>
    </plugin>
  </plugins>
</build>
```

JUnit Test Results Summary						
2009-03-27						
Summary						
name	Tests	Errors	Failures	Skipped	Success Rate	Time (s)
all classes	44	6	0	4	77.27%	36.172
Package						
name	Tests	Errors	Failures	Skipped	Success Rate	Time (s)
egovframework.dev.samples.guestbook.category.service	6	6	0	0	0.00%	2.984
egovframework.dev.samples.junit4	6	0	0	2	66.67%	0.156
egovframework.dev.samples.guestbook.service.impl	5	0	0	0	100.00%	3.313
egovframework.dev.samples.guestbook.service.impl.jdbc	18	0	0	2	88.89%	16.859
egovframework.dev.samples.guestbook.service.impl.hibernate	9	0	0	0	100.00%	12.860

테스트 코드가 대상 소스 코드에 대해 테스트하는 코드를 작성했는지 그 커버하는 정도를 수치나 코드 라인을 통해 알려줌



EclEmma 화면

EMMA HTML 리포팅 화면

EMMA Coverage Report (generated Thu Mar 12 11:20:39 KST 2009)				
COVERAGE SUMMARY				
	class, %	method, %	block, %	line, %
	89% (16/18)	87% (74/85)	59% (479/814)	59% (133/225)
SUMMARY				
Source files:	10			
Bytecode files:	16			
Classes:	18			
Methods:	85			
Block lines:	225			
BREAKDOWN BY PACKAGE				
name	class, %	method, %	block, %	line, %
dev.samples.guestbook.utils	0% (0/1)	0% (0/2)	0% (0/24)	0% (0/6)
dev.samples.guestbook.category.controller	100% (1/1)	50% (1/2)	33% (3/9)	33% (1/3)
dev.samples.guestbook.service.impl.jdbc	80% (4/5)	71% (15/21)	40% (170/427)	31% (33/108)
dev.samples.guestbook.category.service	100% (1/1)	92% (11/12)	49% (38/77)	70% (16/23)
egovframework.dev.samples.guestbook.controller	100% (2/2)	86% (6/7)	76% (29/38)	83% (10/12)
egovframework.dev.samples.guestbook.category.service.impl	100% (2/2)	100% (12/12)	100% (67/67)	100% (18/18)
egovframework.dev.samples.guestbook.service	100% (2/2)	100% (12/12)	100% (57/57)	100% (25/25)
egovframework.dev.samples.guestbook.service.impl	100% (2/2)	100% (7/7)	100% (38/38)	100% (11/11)
egovframework.dev.samples.guestbook.service.impl.hibernate	100% (1/1)	100% (5/5)	100% (43/43)	100% (11/11)
egovframework.dev.samples.guestbook.service.impl.ibatis	100% (1/1)	100% (5/5)	100% (34/34)	100% (8/8)
[all classes]				
EMMA 2.0.5312 (C) Vladimir Roubtsov				

❑ Unitils Guildelines

- <http://unitils.sourceforge.net/guidelines.html>

❑ Mock Object를 사용해서 쉽게 테스트하기

- <http://www.ibm.com/developerworks/kr/event/screencast/final/01/>

❑ mock object

- <http://www.mockobjects.com/>

❑ Effective Unit Testing with DbUnit

- <http://www.onjava.com/pub/a/onjava/2004/01/21/dbunit.html>

❑ What is a mock objects

- <http://www.shinetech.com/display/www/What+Are+Mock+Objects%3F>

❑ An early look at JUnit4

- <http://www.ibm.com/developerworks/java/library/j-junit4.html>

❑ JUnit FAQ

- <http://junit.sourceforge.net/doc/faq/faq.htm>