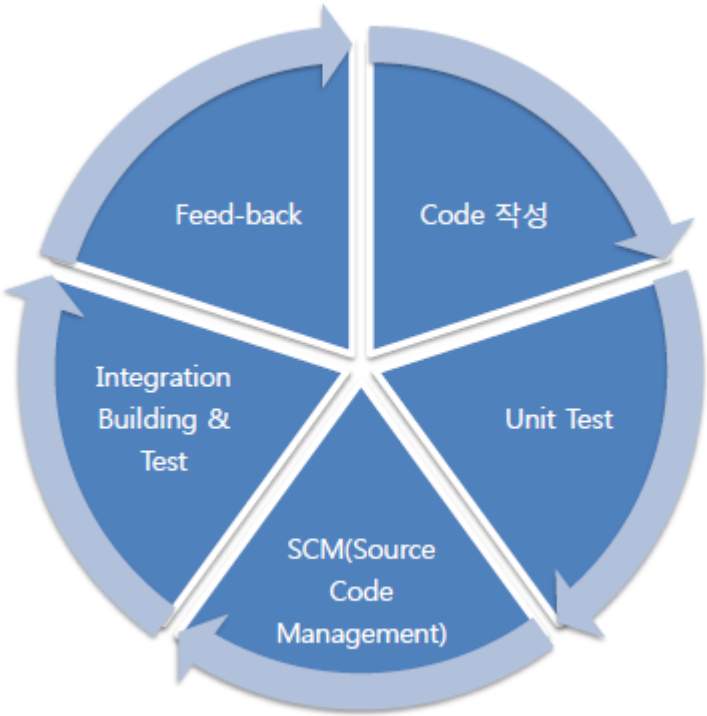
CTIP 환경 구축 & Unit Test “NuSRS”

200511305 김성규
200511306 김성훈
200614164 김효석
200611124 유성배
200518036 곡진화

CTIP

- CTIP
  - Continuous Test & Integration Platform
    - CI 서버를 통한 지속적 통합 및 빌드
    - 품질 도구를 통한 코드 품질 검토 (테스트 및 정적 분석)
    - 빌드 결과의 배포 및 관련자에게 통보
  - 구축 환경
    - 코드 작성 : Eclipse
    - SCM : Subclipse
    - Unit Test : JUnit
    - Build 자동화 : Ant
    - CI 도구 : Hudson



Ant

- ANT
  - Java 기반의 빌드 프로세스 자동화 도구
  - Eclipse는 Ant 플러그인 기본 내장
  - 플랫폼 독립적

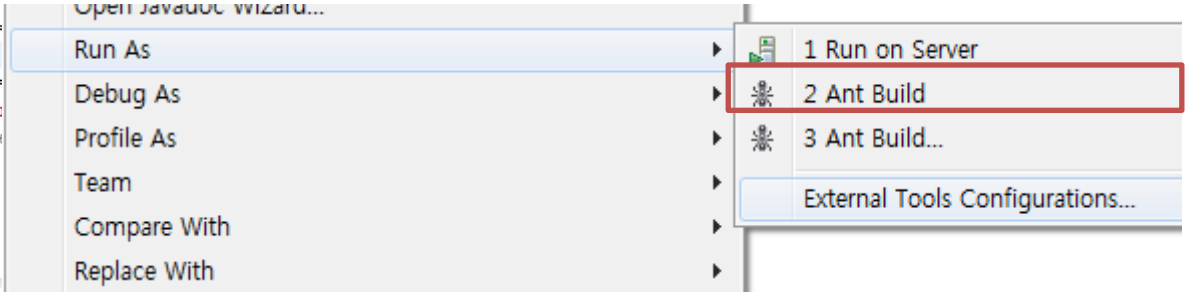
- Ant 요소
  - project : 루트 태그, 프로젝트 설명
  - target : 수행할 태스크를 지정
  - property : 사용하게 될 프로퍼티 지정
  - 수행 작업 : 태스크
    - mkdir : 디렉토리 생성
    - Copy, Delete : 파일/ 디렉토리 복사, 삭제
    - Javac : 자바 파일 컴파일
    - jar : jar 파일 생성
    - Javadoc : javadoc을 생성

Ant

- Ant 작성 & 수행
  - File>New>File
  - build.xml 파일 생성 후 스크립트 작성
  - xml 파일 선택 후 Run As>Ant Build로 수행

build.xml

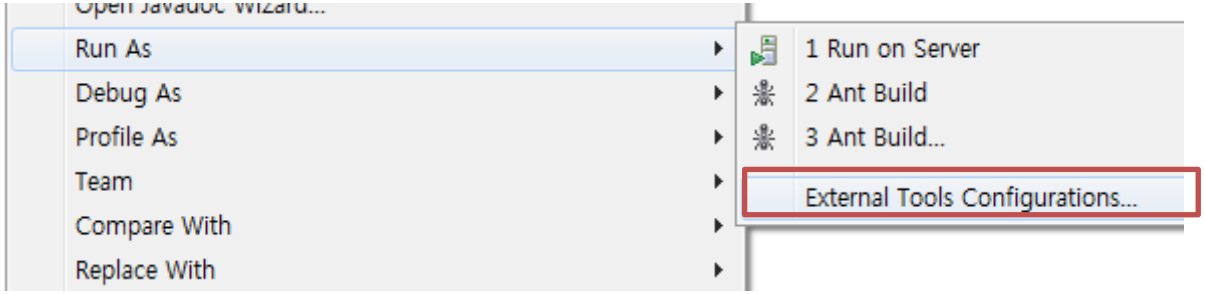
```
<?xml version="1.0" encoding="UTF-8"?>

<project name="NuSRS_2.1" default="junit-test" basedir=".">
    <target name ="test" depends = "init, junit-test, junit-test-report"/>

    <target name="init">
        <delete dir="${basedir}/TestReport" quiet="true"/>
        <mkdir dir="${basedir}/TestReport"/>
    </target>

    <target name="junit-test" description="Runs JUnit Tests">
        <junit printsummary="yes">
            <classpath>
                <pathelement location="${basedir}/bin"/>
                <pathelement location="junit-4.8.1.jar"/>
            </classpath>
            <formatter type=
            <batchtest fork=
                <fileset di:
                    <includ
                </fileset>
            </batchtest>
        </junit>
    </target>
```

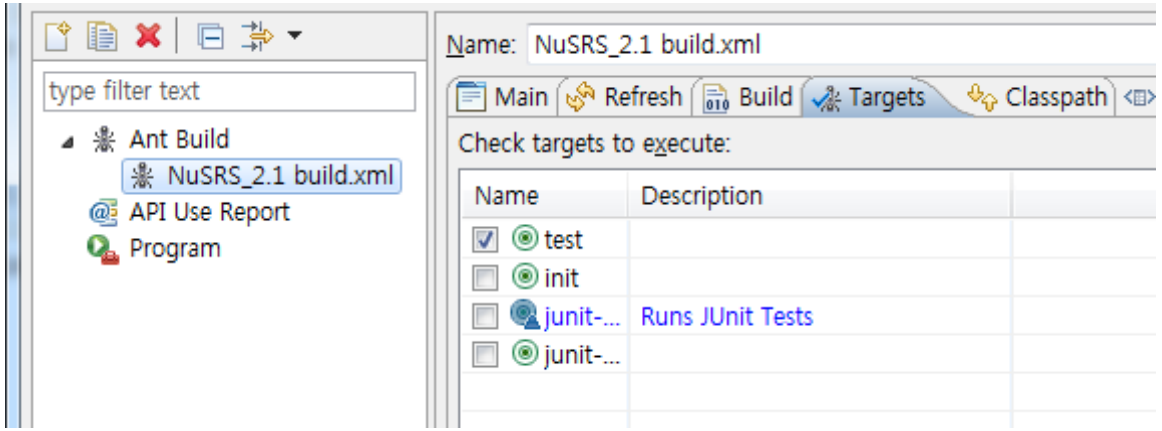


- 3개의 태스크 등록
  - 폴더 초기화(삭제-생성)
  - JUnit Test 수행(*Test.java)
  - JUnit Test Report 생성 (xml 형태, html 문서)

Ant

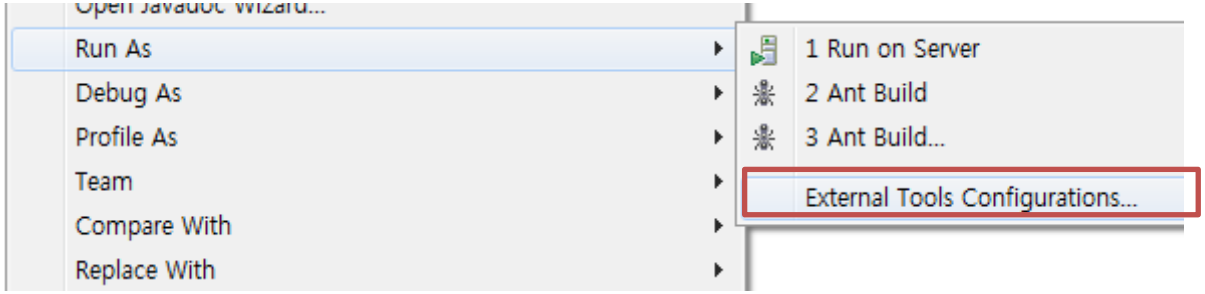
- Ant 설정
  - 실행시킬 target 선택 가능

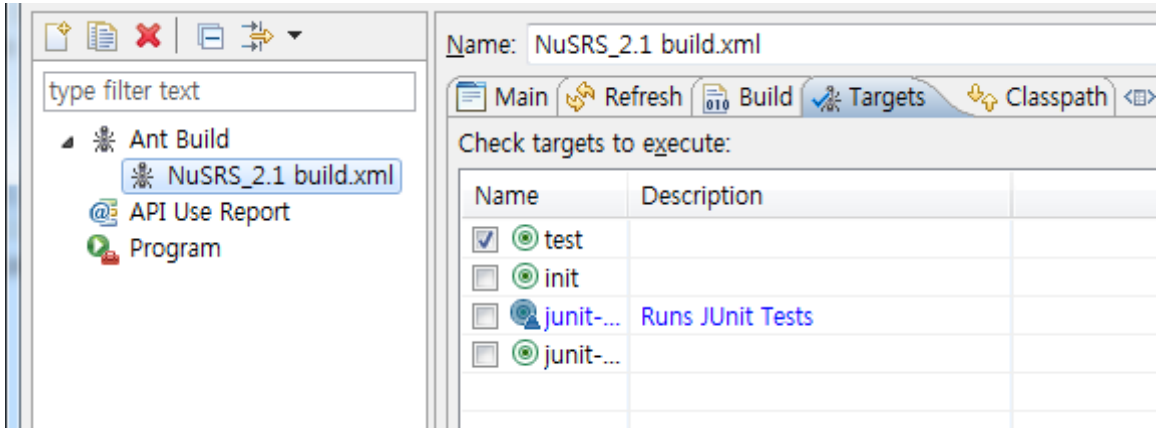




Ant

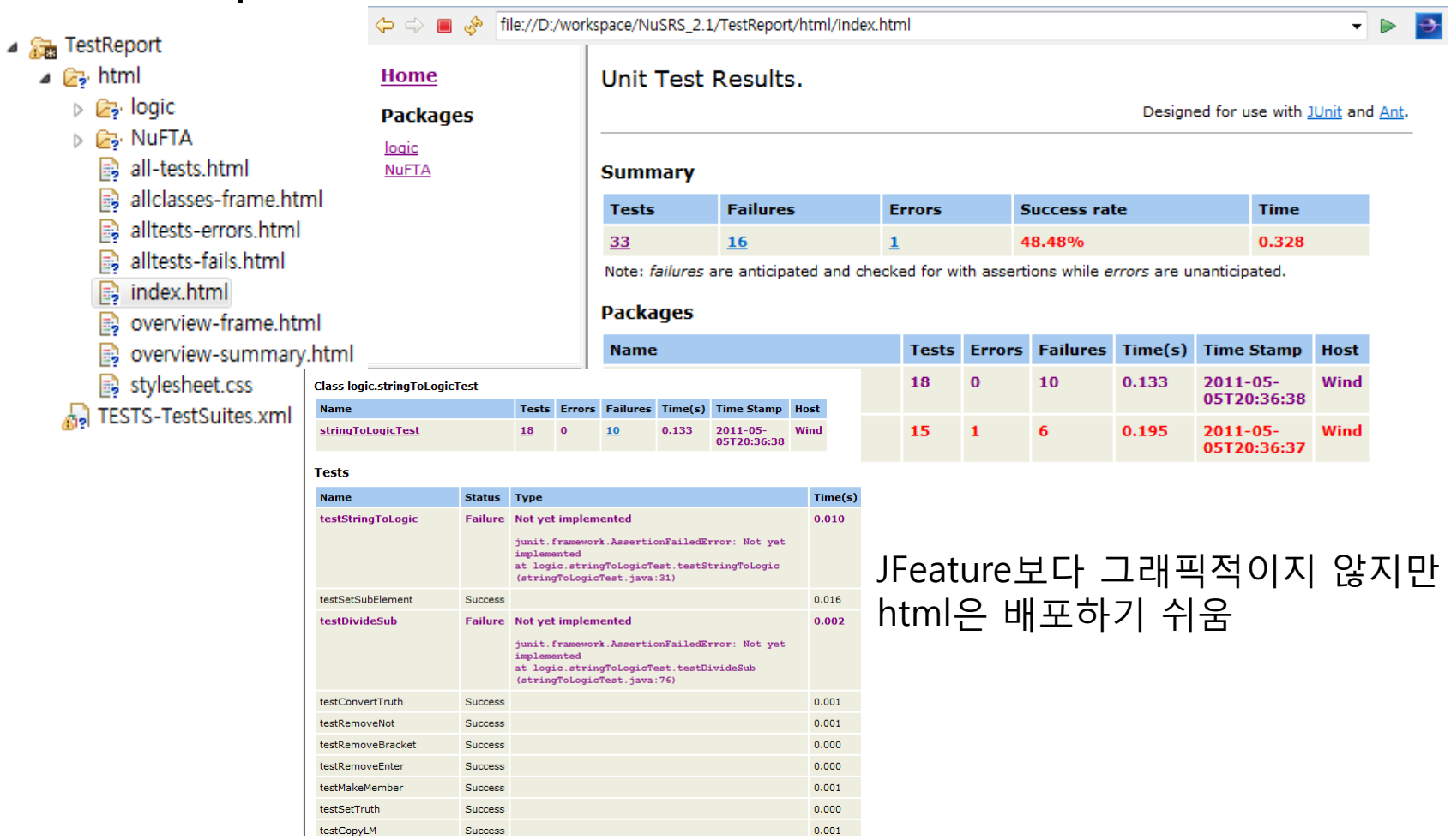
- Ant 설정
  - 실행시킬 target 선택 가능





Ant

- JUnit Report - Html 생성



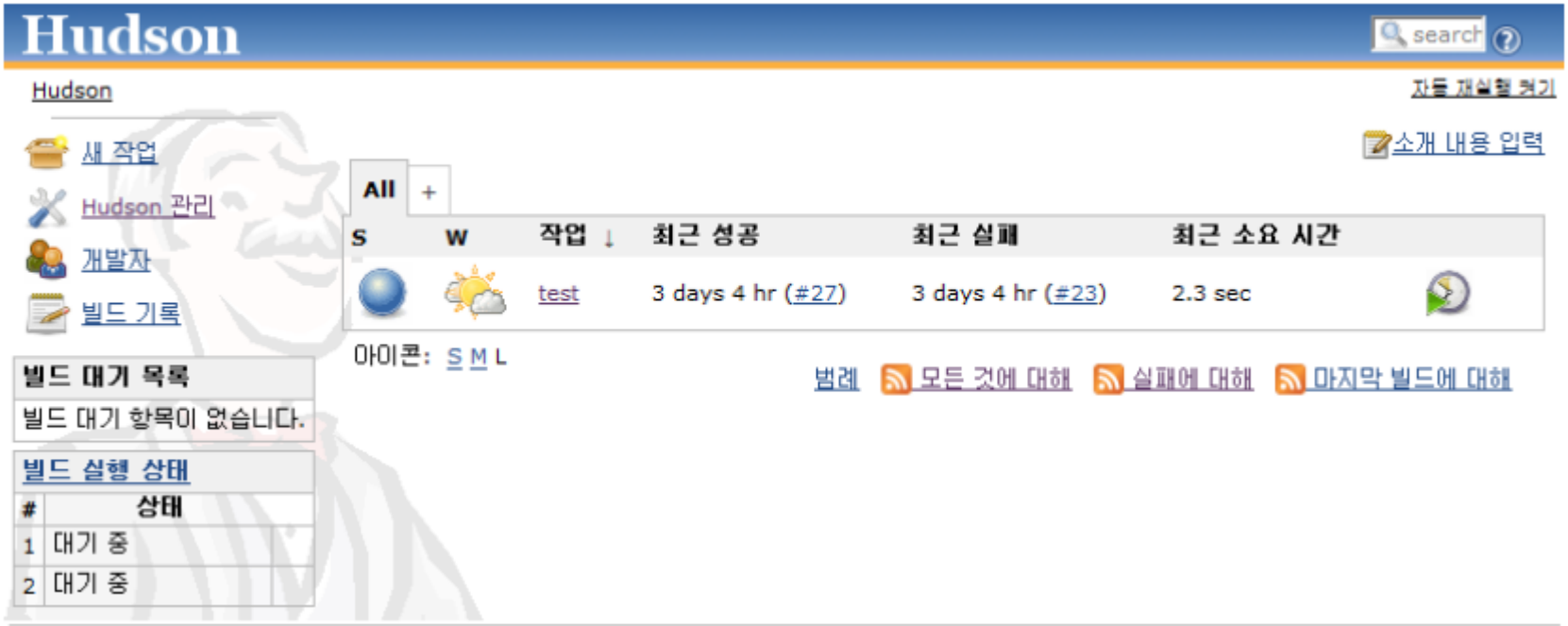
JFeature보다 그래픽적이지 않지만 html은 배포하기 쉬움

Hudson

- Hudson
  - Continuous Integration을 위한 Java build 지원 도구
  - 특징
    - 쉬운 설치
    - 웹 기반 UI로 Client에서 별도의 소프트웨어 설치 불필요
    - 동시 여러 프로젝트 관리 가능
  - 기능
    - Source 관리 프로그램을 이용하여 일관성 유지
    - 자동 build (commit / 시간 간격)
    - 자동 테스팅
    - 일일 Check out과 build

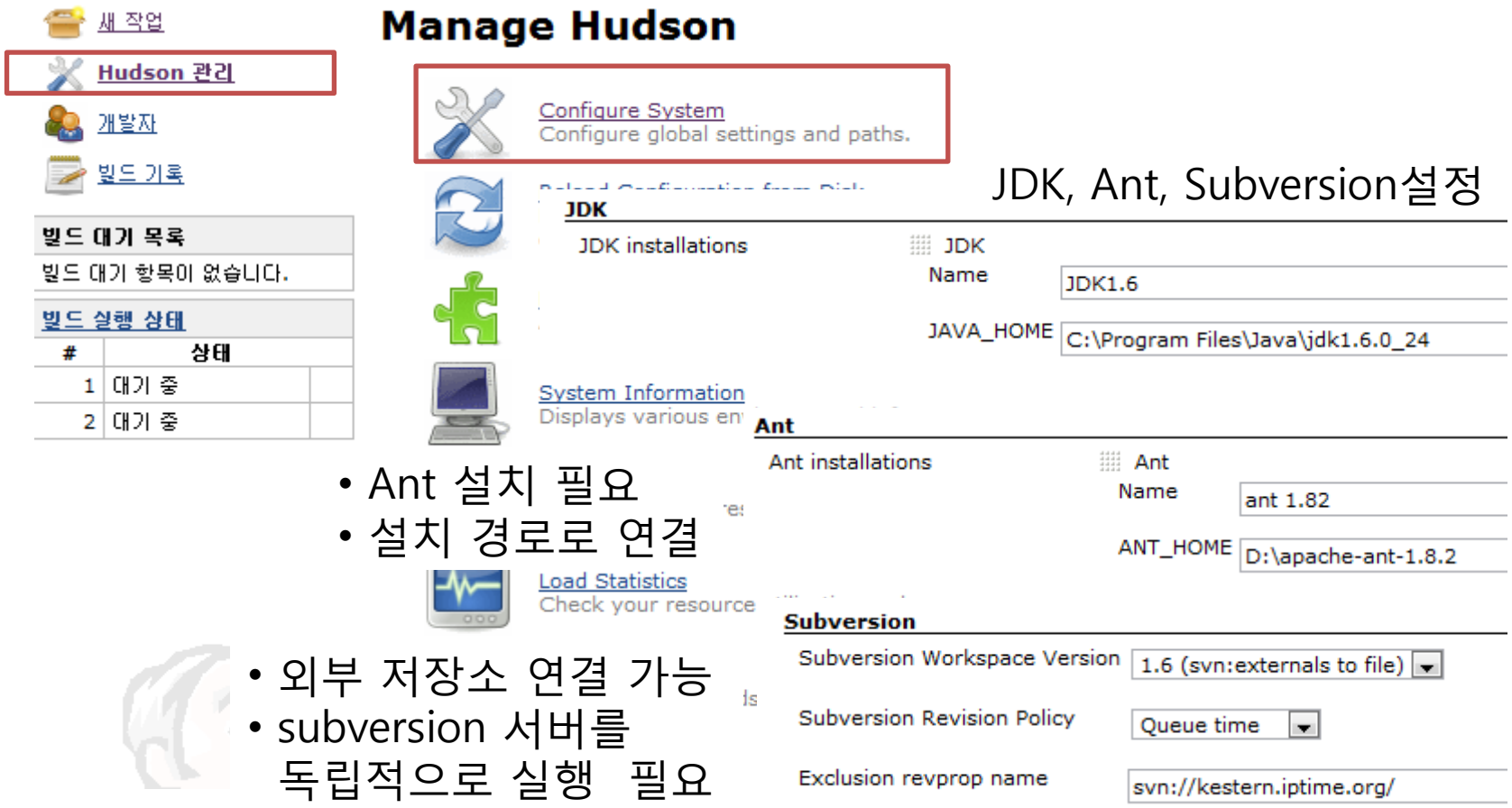
Hudson

- 설치 & 실행
  - hudson-ci.org 에서 hudson.war 파일 download
  - java -jar hudson.war 명령어로 실행
  - 8080 port번호로 접속 가능



Hudson

❖ Hudson 설정



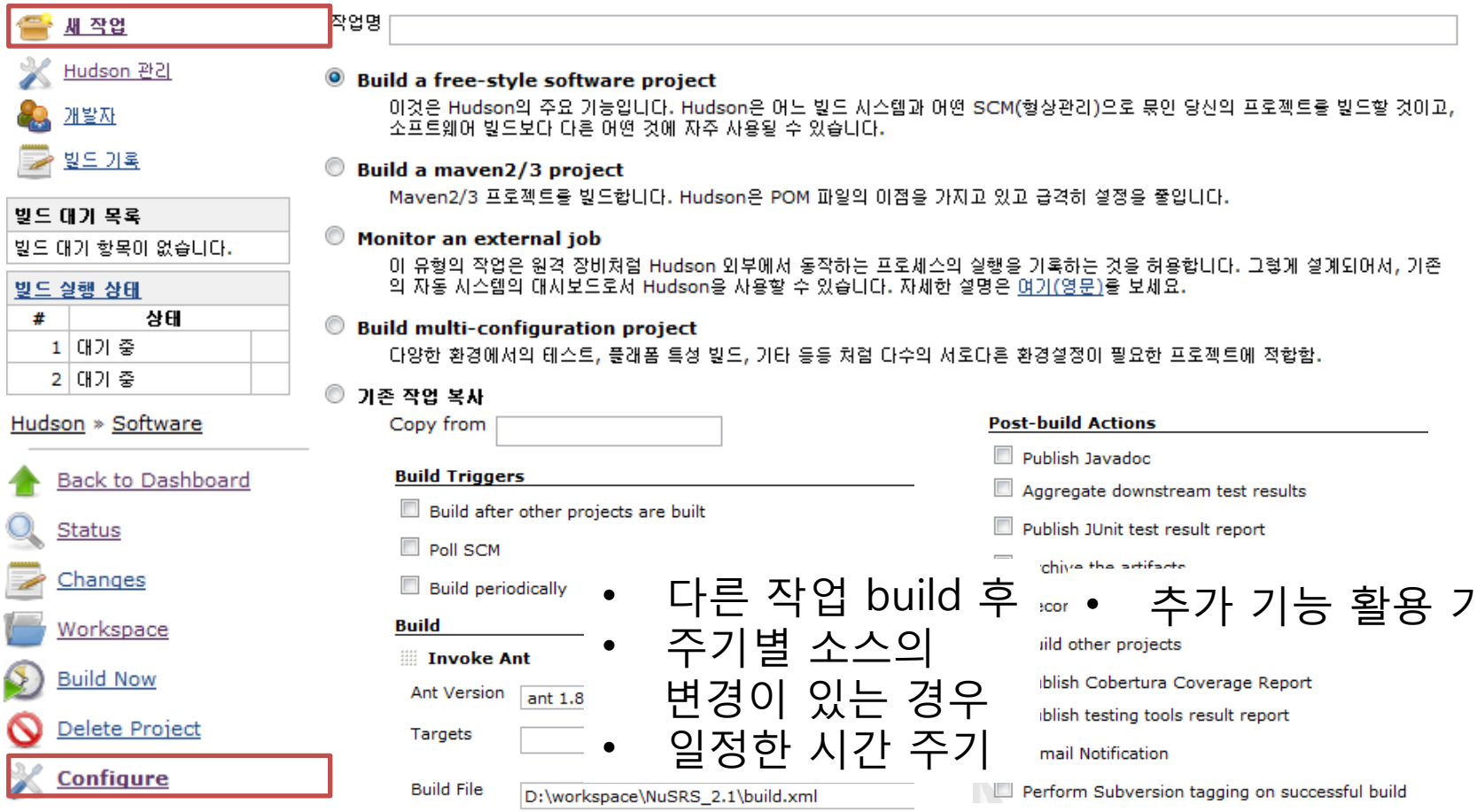
JDK, Ant, Subversion설정

- Ant 설치 필요
- 설치 경로로 연결

- 외부 저장소 연결 가능
- subversion 서버를 독립적으로 실행 필요

Hudson

❖ 작업 생성 & 설정



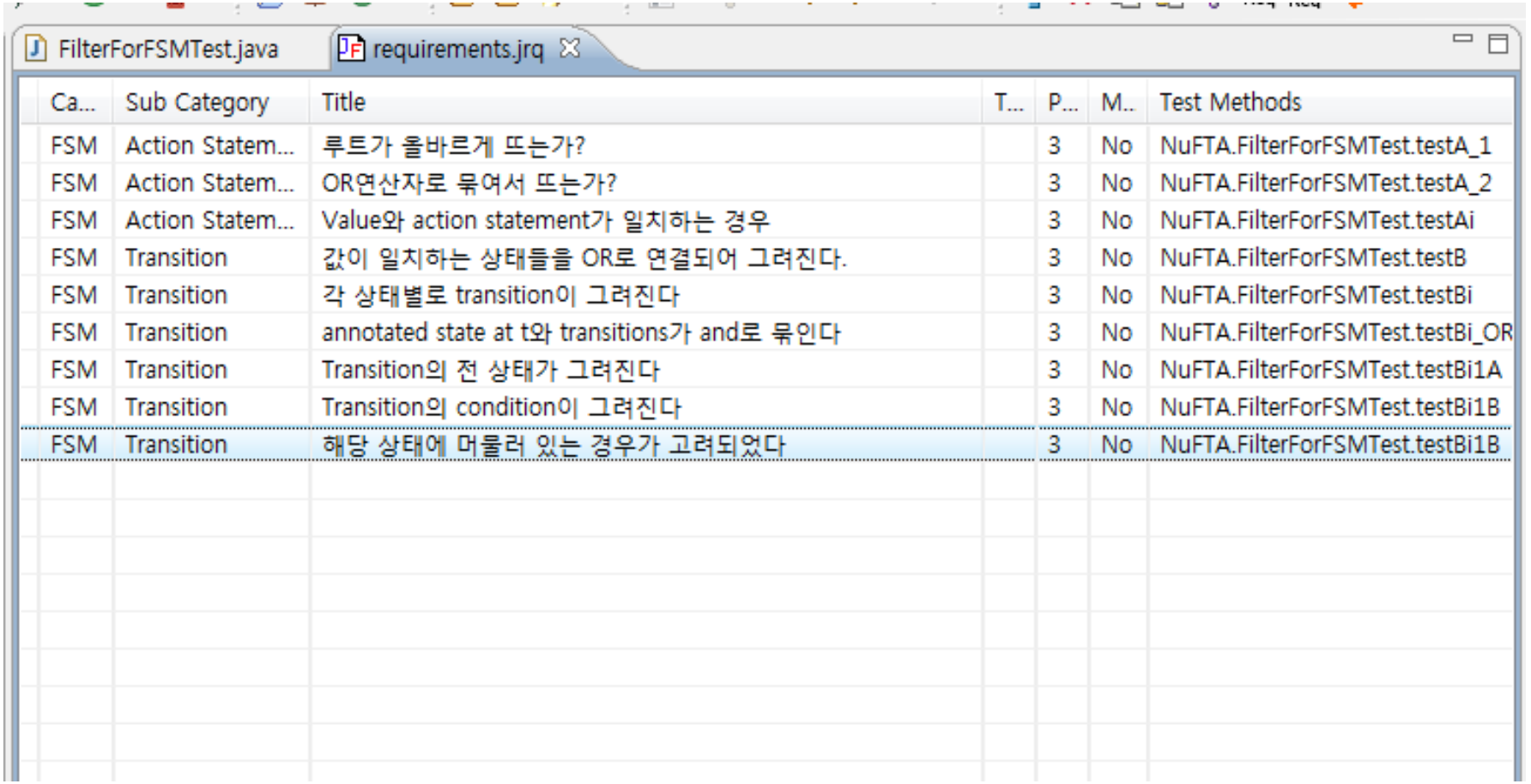
- 다른 작업 build 후
- 주기별 소스의 변경이 있는 경우
- 일정한 시간 주기
- 추가 기능 활용 가능

Unit Test “NuSRS”

Requirement 1.
Generated Annotated FSM

Generate Fault Tree(FSM)

❖ 요구사항 작성

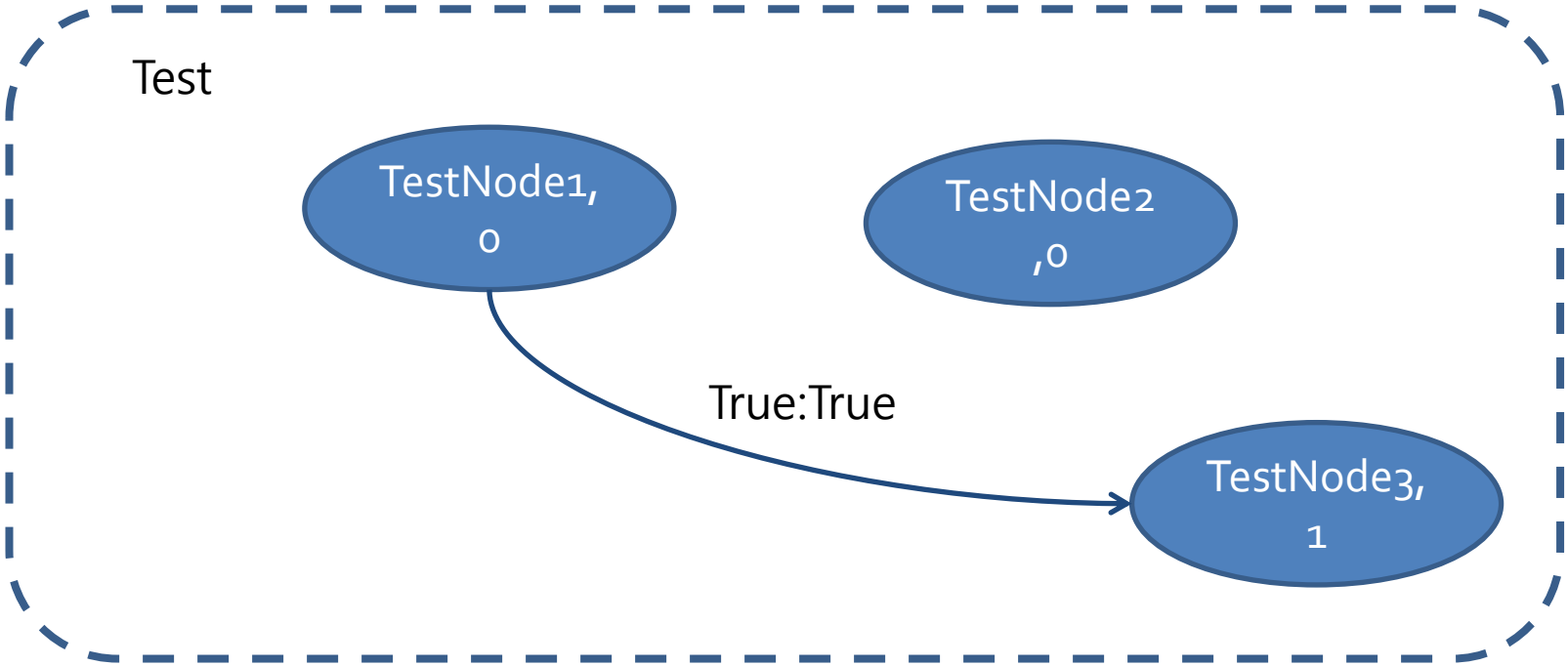


FilterForFSMTest.java | requirements.jrq

Ca...	Sub Category	Title	T...	P...	M..	Test Methods
FSM	Action Statem...	루트가 올바르게 뜨는가?		3	No	NuFTA.FilterForFSMTest.testA_1
FSM	Action Statem...	OR연산자로 묶여서 뜨는가?		3	No	NuFTA.FilterForFSMTest.testA_2
FSM	Action Statem...	Value와 action statement가 일치하는 경우		3	No	NuFTA.FilterForFSMTest.testAi
FSM	Transition	값이 일치하는 상태들을 OR로 연결되어 그려진다.		3	No	NuFTA.FilterForFSMTest.testB
FSM	Transition	각 상태별로 transition이 그려진다		3	No	NuFTA.FilterForFSMTest.testBi
FSM	Transition	annotated state at t와 transitions가 and로 묶인다		3	No	NuFTA.FilterForFSMTest.testBi_OR
FSM	Transition	Transition의 전 상태가 그려진다		3	No	NuFTA.FilterForFSMTest.testBi1A
FSM	Transition	Transition의 condition이 그려진다		3	No	NuFTA.FilterForFSMTest.testBi1B
FSM	Transition	해당 상태에 머물러 있는 경우가 고려되었다		3	No	NuFTA.FilterForFSMTest.testBi1B

Generate Fault Tree(FSM)

- Set Up 코드
  - 테스트에 사용하기 위한 간단한 형태의 FSM
  - FSM : Node 3, Transition 1



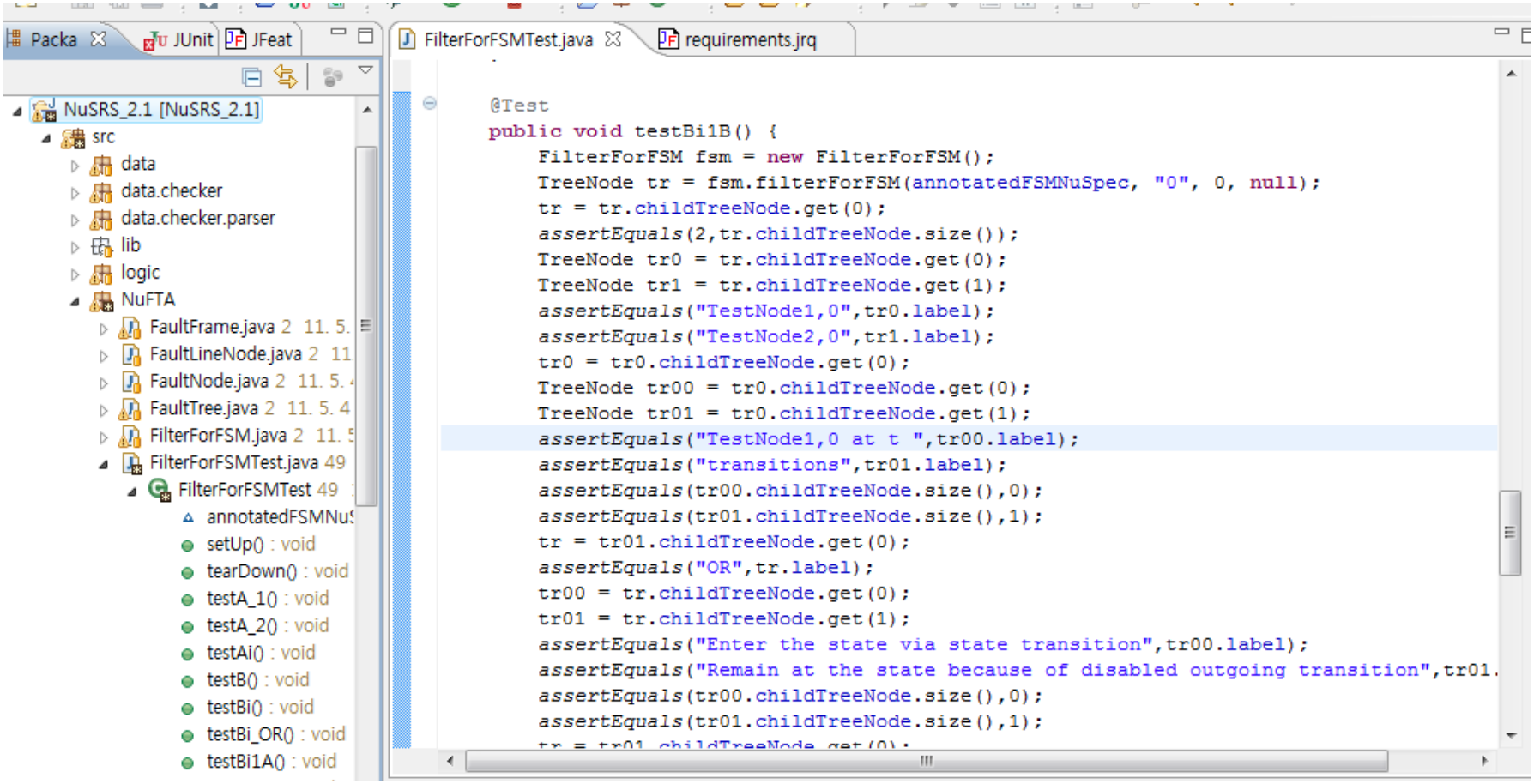
Generate Fault Tree(FSM)

- Set Up 코드
  - setUp 함수는 매 Test 시작전에 호출되는 함수
  - 테스트 전에, 다음의 소스코드로 FSM 생성

```
@Before
public void setUp() throws Exception {
    annotatedFSMNuSpec = new FSM("Test");
    NuNode node1 = new NuNode("TestNode1,0");
    ((FSM)annotatedFSMNuSpec).addNode(node1);
    ((FSM)annotatedFSMNuSpec).addInput(node1);
    NuNode node2 = new NuNode("TestNode2,0");
    ((FSM)annotatedFSMNuSpec).addNode(node2);
    NuNode node3 = new NuNode("TestNode3,1");
    ((FSM)annotatedFSMNuSpec).addNode(node3);
    FSMTransition transition1 = new FSMTransition(node1,node3);

    transition1.setConditions("true := true");
    ((FSM)annotatedFSMNuSpec).addTransition(transition1);
}
@After
public void tearDown() throws Exception {
    //System.out.println("@After");
}
```

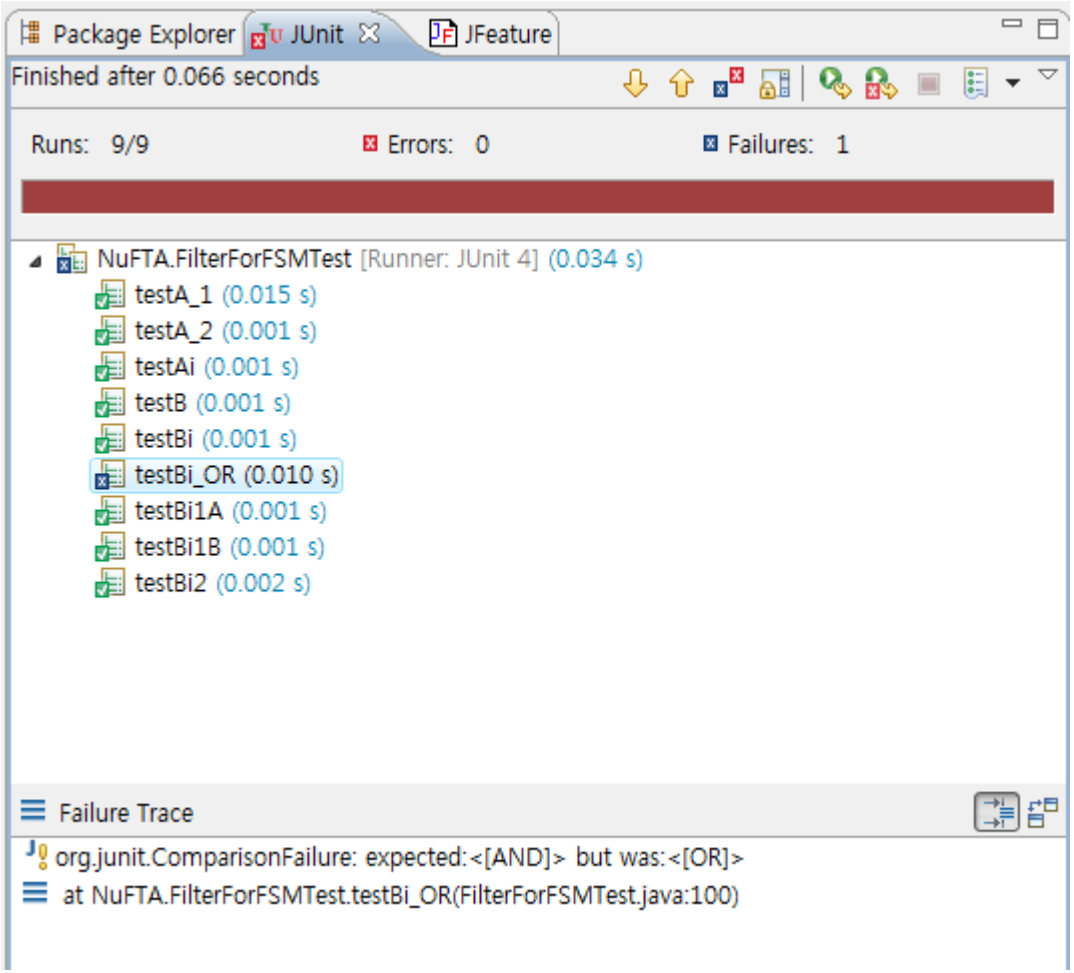
Generate Fault Tree(FSM)

❖ 테스트케이스 작성



Generate Fault Tree(FSM)

❖ Junit 실행



Generate Fault Tree(FSM)

❖ JFeature 실행

Home

All Categories

FSM *(50%)*

All Requirements

루트가 올바르게 뜨는가? *(100%)*
OR연산자로 묶여서 뜨는가? *(100%)*
Value와 action st... *(100%)*
Transition의 cond... *(100%)*
값이 일치하는 상태들을 OR로... *(100%)*
Transition의 전 상태... *(100%)*
각 상태별로 transitio... *(100%)*
해당 상태에 머물러 있는 경우... *(100%)*
annotated state ... *(0%)*

Requirement Coverage Report

Requirement Coverage Summary

FSM (2)	1 (50%)	1 (50%)

Number of Requirements	9
Unique Test Methods	8
Requirements:Test Methods Ratio	1:1
Missing Test Methods	None

Requirement Coverage Details

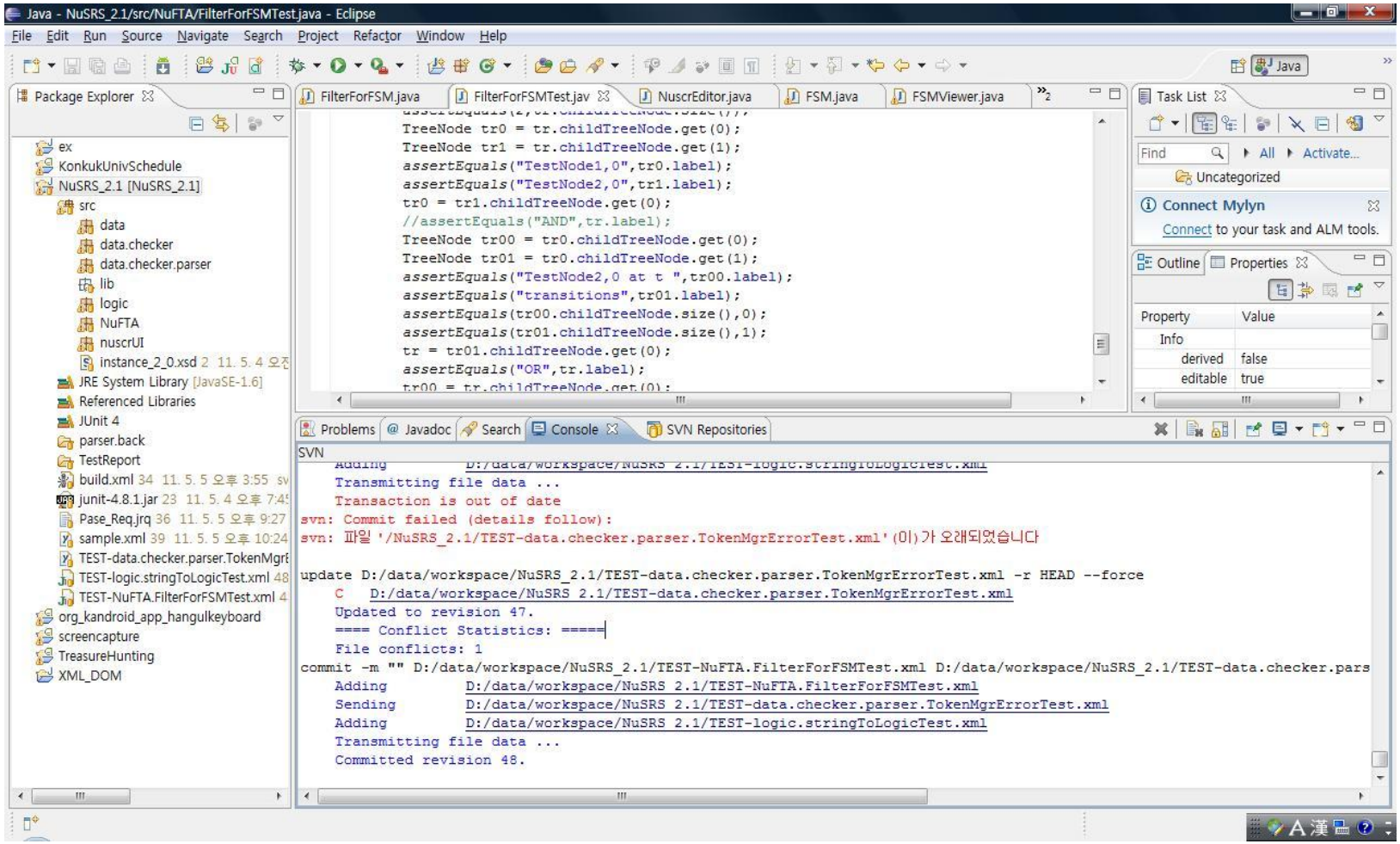
Sr#	Coverage Item	Coverage
1.	Action Statement (3)	3 (100%)
2.	Transition (6)	5 (83.33%) / 1 (16.67%)

Report generated on 금, 06 5월 2011 01:48:06 KST

Powered By JFeature

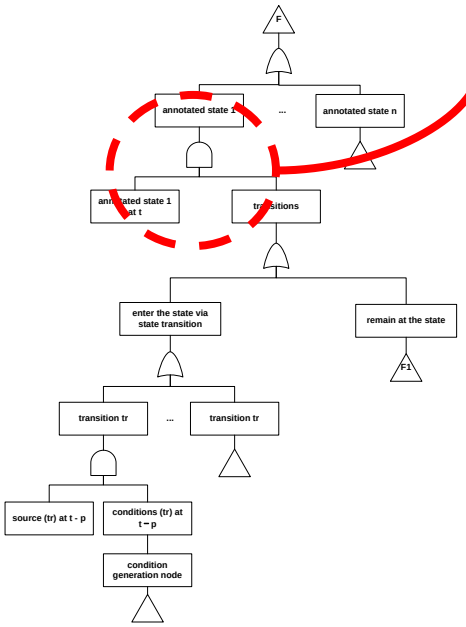
Generate Fault Tree(FSM)

❖ Commit



Generate Fault Tree(FSM)

- Requirements
  - 테스트 실패



실패한 테스트 케이스 : annotated state at t와 transitions가 and로 묶인다 (이부분이 실제로는 OR로 묶이는걸 확인함.)

  - 그 외의 모든 테스트 케이스는 통과

Unit Test “NuSRS”

Requirement 2.
Analyze Logic by expression

Analyze Logic by expression

- 입력받은 수식을 논리적 구조로 분석
- 수식에 사용되는 기호
  - |, &, +, -, =, !, <, >, <=, >=, :=, (,)
- 수식
  - 연산자와 변수로 나누어짐 (A=ture 형태)
  - 불값을 나타냄
  - 수식 내 '(', ')' 가 발생한다면 우선 확인

Analyze Logic by expression

- logic.stringToLogic.java
  - 입력받은 수식 문자열 s를 트리 형태로 Parsing하는 파일

```
public class stringToLogic {

    public String fullString;
    public ArrayList<logicMember> lm = new ArrayList<logicMember>();
    public ArrayList<treeMember> tm = new ArrayList<treeMember>();
//  protected ArrayList<subList> sl = new ArrayList<subList>();
    protected int numNot=0;
    protected int maxLevel=0;

    public stringToLogic(String s)
    {
        //logicMember setting
        fullString = s;
        s = s.trim();
        makeMember(s, 0, 0);
        setTruth();
        removeNot();
        removeBracket();
        removeEnter();
        convertTruth();
        setSubElement();

        //treeMember setting
        copyLM();
        setLevelwithPriority();
        setLevelwithGate();
        getMaxLevel();
        setChild(0);
        removeDuplication();
    }
```

생성자에서 변환 과정 수행

Parse_Req.jrq

Require...	Category	Sub Category	Title	T...	Priority	Must Have	Test Methods	Depe
R2_001	Parse	logicMember	makeMember		3	No	logic.stringToLogicTest.testMakeMember	
R2_002	Parse	logicMember	setTruth		3	No	logic.stringToLogicTest.testSetTruth	
R2_003	Parse	logicMember	removeNot		3	No	logic.stringToLogicTest.testRemoveNot	
R2_004	Parse	logicMember	removeBracket		3	No	logic.stringToLogicTest.testRemoveBracket	
R2_005	Parse	logicMember	removeEnter		3	No	logic.stringToLogicTest.testRemoveEnter	
R2_006	Parse	logicMember	convertTruth		3	No	logic.stringToLogicTest.testConvertTruth	
R2_007	Parse	logicMember	setSubElement		3	No	logic.stringToLogicTest.testSetSubElement	
R2_008	Parse	logicMember	copyLM		3	No	logic.stringToLogicTest.testCopyLM	
R2_009	Parse	treeMember	setLevelwithPriority		3	No	logic.stringToLogicTest.testSetLevelwithPriority	
R2_010	Parse	treeMember	setLevelwithGate		3	No	logic.stringToLogicTest.testSetLevelwithGate	
R2_011	Parse	treeMember	getMaxLevel		3	No	logic.stringToLogicTest.testGetMaxLevel	
R2_012	Parse	treeMember	setChild		3	No	logic.stringToLogicTest.testSetChild	
R2_013	Parse	treeMember	removeDuplication		3	No	logic.stringToLogicTest.testRemoveDuplication	

Analyze Logic by expression

예) (f_SG1_LO_FLOW_Val_Out_t1 -f_SG1_LO_FLOW_Val_Out <=k_SG1_LO_FLOW_100ms_Rate) &
(f_SG1_LO_FLOW_Val_Out -k_SG1_LO_FLOW_Trip_Step >=k_SG1_LO_FLOW_SP_Lo_Lim) &
(f_SG1_LO_FLOW_Val_Out -k_SG1_LO_FLOW_Trip_Step <=k_SG1_LO_FLOW_SP_Hi_Lim)

문자열 변환 과정

Step 1. 복합문을 쪼갠다 : &, !, (,), | 로 나누기 - makeMember
Step 2. !가 미치는 구문을 찾아내어 상쇄시켜 중복 !연산을 최소화 - setTruth
Step 3. Step2에서 ! 영향을 변수로 저장했기 때문에 구문에서 ! 제거 - removeNot
Step 4. 괄호, 개행문제 제거 - removeBracket, removeEnter
Step 5. 연산자에 대해 !을 수행 - convertTruth
Step 6. 나누어진 구문들을 비교 연산자를 기준으로 나눔 - setSubElement
Step 7. +, - 연산자를 기준으로 나눔 - divideSub

f_SG1_LO_FLOW_Val_Out_t1 / - / f_SG1_LO_FLOW_Val_Out / <=
k_SG1_LO_FLOW_100ms_Rate / & /f_SG1_LO_FLOW_Val_Out / -
k_SG1_LO_FLOW_Trip_Step / >= /k_SG1_LO_FLOW_SP_Lo_Lim) / &
f_SG1_LO_FLOW_Val_Out / - /k_SG1_LO_FLOW_Trip_Step / <=
k_SG1_LO_FLOW_SP_Hi_Lim)

Analyze Logic by expression

- void makeMember(String input, int not, int level)
  - 입력받은 수식을 구분자 &, !, (,), | 로 나누어 Array에 저장

```
@Test
public void testMakeMember() {
    STL.lm.clear();
    STL.makeMember(STL.fullString, 0, 0);
    int count = STL.fullString.length();
    int index=0, level=0;
    char ch;
    boolean exp = false;
    for(int i=0; i<count ; i++ )
    {
        ch = STL.fullString.charAt(i);
        if(ch== ' ')
            continue;
        if( (ch=='&') || (ch=='!') ||(ch=='(') ||(ch==')') || (ch=='\\') )
        {
            if(ch == '(')   level++;
            if(ch==')')     level--;

            assertEquals( ch+"", STL.lm.get(index).element );
            assertTrue( STL.lm.get(index).truth);
            assertEquals( level, STL.lm.get(index).level);
            index++;
            exp=false;
        }else{
            if(exp==false){
                exp = true;
                index++;
            }
        }
    }
}
```

Analyze Logic by expression

- void setTruth()
  - '!'가 영향을 미치는 요소를 연산
  - 영향받는 '!'의 수가 홀수라면 요소는 false
  - assertFalse, assertTrue 로 검사

```
for(logicMember lm : this.lm)
{
    if((lm.numNot%2)==1)
        lm.truth = false;
}
```

```
@Test
public void testSetTruth() {

    for(logicMember lm : STL.lm)
    {
        if((lm.numNot%2)==1)
            assertFalse(lm.truth);
        else
            assertTrue(lm.truth);
    }
}
```

Analyze Logic by expression

- void removeNot()
  - 수식 요소 중 '!'를 제거
  - 수식 요소 중 '!' 존재 여부를 검사

```
@Test
public void testRemoveNot() {
    int count = STL.lm.size();

    for(int i=0; i<count; i++)
        assertFalse(STL.lm.get(i).element.contains("!"));
}
```

- void removeBracket()
- void removeEnter()
  - removeNot과 동일한 방식으로 검사

Analyze Logic by expression

- void convertTruth()
  - 앞에서 setTruth 함수로 false로 바꾼 요소(!를 갖던 요소)를 true로 변경 & 내부 연산자를 ! 연산
  - 여전히 남은 false가 존재하는 검사

```
@Test
public void testConvertTruth() {
    for(logicMember lm : STL.lm)
        assertTrue(lm.truth);
}
```

- void setSubElement()
  - 요소 중 비교 연산자(>=, >, <=, <, :=, !=,=)가 있다면 연산자 기준으로 좌 / 우로 나누어 divideSub 함수 수행

Analyze Logic by expression

- void setSubElement()
  - 요소 중 비교 연산자(>=, >, <=, <, :=, !=,=)가 있다면
    연산자 기준으로 좌 / 우로 나누어 divideSub 함수 수행 및
    연산자 위치에 sign 할당

```
@Test
public void testSetSubElement() {

    stringToLogic STL2 = new stringToLogic( STL.fullString);
    STL2.makeMember(STL.fullString, 0, 0);
    STL2.setTruth();
    STL2.removeNot();
    STL2.removeBracket();
    STL2.removeEnter();
    STL2.convertTruth();
```

```
int count = STL2.fullString.length();
int index=0;
char ch;
String sign;
boolean exp = false;
for(int i=0; i<count ; i++ )
{
    ch = STL2.lm.toString().charAt(i);
    if(ch== ' ')
        continue;
    if( (ch=='>') || (ch=='<') ||(ch==':') ||(ch=='!') || (ch=='=') )
    {
        if(STL2.fullString.charAt(i+1)=='=')
            sign = ch+"=";
        else
            sign = ch+"";

        assertEquals( sign, STL.lm.get(++index).sign);
        exp=false;
    }else{
        if(exp==false){
            exp = true;
            index++;
        }
    }
}
```

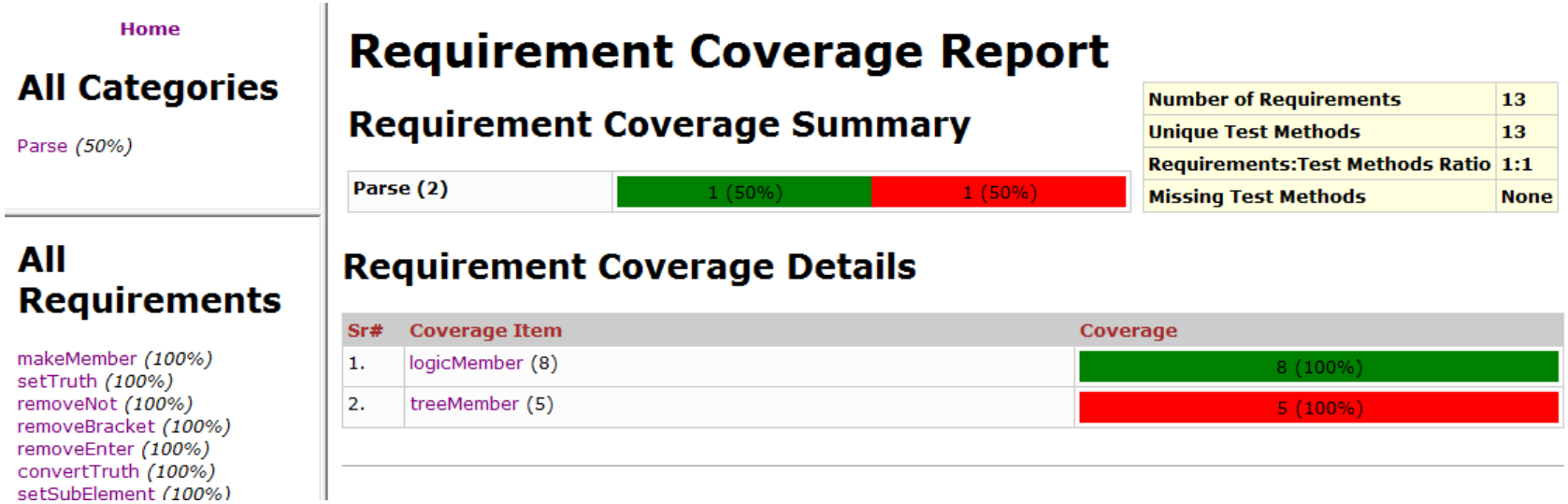
Analyze Logic by expression

- void copyLM()
  - 앞에서 수식을 Parsing 하여 만든 logic 타입을 child를 갖는 tree 타입으로 복사

```
@Test
public void testCopyLM() {
    STL.tm.clear();
    STL.copyLM();
    int count = STL.lm.size();
    for(int i=0; i<count; i++)
    {
        assertEquals(STL.lm.get(i).element, STL.tm.get(i).element);
        assertEquals(STL.lm.get(i).level, STL.tm.get(i).level);
        assertTrue(STL.lm.get(i).truth);
        assertEquals(STL.lm.get(i).leftSubElement, STL.tm.get(i).leftSubElement);
        assertEquals(STL.lm.get(i).sign, STL.tm.get(i).sign);
        assertEquals(STL.lm.get(i).rightSubElement, STL.tm.get(i).rightSubElement);
    }
}
```

Analyze Logic by expression

❖ JFeature



테스트 결과 & 소감

- 문자열을 Parsing하여 의미 있는 단위로 나눈 단계 테스트 완료(검증 완료)
- 계층적 Tree 형태로 변환하는 테스트 케이스 작성은 실패(미검증)
- 입력값을 무조건 받는 생성자에서 모든 연산을 한번에 연산하며
  void 형태의 함수로 인해 테스트케이스 작성을 위한 어려움이 발생
- 테스트를 고려한 코드 작성법도 필요

END