“Priority” Category Partitioning Testing Tool

OOPT Final Report

Demonstration

Project Team T4

201311265 김상원

201210194 김정환

201311269 김제헌

201311297 이상명

Date

2017-06-08

1. Stage 1000 Review

먼저 Stage 1000을 되돌아보면 1000단계에서는 프로젝트를 시작하고, 그에 맞는 목표를 설정하고, 요구사항을 결정한 뒤, 그에 맞는 UseCase Diagram을 도출해 내었다. 아래는 그 과정을 정리한 것이다.

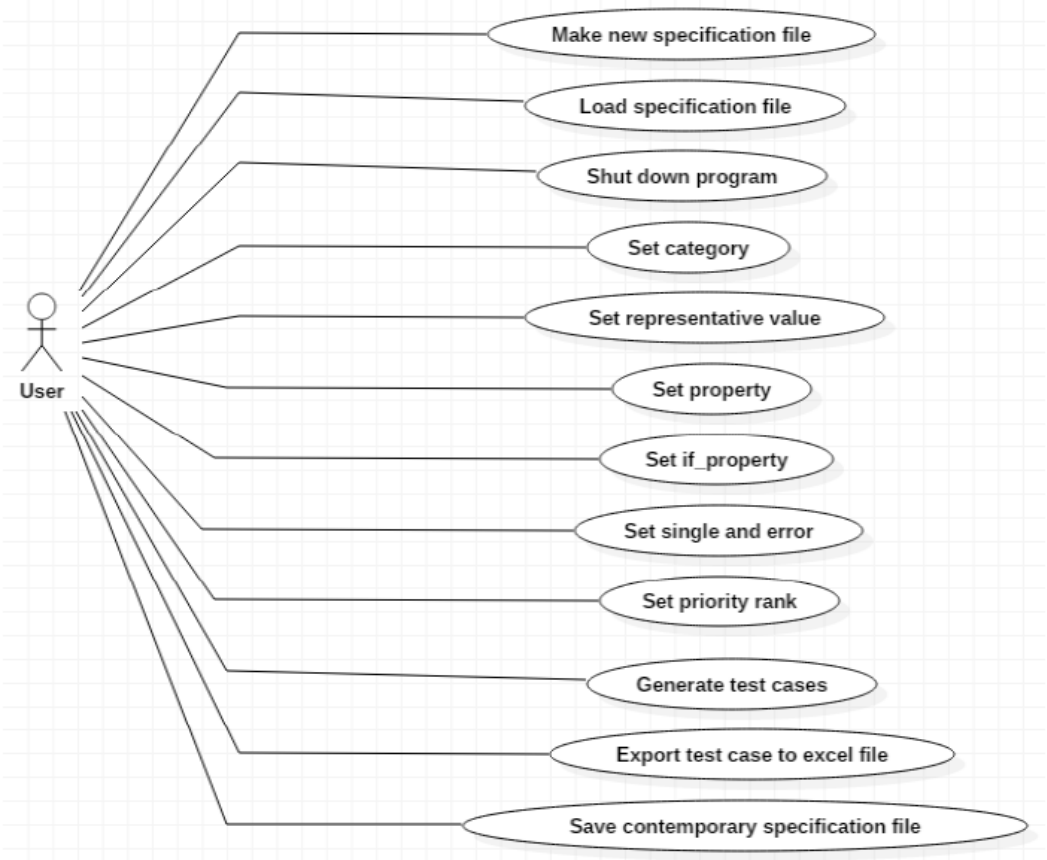
1.1 Project Objectives & Scope

테스트명세(Category, Representative Values, Constraints)를 입력 받아 자동으로 테스트케이스를 생성하고 그 결과를 엑셀로 저장해주는 Category Partitioning Test Case Generating Tool을 만든다. "Priority"는 테스트단계에서, 모든 테스트케이스가 동일한 중요도를 갖지 않는다는 사실에 착안하였다. 더 중요한 테스트케이스를 덜 중요한 케이스보다 시간상 먼저 테스팅 함으로써 테스터는 중요한 시스템 오류를 짧은 시간 안에 파악하여 대처할 수 있다. 이는 소프트웨어의 규모가 클수록 더욱 필요한 기능일 것이다. 따라서 단순히 테스트케이스를 생성시키는 것뿐만 아니라 케이스들간에 우선순위를 매기는 것을 목표로 한다.

1.2 Functional Requirements

Function	Description
New specification file	새로운 명세파일을 만든다.
Load specification file	탐색기를 통해서 또는 최근 실행 목록에서 기존의 명세파일을 읽는다.
Shut down program	프로그램을 종료한다.
Set category	Category를추가또는삭제, 변경한다.
Set representativevalues	representativevalue를추가또는삭제, 변경한다.
Set property	Property constraints를추가또는삭제, 변경한다.
Set if-property	If-property constraints를추가또는삭제, 변경한다.
Set single and error	single, error constraints를변경한다
Set priority rank	priority를정한다.
Generate test cases	주어진 명세 정보로부터 Category Partitioning Test Case를 생성한다.
Export test case to excel file	생성된 Category Partitioning Test Case을 엑셀파일로 생성한다
Save contemporary specification file	지금까지 작성한 명세파일을 저장한다.

1.3 Draw a Use Case Diagram



2. Stage 2030 Review

Stage 2030은 1000단계에서 정의했던 UseCase를 다시 상세하게 재정의 하고, 이를 통해 대략적인 클래스 다이어그램을 그려내고, 시스템을 블랙박스로 보는 시스템 시퀀스 다이어그램을 그려내는 것이 목표였다. 아래는 그에 따른 팀의 수행과정을 요약해서 나열한 것이다.

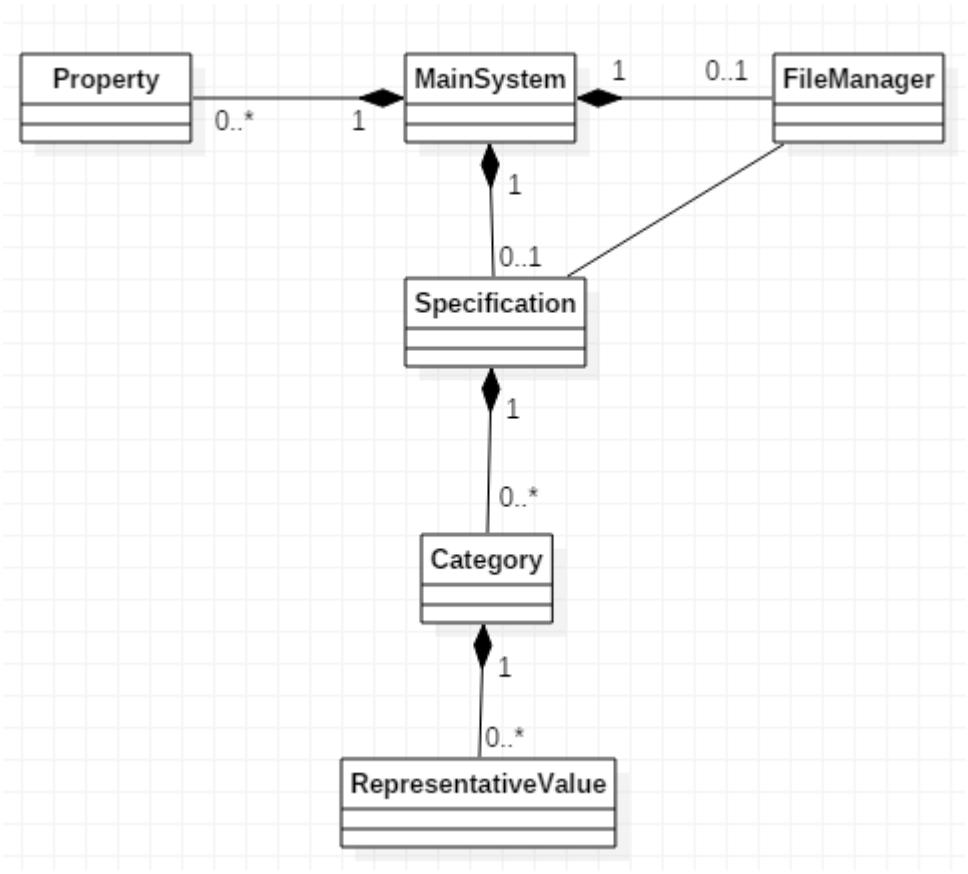
2.1 Define Essential Use Case

아래에는 예시로 두 개 정도 표시해 놓았다.

UseCase	Make new specification file
Actor	User
Purpose	Test case generate를 위해 새로운 specification 파일을 작성한다.
Overview	초기화면에서 „새로 만들기"버튼을 클릭하면 저장할 파일 이름을 입력 받는 화면이 출력되어, 입력 받은 이름으로 새로운 specification파일을 생성한다. 최근 파일 항목에 반영한다.
Type	Primary
CrossReference	FuntionalRequirements:R1.1
Pre-Requisites	N/A
TypicalCoursesofEvents	(A):Actor,(S):System 1. (A) 초기화면에 있는 '새로만들기' 버튼을 누른다. 2. (S) specification 파일 이름을 입력 받는 단계로 넘어간다. 3. (A) 새 파일 이름 항목란에 이름을 입력한다. 4. (A) '이름결정' 버튼을 누른다. 5. (S)지정된 이름의 specification 파일을 만든다. 6. (S)생성된 specification 파일을 recent file list에 갱신한다. 7. (S) Set category page로 이동한다.
AlternativeCoursesofEvents	N/A
ExceptionalCoursesofEvents	E. 입력한 이름과 같은 파일이 존재하면 예외처리한다.

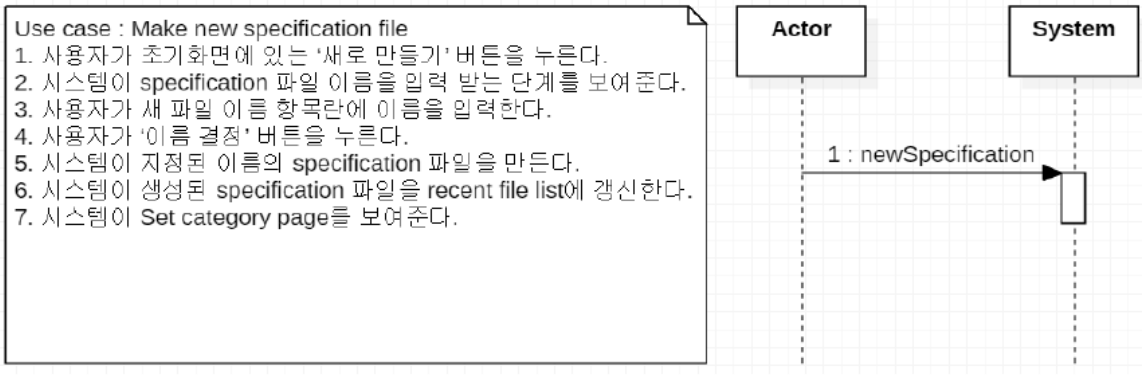
UseCase	Export test case to excel file
Actor	User
Purpose	Test cases를엑셀파일로저장한다.
Overview	Test case generate이끝난후, 생성할 엑셀파일의 이름을 받는다. 이후 생성된 test cases를 엑셀파일로 저장한다. 생성된 엑셀파일은 설정한 Priority 순으로 정렬된다.
Type	Primary
CrossReference	FuntionalRequirements:R4
Pre-Requisites	입력된Specification이test case를만들기위한충분한값이입력되어서, Generate test cases가실행되어야한다.
TypicalCoursesofEvents	(A):Actor,(S):System 1. (S) 'Generate test cases가실행된다. 2. (S) 생성할 엑셀파일의 이름을 받는 창이 뜬다. 3. (A) 생성할 엑셀파일의 이름을 입력한다. 4. (S) 생성된test cases를 Priority순으로 정렬한다. 5. (S) 입력 받은 엑셀파일의 이름으로 엑셀파일을 만든 후, Priority순으로 정렬된 test cases를 저장한다.
AlternativeCoursesofEvents	N/A
ExceptionalCoursesofEvents	N/A

2.2 Define Domain Model

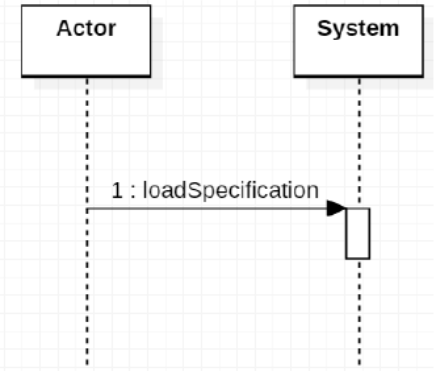


2.3 Define System Sequence Diagrams

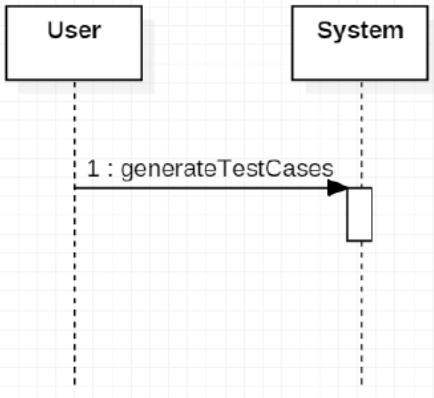
아래에 예시로 몇 개를 표시해 보았다.



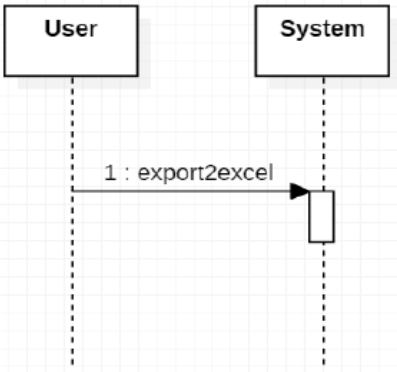
Use case : Load specification file
1. 사용자가 초기화면에서 '불러오기'버튼을 클릭하거나,
   최근 파일 목록 중 하나를 선택한다.
2. 시스템이 선택된 specification 파일을 불러 읽는다.
3. 시스템이 선택된 specification 파일의 recent file list 순위를
   갱신한다.
4. 시스템이 specification 파일이 저장된 시점의 값들을 보여준다.



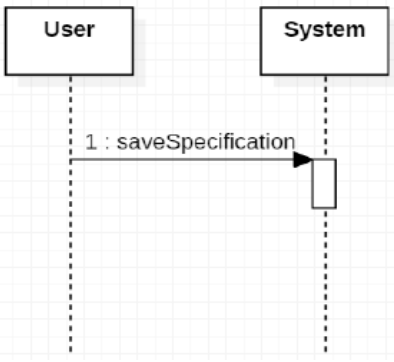
Use case : Generate test cases
1. 사용자가 'Test case 생성 후 엑셀로 저장하기' 버튼을
누른다.
2. 시스템이 입력된 specification을 통해 test case를 만든다.
3. 시스템이 총 개수를 새로운 화면을 통해 사용자에게 알려
   준다.



Use case : Export test case to excel file
1. 'Generate test cases가 실행된다.
2. 시스템이 생성된 test cases를 Priority순으로 정렬 한다.
3. 시스템이 Priority순으로 정렬된 test cases를 엑셀 파일로
   저장한다



Use case : Save contemporary specification file
1. 사용자가 저장 요청을 보낸다.
2. 시스템이 현재까지의 작업 상태를 specification 파일로
   저장한다.

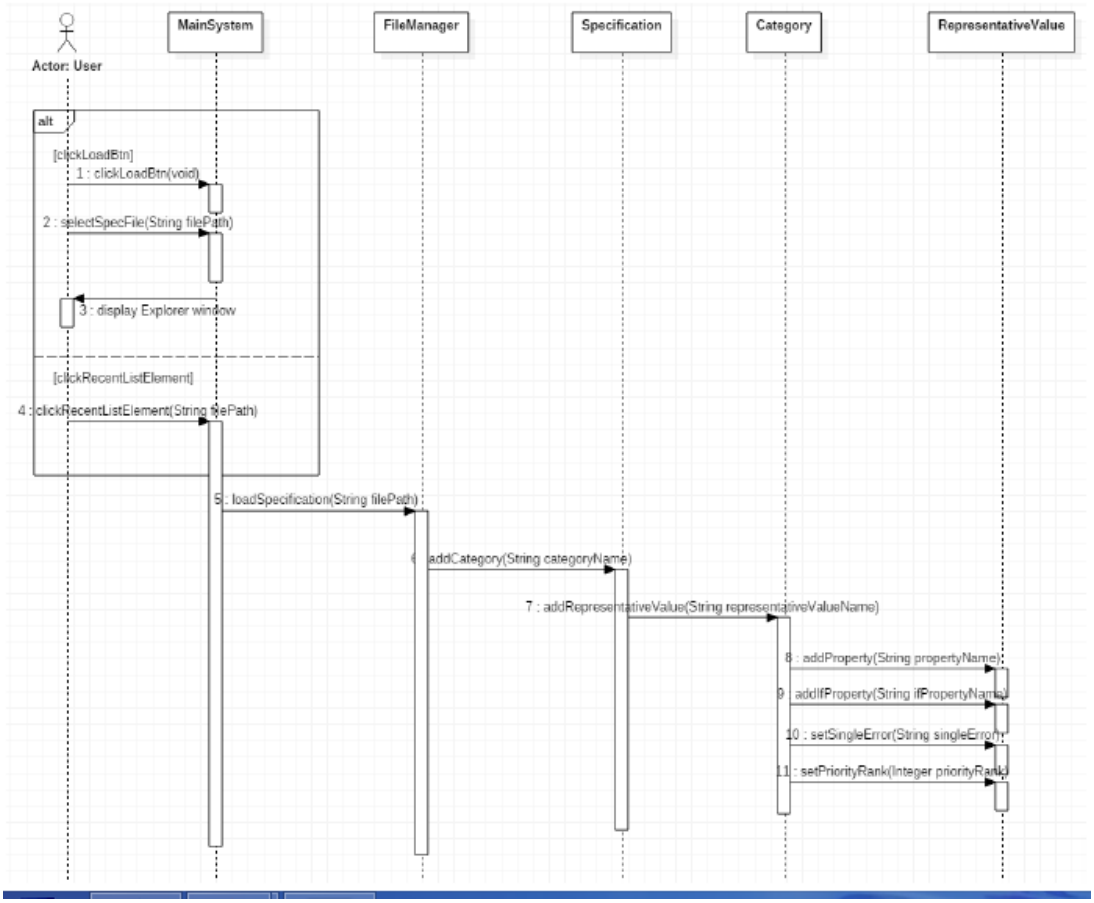


3. Stage 2040 Review

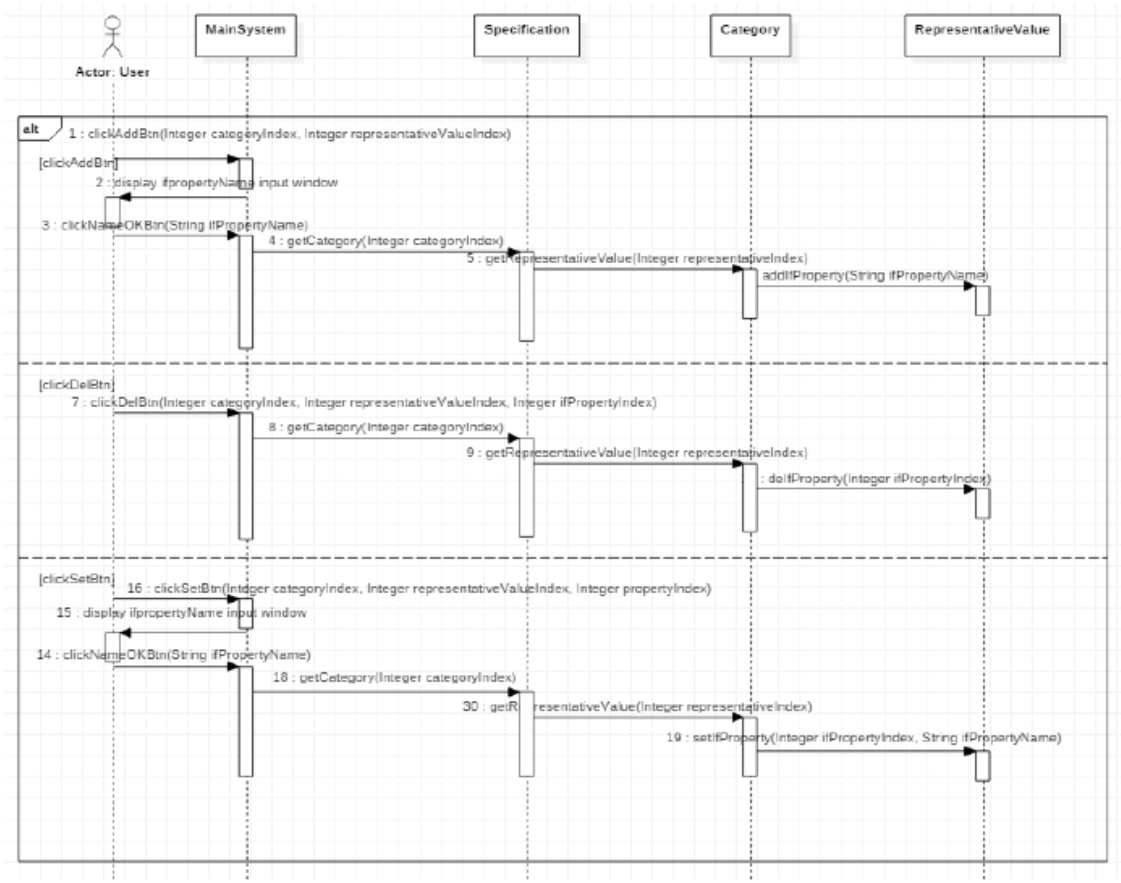
Stage 2040에서의 중요 단계는 Interaction Diagram과 클래스 다이어그램을 그려내는 단계이다. 먼저 시퀀스 다이어그램은 GUI의 부분을 제외하고, 세세하게 표현하고, 이를 바탕으로 클래스 다이어그램을 그려내는 것이 목표인 단계이다. 아래는 그 과정들을 표시하였다.

3.1 Define Interaction Diagrams

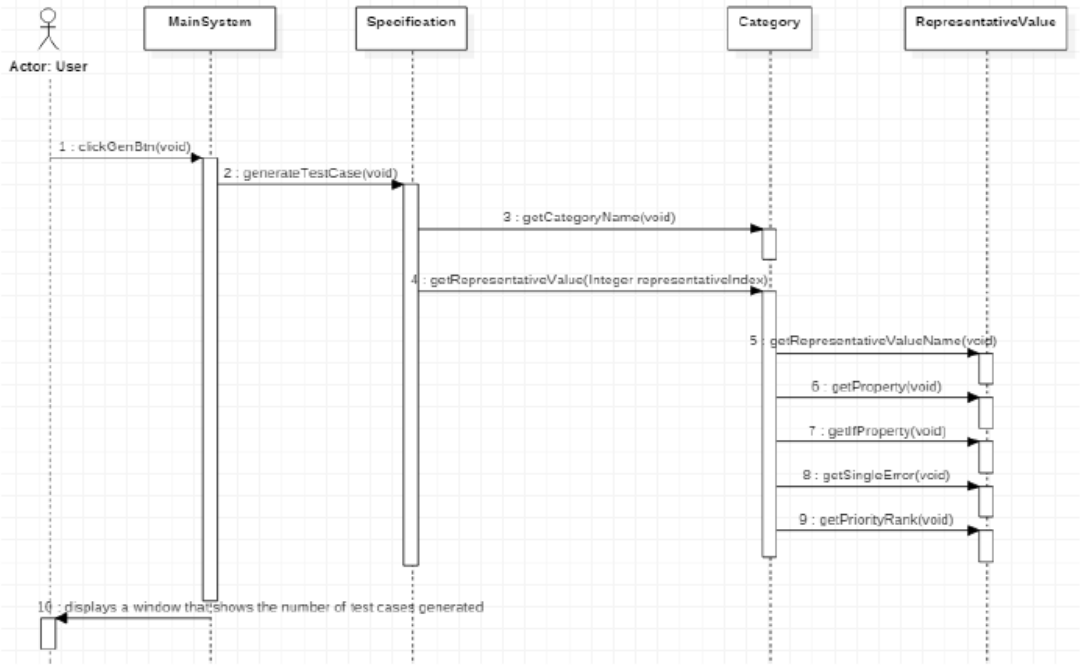
[loadSpecification]



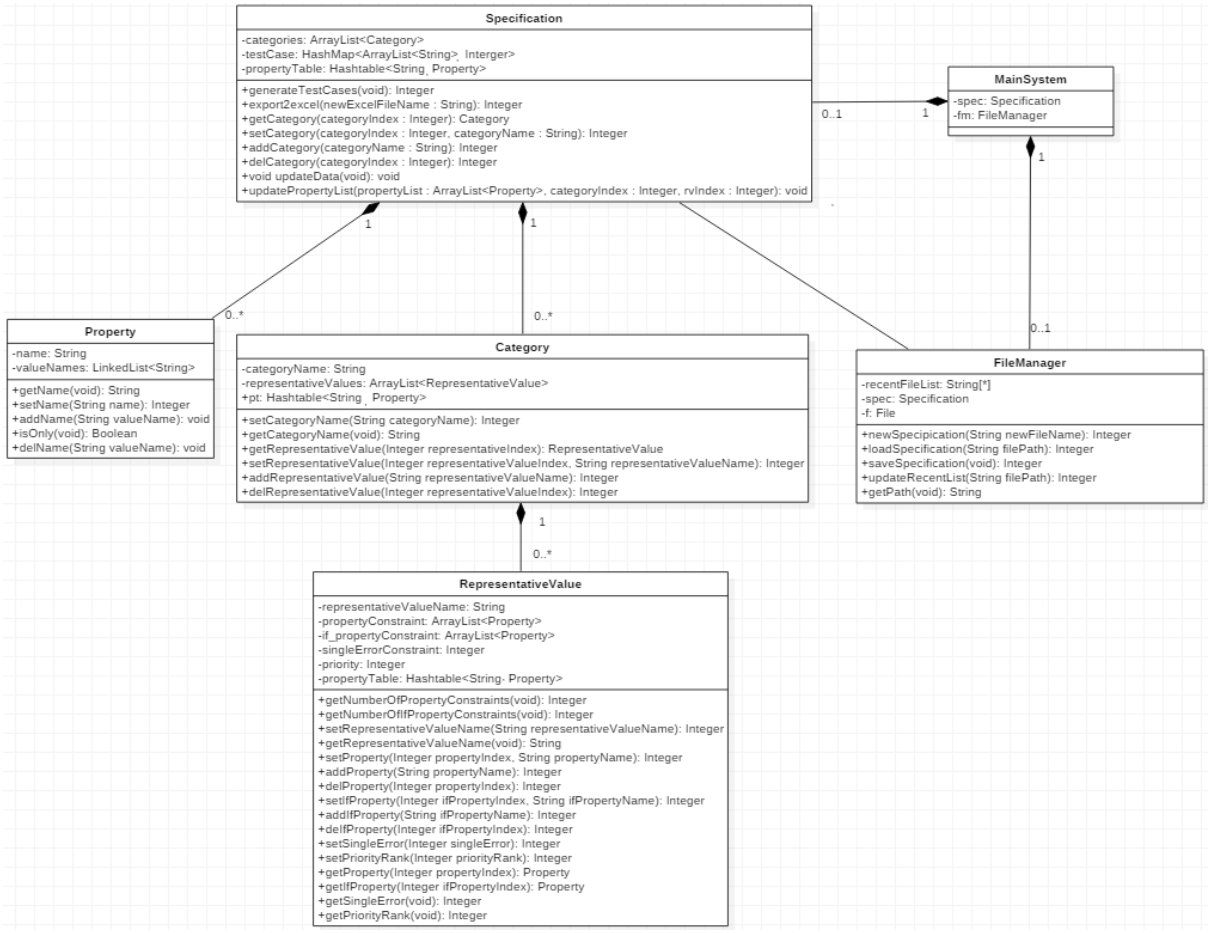
[setIfProperty]



[generateTestCases]



3.2 Define design class diagram



4. Stage 2050 & 2060 Review

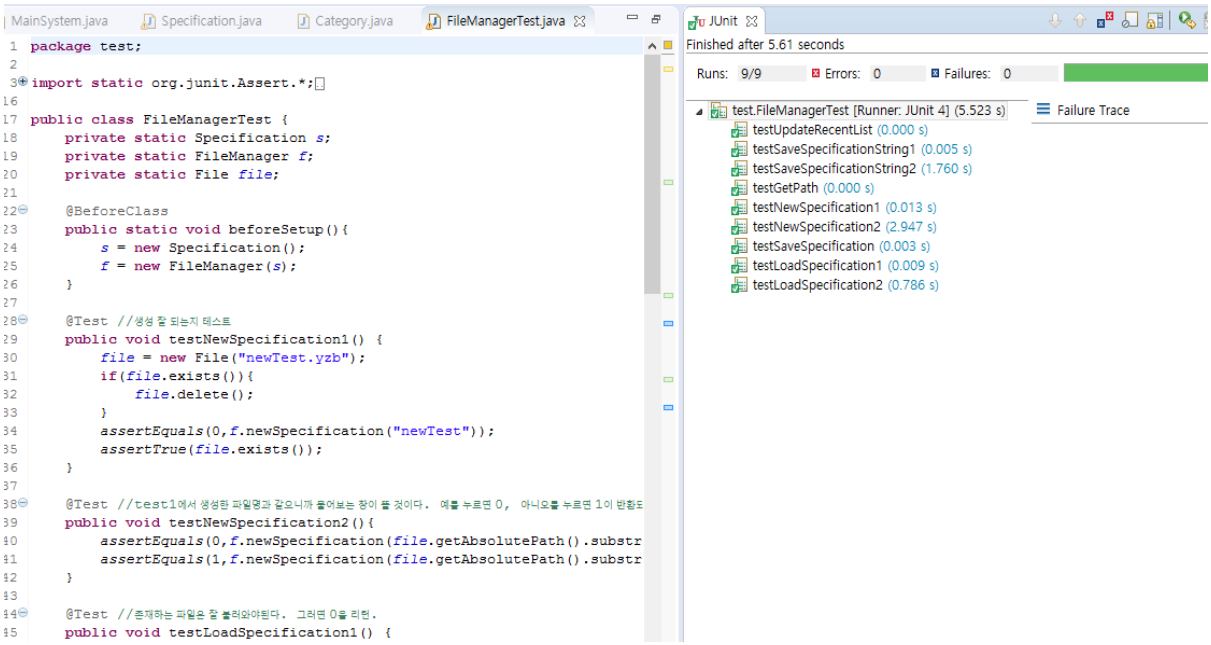
Stage 2050은 이전 단계에서의 작업을 바탕으로 하여 실제로 프로그래밍을 하는 단계이고, Stage 2060은 이러한 완성된 프로그램을 유닛 테스팅, 통합 테스팅, 시스템 테스팅을 거치는 단계이다. 아래는 이러한 단계에서 팀이 구현한 코드 일부와 유닛테스팅 결과의 일부이다.

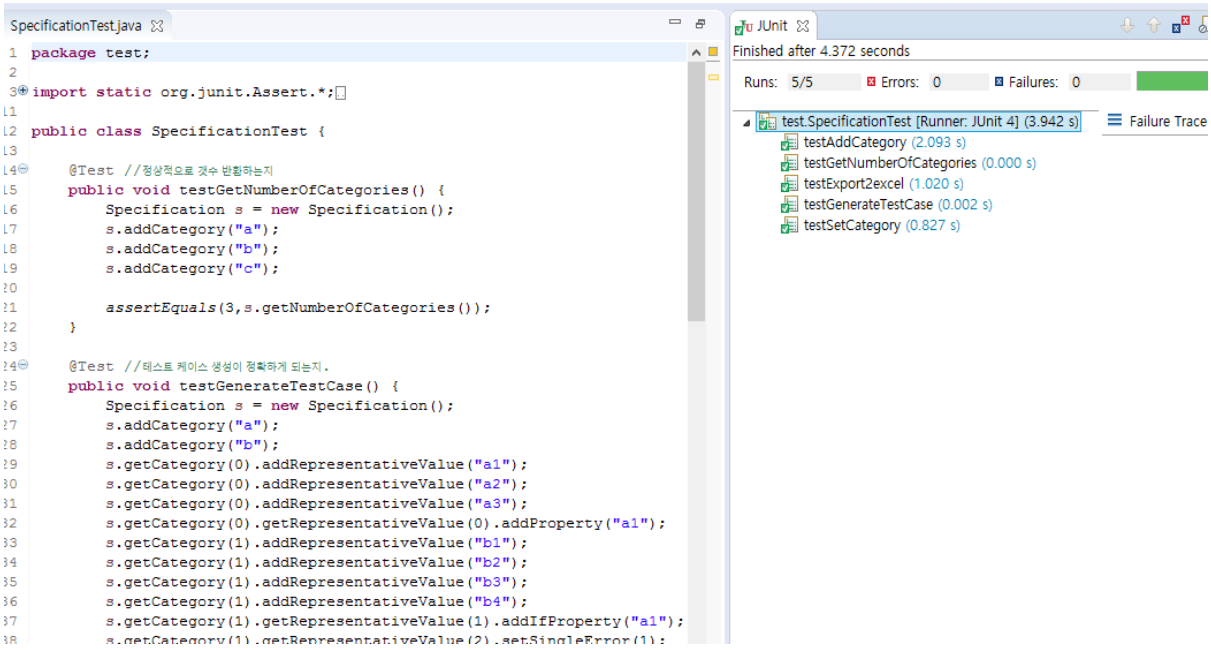
MainSystem.java Specification.java

```java
package System;

import java.util.*;

public class Specification {
    private ArrayList<Category> categories;
    private HashMap<ArrayList<String>, Integer> testCase;
    private Hashtable<String, Property> propertyTable = new Hashtable<String, Property>();

    // Constructor
    public Specification() {
        categories = new ArrayList<Category>();
    }

    // number of categories
    public int getNumberOfCategories() {
        return categories.size();
    }

    // fill TestCase

    public int generateTestCase() {
        boolean isExist=false;
        for(int i=0;i<getNumberOfCategories();i++){
            if(getCategory(i).getNumberOfRepresentativeValues()!=0){
                isExist=true;
                break;
            }
        }
        if(this.getNumberOfCategories()!=0 && isExist){
            testCase = new HashMap<ArrayList<String>, Integer>();
            ArrayList<Property> propertyList = new ArrayList<Property>();
            ArrayList<RepresentativeValue> testCaseList = new ArrayList<RepresentativeValue>();
            f(0, propertyList, testCaseList, testCase, 0);
```

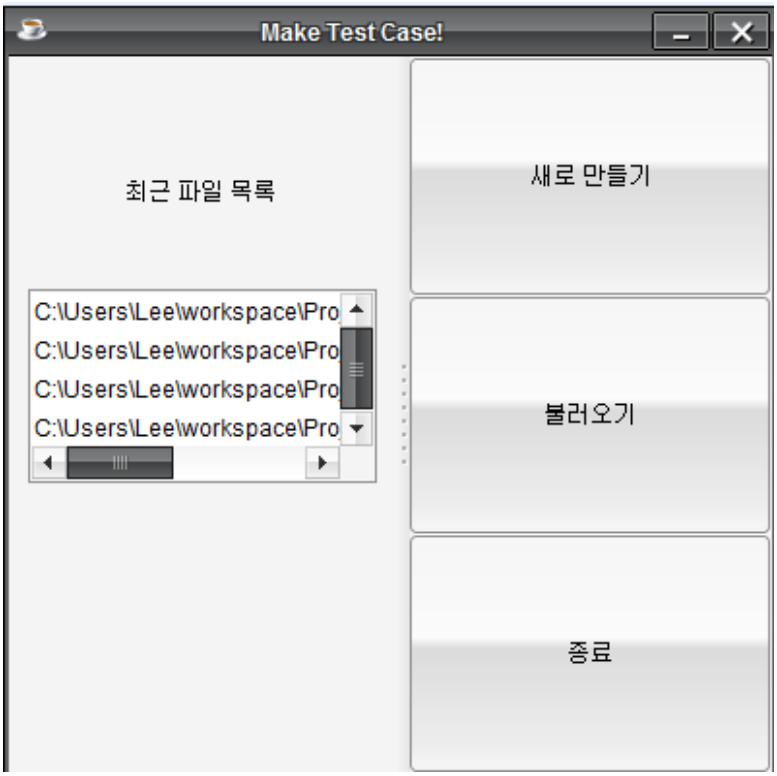
MainSystem.java Specification.java Category.java

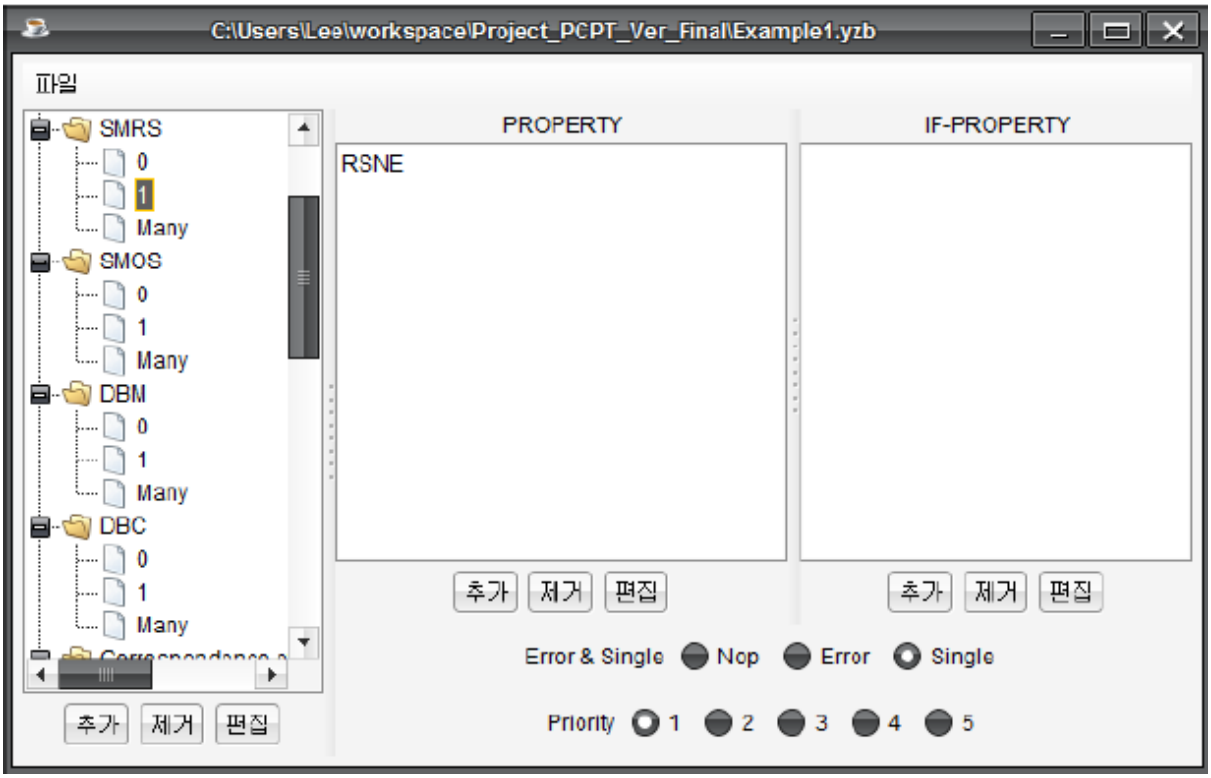
```java
import java.util.ArrayList;

public class Category {
    private String categoryName;
    private ArrayList<RepresentativeValue> representativeValues;
    private Hashtable<String, Property> pt;

    //Constructor
    public Category(){
        this.categoryName = new String();
        this.representativeValues = new ArrayList<RepresentativeValue>();
    }
    public Category(String categoryName, Hashtable<String, Property> pt){
        this.categoryName = new String(categoryName);
        this.representativeValues = new ArrayList<RepresentativeValue>();
        this.pt = pt;
    }
    //number of representativeValues
    public int getNumberOfRepresentativeValues(){
        return representativeValues.size();
    }


    //methods
    public int setCategoryName(String categoryName){
        this.categoryName = categoryName;
        return 0;
    }
    public String getCategoryName(){
        return this.categoryName;
    }
    public RepresentativeValue getRepresentativeValue(int representativeValueIndex){
        return this.representativeValues.get(representativeValueIndex);
```


MainSystem.java
Specification.java
Category.java
FileManagerTest.java
JUnit
Finished after 5.61 seconds
Runs: 9/9
Errors: 0
Failures: 0
test.FileManagerTest [Runner: JUnit 4] (5.523 s)
Failure Trace
testUpdateRecentList (0.000 s)
testSaveSpecificationString1 (0.005 s)
testSaveSpecificationString2 (1.760 s)
testGetPath (0.000 s)
testNewSpecification1 (0.013 s)
testNewSpecification2 (2.947 s)
testSaveSpecification (0.003 s)
testLoadSpecification1 (0.009 s)
testLoadSpecification2 (0.786 s)
package test;
import static org.junit.Assert.*;
public class FileManagerTest {
private static Specification s;
private static FileManager f;
private static File file;
@BeforeClass
public static void beforeSetup(){
s = new Specification();
f = new FileManager(s);
}
@Test //생성 잘 되는지 테스트
public void testNewSpecification1() {
file = new File("newTest.yzb");
if(file.exists()){
file.delete();
}
assertEquals(0,f.newSpecification("newTest"));
assertTrue(file.exists());
}
@Test //test1에서 생성한 파일명과 같으니까 들어보는 창이 뜰 것이다. 예를 누르면 0, 아니오를 누르면 1이 반환도
public void testNewSpecification2(){
assertEquals(0,f.newSpecification(file.getAbsolutePath().substr
assertEquals(1,f.newSpecification(file.getAbsolutePath().substr
}
@Test //존재하는 파일은 잘 불러와야된다. 그러면 0을 리턴.
public void testLoadSpecification1() {

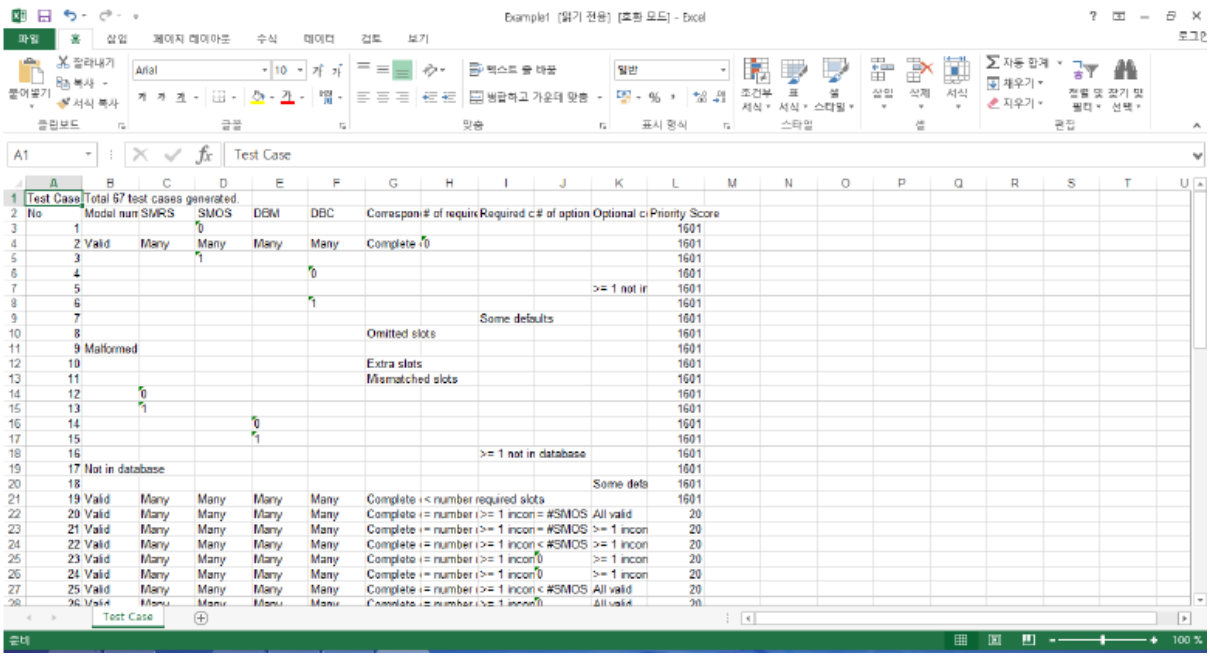

SpecificationTest.java
JUnit
Finished after 4.372 seconds
Runs: 5/5
Errors: 0
Failures: 0
test.SpecificationTest [Runner: JUnit 4] (3.942 s)
Failure Trace
testAddCategory (2.093 s)
testGetNumberOfCategories (0.000 s)
testExport2excel (1.020 s)
testGenerateTestCase (0.002 s)
testSetCategory (0.827 s)
package test;
import static org.junit.Assert.*;
public class SpecificationTest {
@Test //정상적으로 갯수 반환하는지
public void testGetNumberOfCategories() {
Specification s = new Specification();
s.addCategory("a");
s.addCategory("b");
s.addCategory("c");
assertEquals(3,s.getNumberOfCategories());
}
@Test //테스트 케이스 생성이 정확하게 되는지.
public void testGenerateTestCase() {
Specification s = new Specification();
s.addCategory("a");
s.addCategory("b");
s.getCategory(0).addRepresentativeValue("a1");
s.getCategory(0).addRepresentativeValue("a2");
s.getCategory(0).addRepresentativeValue("a3");
s.getCategory(0).getRepresentativeValue(0).addProperty("a1");
s.getCategory(1).addRepresentativeValue("b1");
s.getCategory(1).addRepresentativeValue("b2");
s.getCategory(1).addRepresentativeValue("b3");
s.getCategory(1).addRepresentativeValue("b4");
s.getCategory(1).getRepresentativeValue(1).addIfProperty("a1");

5. Project Output





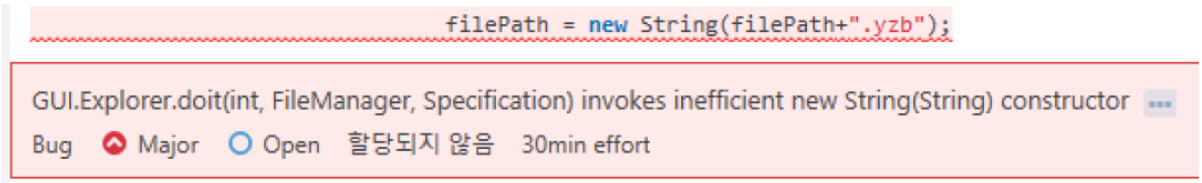

C:\Users\Lee\workspace\Project_PCPT_Ver_Final\Example1.yzb
파일
SMRS
0
1
Many
SMOS
0
1
Many
DBM
0
1
Many
DBC
0
1
Many
추가
저거
편집
PROPERTY
RSNE
IF-PROPERTY
메시지
총 67개의 testCase가 생성되었습니다!
확인
추가
저거
편집
추가
저거
편집
Error & Single
Nop
Error
Single
Priority
1
2
3
4
5


Example1 [읽기 전용] [호환 모드] - Excel
Test Case
Test Case Total 67 test cases generated.
No Model num SMRS SMOS DBM DBC Correspon # of requir Required c # of option Optional c Priority Score
Test Case

6. Static Analysis FeedBack

6.1 T1's Analysis

- Sonar Qube



T1에서 Sonar Qube를 통해 잡아낸 Bug를 보니 거의 대부분 String을 할당할 때, new를 통해 String 객체를 생성해서 할당하는 방식에 문제가 있음을 확인하였으며, 이를 후에 반영한다고 하면, new String을 이용하지 말고 +연산자나 concat 메소드를 이용하여 연결하고, 할당도 그냥 "example" 이런 식으로 할당할 것이다.

- Metrics

Metric	Total	Mean	Std. Dev.	Maxim...	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per method)		2.516	3.185	20	/T4_source/src/System/FileManager.java	loadSpecification
Number of Parameters (avg/max per method)		0.731	0.985	5	/T4_source/src/System/Specification.java	f
Nested Block Depth (avg/max per method)		1.946	1.498	8	/T4_source/src/System/FileManager.java	loadSpecification
Afferent Coupling (avg/max per packageFragment)		3	3.559	8	/T4_source/src/System	

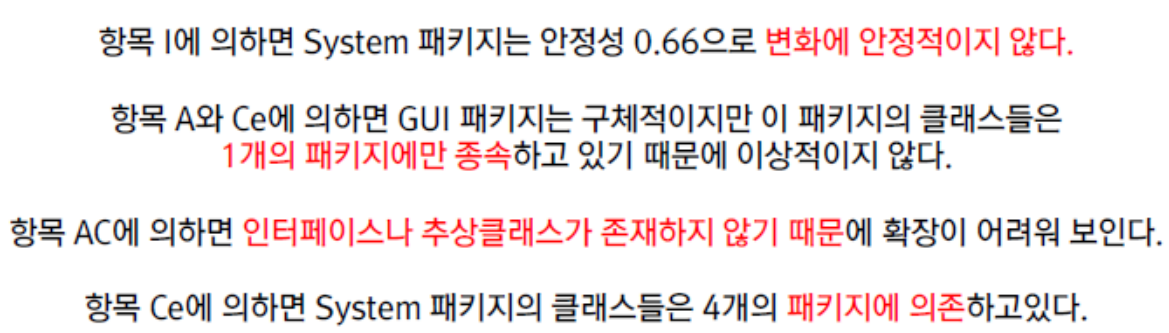
Metrics에서의 Cyclomatic Complexity를 살펴보니 FileManager의 loadSpecification에서 복잡도가 높게 나왔다. 이를 알아보기 위해 코드를 살펴보니

```
for(int i=0;i<sub.length;i++){
    if(i==0){
        categories = sub[i].split("/");
        for(int j=0;j<categories.length;j++){
            spec.addCategory(categories[j]);
        }
    }
    else{
        values = sub[i].split("/");

        for(int j=0;j<values.length;j++){
            temp = values[j].split("\f");
            spec.getCategory(i-1).addRepresentativeValue(temp[0]);
            if(temp[1].length()!=0){
                properties = temp[1].split(",");
                for(int k=0;k<properties.length;k++){
                    spec.getCategory(i-1).getRepresentativeValue(j).addProperty(properties[k]);
                }
            }
            ifPropertybuff.append(i-1+""+j+":"+temp[2]+"/");
            spec.getCategory(i-1).getRepresentativeValue(j).setSingleError(Integer.parseInt(temp[3]));
            spec.getCategory(i-1).getRepresentativeValue(j).setPriorityRank(Integer.parseInt(temp[4]));
        }
    }
}

if(ifPropertybuff.toString().length()!=0){
    ifPropertysemibuff = ifPropertybuff.toString().split("/");
    for(int i=0;i<ifPropertysemibuff.length;i++){
        ifPropertyqbuff = ifPropertysemibuff[i].split(":");
        int cNumber = Integer.parseInt(ifPropertyqbuff[0].substring(0, 1));
        int rNumber = Integer.parseInt(ifPropertyqbuff[0].substring(1));
        if(ifPropertyqbuff.length!=1){
            ifProperties = ifPropertyqbuff[1].split(",");
            for(int j=0;j<ifProperties.length;j++){
                spec.getCategory(cNumber).getRepresentativeValue(rNumber).addIfProperty(ifProperties[j]);
            }
        }
    }
}
```

저장되어 있는 명세 파일을 불러오는 과정에서 각 줄마다 구분자를 추출하여 토큰화 하는 과정에서 if문과 for문이 여러 번 수행되면서 복잡도를 증가시킨 것 같다. 이를 해결하려면 저장 방식을 개선하거나, 아예 데이터베이스를 쓰는 것도 나쁘지 않을 것 같다.

6.2 T2's Analysis

- JDepend



이 분석에 대한 답변을 하자면 우선 현재의 프로젝트는 규모가 매우 크거나 클래스가 여럿 생성되지 않기 때문에 굳이 인터페이스와 추상클래스를 사용하지 않아도 충분히 효율적으로 구성이 가능하고, 오히려 이러한 추가가 더 성능을 저해할 우려가 있다. 또한 System 패키지의 클래스들이 4개의 패키지에 의존하고 있다고 검출이 되었는데, 코드를 다시 살펴본 결과 이는 자기자신의 패키지도 수에 추가된다고 한다면 이것이 하나, Main메소드에서 GUI의 시작 프레임을 불러오기 위해 GUI패키지를 참조한 것이 둘, 테스트 케이스의 Excel 출력을 위한 외부 라이브러리 두 개가 된다. 이는 외부 라이브러리 두 개는 필히 필요한 기능이며 이를 내부적으로 끌어들일 수 없기 때문에 줄일 수 없고, System과 View를 나눈다면 System에서 View를 참조할 수 밖에 없기 때문에 더 줄이기는 힘들 것 같다.

- 그 외

Find Bug나 Check Style은 그렇게 큰 문제가 발견되지 않은 것 같고, Code Review에서 문서와 실제 코드의 불일치라고 지적해준 사항은 문서가 최신화 되기 전에 코드리뷰를 수행하였던 문제가 우선적으로 존재하고, 코드상에 존재하는 변수가 플러그인에 캐치가 안된 경우도 존재하는 것으로 확인되었다. 이 부분은 문서의 최신화로 해결이 되었다고 본다.

7. Epilogue

이전에도 이정도 규모의 프로그램을 만드는 팀 과제는 많이 해보았었지만 이번처럼 문서화 작업까지 단계적으로 거치면서 작업을 수행한 적은 없었다. 이번에 단계적으로 문서를 작성하고, 그에 따라 다음 단계를 수행하면서 느낀 점은 크게 두 가지가 있는데,

거의 장점과 단점의 느낌이다. 먼저 느낀 장점으로는, 단계적으로 문서를 남겨가면서 작업을 수행하게 되니, 막연하게 시작하는 프로젝트가 아니라 차근차근 쌓아가는 느낌으로 다음 단계에서 무엇을 해야 하는지가 명확했고, 진행하면서 이전단계에서 수행하였던 것의 자료가 필요하거나 할 때 즉각적으로 가져다가 반영시킬 수 있었으며, 차근차근 밑그림이 그려지다보니, 실제 구현과정에서 불과 일주일의 시간만이 주어졌고, 게다가 실질적으로 코딩 작업을 수행한 일수는 1박2일 밖에 되지 않음에도 불구하고, 기반이 탄탄한 제대로 작동하는 프로그램이 구현이 됨을 보고 이러한 기반 작업의 중요성을 새삼 깨닫게 되었다.
앞서 말한 이러한 장점들을 많이 느꼈지만, 그에 못지 않게 단점 또한 많이 느꼈는데, 일단 문서화 작업에서 중복되는 작업이 많다 보니 이러한 과정을 거치지 않고 바로 코딩을 시작하면 더 빠르게 수행될 작업들도 한참의 시간이 걸렸다. 특히 후반 단계에서 수정을 하게 되면, 앞선 단계 모두를 수정해야 하는 번거로움은, 이전까지 느껴보지 못하였던 고통이었다. 문서를 작성하기 바쁘다 보니 정작 코드 쪽에는 버그 등을 잡는데 오히려 더욱 소홀해 진 것이 아닌가 하는 회의감도 같이 들었다. 특히 프로젝트의 규모가 크지 않았기에 중간에 굳이 필요하지 않을 과정이나, 효용성이 의심되는 과정도 많았다(특히 후반의 정적 분석 과정에서 더욱 절정을 찍은 것 같다).
종합적으로 장단점이 고루 분포 된 것 같은데, 그만큼 힘들지만 필요한 과정이라는 의미가 아닐까 싶다. 확실히 이러한 경험을 해두면 나중에 공동적으로 큰 프로젝트를 수행해야 할 때에 도움이 될 것이란 것 만은 확실한 것 같다.