Feesual CPT Tool

Final Report

Project Team
T8

Date
2017-06-08

T8 Team Information
201211347 박성근
201211376 임제현
201411270 김태홍

Table of Contents

1. Static Analysis 대응

1.1. Level 1 PMD

FeedbackController.java

P	Line	created	Rule	Error Message
▶	34	Wed May 31 19:30:10 KST 2017	CollapsibleIfStatements	These nested if statements could be combined
▶	42	Wed May 31 19:30:10 KST 2017	CollapsibleIfStatements	These nested if statements could be combined
▶	29	Wed May 31 19:30:10 KST 2017	CollapsibleIfStatements	These nested if statements could be combined
▶	55	Wed May 31 19:30:10 KST 2017	CollapsibleIfStatements	These nested if statements could be combined
▶	29	Wed May 31 19:30:10 KST 2017	SimplifyBooleanExpressions	Avoid unnecessary comparisons in boolean expressions
▶	34	Wed May 31 19:30:10 KST 2017	SimplifyBooleanExpressions	Avoid unnecessary comparisons in boolean expressions

*CollapseibleIfStatements(합칠 수 있는 if문 지적)

➔ 모두 이런 식으로 되어있는 부분을 지적하였다.

```
if(table.getTable()[i][4]!=null){
    if(table.getTable()[i][4].equals(property)){
        f=1;
    }
}
```

&&를 사용하여 if문을 줄일 수 있음을 확인하였다.

*SimplifyBooleanExpressions(축약 가능한 Boolean형 지적)

➔ 개발자의 습관을 확인.

```
if(table.getTable()[i][6]!=null){
    if(table.getTable()[i][6].equals("single")==false){
        return feedbackList[2];
    }
}
```

Table.java

P	Line	created	Rule	Error Message
▶	46	Wed May 31 18:53:48 KST 2017	SimplifyBooleanExpressions	Avoid unnecessary comparisons in boolean expressions
▶	50	Wed May 31 18:53:48 KST 2017	SimplifyBooleanExpressions	Avoid unnecessary comparisons in boolean expressions
▶	1	Wed May 31 18:53:48 KST 2017	UnusedImports	Avoid unused imports such as 'javax.swing.JTable'

*UnusedImports(사용하지 않는 import로 지적)

➔ 실제로 Table.java에서 JTable을 사용하지 않았기에 삭제할 수 있음을 확인하였다.

*SimplifyBooleanExpressions(축약 가능한 Boolean형 지적)

➔ 개발자의 습관을 확인.

TestCaseController.java

P	Line	created	Rule	Error Message
▶	3	Wed May 31 18:53:48 KST 2017	UnusedImports	Avoid unused imports such as 'java.util.ArrayList'

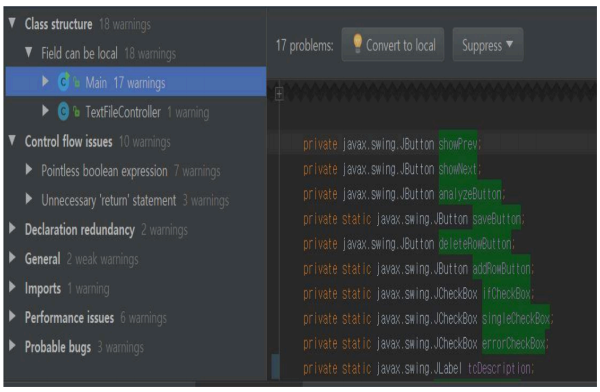
*UnusedImports(사용하지 않는 import로 지적)

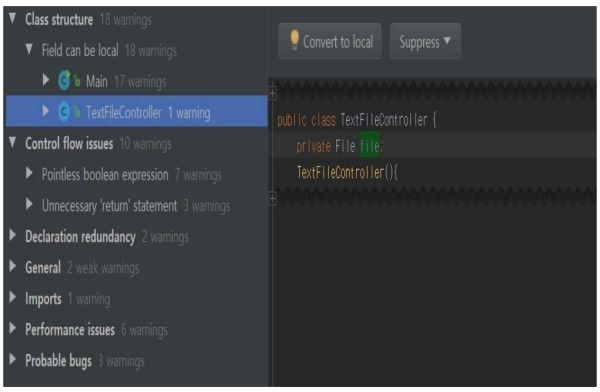
➔ TestCaseController.java에서 사용하므로 import해야 한다.

```
public class TestCaseController {
   private ArrayList<TestCase> tcList;
   private ArrayList<TestCase> propertySub;
   private ArrayList<TestCase> nonPropertySub;
   private ArrayList<TestCase> propertyTc;
   private ArrayList<TestCase> nonPropertyTc;
```

 개발자의 코딩 습관을 확인할 수 있었고, 좀 더 코드가 군더더기 없이 간결하게 발전할 수 있음을 확인 하였다.

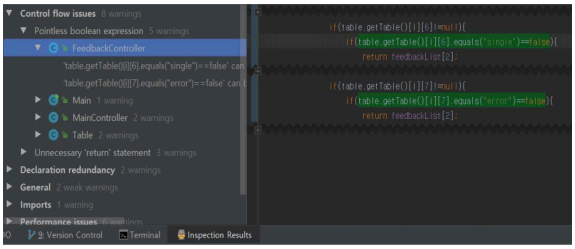
1.2. Level 1 IntelliJ

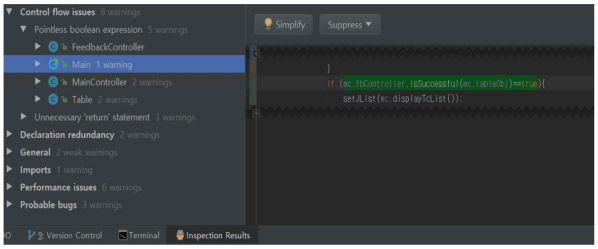


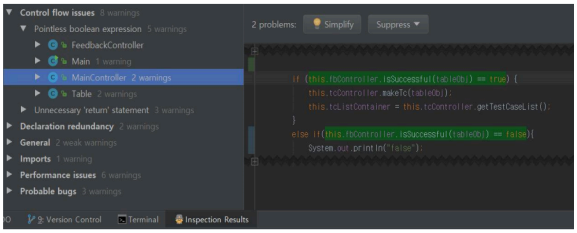


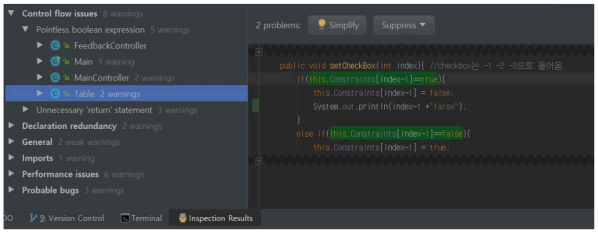
변수가 하나의 메소드에서만 사용 (지역변수 추천)

➔ 지적받은 부분은 수정 가능한 부분임을 확인하였다.



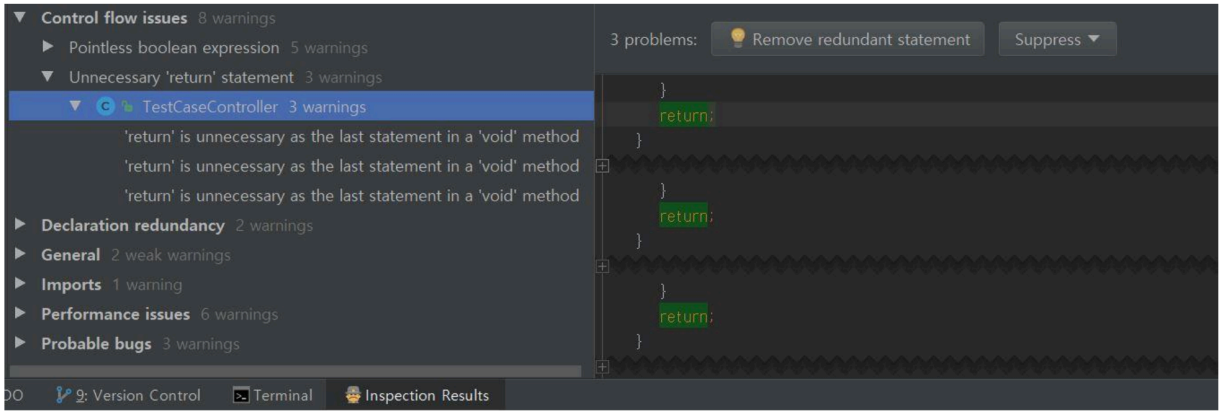






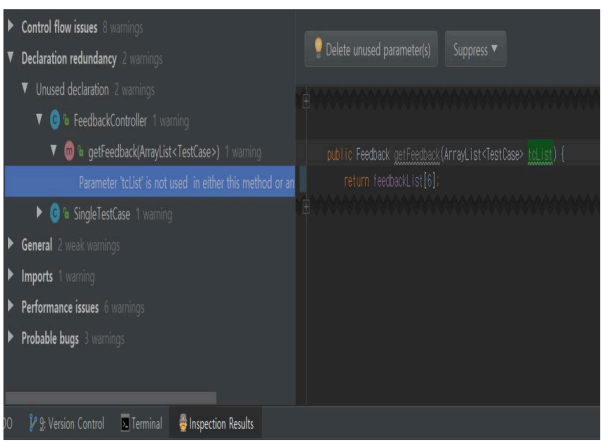
'== true / false' 사용 (단순히 expression 추천)

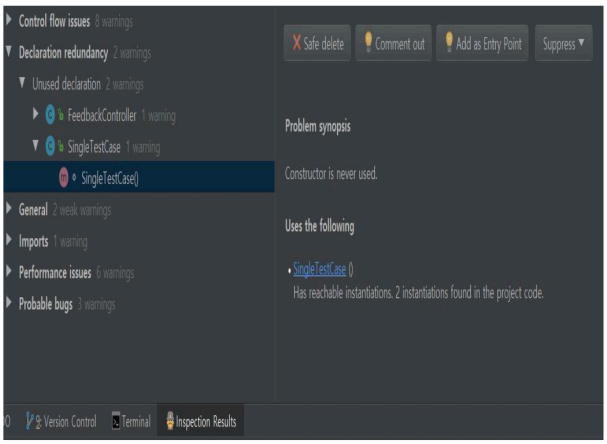
➔ 개발자의 습관임을 확인하였다.



불필요한 return문 사용

➔ 끝을 명시적으로 표현하고 싶어서 사용했다. 고칠 필요는 없다고 생각하였다.

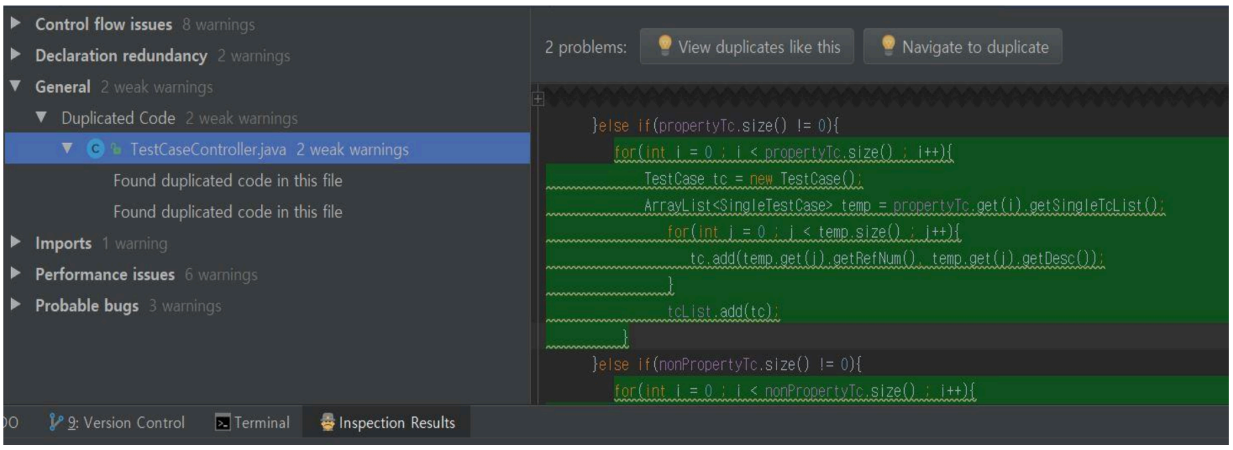




메소드 / 생성자 사용 x

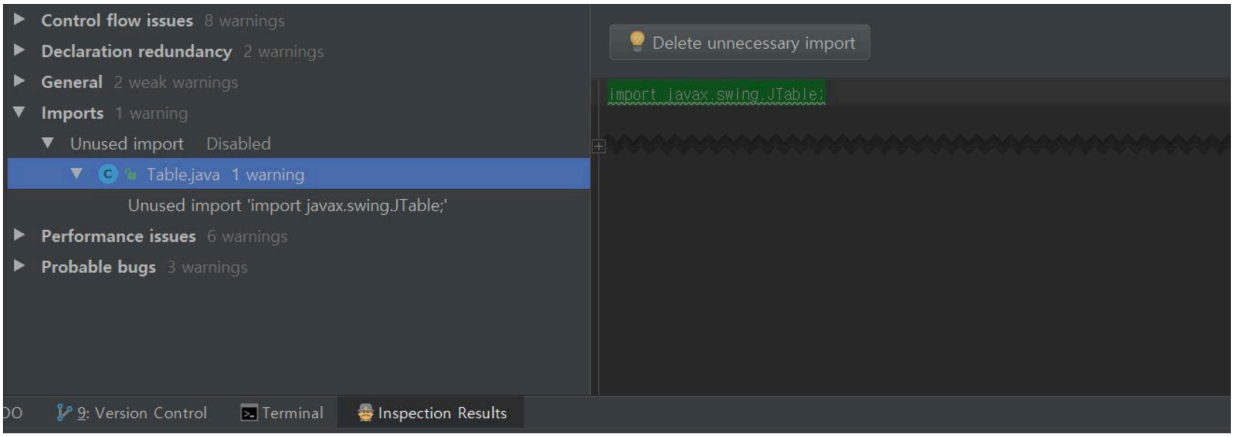
➔ 왼쪽의 경우는 개발 중 확인 용도로 생성된 메소드이며 사용하지 않기 때문에 삭제가 필요함을 확인하였다.

오른쪽의 경우는 개발자의 습관으로 인한 발생한 생성자이며 사용하지 않기 때문에 삭제가 필요함을 확인하였다.



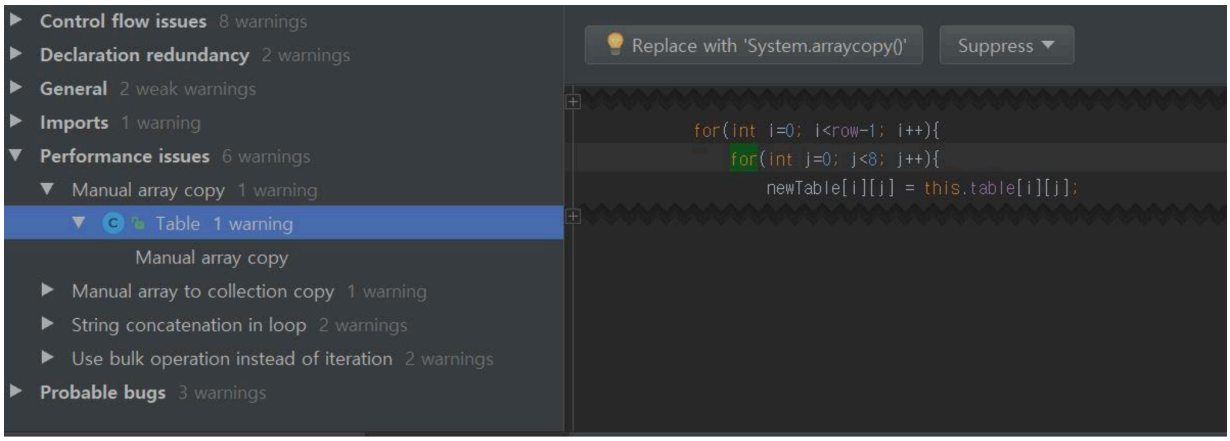
동일한 코드 중복

➔ 코드가 중복된다고 했는데 nonPropertyTc을 사용하냐 propertyTc를 사용하냐 이것만 차이나기 때문에 그렇다고 생각하였다. 코드가 중첩되어도 사용하는 ArrayList가 다르므로 분리해줘야한다고 생각해서 수정하지 않았다.



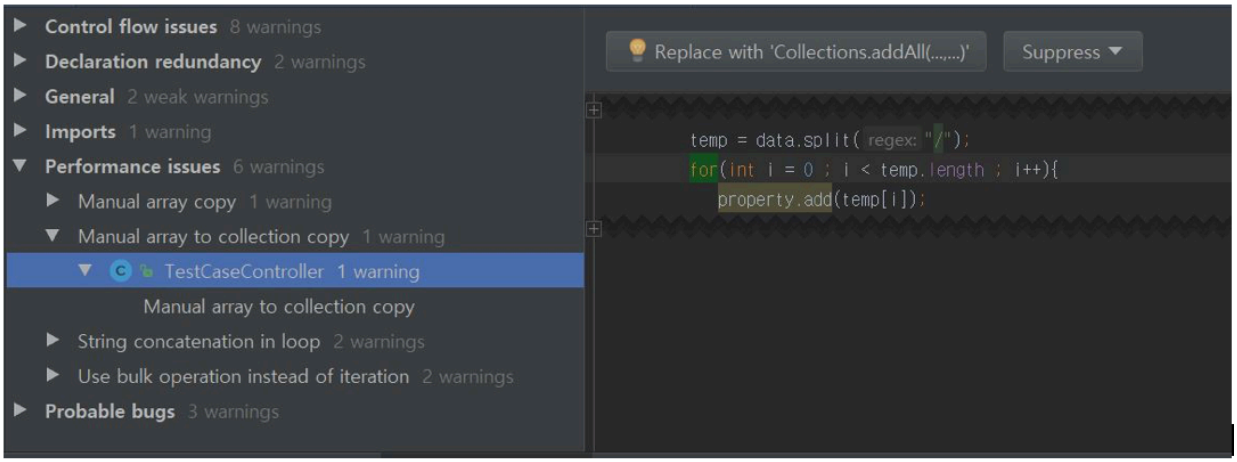
Import한 내용 사용 x

➔ 실제로 사용하지 않기 때문에 삭제가 필요함을 확인하였다.



단순 array 복사 (arraycopy 추천)

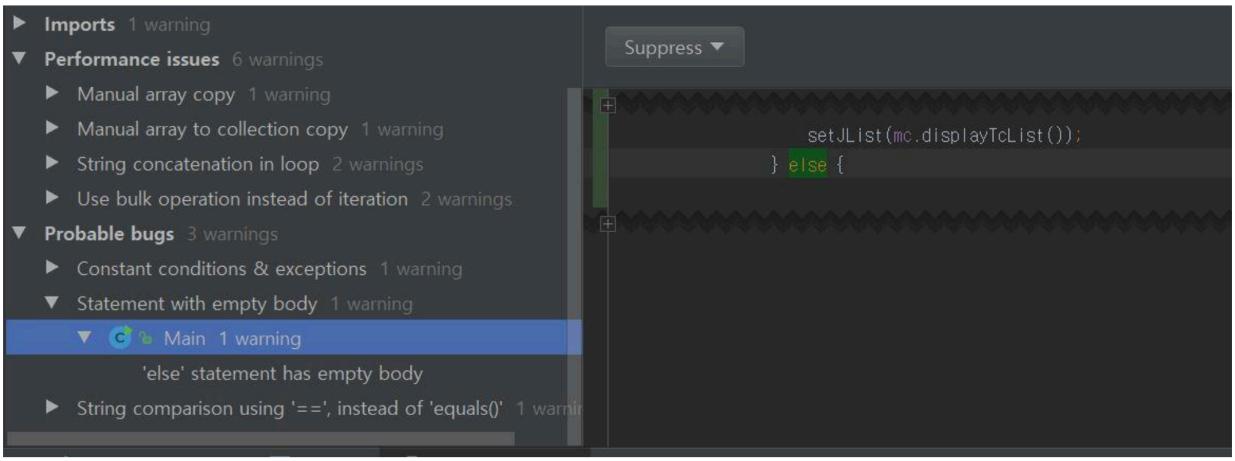
➔ arraycopy의 존재를 알려주었다.




```
public void saveProperty(ArrayList<String> property, String data){
   if(data.contains("/")){
      String[] temp;
      temp = data.split("/");
      for(int i = 0 ; i < temp.length ; i++){
         property.add(temp[i]);
      }
   }else{
      property.add(data);
   }
   return;
}
```

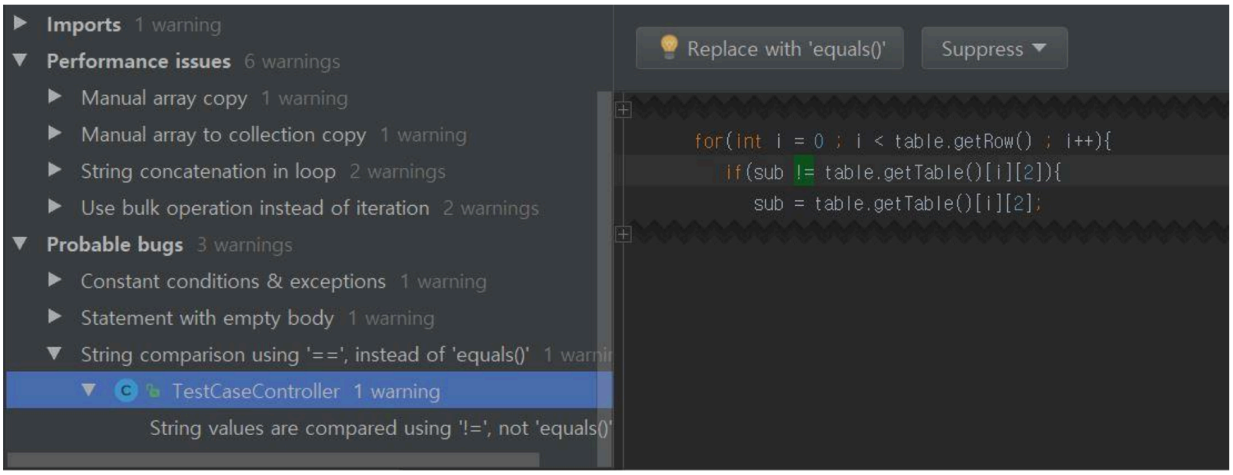
➔ 배열을 통째로 array에 넣는데는 collection.addall 메소드를 추천하는데 굳이 바꿔야할

필요성을 느끼지 못하였다.



else문 body가 비어있음

➔ 실제 코드에서 삭제가 필요함을 확인하였다.

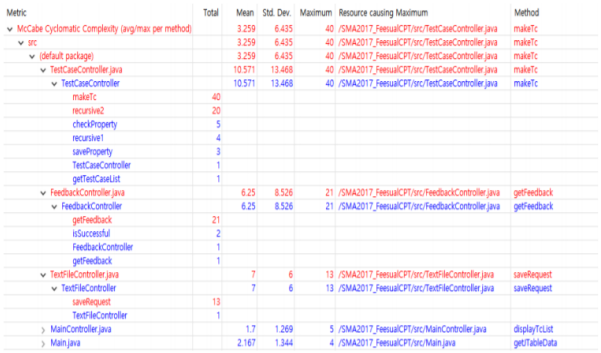


Stirng 비교문에 != / == 사용 (equals()함수 사용)

➔ 수정이 필요함을 확인하였다.

 개발자의 코딩 습관을 확인할 수 있었다. 치명적으로 작용할 수 있는 에러 또한 확인할 수 있었다. 그리고 코드가 좀 더 군더더기 없이 간결하게 발전할 수 있음을 확인 하였다.

1.3. Level 2 Eclipse Metrics Plugin

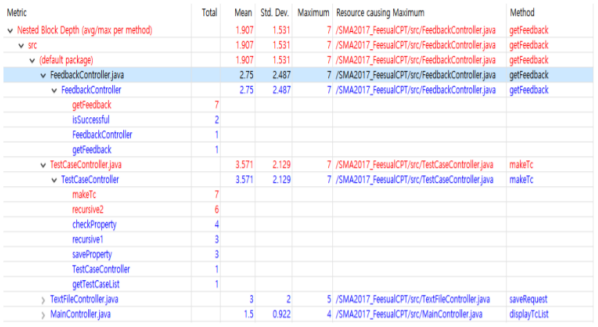


Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
McCabe Cyclomatic Complexity (avg/max per method)		3.259	6.435	40	/SMA2017_FeesualCPT/src/TestCaseController.java	makeTc
src		3.259	6.435	40	/SMA2017_FeesualCPT/src/TestCaseController.java	makeTc
(default package)		3.259	6.435	40	/SMA2017_FeesualCPT/src/TestCaseController.java	makeTc
TestCaseController.java		10.571	13.468	40	/SMA2017_FeesualCPT/src/TestCaseController.java	makeTc
TestCaseController		10.571	13.468	40	/SMA2017_FeesualCPT/src/TestCaseController.java	makeTc
makeTc	40					
recursive2	20					
checkProperty	5					
recursive1	4					
saveProperty	3					
TestCaseController	1					
getTestCaseList	1					
FeedbackController.java		6.25	8.526	21	/SMA2017_FeesualCPT/src/FeedbackController.java	getFeedback
FeedbackController		6.25	8.526	21	/SMA2017_FeesualCPT/src/FeedbackController.java	getFeedback
getFeedback	21					
isSuccessful	2					
FeedbackController	1					
getFeedback	1					
TextFileController.java		7	6	13	/SMA2017_FeesualCPT/src/TextFileController.java	saveRequest
TextFileController		7	6	13	/SMA2017_FeesualCPT/src/TextFileController.java	saveRequest
saveRequest	13					
TextFileController	1					
MainController.java		1.7	1.269	5	/SMA2017_FeesualCPT/src/MainController.java	displayTcList
Main.java		2.167	1.344	4	/SMA2017_FeesualCPT/src/Main.java	getTableData

TestCaseController.makeTc 메소드
TestCaseController.recursive2 메소드
FeedbackController.getFeedback 메소드
TextFileController.saveRequest 메소드

Cyclomatic Complexity가 높게 나옴

➔ TestCase를 생성함에 있어서 table을 탐색하고 모든 Testcase를 돌기때문에 다중for문과 재귀함수가 필요하다고 생각했다. 또 계산하는데 조건들이 많이 필요해서 조건문도 많이 사용하였다. 그러다 보니 Complexity가 높았다고 생각하였다.



Metric	Total	Mean	Std. Dev.	Maximum	Resource causing Maximum	Method
Nested Block Depth (avg/max per method)		1.907	1.531	7	/SMA2017_FeesualCPT/src/FeedbackController.java	getFeedback
src		1.907	1.531	7	/SMA2017_FeesualCPT/src/FeedbackController.java	getFeedback
(default package)		1.907	1.531	7	/SMA2017_FeesualCPT/src/FeedbackController.java	getFeedback
FeedbackController.java		2.75	2.487	7	/SMA2017_FeesualCPT/src/FeedbackController.java	getFeedback
FeedbackController		2.75	2.487	7	/SMA2017_FeesualCPT/src/FeedbackController.java	getFeedback
getFeedback	7					
isSuccessful	2					
FeedbackController	1					
getFeedback	1					
TestCaseController.java		3.571	2.129	7	/SMA2017_FeesualCPT/src/TestCaseController.java	makeTc
TestCaseController		3.571	2.129	7	/SMA2017_FeesualCPT/src/TestCaseController.java	makeTc
makeTc	7					
recursive2	6					
checkProperty	4					
recursive1	3					
saveProperty	3					
TestCaseController	1					
getTestCaseList	1					
TextFileController.java		3	2	5	/SMA2017_FeesualCPT/src/TextFileController.java	saveRequest
MainController.java		1.5	0.922	4	/SMA2017_FeesualCPT/src/MainController.java	displayTcList

FeedbackController.getFeedback 메소드
TestCaseController.makeTc 메소드
TestCaseController.recursive2 메소드

중첩된 Block이 많음

➔ 여러 조건에 따라 수행되는 결과가 달라야 했기 때문에, if문 안에 if문, 또는 for문이 들어가는 경우가 많았다. 이 때문에 중첩된 Block이 많다고 생각하였다.

 작성된 코드의 Cyclomatic Complexity와 {}의 중첩도를 확인할 수 있었고, 우리 팀이 작성한 코드가 얼마나 복잡한지 확인할 수 있었다. 하지만 최선을 다해 짜낸 알고리즘이었기 때문에 현재 코드 이상으로 개선할 수는 없다고 생각하였다.

1.4. Level 2 JDepend

Package	CC(concr.cl.)	AC(abstr.cl.)	Ca(aff.)	Ce(eff.)	A	I	D	Cycle!
Default	25	0	0	1	0.00	1.00	0.00	
org.junit	0	0	1	0	0.00	0.00	1.00	

```
JDepend
----------------------------
- Package: Default
----------------------------

Stats:
    Total Classes: 25
    Concrete Classes: 25
    Abstract Classes: 0

    Ca: 0
    Ce: 1

    A: 0
    I: 1
    D: 0

Depends Upon:
    org.junit

Used By:
    Not used by any packages.
```

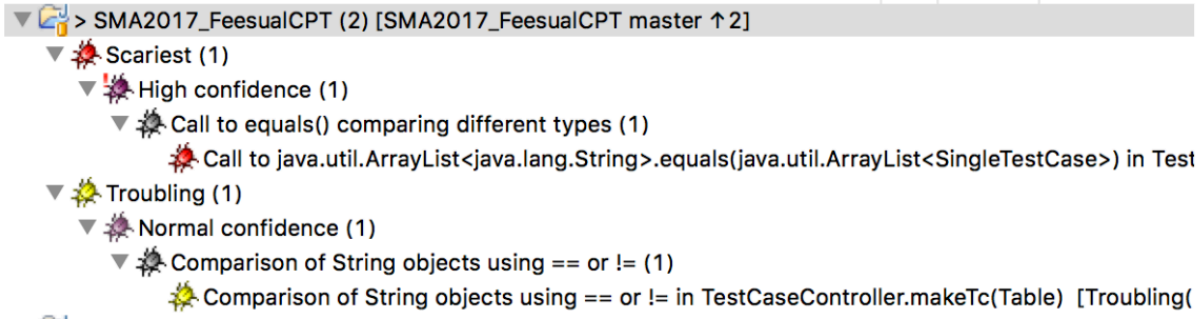
안정적이면서 추상화가 덜 되었음.

➔ 패키지가 1개 밖에 없어서 안정적이며 추상화가 덜 되었다고 평가받은 것 같다. OOPT 2040을 수행할 때, Feesual CPT Tool은 1개의 패키지로 충분하다고 생각했기 때문에 당연한 결과이다.

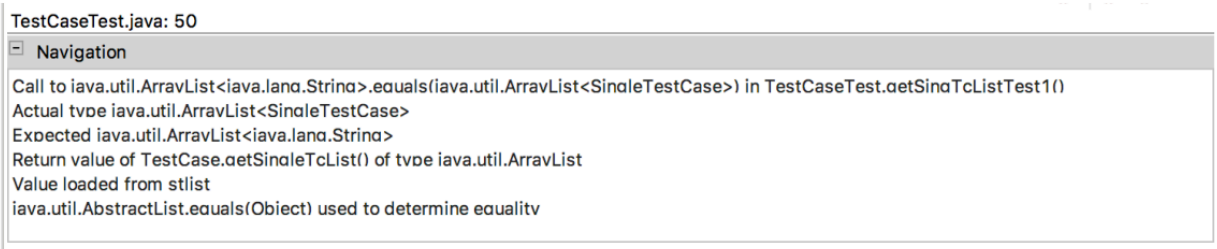
1.5. Level 3 Findbugs

8조 분석결과

- **FindBug 를 사용하여 분석한 결과, 총 2 개의 warning 이 발견 되었다.**



Scariest, High 등급 warning



Bug: Call to java.util.ArrayList<java.lang.String>.equals(java.util.ArrayList<SingleTestCase>) in TestCaseTest.getSingTcListTest1()

This method calls equals(Object) on two references of different class types and analysis suggests they will be to objects of different classes at runtime. Further, examination of the equals methods that would be invoked suggest that either this call will always return false, or else the equals method is not be symmetric (which is a property required by the contract for equals in class Object).

Rank: Scariest (1), **confidence**: High
Pattern: EC_UNRELATED_TYPES
Type: EC, **Category**: CORRECTNESS (Correctness)

Troubling, Normal등급 warning

Bug: Comparison of String objects using == or != in TestCaseController.makeTc(Table)

This code compares `java.lang.String` objects for reference equality using the == or != operators. Unless both strings are either constants in a source file, or have been interned using the `String.intern()` method, the same string value may be represented by two different String objects. Consider using the `equals(Object)` method instead.

Rank: Troubling (11), **confidence**: Normal
Pattern: ES_COMPARING_STRINGS_WITH_EQ
Type: ES, **Category**: BAD_PRACTICE (Bad practice)

T8. Feesual CPT

1. String 파라미터에 == , != 연산자 사용

2. 서로 다른 성격의 자료형간 비교

➔ 1번의 경우는 equals로 수정해야함을 확인하였다.

2번의 경우는 유닛테스트를 수행하면서, 고의적으로 에러 상황을 발생시키고 확인하려는 코드였기 때문에 저러한 지적이 나온것 같다. 완성된 코드를 보낼 때, 유닛 테스트 코드는 제외하고 보내야함을 확인하였다.

 실제로 코드가 돌아가면서 발생할 수 있는 치명적인 상황을 지적해주었다. 유닛 테스트 코드에서 고의적으로 발생시킨 에러 코드를 제외하면, 상당히 적은 수의 지적이 있었다고 생각한다. 좀 더 완벽한 코드를 작성하도록 꼼꼼히 확인해야한다.

2. 느낀 점

[OOPT_1000]
시작할 당시에는 가벼운 마음으로 프로젝트를 시작했다.
개개인의 객관적 실력 평가, 시장 조사 및 Use Case 를 작성할 때, 올바르게 하는지 확신은 없었지만 자신감 있게 팀원들과 모든 부분을 일사천리로 진행했다.

[OOPT_2030]
첫번째 발표 이후 Use Case 에 대한 개념이 아예 틀렸다는 것을 알 수 있었다. 교수님과 조교님의 조언을 적극적으로 적용하여 User 와 System 간의 관계를 명확히 하였고 Use Case 를 수정하였다. 2034 에서 Step 별로 생각하여 작성하다보니 시간이 많이 걸렸다. 또, 2035 와 2036 이 중요하다고 하셨기에 이 부분을 작성하면서 많은 시간이 걸렸다.

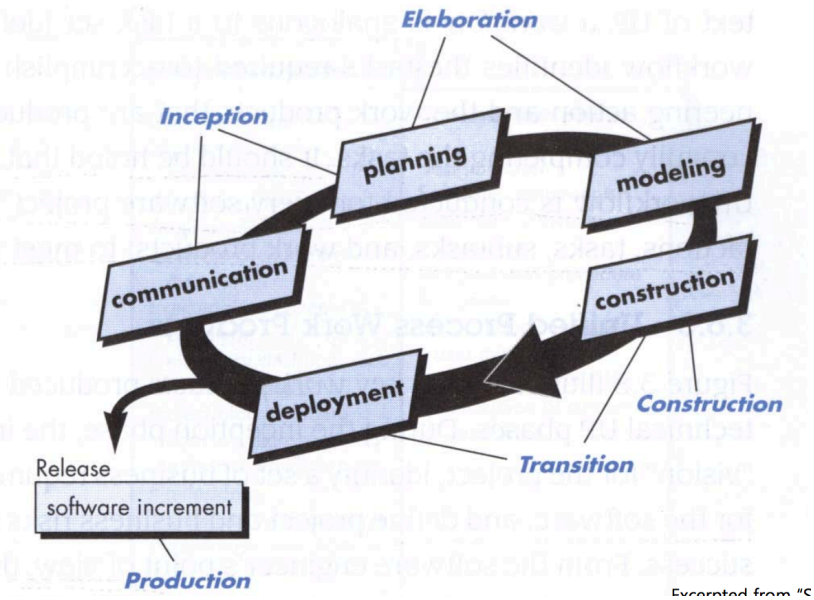
[OOPT_2040]

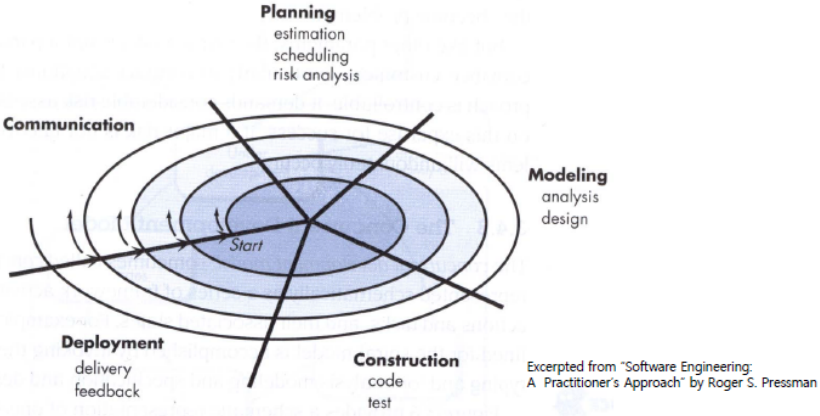
UI 프로토타입을 만들어야 했다. 남자 공학생 세명이서 이쁜 디자인을 구성하기 어려워 제일 간단하고 직관적인 디자인을 뽑아내었다.
이제부터 좀 더 명확한 Use Case 들의 실행 과정 및 Use Case 간의 관계를 정의하였다.
또한 Interaction Diagram 을 하면서 부터 별거 아니라 생각한 프로그램이 되게 복잡하다는 것을 느끼게 되었고 이 때 부터 많은 시간을 소모에 사용해야 했으며 정신적 스트레스와 압박감이 증가하였다.
발표 전날 교수님과의 면담으로 Class Diagram 같은 경우에는 너무 Low Level 인데 Interaction Diagram 이 너무 High Level 이라 중간이 없다는 교수님의 피드백을 받았다. GUI 를 2040 에서는 생각할 필요가 없다고 하셨다.(구현을 생각하여 작성, reverse engineering 라는 느낌을 받았다고 하셨다.) 발표 전날 이것을 반영하여 수정하느라 너무 힘들었다.

[OOPT_2050/2060]
실제로 구현을 하려고 보니 생각했던 Interaction Diagram 과 Class Diagram 이 실제 구현과 약간 다르다는 것을 파악했다.
GUI 가 생김으로써 2040 에서 지적받은 문제 또한 해결하여 수정하였다.
짧은 시간에 프로젝트 개발, 유닛 테스트, 문서 작업, 및 발표 자료를 작성하느라 팀원들과 매우 늦은 시간까지 고생하였다.

[Static Analysis]
Level 1 은 코드 자체의 룰체킹을 하는 등 코드에 있는 불필요한 부분을 알려주었고, level 이 높아질수록 프로그램이 컴파일되고 실행될 때 발생할 수 있는 오류를 알려주었다. 실제로 실행하진 않지만 코드만으로 검사할 수 있는 도구를 사용하고 결과를 확인하니 놀라웠다.
보고서를 작성할 때에는 간만에 너무 쉬운 주제가 나와 너무 가볍게 생각하고 작성한것이 아닌가 싶다. 내용이 부실하고 분석이 부족하다는 피드백을 받았다.

[OOPT]
각 단계를 수행함에 있어서, 완벽하다고 생각했던 부분이 다음 단계로 넘어가면서 부족한 부분들이 보였다. 이전에 했단 단계들로 돌아가 수정하고 보완하면서 점점 완성에 가까워졌다.
이런 과정을 반복하다보니 다음과 같은 그림과 비슷하게 진행되고 있음을 깨달았다.





현재 단계에서 문제가 발생하여 이전의 단계들을 수정하다 보면, 이전 단계에서 생각했던 것들이 조금씩 바뀌고 구조가 변경되었다. 이에 따라 상당히 까다롭다고 느꼈던 문제점들이 의외로 간단히 해결되는 부분도 있었다.

하지만 이전 단계에 있던 작은 부분이 다음 단계로 가면 큰 문제를 야기하는 나비효과가 발생할 수 있기 때문에, 첫 단추부터 신중하게 생각하여 작성해야만 했다. 이에 따라 시간이 많이 부족하여 힘들었다.

이전에 개발할 때 design 단계를 거치지 않고 무작정 개발을 하게되면 문서를 작성하는 시간과 비용이 들지 않았지만, 수정사항이 발생하게 되면 프로젝트를 뒤엎거나 수정하는데 상당한 시간과 비용이 들었다.

이번 강의에서 개발할 때에는 단계 별로 문서를 작성하고 개발을 하다보니, 개발 전 많은 시간과 비용이 들었지만, 실제로 구현할 때에는 별다른 문제없이 작성된 문서만으로 개발을 끝마칠 수 있었다. 오히려 지금까지 해왔던 개발 방식보다 편한 점이 있었다.