2016

소프트웨어 검증

정적 분석 결과 보고서 #1

박수민, 원정일, 조경래

2016년 6월 9일

내용

개요

이 문서는 건국대학교 '소프트웨어 검증' 과목의 팀 프로젝트 4조(조원 박수민, 원정일, 조경래)의 5차 발표를 위하여 작성된 보고서이다.

'소프트웨어 검증' 팀 프로젝트 4조는 '소프트웨어 모델링' 팀 프로젝트 4조(조원 강태준, 김서우, 홍유리)의 프로젝트 산출물 '그놈! Clone Checker (Ver. 2016.06.05, 빌드 번호 #83)'를 대상으로 하여 2016년 6월 2일 ~ 2016년 6월 9일의 기간 중 정적 분석을 진행하였다.

이 문서는 대상 소프트웨어에 대한 정적 분석 결과를 포함하고 있다.

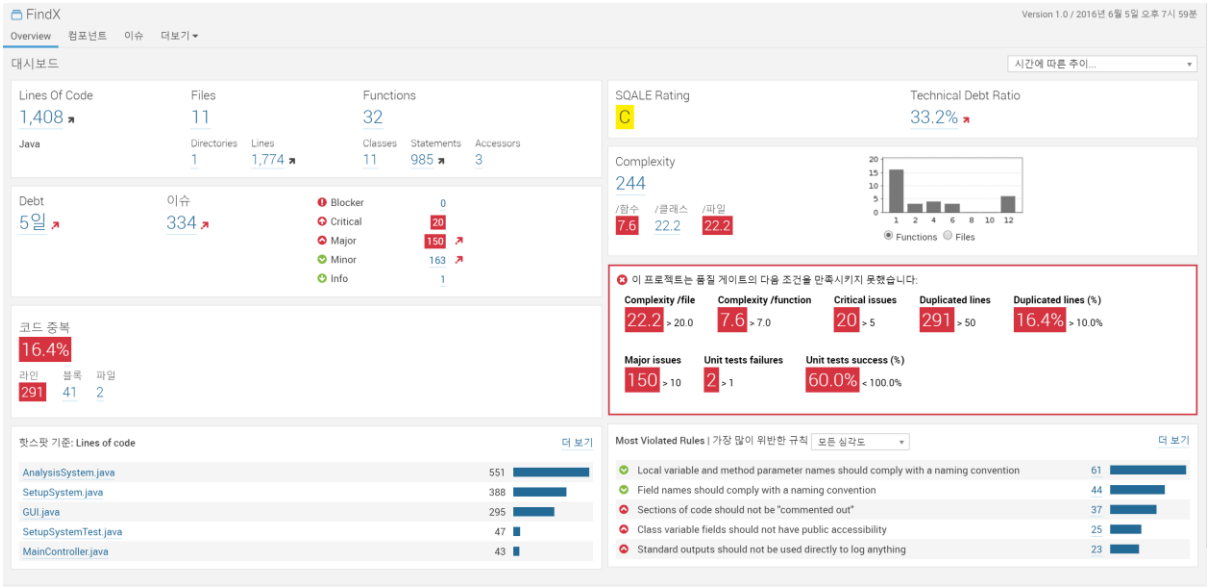
지난 판에서 달라진 점

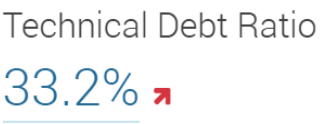
이 문서는 이 보고서의 첫 번째 판이다.

정적 분석

SonarQube

요약





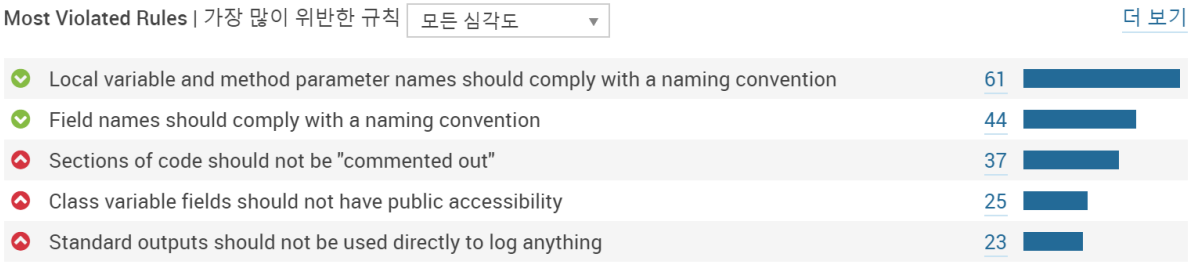
이 프로젝트는 품질 게이트의 다음 조건을 만족시키지 못했습니다:

Complexity /file	Complexity /function	Critical issues	Duplicated lines	Duplicated lines (%)
22.2 > 20.0	7.6 > 7.0	20 > 5	291 > 50	16.4% > 10.0%

Major issues	Unit tests failures	Unit tests success (%)
150 > 10	2 > 1	60.0% < 100.0%

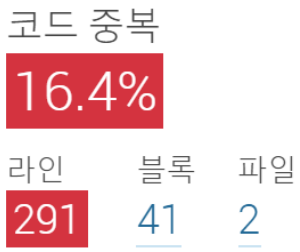
SonarQube를 사용한 정적 분석 결과, 전체 코드에 대한 Technical Debt의 비율이 33.2%로, 종합 판정 결과는 C이다.

분석 결과, 334개의 issue를 발견하였고, 이 중 Major issue의 수는 150개, Critical issue의 수는 20개이다.



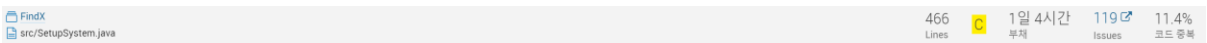
가장 많이 위반한 규칙은 위와 같다.

분석 결과: Duplicated lines



분석 대상 프로젝트의 전체 코드 중복은 16.4%이다. AnalysisSystem.java 파일과 SetupSystem.java에서 중복이 검출되었다.

AnalysisSystem.java 파일의 전체 659 라인 중 238 라인(36.1%)이 중복되어 있음이 검출되었다.



SetupSystem.java 파일의 전체 466 라인 중 52 라인(11.4%)이 중복되어 있음이 검출되었다.

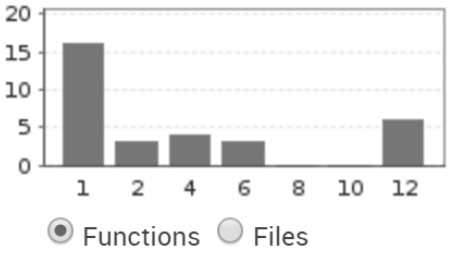
분석 결과: Complexity

Complexity

244

/함수 /클래스 /파일

7.6 22.2 22.2



분석 대상 프로젝트의 전체 Complexity는 244이다.

- AnalysisSystem.java 131
- SetupSystem.java 77
- GUI.java 21
- MainController.java 5
- SetupSystemTest.java 4
- Variable.java 1
- SourceCode.java 1
- Loop.java 1
- Function.java 1
- Conditional.java 1
- AnalysisSystemTest.java 1

분석 결과: Issue

Debt

5일

이슈

334

Blocker 0
Critical 20
Major 150
Minor 163
Info 1

분석 결과, 분석 대상 프로젝트에서 334개의 이슈를 발견하였다. 이 중, Critical 등급의 이슈는 150개, Major 등급의 이슈는 150개이다.

파일	Critical	Major	Minor	Info	등급
Variable.java	1	3	3	0	E
SourceCode.java	1	13	10	0	E
SetupSystemTest.java	1	3	4	0	D
Loop.java	1	0	3	0	D
SetupSystem.java	5	59	55	0	C
MainController.java	1	7	16	0	D

GUI.java	6	28	35	1	D
Function.java	1	5	2	0	E
Conditional.java	1	4	5	0	E
AnalysisSystemTest.java	1	4	1	0	E
AnalysisSystem.java	1	24	29	0	B

CheckStyle

요약

CheckStyle Result

Warnings Trend

All Warnings	New Warnings	Fixed Warnings
1732	0	0

Summary

Total	High Priority	Normal Priority	Low Priority
1732	1732	0	0

Details

Files | **Categories** | Types | Warnings | Details

Category	Total	Distribution
Blocks	15	
Checks	57	
Coding	424	
Design	91	
Imports	13	
Javadoc	91	
Metrics	17	
Naming	120	
Regexp	160	
Sizes	166	
Whitespace	578	
Total	1732	

CheckStyle를 사용한 정적 분석 결과 1732개의 Warning을 발견하였다.

분석 결과: Blocks

Type	Total	Distribution
EmptyBlockCheck	1	
LeftCurlyCheck	1	
NeedBracesCheck	6	
RightCurlyCheck	7	
Total	15	

분석 결과: Checks

Type	Total	Distribution
ArrayTypeStyleCheck	4	
FinalParametersCheck	45	
NewlineAtEndOfFileCheck	8	
Total	57	

분석 결과: Coding

Type	Total	Distribution
HiddenFieldCheck	7	
MagicNumberCheck	415	
MissingSwitchDefaultCheck	1	
SimplifyBooleanExpressionCheck	1	
Total	424	

분석 결과: Design

Type	Total	Distribution
DesignForExtensionCheck	27	
VisibilityModifierCheck	64	
Total	91	

분석 결과: Imports

Type	Total	Distribution
RedundantImportCheck	2	
UnusedImportsCheck	11	
Total	13	

분석 결과: Javadoc

Type	Total	Distribution
JavadocMethodCheck	34	
JavadocPackageCheck	1	
JavadocTypeCheck	11	
JavadocVariableCheck	45	
Total	91	

분석 결과: Metrics

File	Total	Distribution
AnalysisSystem.java	11	
GUI.java	2	
SetupSystem.java	4	
Total	17	

분석 결과: Naming

Type	Total	Distribution
LocalVariableNameCheck	57	
MemberNameCheck	44	
MethodNameCheck	14	
ParameterNameCheck	5	
Total	120	

분석 결과: Regexp

File	Total	Distribution
AnalysisSystem.java	20	
AnalysisSystemTest.java	1	
Conditional.java	1	
Function.java	2	
GUI.java	70	
Loop.java	1	
MainController.java	11	
SetupSystem.java	48	
SourceCode.java	6	
Total	160	

분석 결과: Sizes

Type	Total	Distribution
LineLengthCheck	164	
MethodLengthCheck	2	
Total	166	

분석 결과: Whitespace

Type	Total	Distribution
FileTabCharacterCheck	11	
NoWhitespaceBeforeCheck	10	
ParenPadCheck	13	
WhitespaceAfterCheck	102	
WhitespaceAroundCheck	442	
Total	578	

Findbugs

요약

FindBugs Result

Warnings Trend

All Warnings	New this build	Fixed Warnings
14	0	0

Summary

Total	High Priority	Normal Priority	Low Priority
14	2	12	0

Details

Files | **Categories** | Types | Warnings | Details | High | Normal

Category	Total	Distribution
BAD_PRACTICE	1	
CORRECTNESS	1	
I18N	1	
MALICIOUS_CODE	1	
PERFORMANCE	6	
STYLE	4	
Total	14	

FindBugs를 사용한 정적 분석 결과, 총 14개의 Warning을 발견하였으며, 이 중 Low 등급은 없었으며, Normal 등급은 12개, High 등급은 2개이다.

분석 결과: High

AnalysisSystem.java:454, DM_DEFAULT_ENCODING, Priority: High

Dm: Found reliance on default encoding in AnalysisSystem.make_Detail(int, int, double[][], double): new java.io.FileWriter(String, boolean)

Found a call to a method which will perform a byte to String (or String to byte) conversion, and will assume that the default platform encoding is suitable. This will cause the application behaviour to vary between platforms. Use an alternative API and specify a charset name or Charset object explicitly.

SetupSystem.java:145, RV_RETURN_VALUE_IGNORED, Priority: High

RV: Return value of String.trim() ignored in SetupSystem.start(String, int)

The return value of this method should be checked. One common cause of this warning is to invoke a method on an immutable object, thinking that it updates the object. For example, in the following code fragment,

```
String dateString = getHeaderField(name);
dateString.trim();
```

the programmer seems to be thinking that the trim() method will update the String referenced by dateString. But since Strings are immutable, the trim() function returns a new String value, which is being ignored here. The code should be corrected to:

```
String dateString = getHeaderField(name);
dateString = dateString.trim();
```

분석 결과: Normal

AnalysisSystem.java:85, SF_SWITCH_NO_DEFAULT, Priority: Normal

SF: Switch statement found in AnalysisSystem.analyzeVar(SourceCode, SourceCode) where default case is missing

This method contains a switch statement where default case is missing. Usually you need to provide a default case.

Because the analysis only looks at the generated bytecode, this warning can be incorrect triggered if the default case is at the end of the switch statement and the switch statement doesn't contain break statements for other cases.

AnalysisSystem.java:225, SBSC_USE_STRINGBUFFER_CONCATENATION, Priority: Normal

SBSC: AnalysisSystem.analyzeFunc(SourceCode, SourceCode) concatenates strings using + in a loop

The method seems to be building a String using concatenation in a loop. In each iteration, the String is converted to a StringBuffer/StringBuilder, appended to, and converted back to a String. This can lead to a cost quadratic in the number of iterations, as the growing string is recopied in each iteration.

Better performance can be obtained by using a StringBuffer (or StringBuilder in Java 1.5) explicitly.

For example:

```
// This is bad
String s = "";
for (int i = 0; i < field.length; ++i) {
  s = s + field[i];
}

// This is better
StringBuffer buf = new StringBuffer();
for (int i = 0; i < field.length; ++i) {
  buf.append(field[i]);
}
String s = buf.toString();
```

GUI.java:346, DLS_DEAD_LOCAL_STORE, Priority: Normal

DLS: Dead store to e1 in GUI$6.actionPerformed(ActionEvent)

This instruction assigns a value to a local variable, but the value is not read or used in any subsequent instruction. Often, this indicates an error, because the value computed is never used.

Note that Sun's javac compiler often generates dead stores for final local variables. Because FindBugs is a bytecode-based tool, there is no easy way to eliminate these false positives.

GUI.java:392, DM_EXIT, Priority: Normal

Dm: GUI$7$1.actionPerformed(ActionEvent) invokes System.exit(...), which shuts down the entire virtual machine

Invoking System.exit shuts down the entire Java virtual machine. This should only been done when it is appropriate. Such calls make it hard or impossible for your code to be invoked by other code. Consider throwing a RuntimeException instead.

Loop.java:6, URF_UNREAD_FIELD, Priority: Normal

UrF: Unread field: Loop.loop_type

This field is never read. Consider removing it from the class.

Loop.java:7, URF_UNREAD_FIELD, Priority: Normal

UrF: Unread field: Loop.loop_body

This field is never read. Consider removing it from the class.

MainController.java:13, URF_UNREAD_FIELD, Priority: Normal

UrF: Unread field: MainController.folder_path

This field is never read. Consider removing it from the class.

SetupSystem.java:30, NP_NULL_ON_SOME_PATH_FROM_RETURN_VALUE, Priority: Normal

NP: Possible null pointer dereference in SetupSystem.folder_list(String) due to return value of called method

The return value from a method is dereferenced without a null check, and the return value of that method is one that should generally be checked for null. This may lead to a `NullPointerException` when the code is executed.

SetupSystem.java:83, SBSC_USE_STRINGBUFFER_CONCATENATION, Priority: Normal

SBSC: SetupSystem.setting_files(String, int) concatenates strings using + in a loop

The method seems to be building a String using concatenation in a loop. In each iteration, the String is converted to a StringBuffer/StringBuilder, appended to, and converted back to a String. This can lead to a cost quadratic in the number of iterations, as the growing string is recopied in each iteration.

Better performance can be obtained by using a StringBuffer (or StringBuilder in Java 1.5) explicitly.

For example:

```
// This is bad
String s = "";
for (int i = 0; i < field.length; ++i) {
  s = s + field[i];
}

// This is better
StringBuffer buf = new StringBuffer();
for (int i = 0; i < field.length; ++i) {
  buf.append(field[i]);
}
String s = buf.toString();
```

SetupSystem.java:271, SBSC_USE_STRINGBUFFER_CONCATENATION, Priority: Normal

SBSC: SetupSystem.start(String, int) concatenates strings using + in a loop

The method seems to be building a String using concatenation in a loop. In each iteration, the String is converted to a StringBuffer/StringBuilder, appended to, and converted back to a String. This can lead to a cost quadratic in the number of iterations, as the growing string is recopied in each iteration.

Better performance can be obtained by using a StringBuffer (or StringBuilder in Java 1.5) explicitly.

For example:

```
// This is bad
String s = "";
for (int i = 0; i < field.length; ++i) {
  s = s + field[i];
}

// This is better
StringBuffer buf = new StringBuffer();
for (int i = 0; i < field.length; ++i) {
  buf.append(field[i]);
}
String s = buf.toString();
```

SetupSystem.java:463, EI_EXPOSE_REP, Priority: Normal

EI: SetupSystem.getSc() may expose internal representation by returning sc

Returning a reference to a mutable object value stored in one of the object's fields exposes the internal representation of the object. If instances are accessed by untrusted code, and unchecked changes to the mutable object would compromise security or other important properties, you will need to do something different. Returning a new copy of the object is better approach in many situations.

Variable.java:8, URF_UNREAD_PUBLIC_OR_PROTECTED_FIELD, Priority: Normal

UrF: Unread public/protected field: Variable.count

This field is never read. The field is public or protected, so perhaps it is intended to be used with classes not seen as part of the analysis. If not, consider removing it from the class.

PMD

요약

PMD Result

Warnings Trend

All Warnings	New Warnings	Fixed Warnings
13	0	0

Summary

Total	High Priority	Normal Priority	Low Priority
13	0	13	0

Details

Files | **Types** | Warnings | Details

Type	Total	Distribution
DontImportJavaLang	2	
UnusedImports	11	
Total	13	

PMD를 사용한 정적 분석 결과, 13개의 Warning를 발견하였으며, 13개 모두 Normal 등급이다.

분석 결과

AnalysisSystem.java:4, UnusedImports, Priority: Normal

Avoid unused imports such as 'java.util.ArrayList'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

GUI.java:3, UnusedImports, Priority: Normal

Avoid unused imports such as 'java.awt.FileDialog'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

GUI.java:7, UnusedImports, Priority: Normal

Avoid unused imports such as 'javax.swing.JScrollBar'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

GUI.java:8, UnusedImports, Priority: Normal

Avoid unused imports such as 'javax.swing.JScrollPane'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

GUI.java:9, UnusedImports, Priority: Normal

Avoid unused imports such as 'javax.swing.JTextArea'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

GUI.java:15, UnusedImports, Priority: Normal

Avoid unused imports such as 'java.io.BufferedWriter'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

GUI.java:16, UnusedImports, Priority: Normal

Avoid unused imports such as 'java.io.FileWriter'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

GUI.java:24, UnusedImports, Priority: Normal

Avoid unused imports such as 'javax.swing.ScrollPaneConstants'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

GUI.java:26, DontImportJavaLang, Priority: Normal

Avoid importing anything from the package java.lang.

Avoid importing anything from the package 'java.lang'. These classes are automatically imported (JLS 7.5.3).

```
import java.lang.String;        // this is unnecessary

public class Foo {}

// --- in another source code file...

import java.lang.*;     // this is bad

public class Foo {}
```

SetupSystem.java:7, UnusedImports, Priority: Normal

Avoid unused imports such as 'java.io.StringReader'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

SetupSystem.java:8, DontImportJavaLang, Priority: Normal

Avoid importing anything from the package java.lang.

Avoid importing anything from the package 'java.lang'. These classes are automatically imported (JLS 7.5.3).

```
import java.lang.String;        // this is unnecessary

public class Foo {}

// --- in another source code file...

import java.lang.*;     // this is bad

public class Foo {}
```

SetupSystem.java:9, UnusedImports, Priority: Normal

Avoid unused imports such as 'java.util.LinkedList'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

SourceCode.java:2, UnusedImports, Priority: Normal

Avoid unused imports such as 'java.util.LinkedList'.

Avoid the use of unused import statements to prevent unwanted dependencies.

```
// this is bad
import java.io.File;
public class Foo {}
```

결론

'그놈! Clone Checker (Ver. 2016.06.05, 빌드 번호 #83)'에 대한 정적 분석 결과, SonarQube에서 334개, CheckStyle에서 1732개, FindBugs에서 14개, PMD에서 13개의 문제점을 발견하였다.