Università Degli Studi Di Salerno

Dipartimento di Informatica

Corso di Laurea Magistrale in Softaware Engeneering and IT Mangament

INGEGNERIA, GESTIONE ED EVOLUZIONE DEL SOFTWARE

ANNO ACCADEMICO 2021/22

DARTS_2.0

Test Report

Sommario

Versione	***Modifiche***
0.1	Test Report
0.2	Test Summary
0.3	Black Box Testing Report
0.4	Regression Testing
1.0	White Box Testing Report

Autori	***Matricola***
Gerardo Iuliano	0522501329
Alessandro Bacco	0522501104
Tiziano La Monica	0522501258

Test Report

Test Summary

77	0	0	23.423s
tests	failures	ignored	duration

100%
successful

Packages | Classes

Package	Tests	Failures	Ignored	Duration	Success rate
dataFunctionality	8	0	0	14.652s	100%
testSmellDetection	69	0	0	8.771s	100%

Black Box Testing Report

Unit Testing - Integration Testing - System Testing

69	0	0	8.771s
tests	failures	ignored	duration

100%
successful

Classes

Class	Tests	Failures	Ignored	Duration	Success rate
GodTestLackStructuralTest	21	0	0	1.825s	100%
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
GodClassMethodTest	4	0	0	0.402s	100%
[illegible]	5	0	0	[illegible]	[illegible]
[illegible]	17	0	0	[illegible]	[illegible]
IsolatedTest	3	0	0	0.202s	100%
Integration Utilities	5	0	0	0.527s	100%
MagicNumberStructuralTest	5	0	0	0.397s	100%
TestSmellUtilitiesTest	4	0	0	0.362s	100%

Regression Testing

8	0	0	1m16.16s
tests	failures	ignored	duration

100%
successful

Classes

Class	Tests	Failures	Ignored	Duration	Success rate
EagerTestStructuralTest	2	0	0	1m10.82s	100%
LazyTestTextualTest	2	0	0	1.640s	100%
GeneralFixtureTextualTest	2	0	0	1.308s	100%
LackOfCohesionOfTestSmellTextualTest	2	0	0	2.180s	100%

White Box Testing Report

ExceptionHandlingStructural

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
checkMethodsThatContainExceptions(PslClassBean, int)		97%		94%	1	10	0	17	0	1

ConstructorInitStructural

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
checkMethodsThatCauseConstructorInitialization(PslClassBean)		100%		100%	0	4	0	9	0	1

CondTestLogicStructural

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
checkMethodsThatCauseCondTestLogic(PslClassBean, int)		100%		86%	1	12	0	21	0	1

MagicNumberStructural

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
checkMethodsThatCauseMagicNumber(PslClassBean)		100%		100%	0	10	0	19	0	1

IgnoredTestStructural

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
checkMethodsThatIgnoredTest(PslClassBean)		100%		83%	2	7	0	14	0	1

DuplicateAssertStructural

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
checkMethodsThatCauseDuplicateAssert(PslClassBean)		100%		77%	4	10	0	21	0	1

```
public abstract class MagicNumberStructural {

  public static ArrayList<MethodWithMagicNumber> checkMethodsThatCauseMagicNumber(PsiClassBean testClass){
    ArrayList<MethodWithMagicNumber> methodsWithMagicNumbers = new ArrayList<>();
    for(PsiMethodBean psiMethodBeanInside : testClass.getPsiMethodBeans()){
      String methodName = psiMethodBeanInside.getPsiMethod().getName();
      if(!methodName.equals(testClass.getPsiClass().getName()) &&
              !methodName.equalsIgnoreCase("setup") &&
              !methodName.equalsIgnoreCase("teardown")){
        ArrayList<PsiMethodCallExpression> methodCalls = psiMethodBeanInside.getMethodCalls();
        if(methodCalls.size() >= 1){
          for(PsiMethodCallExpression callExpression : methodCalls){
            /*Prendo il nome del metodo */
            String methodCallName = callExpression.getMethodExpression().getQualifiedName();
            if(methodCallName.toLowerCase().contains("assert")){
              /*Prendo il primo argomento (expected) dall'assert */
              String firstArg = callExpression.getArgumentList().getExpressions()[0].getText();
              if(firstArg.matches("[0-9]+")){
                methodsWithMagicNumbers.add(new MethodWithMagicNumber(psiMethodBeanInside));
              }
            }
          }
        }
      }
    }
    if(methodsWithMagicNumbers.isEmpty()){
      return null;
    } else {
      return methodsWithMagicNumbers;
    }
  }
}
```

```
public abstract class ExceptionHandlingStructural {

    public static ArrayList<MethodWithExceptionHandling> checkMethodsThatContainExceptions(@NotNull PsiClassBean testClass, int threshold) {
        if(threshold<0 || threshold>5)
            return null;
        ArrayList<MethodWithExceptionHandling> methodsWithExceptHandling = new ArrayList<>();
        int count = 0;
        for(PsiMethodBean method : testClass.getPsiMethodBeans()) {
            PsiCodeBlock body = method.getPsiMethod().getBody();
            final PsiStatement[] statements = body.getStatements();
            for(PsiStatement statement : statements){
                if(statement instanceof PsiTryStatement || statement instanceof PsiThrowStatement){
                    count++;
                }
            }
            if(count > threshold){
                methodsWithExceptHandling.add(new MethodWithExceptionHandling(method));
            }
            count = 0;
        }
        if (methodsWithExceptHandling.isEmpty()){
            return null;
        } else{
            return methodsWithExceptHandling;
        }
    }
}
```

```java
public abstract class CondTestLogicStructural {

    public static ArrayList<MethodWithCondTestLogic> checkMethodsThatCauseCondTestLogic(PsiClassBean testClass, int threshold){
        if(threshold<0 || threshold>5)
            return null;
        ArrayList<MethodWithCondTestLogic> methodsWithCondTestLogics = new ArrayList<>();
        int count = 0;
        for(PsiMethodBean psiMethodBeanInside : testClass.getPsiMethodBeans()){
            PsiStatement[] psiStatements = psiMethodBeanInside.getPsiMethod().getBody().getStatements();
            if(psiStatements != null && psiStatements.length!=0){
                for(int i=psiStatements.length-1; i>=0; i--){
                    if(psiStatements[i] instanceof PsiIfStatement){
                        count++;
                    }
                    if(psiStatements[i] instanceof PsiSwitchStatement){
                        count++;
                    }
                    if(psiStatements[i] instanceof PsiConditionalLoopStatement){
                        count++;
                    }
                }
                if(count > threshold){
                    methodsWithCondTestLogics.add(new MethodWithCondTestLogic(psiMethodBeanInside));
                }
                count = 0;
            }
        }
        if(methodsWithCondTestLogics.isEmpty()){
            return null;
        } else {
            return methodsWithCondTestLogics;
        }
    }
}
```

```java
public abstract class ConstructorToItStructural {

    public static ArrayList<MethodWithConstructorInitialization> checkMethodsThatCauseConstructorInitialization(PsiClassBean testClass){
        ArrayList<MethodWithConstructorInitialization> methodWithConstructorInitializationArrayList = new ArrayList<>();

        for(PsiMethodBean psiMethodBeanInside : testClass.getPsiMethodBeans()){
            String methodName = psiMethodBeanInside.getPsiMethod().getName();
            if(methodName.equals(testClass.getPsiClass().getName())) {
                methodWithConstructorInitializationArrayList.add(new MethodWithConstructorInitialization(psiMethodBeanInside));
            }
        }
        if(methodWithConstructorInitializationArrayList.isEmpty()){
            return null;
        } else {
            return methodWithConstructorInitializationArrayList;
        }
    }
}
```

```java
public abstract class DuplicateAssertStructural {

    public static ArrayList<MethodWithDuplicateAssert> checkMethodsThatCauseDuplicateAssert(PsiClassBean testClass) {
        ArrayList<MethodWithDuplicateAssert> methodsWithDuplicateAsserts = new ArrayList<>();
        int counter = 0;
        for(PsiMethodBean psiMethodBeanInside : testClass.getPsiMethodBeans()){
            String methodName = psiMethodBeanInside.getPsiMethod().getName();
            if(!methodName.equals(testClass.getPsiClass().getName()) &&
                    !methodName.toLowerCase().equals("setup") &&
                    !methodName.toLowerCase().equals("teardown")){
                ArrayList<PsiMethodCallExpression> methodCalls = psiMethodBeanInside.getMethodCalls();
                if(methodCalls.size() >= 1){
                    for(PsiMethodCallExpression callExpression : methodCalls){
                        /*Prendo il nome del metodo */
                        String methodCallName = callExpression.getMethodExpression().getQualifiedName();
                        if(methodCallName.toLowerCase().contains("assert")){
                            counter++;
                        }
                    }
                    if (counter>1){
                        methodsWithDuplicateAsserts.add(new MethodWithDuplicateAssert(psiMethodBeanInside));
                    }
                }
            }
            counter=0;
        }
        if (methodsWithDuplicateAsserts.isEmpty()) {
            return null;
        } else {
            return methodsWithDuplicateAsserts;
        }
    }

}
```

```
16. public class IgnoredTestStructural {
17.
18.     public static ArrayList<MethodWithIgnoredTest> checkMethodsThatIgnoredTest(PsiClassBean testClass) {
19.         ArrayList<MethodWithIgnoredTest> methodWithIgnoredTests = new ArrayList<>();
20.         for(PsiMethodBean psiMethodBeanInside : testClass.getPsiMethodBeans()){
21.             String methodName = psiMethodBeanInside.getPsiMethod().getName();
22.             if(!methodName.equals(testClass.getPsiClass().getName()) &&
23.                     !methodName.toLowerCase().equals("setup") &&
24.                     !methodName.toLowerCase().equals("teardown")) {
25.
26.                 PsiModifierList psiModifierList = psiMethodBeanInside.getPsiMethod().getModifierList();
27.
28.                 PsiAnnotation[] annotations = psiModifierList.getAnnotations();
29.
30.                 if (psiModifierList.toString().contains("Ignore")){
31.                     methodWithIgnoredTests.add(new MethodWithIgnoredTest(psiMethodBeanInside));
32.                 }
33.             }
34.         }
35.         if(methodWithIgnoredTests.isEmpty()){
36.             return null;
37.         } else {
38.             return methodWithIgnoredTests;
39.         }
40.     }
41. }
```