Laurea Magistrale in informatica
Università di Salerno

Test Plan
Unit testing

Team member
Saverio Napolitano - 0522501400
Gerardo Festa - 0522501452
Alessandra Parziale - 0522501422

https://github.com/GerardoFesta/infozilla

Sommario

1. Approccio generale per il testing dei filtri

Il testing è stato condotto sulla base di:
Coverage, dunque in maniera white-box. La coverage ci ha guidato nell'attività di testing.
Regular Expression/Patterns: i filtri utilizzano pattern con i quali fanno il match delle stringhe in input. Abbiamo fornito quanti più input per mettere in difficoltà il sistema in tal senso, utilizzando delle stringhe che siano realistiche, ma che possano essere male interpretate dalle regex.

1.1. FilterPatches

Classi Testate: *FilterPatches* e *PatchParser.*
Nome classi di Test: *FilterPatchesTest* e *PatchParserTest*.

FilterPatchesTest

In *FilterPatchesTest* abbiamo verificato il riconoscimento da parte del filtro per un numero di patches uguale a: zero, due e tre. Negli ultimi due casi è stato verificato anche il numero di *Hunk* per ogni patch. Infine per ogni Test abbiamo verificato la corretta individuazione degli elementi.

Test:

TestRunFilterWith_TwoPatches()

Input	**Oracolo**
2 patches	Size patches=2 Size hunk prima patch = 2 Size hunk seconda patch =1

TestRunFilterWith_ThreePatches()

Input	**Oracolo**
3 Patches	Size patches = 3 Size hunk prima patch = 2 Size hunk seconda patch =1 Size hunk terza patch =1

TestRunFilterWithoutPatches()

Input	**Oracolo**
“Index: file_modificato.txt\n ==========”	Size patches = 0

TestRunFilterWithoutPatches2()

Input	**Oracolo**
“Ihghghgh”	Size Patches = 0

TestGetOutputText()

Input	**Oracolo**
"Test output"	“Test output”

TestGetOutputText2()

Input	**Oracolo**
2 Patches	“ ”

TestGetOutputText3()

Input	**Oracolo**
1 patch	“ “

PatchParserTest

Test:

TestParseForPatches()

Input	**Oracolo**
1 patch	Size patches = 1 Size hunk = 2 Index = “file_modificato.txt” Original File = “file_modificato.txt” Modified File = “file_modificato.txt” Testo primo hunk corrispondente all’input Testo secondo hunk corrispondente all’input

TestPatchWithLeadingTrailingWhitespace()

Input	**Oracolo**
1 patch contenente spazi vuoti prima dell'hunk e alla fine	Testo degli hunk uguale all'input

TestDiffPatch1()

Input	**Oracolo**
1 patch in formato unified diff	Size patch = 1

TestDiffPatchHunks()

Input	**Oracolo**
2 patches 1 in formato unified diff	Size hunk prima patch = 1

TestParsePatchWithWhitespace()

Input	**Oracolo**
1 patch con doppio all'interno del testo degli hunks \n	Testo degli hunk uguale all'input

TestParsePatch_TextWithoutLeadingWhitespace()

Input	**Oracolo**
1 patch con riga nella sezione testo degli hunk che inizia senza uno spazio.	Size patch = 1 Testo senza la riga che inizia senza uno spazio.

TestParsePatch_WithLeadingWhitespace_Hunk()

Input	**Oracolo**
1 patch con un hunk header che inizia con uno spazio	Size patch = 0

TestParsePatch_WithLeadingWhitespace_index()

Input	**Oracolo**
1 patch con intestazione index con spazio iniziale	Size patch = 0

TestParsePatch_WithLeadingWhitespace_OriginalFile()

Input	**Oracolo**
1 patch con file originale che inizia con uno spazio	Size patch = 0

TestParsePatch_SeparationLines_Number()

Input	**Oracolo**
1 patch con numero di separation line dopo index uguale a 3	Size patch = 0

TestParsePatch_SeparationLines_MinimumNumber()

Input	**Oracolo**
1 patch con numero di separation line dopo index uguale a 4	Size patch = 1

1.2. FilterEnumeration

Classi Testate: *FilterEnumeration.*
Nome classi di Test: *FilterEnumerationTest.*

FilterEnumerationTest

In *FilterEnumerationTest* è stato verificato il riconoscimento da parte del filtro per un numero di enumerazioni uguale a zero, a due e a N per ogni tipologia *(Char, Num, Itemization)*.

Test:

TestGetWithoutEnums()

Input	**Oracolo**
Input senza enumeration	Size = 0

TestGetCharEnums()

Input	Oracolo
A.Item 1 B.Item 2 C.Item 3	Size = 1 Type = Char

TestGetNumEnums()

Input	Oracolo
1. Item 1 2. Item 2 3. Item 3	Size = 1 Type = Num

TestGetItemizations()

Input	Oracolo
- Item 1 - Item 2 - Item 3	Size = 1 Type = Itemizations

TestRunFilterCharEnumsANDItemizations()

Input	Oracolo
A.Item 1 B.Item 2 C.Item 3 - Item 4 - Item 5	Size = 2 Type = Char - Itemizations

TestRunFilterCharEnumsANDNumEnums()

Input	Oracolo
A.Item 1 B.Item 2 C.Item 3 - Item 4 - Item 5	Size = 2 Type = Char - Itemizations

TestRunFilterCharEnumsANDCharEnums()

Input	Oracolo
A.Item 1 B.Item 2 C.Item 3 A.Item 4 B.Item 5	Size = 2 Type = Char - Char

TestRunFilter2CharEnumsAND2CharEnums()

Input	Oracolo
A.Item 1 B.Item 2 A.Item 4 B.Item 5	Size = 2 Type = Char - Char

TestRunFilterItemizationsANDCharEnums()

Input	Oracolo
- Item 1 - Item 2 - Item 3 A.Item 4 B.Item 5	Size = 2 Type = Itemizations - Char

TestRunFilterNumEnumsANDItemizations()

Input	Oracolo
1.Item 1 2.Item 2 - Item 3 - Item 4	Size = 2 Type = Num - Itemizations

TestRunFilterNumEnumsANDCharEnums()

Input	Oracolo
1. Item 1 2. Item 2 A. Item 3 B. Item 4	Size = 2 Type = Num - Char

TestRunFilterNumEnumsANDNumEnums()

Input	Oracolo
1. Item 1 2. Item 2 3. Item 3 1. Item 4 2. Item 5	Size = 2 Type = Num - Num

TestRunFilterCharEnumsANDNumEnumsANDItemization()

Input	**Oracolo**
A. Item 1 B. Item 2 6. Item 3 7. Item 4 1. Item 5 2. Item 6 - Item 7 - Item 8	Size = 4 Type = Char - Num - Itemizations

TestGetOutputText()

Input	**Oracolo**
1. Item 1 2. Item 2 - Item 3 - Item 4	“ ”

1.3. FilterSourceCodeJAVA

Classi Testate: *FilterSourceCodeJAVA*.
Nome classi di Test: *FilterSourceCodeJAVATest*

Test:

TestRunFilterWithMinimalSet()

Input	**Oracolo**
if (condition) { doSomething(); } else { doSomethingElse(); }	Size = 2 Keyword = ifstatement - elsestatement text[0] = if (condition) { doSomething(); } text[1] = else { doSomethingElse(); }

TestRunFilterWithNestedCode()

Input	Oracolo

import java.util.*;
public class Test {
  private int x;
  public void method() {
  if (condition) {
  doSomething();
  } else {
  doSomethingElse();
  }
  }
}
	Size = 2
	Keyword = import- class
	
	text[0] = import java.util.*;
	
	text[1] = public class Test {
	  private int x;
	  public void method() {
	  if (condition) {
	  doSomething();
	  } else {
	  doSomethingElse();
	  }
	  }
	}

TestRunFilterWithIncompleteSyntax()

Input	Oracolo
import java.util.*; public class Test { int x;	Size = 2 Keyword = import- class text[0] = import java.util.*; text[1] = public class Test {

TestPackageDeclaration()

Input	Oracolo
package io.example;	Size = 1 Keyword = package text[0] = package io.example;

TestSingleLineComment()

Input	Oracolo
//This is a comment abc	Size = 1 Keyword = singlecomment text[0] = //This is a comment

TestMultiLineComment()

Input	Oracolo
/* This is a multi-line comment */ and it's over	Size = 1 Keyword = multicomment text[0] = /* This is a multi-line comment */

TestFunctionDefAndCall()

Input	Oracolo

public void method() {
if (condition) {
doSomething();
} else {
doSomethingElse();
}
}
This is just random text. When i call method(); it crashes
Then, here is an invalid method(); call. This happens because it is not at the end of the line
This other one is bad because it is missing the ; method()
Yet, it works with parameters method(a,b,c);
Just a reminder, the regex needs the call to be on a newline, otherwise it will take whatever is before the call as well.
	Size = 3
	Keyword = functiondef - functioncall - functioncall
	
	text[0] = public void method() {
	if (condition) {
	doSomething();
	} else {
	doSomethingElse();
	}
	}
	
	text[1] = method();
	
	text[2] = method(a,b,c);

TestWithAllSegments

Input	Oracolo

package java.io;
import java.lang.*;
//this is a test comment
public class randomClass{
//doesn't matter what's in there
}
Some break text
public void method() {
}
/*Some break text
for multiline comment */
if (condition) {
doSomething();
}
else {
doSomethingElse();
}
 	
This is just random text.
method();
	Size = 9
	Keyword = package - import - singlecomment - class - functiondef - multicomment - ifstatement - elsestatement - functioncall
	
	text[0] = package java.io;
	
	text[1] = import java.lang.*;
	
	text[2] = //this is a test comment
	
	text[3] =public class randomClass{
	//doesn't matter what's in there
	}
	
	text[4] = public void method() {
	}
	
	text[5] = /*Some break text
	for multiline comment */
	
	text[6] = if (condition) {
	doSomething();
	}
	
	text[7] = else {
	doSomethingElse();
	}
	
	text[8] = method();

1.4. FilterStackTraceJAVA

Classi Testate: *FilterStackTraceJAVA.*
Nome classi di Test: *FilterStackTraceJAVATest*

Test:

TestFindStackTracesSingleDepth()

Input	**Oracolo**
"Some text with a stack trace: java.lang.TestException: TestException at com.example.TestClass.method(Test Class.java:42)"	Size = 1 trace[0].isCause = false trace[0].frames.size = 1 trace[0].reason = TestException; text[0].exception = java.lang.TestException

TestFindStackTracesMultipleDepth()

Input	**Oracolo**

Some text with a stack trace:
java.lang.TestException:
TestException
  at
com.example.TestClass.method1(Te
stClass.java:42)
  at
com.example.TestClass.method2(Te
stClass.java:50)
	Size = 1
	
	trace[0].isCause = false
	
	trace[0].frames.size = 2
	trace[0].reason = TestException;
	
	text[0].exception =
	java.lang.TestException

TestFindStackTracesNoStackTrace()

Input	**Oracolo**
“”	Size = 0

TestFindStackTracesMultipleStackTraces()

Input	**Oracolo**

First stack trace:
java.lang.TestException:
TestException1
at
com.example.TestClass.method1(TestClass.java:42)
at
com.example.TestClass.method4(TestClass.java:112)
Second stack trace:
java.lang.RuntimeException:
TestException2
at
com.example.TestClass.method2(TestClass.java:50)
	Size = 2
	
	trace[0].isCause = false
	
	trace[0].frames.size = 2
	trace[0].reason = TestException1;
	trace[0].exception =
	java.lang.TestException
	
	trace[1].isCause = false
	
	trace[1].frames.size = 1
	trace[1].reason = TestException2;
	trace[1].exception =
	java.lang.RuntimeException

TestFindStackTracesWithCause()

Input	Oracolo
Root Exception: java.lang.TestException: RootException at com.example.TestClass.method(TestClass.java:42) Caused by: java.lang.RuntimeException: CauseException at com.example.TestClass.method2(TestClass.java:50)	trace[0].isCause = false trace[1].isCause = true Size = 2 trace[0].exception = java.lang.TestException trace[0].reason = RootException trace[0].frames.size = 1 trace[1].exception = java.lang.RuntimeException trace[1].reason = CauseException trace[1].frames.size = 1

2. Coverage raggiunta

La percentuale di copertura della branch raggiunta per ciascun filtro è:

- Per *FilterPatches*: 75%
- Per *PatchParser*: 86%
- Per *FilterStackTrace*: 73%
- Per *FilterSourceCode*: 97%
- Per *FilterEnumeration*: 86%