

ROTTEN GREEN TESTS

(FROM ICSE'19)

*J. Delplanque, S. Ducasse, G. Polito, A. P. Black
and A. Etien*

*Univ. Lille, CNRS, Centrale Lille, Inria, UMR 9189 - CRIStAL
Dept of Computer Science, Portland State University, Oregon, USA*



WHAT IS A ROTTEN GREEN TEST?

ANATOMY OF A TEST

SetTest » testSetAdd

| s |

s := Set new.

s add: 1.

s add: 1.

self **assert:** s size **equals:** 1.

self **assert:** (s includes: 1)

```
class SetTest {  
    method testSetAdd {  
        def s = Set.new()  
        s.add(1)  
        s.add(1)  
        self.assertEquals(s.size(),1)  
        self.assert(s.includes(1))  
    }  
}
```

In Pharo

s add: 1. <=> s.add(1);

ANATOMY OF A SMOKE TEST

SetTest » testSetAddSmokeTest

| s |

s := Set new.

s add: 1.

s add: 1

- No assertion
- Not a rotten green test

A ROTTEN GREEN TEST IS

- A test *passing (green)*
- A test that contains at least one *assertion*
- One or more assertions is *not* executed when test runs



A LITTLE SKETCH OF A ROTTEN GREEN TEST

```
class RottenTest {  
  method testABC {  
    if (false) then {self.assert(x)}  
  }  
}
```

.....

TPrintOnSequencedTest » testPrintOnDelimiter

| aStream result allElementsAsString |

result := ''.

aStream := ReadWriteStream on: result.

self nonEmpty printOn: aStream delimiter: ', '.

allElementsAsString := result findBetweenSubstrings: ', ', ' '.

allElementsAsString withIndexDo: [:el :i |

self assert: el equals: ((self nonEmpty at:i) asString)]

A REAL ONE



.....

TPrintOnSequencedTest » testPrintOnDelimiter

| aStream result allElementsAsString |

result := ''.

aStream := ReadWriteStream on: result.

self nonEmpty printOn: aStream delimiter: ', '.

allElementsAsString := result findBetweenSubstrings: ', ', '.

allElementsAsString withIndexDo: [:el :i |

self **assert:** el **equals:** ((self nonEmpty at:i) asString)]

Not executed!

The programmer believed that the object on which the stream is working is “magically” mutated on stream growth

```
TPrintOnSequencedTest » testPrintOnDelimiter
```

```
| aStream result allElementsAsString |
```

```
result := "".
```

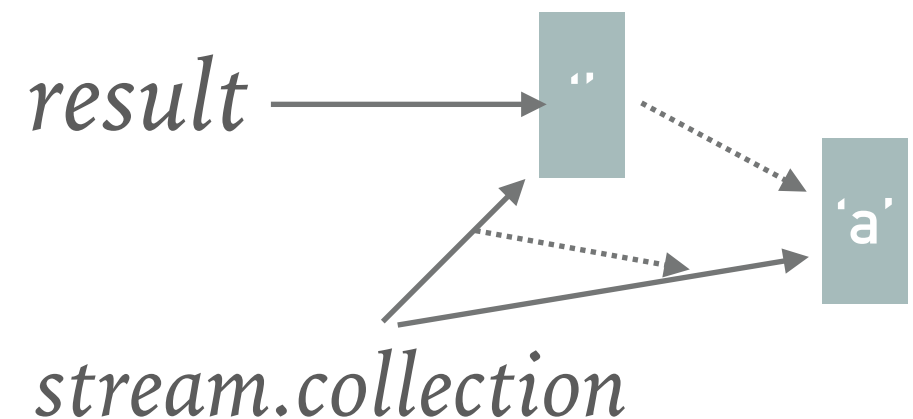
```
aStream := ReadWriteStream on: result.
```

```
self nonEmpty printOn: aStream delimiter: ', '.
```

```
allElementsAsString := result findBetweenSubstrings: ', '.
```

```
allElementsAsString withIndexDo: [:el :i |
```

```
self assert: el equals: ((self nonEmpty at:i) asString) ]
```



result stays empty

Iterator does not run

ROTTEN GREEN TEST WRITERS

- Rotten green tests are NOT intentional
- We say: this is *not* the programmer's fault
- Instead: it is the fault of testing tools that **do not report** them

WHY ARE ROTTEN GREEN TESTS BAD?

- Give a false sense of security
- Can easily pass unnoticed
- Not reported by testing frameworks prior to *DrTest*

ROTTEN GREEN TEST IS...

- A test *passing* (green)
- A test that contains at least one *assertion*
- One or more assertions is *not* executed when test runs

MAINLY CAUSED BY

- Conditional code not executing a branch
- Iterating over an empty collection

ROTTEN GREEN TEST IS...

- A test *passing* (green)
- A test that contains at least one *assertion*
- One or more assertions is *not* executed when test runs

HOW TO IDENTIFY THEM?

HANDLING HELPERS

```
class RottenTest {  
  method testABC {  
    if (false) then {self.helper()}  
  }  
  
  method helper {  
    self.secondHelper()  
  }  
  
  method secondHelper {  
    self.assert(x)  
  }  
}
```

HANDLING HELPERS

```
class RottenTest {  
  method testABC {  
    if (false) then {self.helper()}  
  }  
  
  method helper {  
    self.secondHelper()  
  }  
  
  method secondHelper {  
    self.assert(x)  
  }  
}
```

Not executed!

Not executed!

ABOUT THE NEED FOR CALL SITE ANALYSIS

```
class RottenTest {  
  method testDEF {  
    self.badHelper()  
    self.assert(true)  
  }  
  
  method badHelper {  
    if (false) then {  
      self.secondHelper()  
    }  
  }  
  
  method secondHelper {  
    self.assert(x)  
  }  
}
```

ABOUT THE NEED FOR CALL SITE ANALYSIS

```
class RottenTest {  
  method testDEF {  
    self.badHelper()  
    self.assert(true) Executed!  
  }  
  
  method badHelper {  
    if (false) then {  
      self.secondHelper() Not executed!  
    }  
  }  
  
  method secondHelper {  
    self.assert(x) Not executed!  
  }  
}
```

IDENTIFYING ROTTEN GREEN TESTS

- We use both
 - *Static analysis*, to identify helpers and inherited methods
 - *Dynamic analysis*, to identify *call* sites that are not executed

IDENTIFYING ROTTEN GREEN TESTS

- **Static Analysis**
 - Identify “testing primitives” (assert:, deny:...)
 - Identify helper methods (abstract interpreter)
- **Dynamic Analysis** through instrumentation
 - Instrument the **call-sites** of the “test primitives”
- **Run the test suite**
 - Record green tests whose test primitives are not executed
- **Generate Report**



BEFORE TEST EXECUTION: FIRST IDENTIFYING THE HELPERS

```
class RottenTest {  
  method testDEF {  
    self.badHelper()  
    self.assert(true)  
  }  
  
  method badHelper {  
    if (false) then {  
      self.secondHelper()  
    }  
  }  
  
  method secondHelper {  
    self.assert(x)  
  }  
}
```



BEFORE TEST EXECUTION: FIRST IDENTIFYING THE HELPERS

```
class RottenTest {  
  method testDEF {  
    self.badHelper()  
    self.assert(true)  
  }  
  
  method badHelper { is an helper  
    if (false) then {  
      self.secondHelper()  
    }  
  }  
  
  method secondHelper { is an helper  
    self.assert(x)  
  }  
}
```

BEFORE TEST EXECUTION: INSTALLING CALL SITE SPIES

```
class RottenTest {  
  method testDEF {  
    self.badHelper()  
    self.assert(true)   
  }  
  
  method badHelper {  
    if (false) then {  
      self.secondHelper()  
    }  
  }  
  
  method secondHelper {  
    self.assert(x)   
  }  
}
```

AT EXECUTION

```
class RottenTest {  
  method testDEF {  
    self.badHelper()  
    self.assert(true)   
  }  
  
  method badHelper {  
    if (false) then {  
      self.secondHelper()  
    }  
  }  
  
  method secondHelper {  
    self.assert(x)   
  }  
}
```


CASE STUDIES (CHECK THE PAPER)

- 19,905 tests analysed on mature projects
- 294 rotten (25 fully rotten)

Project	Description	#pack.	#classes	#test	#tests classes	#helpers	found rotten tests			
							missed fail	missed skip	context dependent	fully rotten
Compiler	AST model and compiler of Pharo.	6	232	51	859	10	0	0	1	4
Aconcagua	Model representing measures.	2	84	27	661	2	0	0	0	0
Buoy	Various package extensions	12	51	19	185	0	0	0	0	0
Calypso	Pharo IDE.	58	705	157	2692	4	88	0	0	0
Collections	Pharo collection library.	16	222	59	5850	32	0	5	119	17
Fuel	Object serialization library.	6	131	30	518	4	0	0	5	0
Glamour	UI framework.	19	463	65	458	9	0	0	0	0
Moose	Software analysis platform.	66	491	120	1091	6	1	0	0	1
PetitParser2	Parser combinator framework.	14	319	78	1499	349	0	0	0	1
Pillar	Document processing platform.	32	354	127	3179	136	0	0	0	1
Polymath	Advanced maths library.	54	299	91	767	3	0	0	0	0
PostgreSQL	PostgreSQL Parser.	4	130	11	130	2	0	0	0	0
RenoirSt	DSL to generate CSS.	4	103	42	157	4	0	0	0	0
Seaside	Web application framework.	49	837	134	806	44	35	17	0	1
System	Low-level system packages	40	260	46	553	11	0	1	9	0
Telescope	Visualisation framework.	6	173	21	87	0	0	0	0	0
Zinc	HTTP library.	9	184	43	413	12	0	0	0	0

RESULTS

- **Missing fail:** test passes false to **assert:**, instead of using **fail**
- **Missed skip:** test contains guards to stop its execution early (under certain conditions), instead of using **skip**
- **Context-dependent logic:** complex logic with different assertions in different branches — some may be rotten too
- **Fully rotten:** other tests that do not execute one or more assertions

MISSED FAIL

- Test passes false to **assert:**, instead of using **fail**

```
TSequencedElementAccessTest » test0FixtureSequencedElementAccessTest
  self moreThan4Elements.
  self assert: self moreThan4Elements size >= 4.
  self subCollectionNotIn
    detect: [:each | (self moreThan4Elements includes: each) not]
    ifNone: [ self fail ].
  self elementNotInForElementAccessing.
  self deny: (self moreThan4Elements includes: self
    elementNotInForElementAccessing).
  self elementInForElementAccessing.
  self assert: (self moreThan4Elements includes: self
    elementInForElementAccessing)
```

SOLVING MISSING FAIL

- Test passes false to **assert**:
- Check on a case by case basis
- Use **fail** to really indicate that the assertion should not be executed

MISSED SKIP

- Test contains guards to stop its execution early (under certain conditions), instead of using **skip**

OCContextTempMappingTest »

testAccessingArgOfOuterBlockFromAnotherDeepBlock

| actual |

"Check source code availability; do not fail on images without sources"

thisContext method hasSourceCode

ifTrue: **[self skip]**

actual := [:outerArg | outerArg asString.

 [:innerArg | innerArg asString.

 thisContext tempNamed: #outerArg] value: #innerValue.

] value: #outerValue.

self assert: actual equals: #outerValue

SOLVING MISSED SKIP

- Test contains guards to stop its execution early (under certain conditions), instead of using **skip**
- Easy to fix: use **self skip**
- Runner can then report correctly

CONTEXT DEPENDENT LOGIC

- Complex logic with different assertions in different branches — some may be rotten too

FLBasicSerializationTest » testCharacter

"Test character serialization. If the code is less than 255 the same instance is used. But if it is bigger, new ones are created."

self assertSerializationIdentityOf: \$a.

FLPlatform current isSpur

ifTrue: [self assertSerializationIdentityOf: (Character value: 12345).

"Japanese Hiragana 'A' "]

ifFalse: [self assertSerializationEqualityOf: (Character value: 12345).

"Japanese Hiragana 'A' "].

self assertSerializationEqualityOf: Character allCharacters.

self assertSerializationEqualityOf: (Array with: \$a with: (Character value: 12345)).

SOLVING CONTEXT DEPENDENT LOGIC

- Complex logic with different assertions in different branches
— some may be rotten too
- How to fix:
 - create a separate test for each branch
 - use **self skip** to execute the test only when it applies

FULLY ROTTEN TESTS

- Other tests that do not execute one or more assertions

```
testFamixPackageNamespace
```

```
  self
```

```
    should: [ self assert: self packReferee ]
```

```
    raise: Error
```

```
testFamixPackageNamespace {
```

```
  try{
```

```
    self.assert(self.packReferee())
```

```
    self.fail()
```

```
  } catch(Error e){ }
```

```
}
```

- packageP5FullReferee *did* raise Error, so assert: is never invoked!

FULLY ROTTEN

- Other tests that do not execute one or more assertions

MustBeBooleanTests » testAnd (original)

```
I myBooleanObject I  
myBooleanObject := MyBooleanObject new.  
self deny: (myBooleanObject and: [true])
```

- Compiled method bytecode was dynamically rewritten by the compiler as:

MustBeBooleanTests » testAnd (rewritten)

```
I myBooleanObject I  
myBooleanObject := MyBooleanObject new.  
^ (myBooleanObject) and: [ true ]
```

Oops the deny: disappeared!

SOLVING FULLY ROTTEN

- Case-by-case analysis and repair

CONCLUSION

- Rotten Green Tests exist. Yes M'dam
- Every Unit-testing framework should report them
- *DrTests* the new Pharo Unit framework reports them



- Working on Java and Python replication
 - Found many of them in great Java frameworks :)
 - Looking for extra case studies