ROTTEN GREEN TESTS

(ICSE'19)

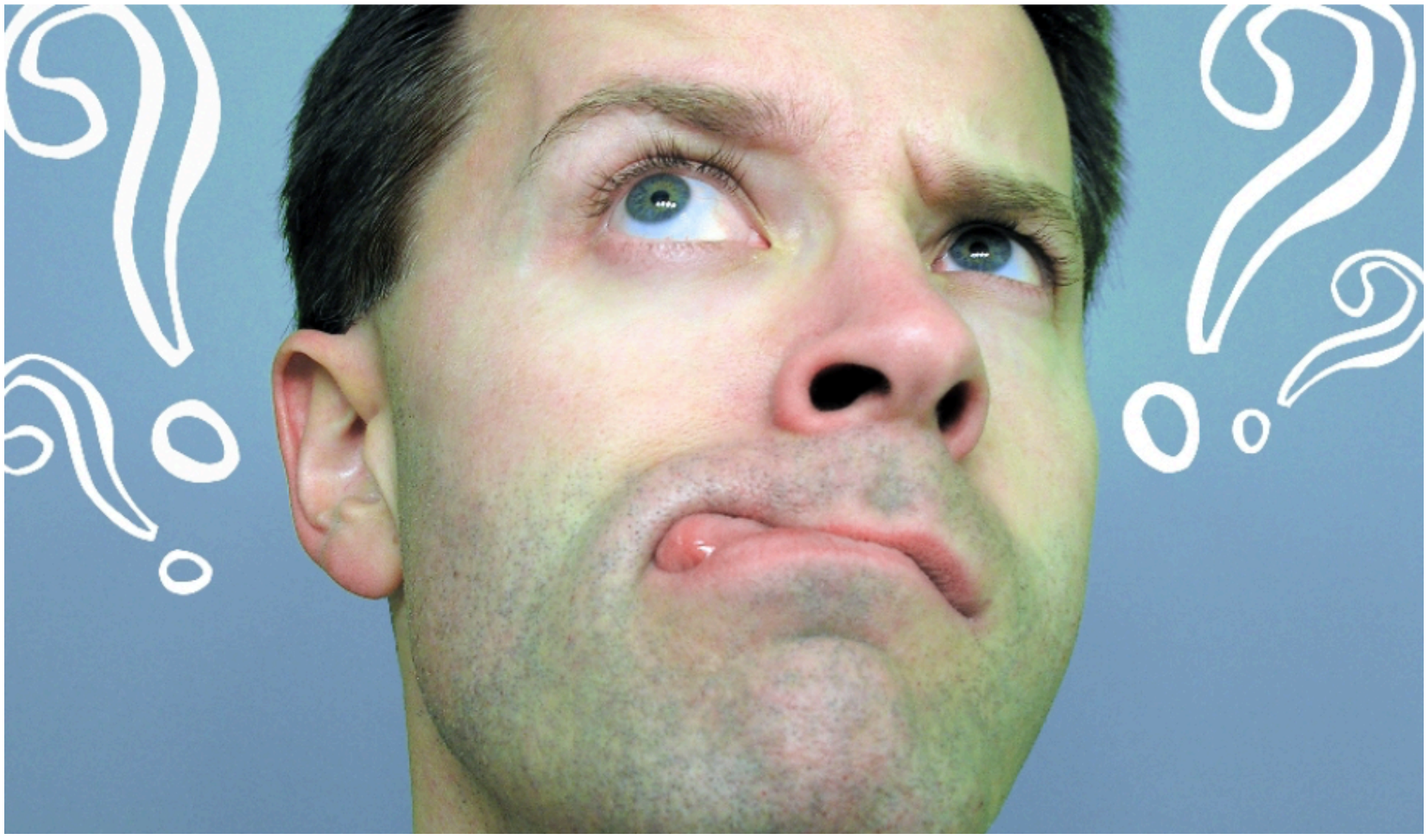
*J. Delplanque, **S. Ducasse**, G. Polito, A. P. Black and A. Etien*

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

WHAT IS A ROTTEN GREEN TEST?

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)}
    }

}
```

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) ]
```

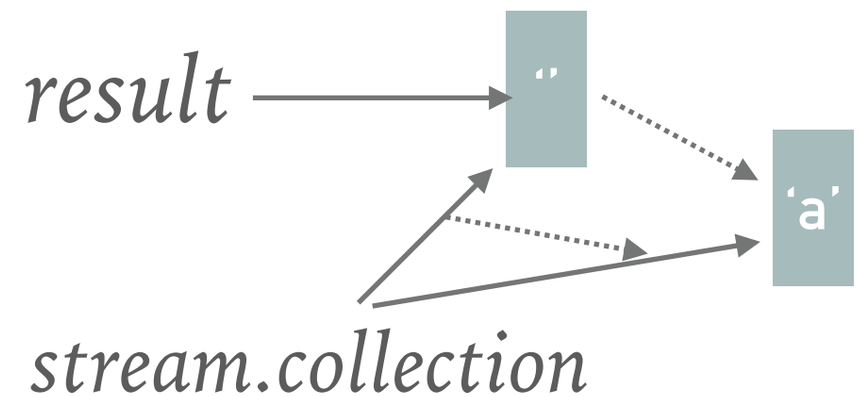
A REAL ONE


```
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()}
    }
```

Not executed!

```
    method helper {
        self.secondHelper()
    }

    method secondHelper {
        self.assert(x)
    }
```

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)
    }

    method badHelper {
        if (false) then {
            self.secondHelper()
        }
    }

    method secondHelper {
        self.assert(x)
    }
}
```

Executed!

Not executed!

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 {
        if (false) then {
            self.secondHelper()
        }
    }
    method secondHelper {
        self.assert(x)
    }
}
```

is an helper

is an helper

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)
    }
}
```

spy

spy

AT EXECUTION

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

spy

spy

CASE STUDIES (CHECK THE PAPER AND THE FOLLOWING ONE)

- 19,905 tests analysed on mature projects
- 294 rotten (25 fully rotten)
- Found rotten green tests in Java and Python projects

							found rotten tests			
Project	Description	#pack.	#classes	#test	#tests classes	#helpers	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