Rotten Green Tests

And a discussion about tests in Pharo

Julien Delplanque

julien.delplanque@inria.fr

https://rmod.inria.fr/web/

Roadmap

1. **Rotten Green Tests**
    - Definitions
    - Detect Rotten Tests
    - A Vicious Rotten Test
    - Future Work
2. **Discussion around testing in Pharo**

Rotten Green Tests

A First Analysis

Julien Delplanque
Université de Lille, CRIStAL, CNRS, UMR 9189,
RMoD Team, Inria Lille Nord Europe
Lille, France
julien.delplanque@inria.fr

Stéphane Ducasse
RMoD Team, Inria Lille Nord Europe
France
stephane.ducasse@inria.fr

Andrew P. Black
Department of Computer Science
Portland State University
Oregon, USA &
RMoD Team, Inria Lille Nord Europe
France
apblack@pdx.edu

Guillermo Polito
CNRS, Université de Lille, CRIStAL, UMR 9189,
RMoD Team, Inria Lille Nord Europe
Lille, France
guillermo.polito@inria.fr

Abstract

Unit tests are a tenant of agile programming methodologies, and are widely used to improve code quality and prevent code regression. A passing (green) test is usually taken as a robust sign that the code under test is valid. However, we have noticed that some green tests contain assertions that are never executed; these tests pass not because they assert properties that are true, but because they assert nothing at all. We call such tests *Rotten Green Tests.*

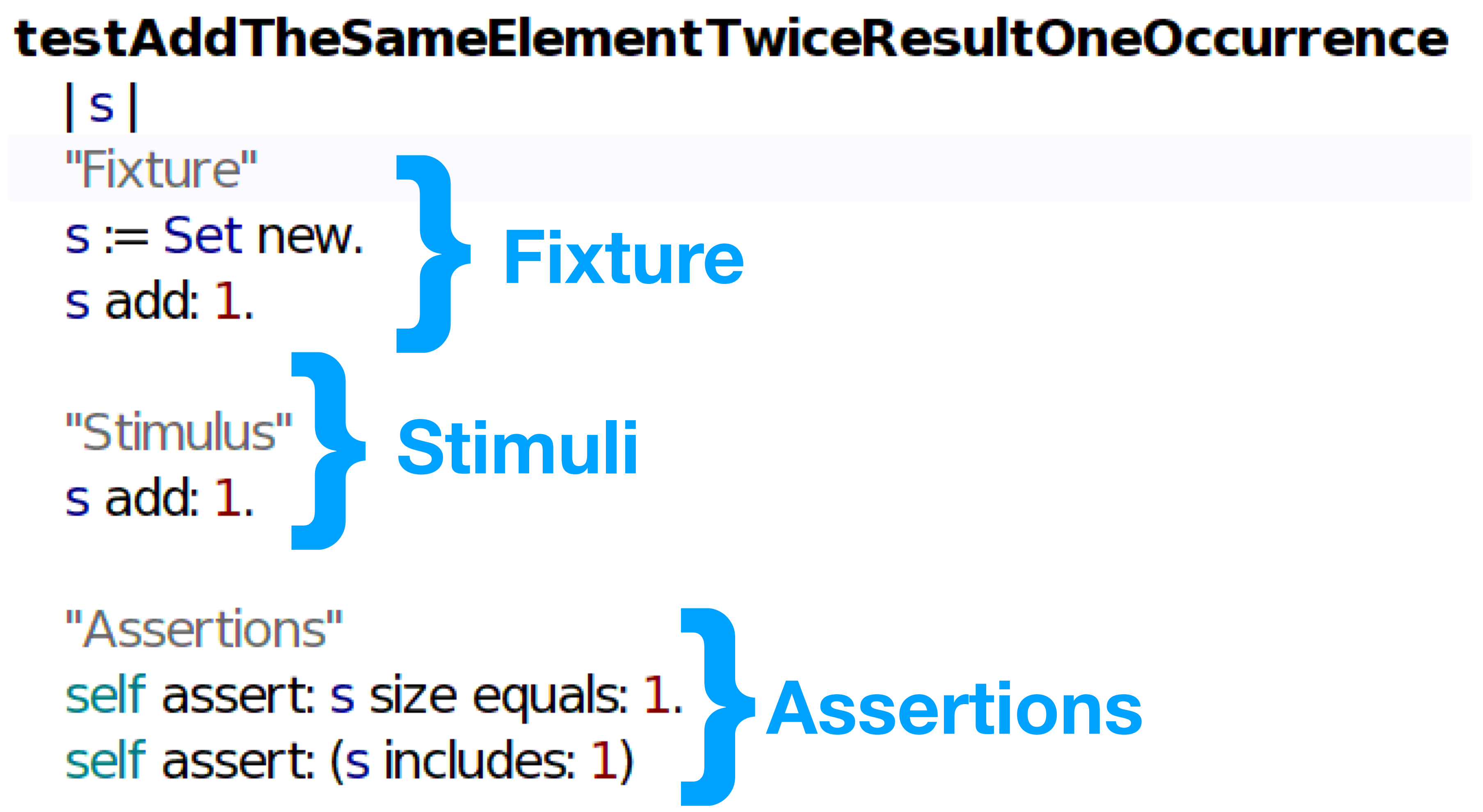
Rotten Green Tests represent a worst case: they report that the code under test is valid, but in fact do nothing to test that validity, beyond checking that the code does not crash. We describe an approach to identify rotten green tests by combining simple static and dynamic analyses. Our approach

i.e., tests that are passing, because they provide assurance that the software is working as expected.

Our concern in this work is with tests that were intended by their designer to execute some assertions, but do not actually do so — we call them *rotten green tests*. Such tests are insidious because they pass, and they contain assertions; they therefore give the *impression* that some useful property is being validated. In fact, rotten green tests guarantee nothing: they are worse than having no tests at all!

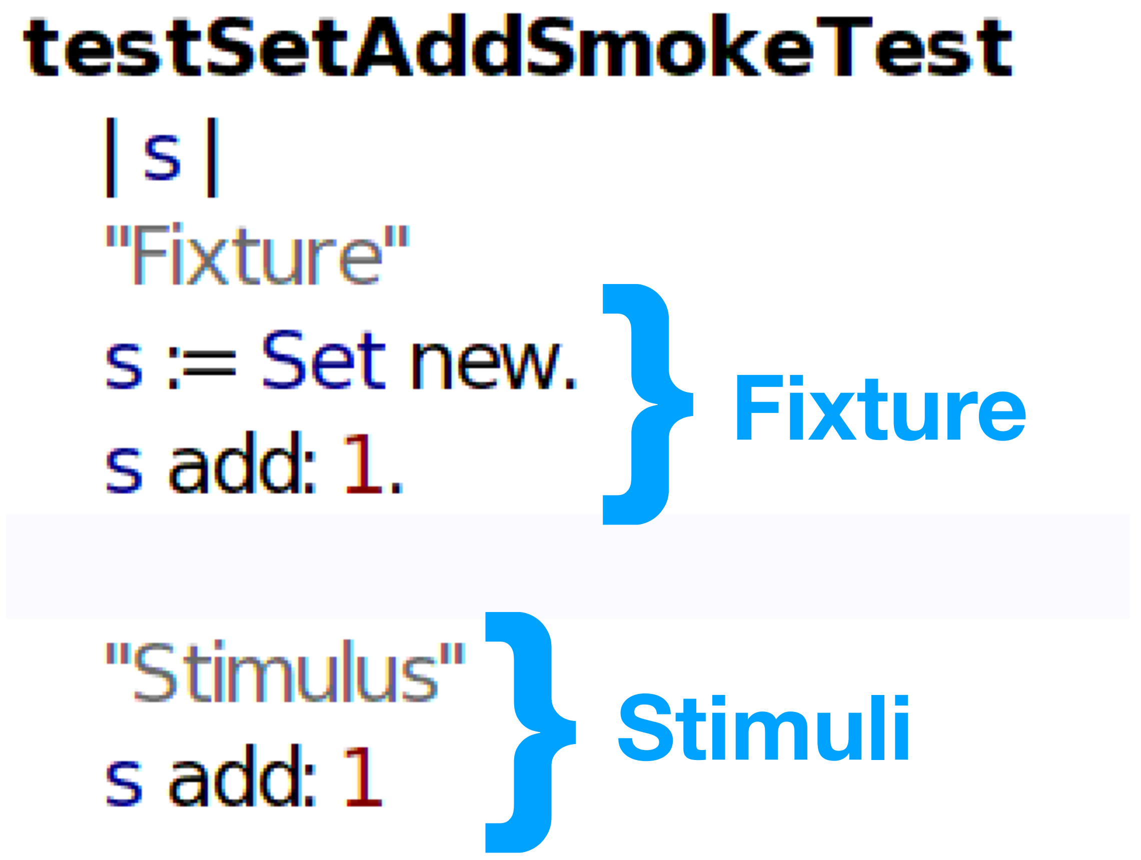
Our approach is based on a combination of static analysis and dynamic monitoring of method execution. We identify whether or not a test is rotten, even in presence of helper methods and trait compositions. A limitation of our current implementation is that a test with multiple assertions will

Definition: Unit test



```
testAddTheSameElementTwiceResultOneOccurrence
   | s |
   "Fixture"
   s := Set new.
   s add: 1.

   "Stimulus"
   s add: 1.

   "Assertions"
   self assert: s size equals: 1.
   self assert: (s includes: 1)
```

Definition: Smoke test

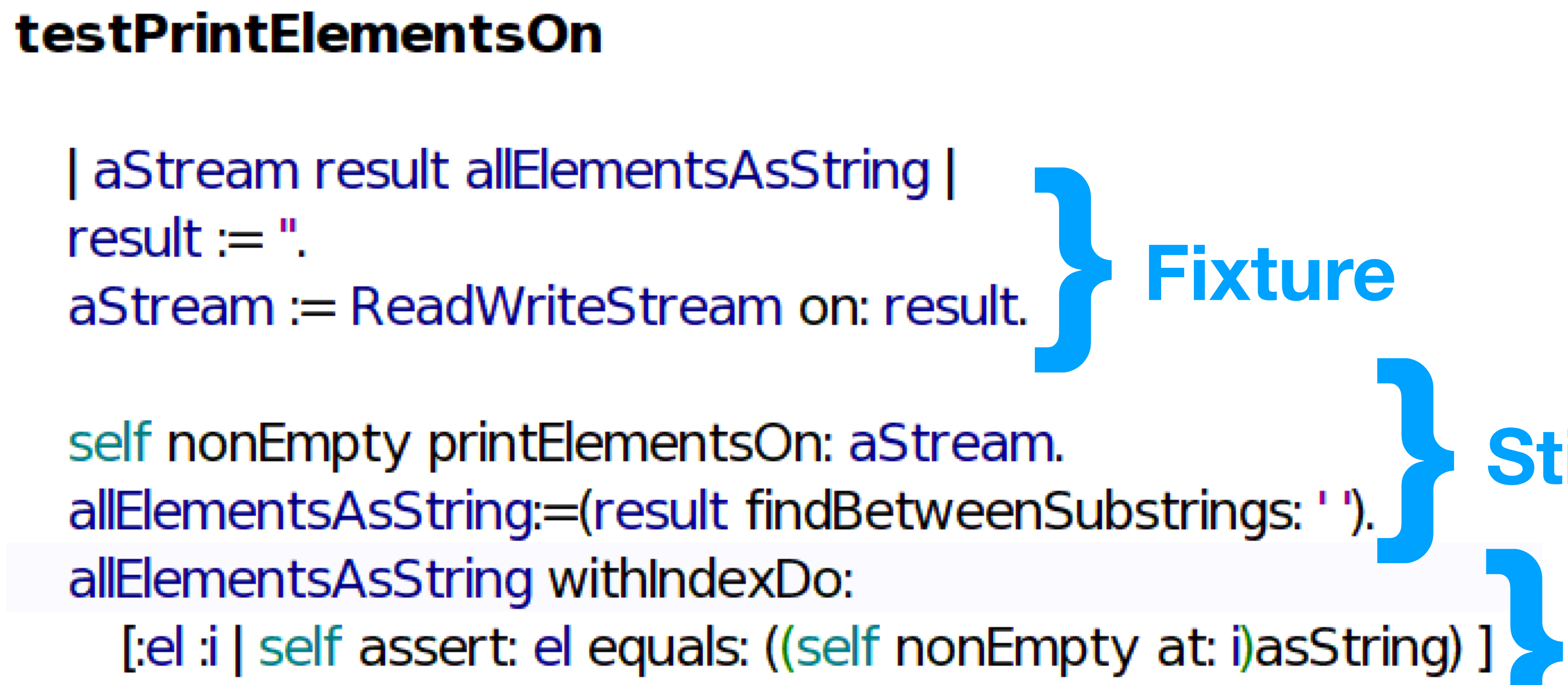


```
testSetAddSmokeTest
    | s |
    "Fixture"
    s := Set new.
    s add: 1.

    "Stimulus"
    s add: 1
```

The goal here is to ensure the source code can be run **without any exception thrown**

Definition: Rotten test

(1/2)



```
testPrintElementsOn

    | aStream result allElementsAsString |
    result := ''.
    aStream := ReadWriteStream on: result.

    self nonEmpty printElementsOn: aStream.
    allElementsAsString:=(result findBetweenSubstrings: ' ').
    allElementsAsString withIndexDo:
       [:el :i | self assert: el equals: ((self nonEmpty at: i)asString) ]
```

Fixture

Stimuli

Assertions

Additionally, the test is green so, what’s wrong?

Definition: Rotten test

(2/2)

```
testPrintElementsOn

    | aStream result allElementsAsString |
    result := ''.
    aStream := ReadWriteStream on: result.

    self nonEmpty printElementsOn: aStream.
    allElementsAsString:=(result findBetweenSubstrings: ' ').
    allElementsAsString withIndexDo:
       [:el :i | self halt.
          self assert: el equals: ((self nonEmpty at: i)asString) ]
```

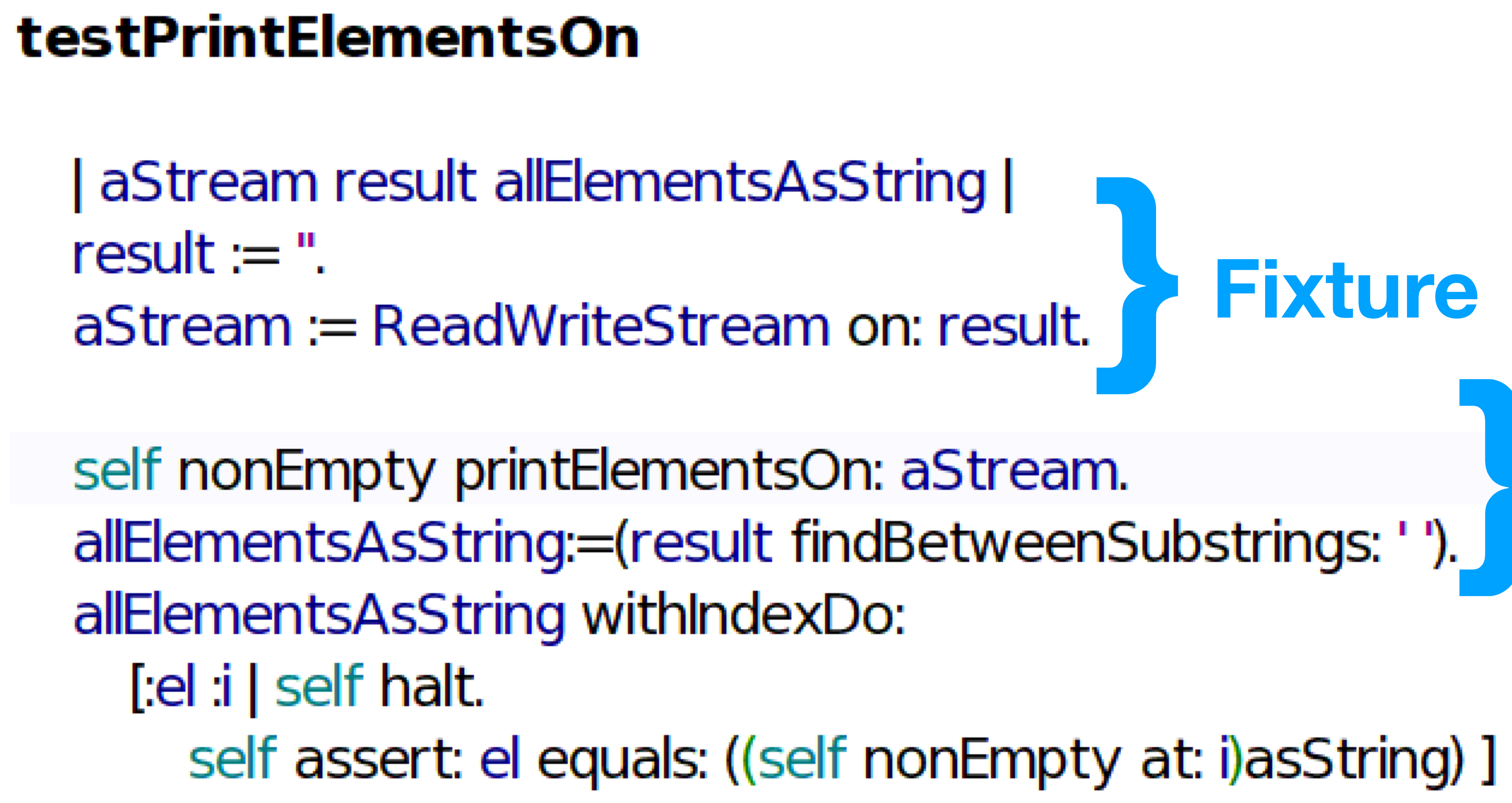
Fixture

Stimuli

Assertions

Definition: Rotten test

(2/2)



```
testPrintElementsOn

   | aStream result allElementsAsString |
   result := ''.
   aStream := ReadWriteStream on: result.

   self nonEmpty printElementsOn: aStream.
   allElementsAsString:=(result findBetweenSubstrings: ' ').
   allElementsAsString withIndexDo:
      [:el :i | self halt.
         self assert: el equals: ((self nonEmpty at: i)asString) ]
```

Fixture

Stimuli

Assertions

This modified version of the test is still green!

Definition: Rotten test

(2/2)

testPrintElementsOn

```
| aStream result allElementsAsString |
result := ''.
aStream := ReadWriteStream on: result.

self nonEmpty printElementsOn: aStream.
allElementsAsString:=(result findBetweenSubstrings: ' ').
allElementsAsString withIndexDo:
   [:el :i | self halt.
      self assert: el equals: ((self nonEmpty at: i)asString) ]
```

Fixture

Stimuli

Assertions

Not executed at runtime!

This modified version of the test is still green!

Definition:
Assertion primitive

A method of the unit-testing framework that performs the actual check.

#assert:

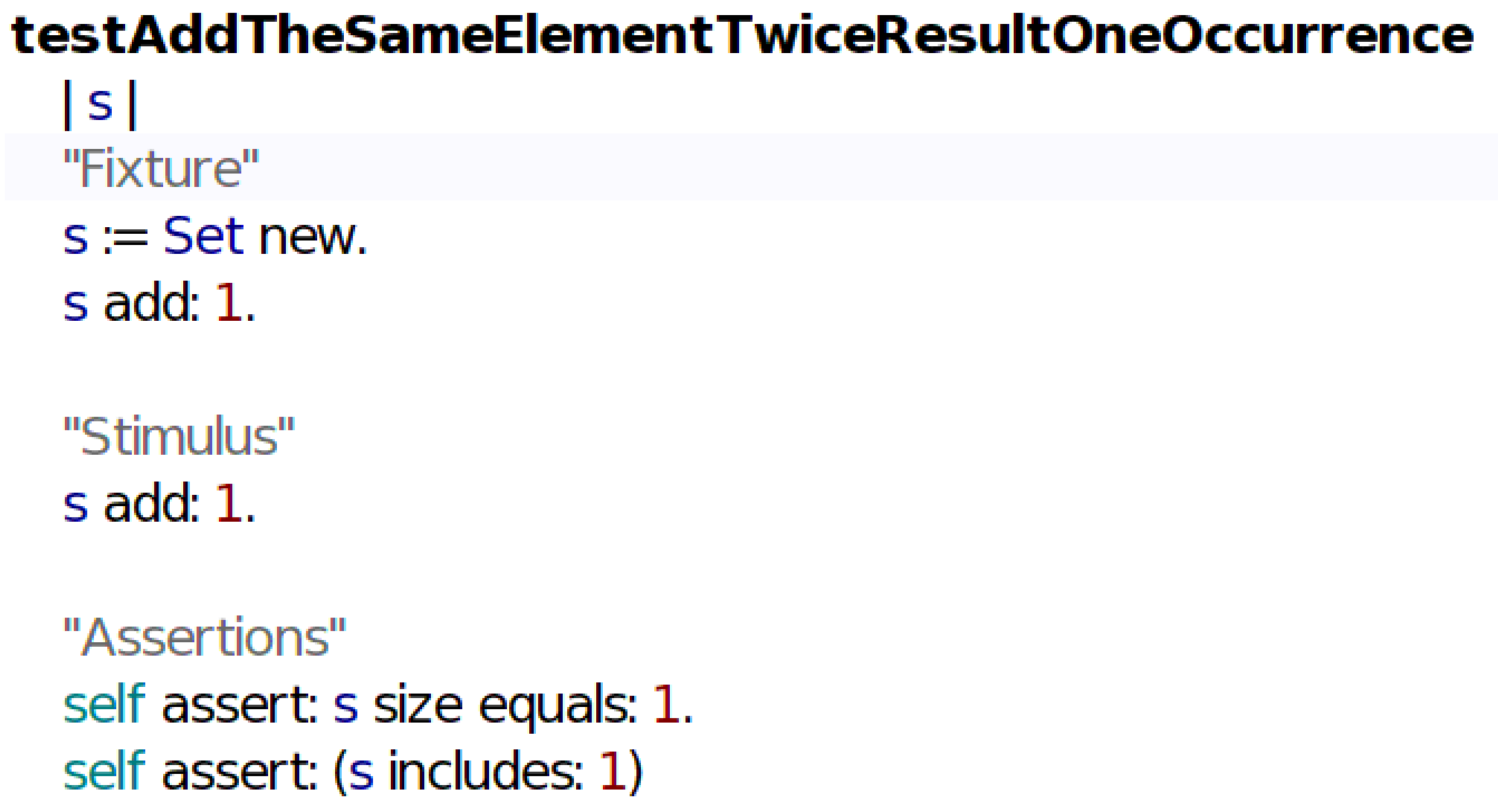
#assert:equals:

#deny:

⋮

Definition: Test

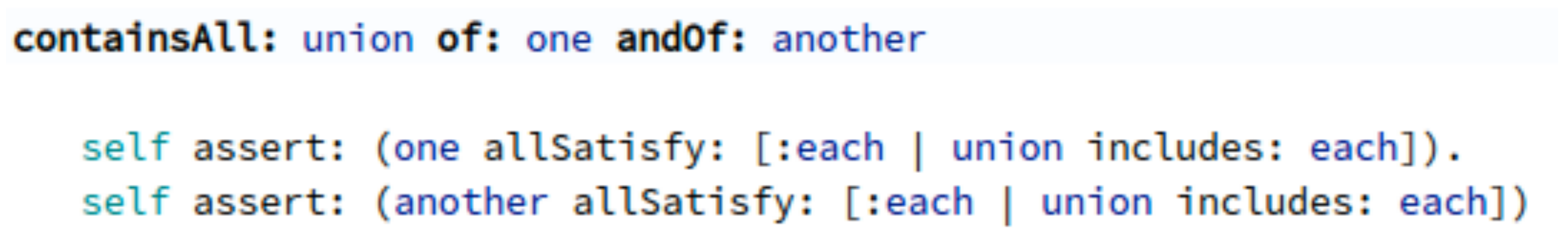
A method identified as such by the unit-testing framework. In Pharo, test methods are zero-argument methods defined in a subclass of TestCase whose names start with 'test'.



```
testAddTheSameElementTwiceResultOneOccurrence
  | s |
  "Fixture"
  s := Set new.
  s add: 1.

  "Stimulus"
  s add: 1.

  "Assertions"
  self assert: s size equals: 1.
  self assert: (s includes: 1)
```

Definition:
Test helper

A method that makes an assertion directly (by invoking an assertion primitive) or indirectly (by invoking another helper method), but that is not a test method.



```
containsAll: union of: one andOf: another

    self assert: (one allSatisfy: [:each | union includes: each]).
    self assert: (another allSatisfy: [:each | union includes: each])
```

Detect Rotten Tests

1. Identification of assertion primitives
2. Identification of helper methods
3. Install assertion primitives and helper methods call watchers
4. Test execution
5. Classification (good test, rotten test, smoke test)
6. Report generation

Detect Rotten Tests

Classification

	Dynamic Analysis		Static Analysis		
Row №	Helper Executed	Assertion Executed	Test contains helper	Test contains assertion	Classification
1	✓	✓	✓	✓	✓Good test
2	✗	✓	✓	✓	✗ Rotten test
3	✓	✗	✓	✓	✗ Rotten test & rotten helper
4	✗	✗	✓	✓	✗ Rotten test
5	✓	✓	✗	✓	✓Good test (dynamic helper invocation)
6	✗	✓	✗	✓	✓Good test
7	✓	✗	✗	✓	✗ Rotten test & rotten helper (dynamic helper invocation)
8	✗	✗	✗	✓	✗ Rotten test
9	✓	✓	✓	✗	✓Good test
10	✗	✓	✓	✗	✗ Rotten test (dynamic assert invocation)
11	✓	✗	✓	✗	✗ Rotten helper
12	✗	✗	✓	✗	✗ Rotten test
13	✓	✓	✗	✗	✓Good test (dynamic assertion & helper)
14	✗	✓	✗	✗	✓Good test (dynamic assertion invocation)
15	✓	✗	✗	✗	✓Good test (dynamic helper invocation)
16	✗	✗	✗	✗	✓Smoke test

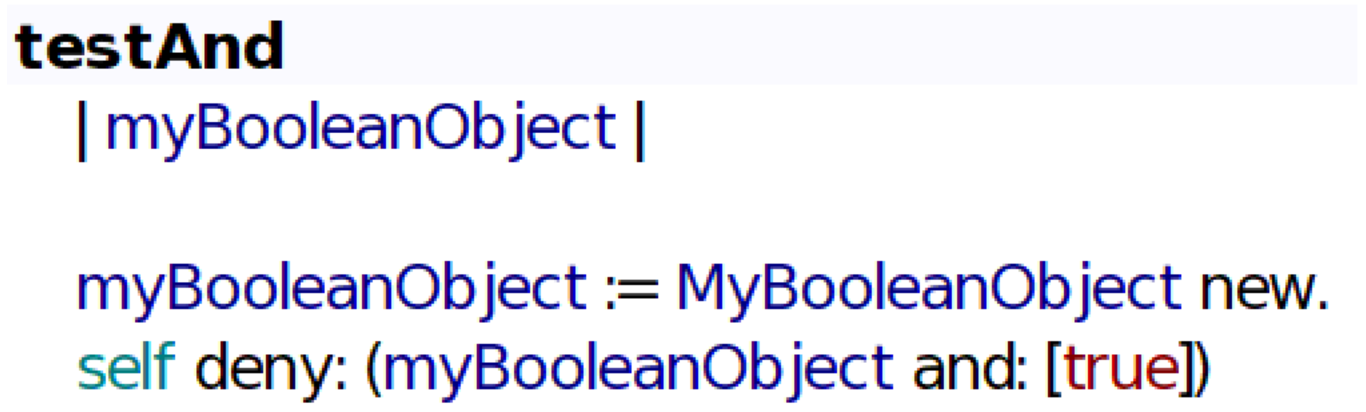
Detect Rotten Tests

Preliminary results

Subsystem	Packages	Classes	Test classes	Tests	Rotten tests
Calypso	58	705	128	2671	4
Collections	16	224	59	5858	7
Glamour	19	463	65	449	3
Iceberg	16	565	44	555	0
Opal Compiler	7	227	49	854	15
Pillar	33	358	112	3188	1
System	48	330	44	552	1
Zinc	9	184	43	412	3

A Vicious Rotten Test

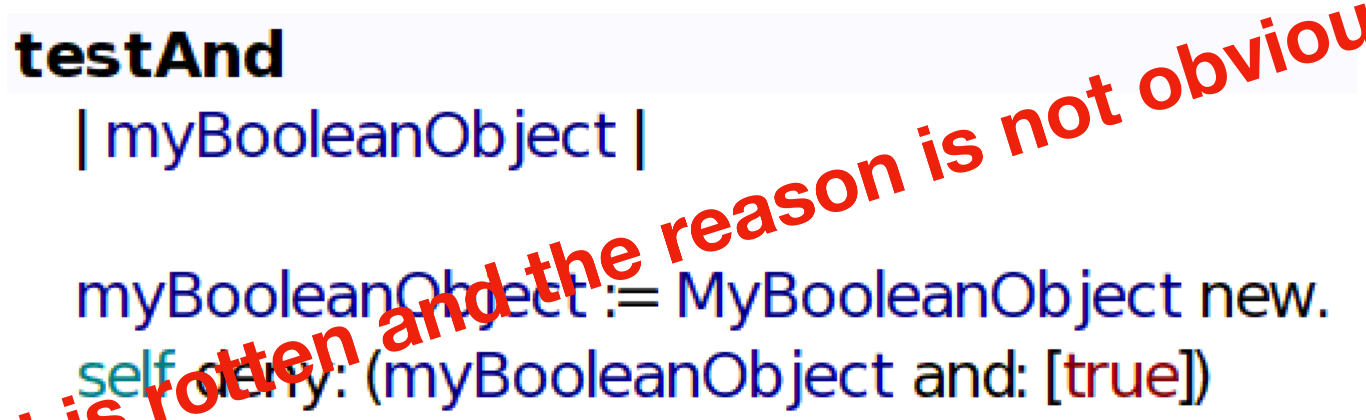
Pharo allows to use **any object** implementing the right interface as a *Boolean*. There are some tests for this feature in *MustBeBooleanTests.*



```
testAnd
    | myBooleanObject |

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

A Vicious Rotten Test

Pharo allows to use **any object** implementing the right interface as a *Boolean*. There are some tests for this feature in *MustBeBooleanTests.*



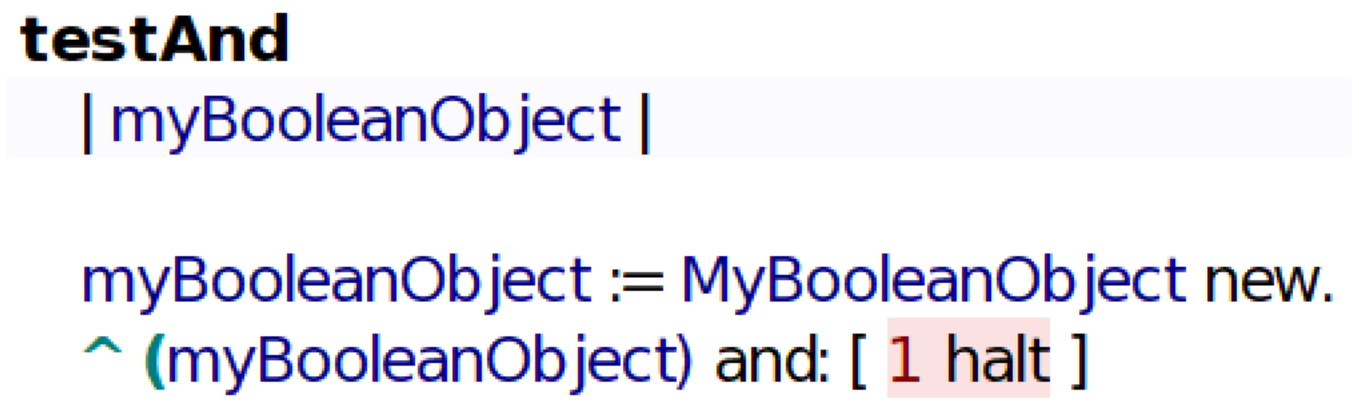
```
testAnd
    | myBooleanObject |

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

This test is rotten and the reason is not obvious at all!

A Vicious Rotten Test

Common boolean methods (e.g. #ifTrue:) are compiled to optimized bytecode which raise an exception when they are evaluated for non-booleans.

Pharo dynamically catches this exception and it rewrites this with a de-optimization allowing to use the receiver as a boolean.



```
testAnd
    | myBooleanObject |

    myBooleanObject := MyBooleanObject new.
    ^ (myBooleanObject) and: [ 1 halt ]
```

It seems that there is a bug in this process, de-optimized source code generated is **incorrect**.

Future Work

- Watch assertion primitives/helpers call at AST-node level
- Run the finder on more Pharo projects

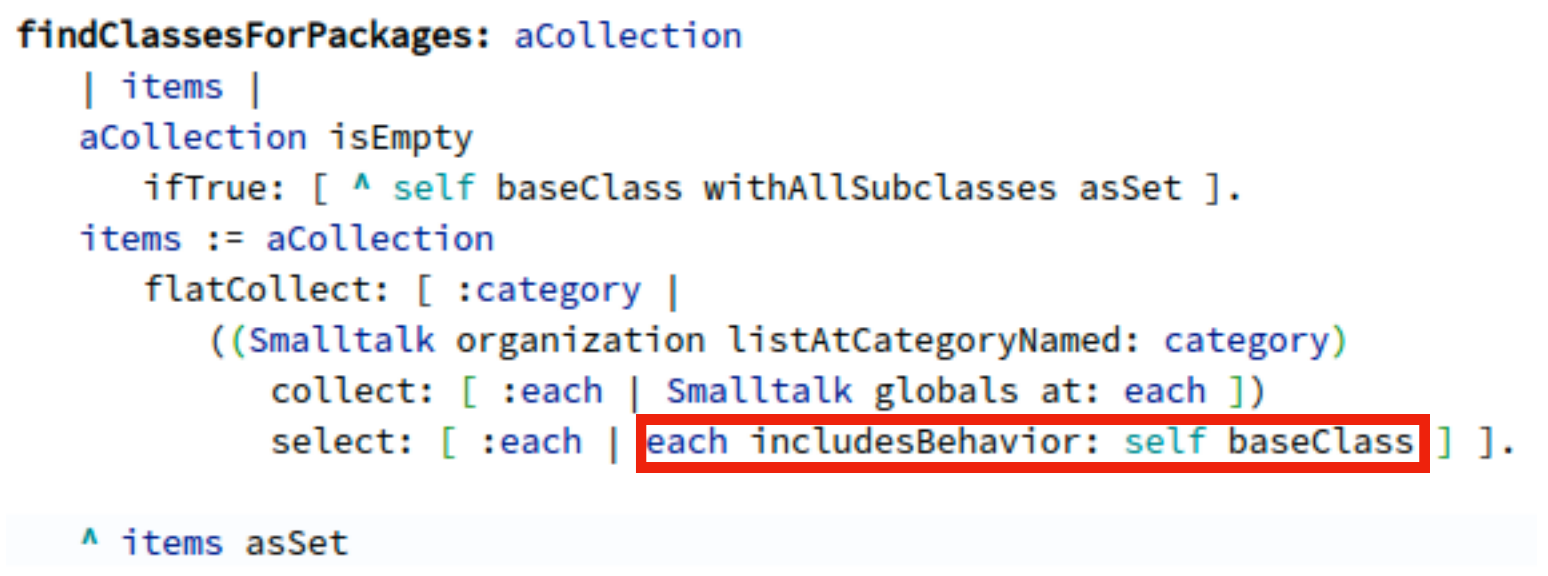
Testing in Pharo

Discussion

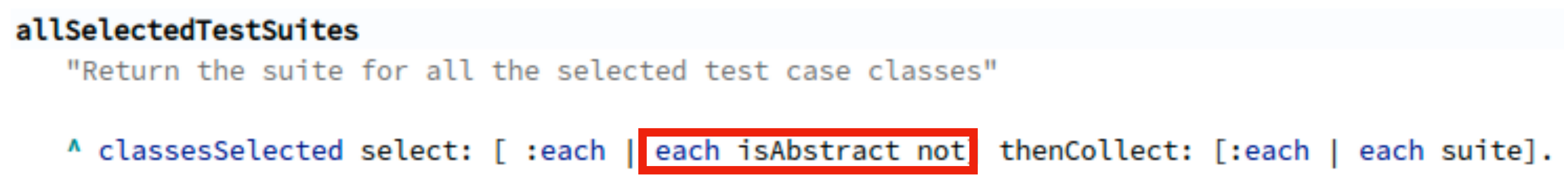
Observations on SUnit

- The API that should be used is not always well defined (*TestRunner*, *TestCommandLineHandler*, *Calypso*, etc… sometimes behave differently because of that)
- The TestRunner UI is not easily extensible

SUnit API:
How to visit tests in a package?



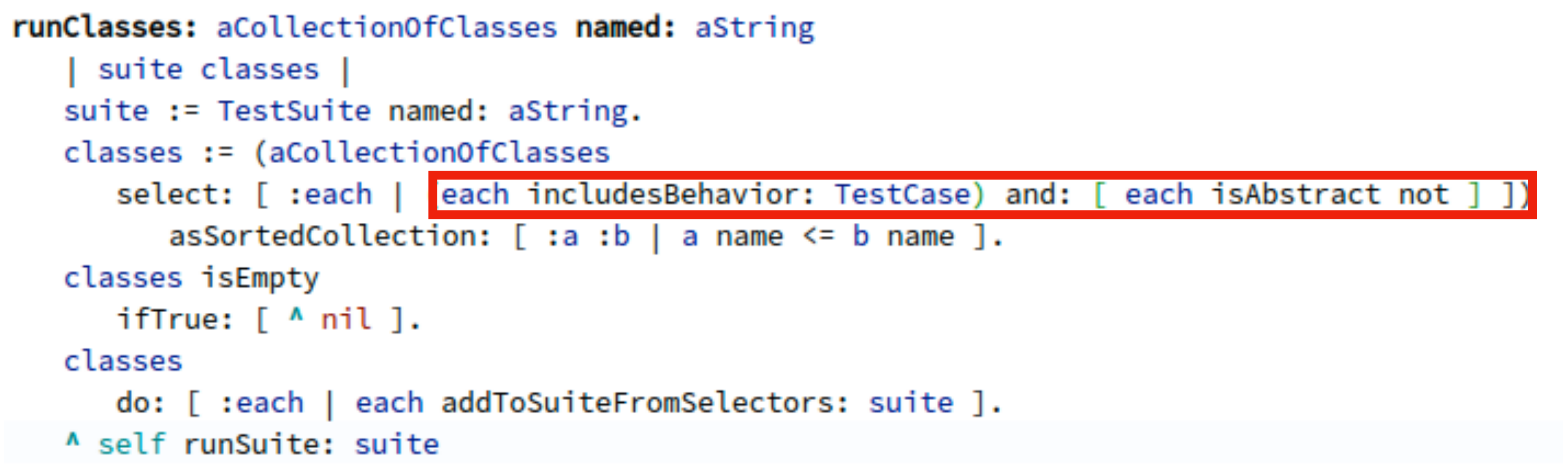
```
findClassesForPackages: aCollection
    | items |
    aCollection isEmpty
        ifTrue: [ ^ self baseClass withAllSubclasses asSet ].
    items := aCollection
        flatCollect: [ :category |
            ((Smalltalk organization listAtCategoryNamed: category)
                collect: [ :each | Smalltalk globals at: each ])
                select: [ :each | each includesBehavior: self baseClass ] ].

    ^ items asSet
```



```
allSelectedTestSuites
    "Return the suite for all the selected test case classes"

    ^ classesSelected select: [ :each | each isAbstract not] thenCollect: [:each | each suite].
```

TestRunner

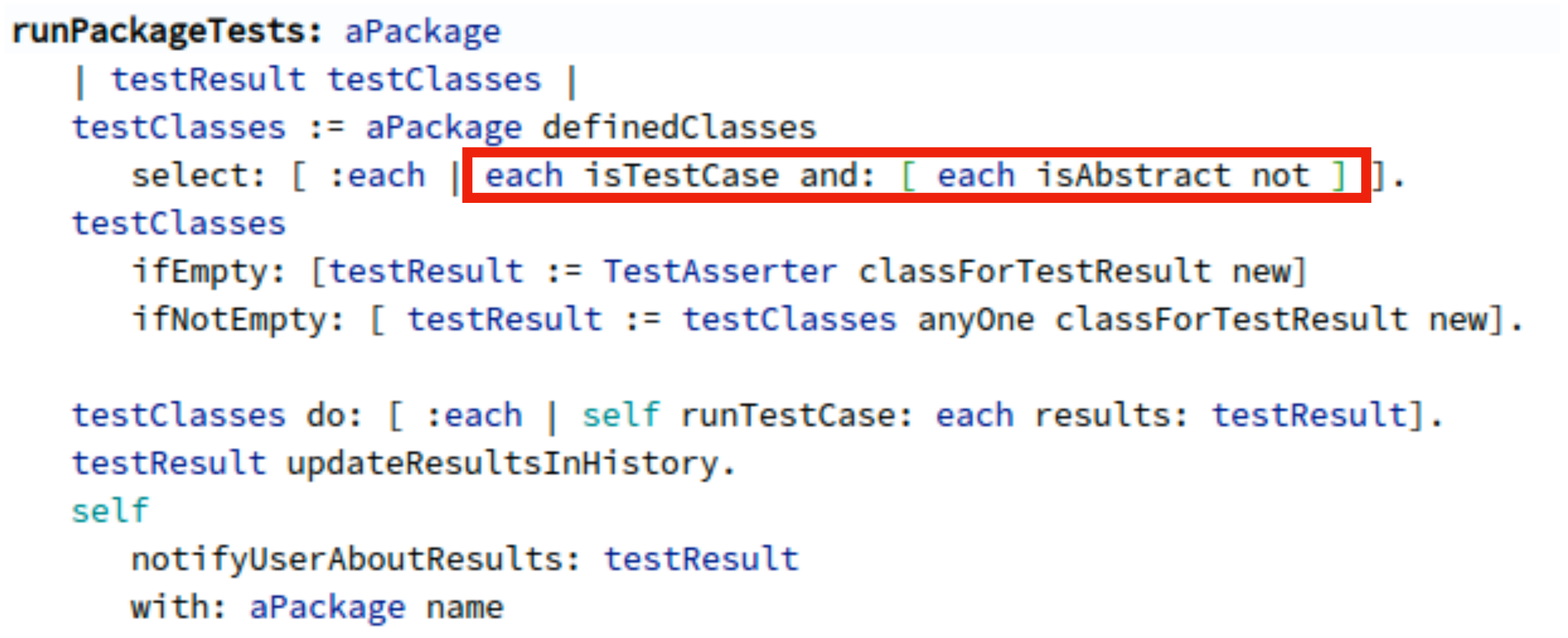
SUnit API: How to visit tests in a package?



```
runClasses: aCollectionOfClasses named: aString
    | suite classes |
    suite := TestSuite named: aString.
    classes := (aCollectionOfClasses
        select: [ :each |  each includesBehavior: TestCase) and: [ each isAbstract not ] ])
            asSortedCollection: [ :a :b | a name <= b name ].
    classes isEmpty
        ifTrue: [ ^ nil ].
    classes
        do: [ :each | each addToSuiteFromSelectors: suite ].
    ^ self runSuite: suite
```

CommandLineTestRunner

SUnit API:
How to visit tests in a package?

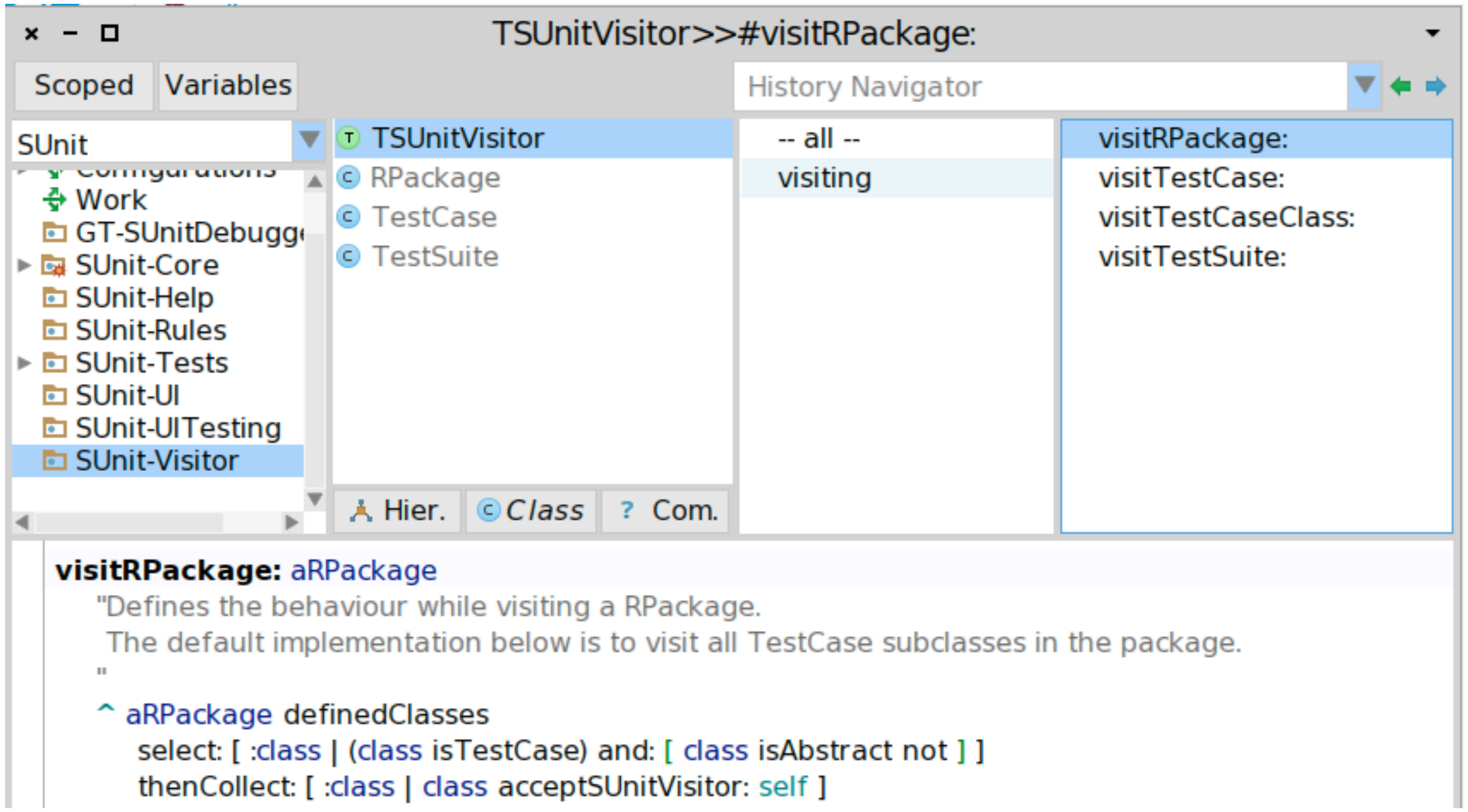


```
runPackageTests: aPackage
   | testResult testClasses |
   testClasses := aPackage definedClasses
      select: [ :each | each isTestCase and: [ each isAbstract not ] ].
   testClasses
      ifEmpty: [testResult := TestAsserter classForTestResult new]
      ifNotEmpty: [ testResult := testClasses anyOne classForTestResult new].

   testClasses do: [ :each | self runTestCase: each results: testResult].
   testResult updateResultsInHistory.
   self
      notifyUserAboutResults: testResult
      with: aPackage name
```

Calypso

SUnit API:
How to visit tests in a package?

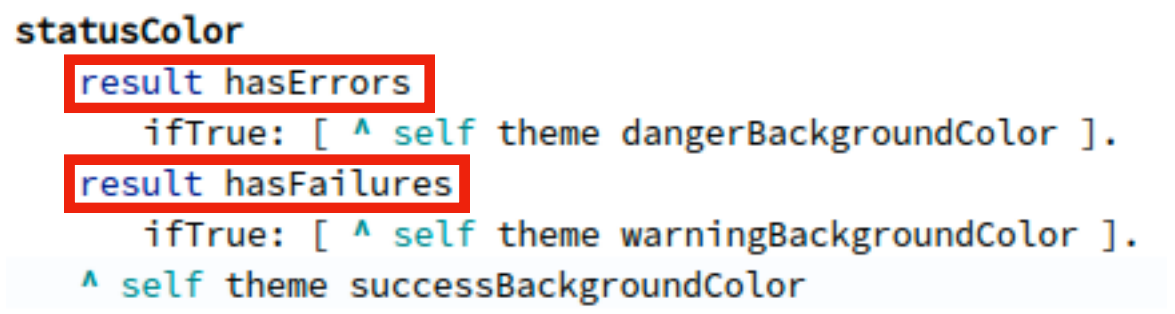
Possible solution is to create a visitor.



```
visitRPackage: aRPackage
    "Defines the behaviour while visiting a RPackage.
     The default implementation below is to visit all TestCase subclasses in the package.
    "
    ^ aRPackage definedClasses
        select: [ :class | (class isTestCase) and: [ class isAbstract not ] ]
        thenCollect: [ :class | class acceptSUnitVisitor: self ]
```

https://github.com/juliendelplanque/SUnit-Visitor

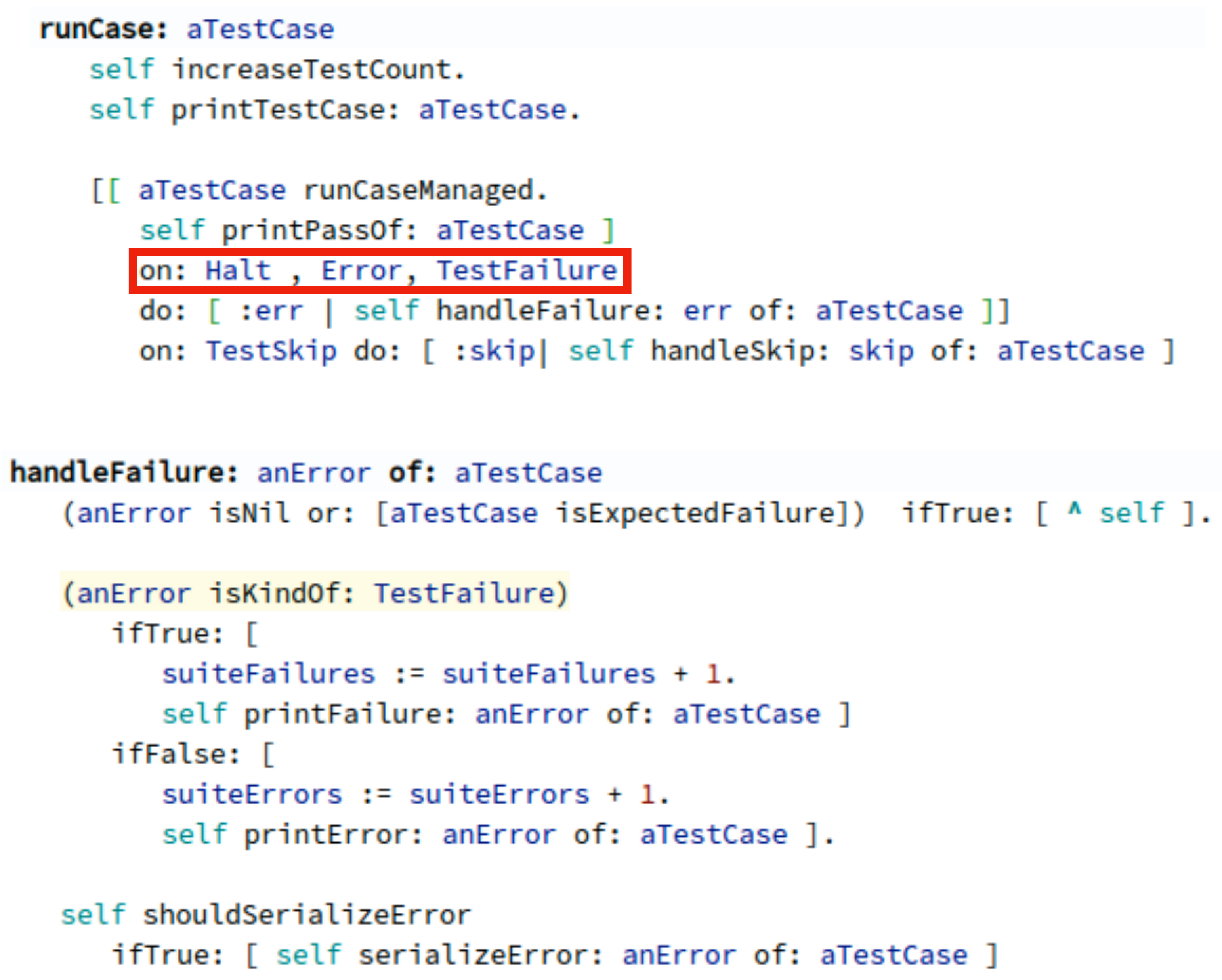
SUnit API:
How to manage exceptions?



```
statusColor
    result hasErrors
        ifTrue: [ ^ self theme dangerBackgroundColor ].
    result hasFailures
        ifTrue: [ ^ self theme warningBackgroundColor ].
    ^ self theme successBackgroundColor
```

TestRunner

SUnit API:
How to manage exceptions?

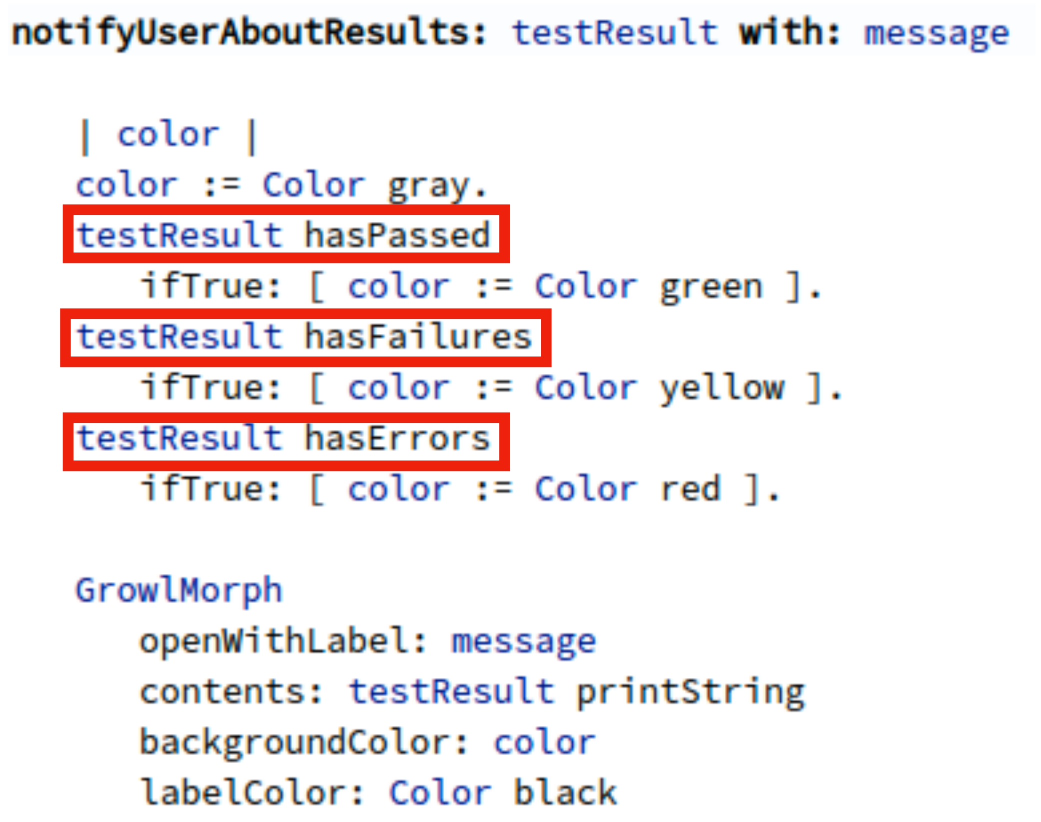


```
runCase: aTestCase
    self increaseTestCount.
    self printTestCase: aTestCase.

    [[ aTestCase runCaseManaged.
       self printPassOf: aTestCase ]
       on: Halt , Error, TestFailure
       do: [ :err | self handleFailure: err of: aTestCase ]]
       on: TestSkip do: [ :skip| self handleSkip: skip of: aTestCase ]
```

```
handleFailure: anError of: aTestCase
    (anError isNil or: [aTestCase isExpectedFailure])  ifTrue: [ ^ self ].

    (anError isKindOf: TestFailure)
        ifTrue: [
            suiteFailures := suiteFailures + 1.
            self printFailure: anError of: aTestCase ]
        ifFalse: [
            suiteErrors := suiteErrors + 1.
            self printError: anError of: aTestCase ].

    self shouldSerializeError
        ifTrue: [ self serializeError: anError of: aTestCase ]
```

CommandLineTestRunner

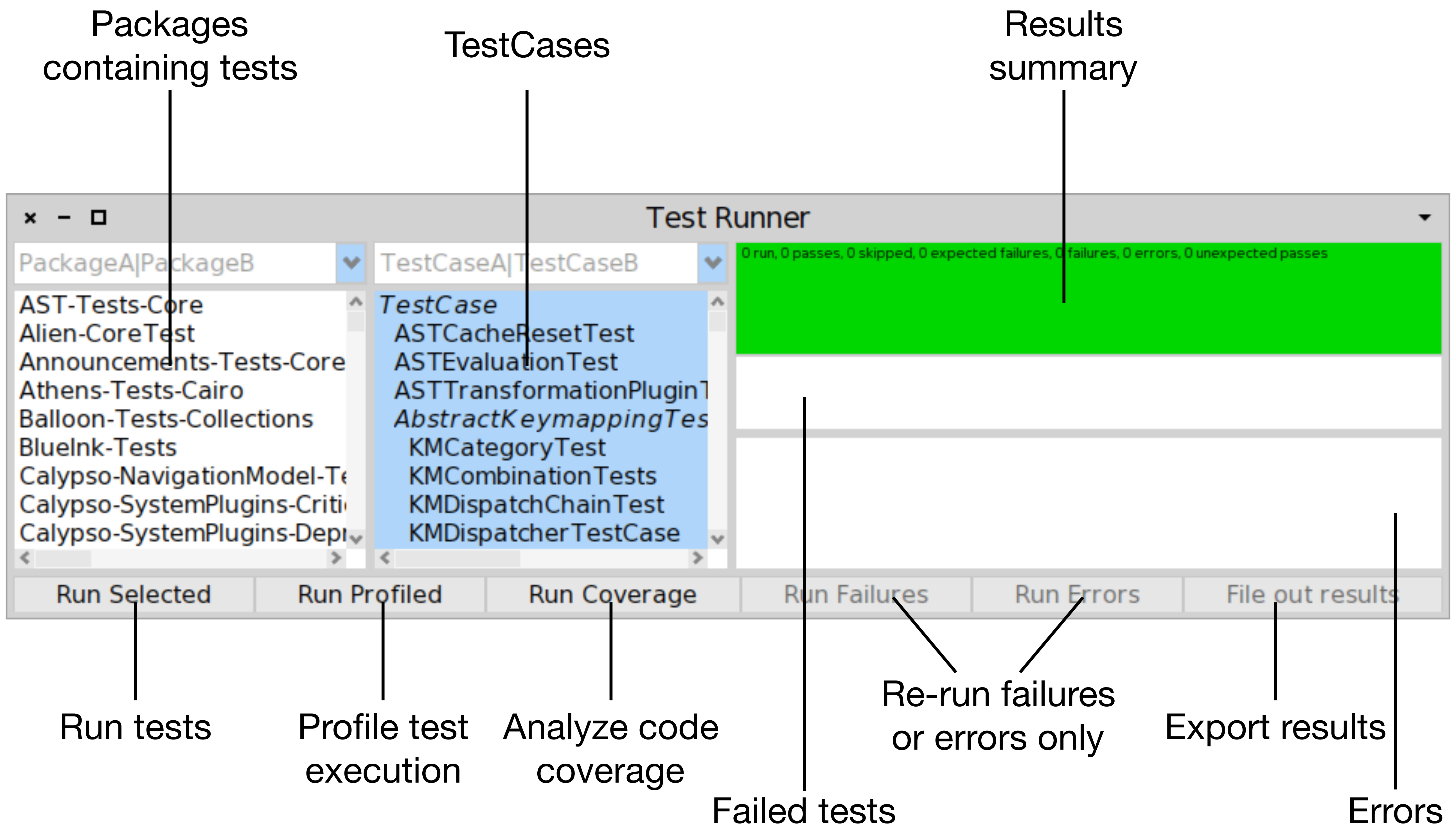
SUnit API:
How to manage exceptions?



```
notifyUserAboutResults: testResult with: message

  | color |
  color := Color gray.
  testResult hasPassed
     ifTrue: [ color := Color green ].
  testResult hasFailures
     ifTrue: [ color := Color yellow ].
  testResult hasErrors
     ifTrue: [ color := Color red ].

  GrowlMorph
     openWithLabel: message
     contents: testResult printString
     backgroundColor: color
     labelColor: Color black
```

Calypso

TestRunner UI



```
AST-Tests-Core
Alien-CoreTest
Announcements-Tests-Core
Athens-Tests-Cairo
Balloon-Tests-Collections
BlueInk-Tests
Calypso-NavigationModel-Te
Calypso-SystemPlugins-Criti
Calypso-SystemPlugins-Depi
```

```
TestCase
  ASTCacheResetTest
  ASTEvaluationTest
  ASTTransformationPluginT
  AbstractKeymappingTes
    KMCategoryTest
    KMCombinationTests
    KMDispatchChainTest
    KMDispatcherTestCase
```

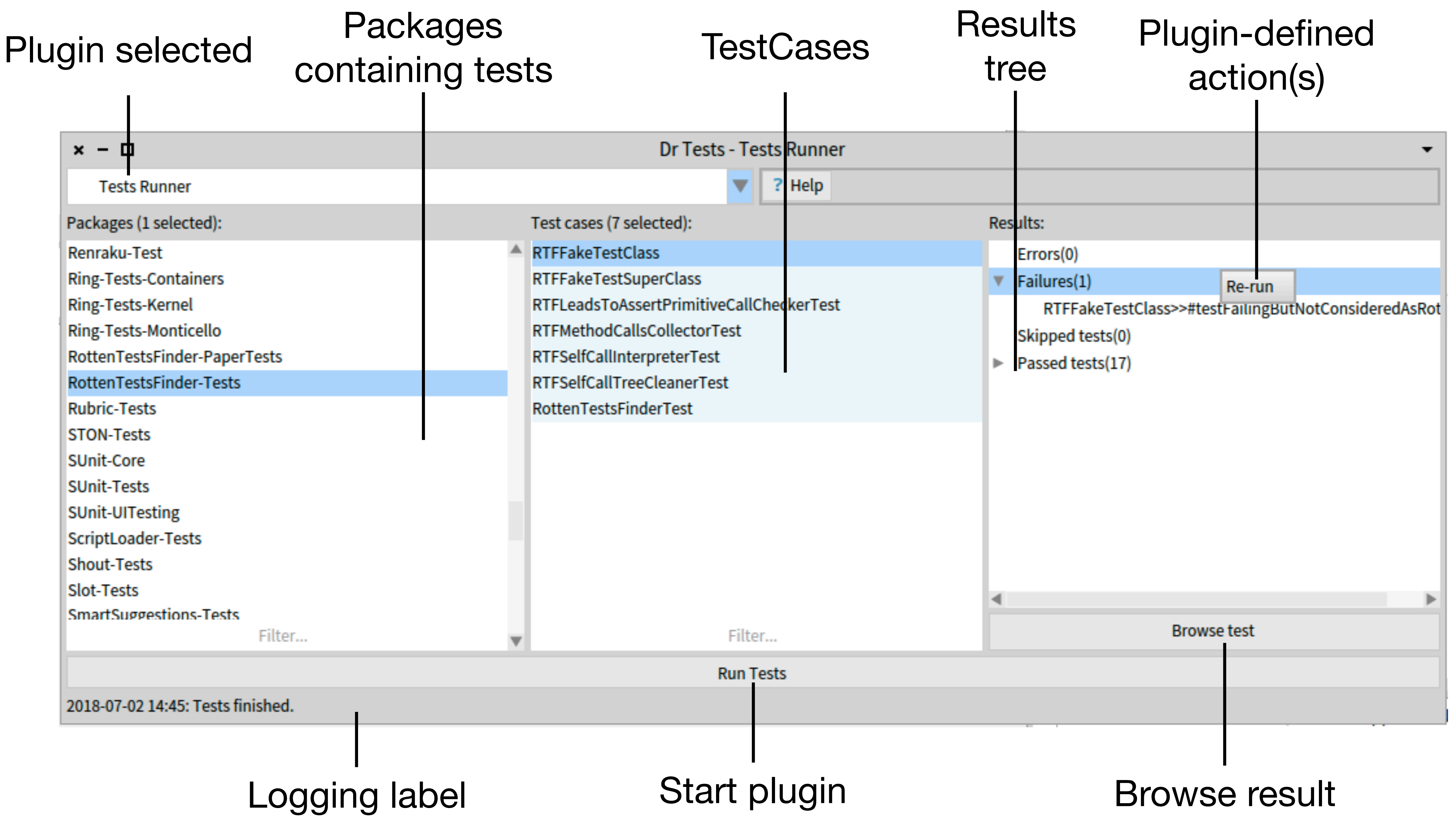
```
0 run, 0 passes, 0 skipped, 0 expected failures, 0 failures, 0 errors, 0 unexpected passes
```

TestRunner UI discussion

Multiple things can be done around tests in Pharo environment:

- Run tests
- Profile tests execution } **Covered by TestRunner**
- Analyse code coverage
- Mutation testing
- Find rotten tests
- Analyse code example in comments
- …

Proposal: DrTests



Dr Tests - Tests Runner

Tests Runner

? Help

Packages (1 selected):

Renraku-Test
Ring-Tests-Containers
Ring-Tests-Kernel
Ring-Tests-Monticello
RottenTestsFinder-PaperTests
RottenTestsFinder-Tests
Rubric-Tests
STON-Tests
SUnit-Core
SUnit-Tests
SUnit-UITesting
ScriptLoader-Tests
Shout-Tests
Slot-Tests
SmartSuggestions-Tests

Filter...

Test cases (7 selected):

RTFFakeTestClass
RTFFakeTestSuperClass
RTFLeadsToAssertPrimitiveCallCheckerTest
RTFMethodCallsCollectorTest
RTFSelfCallInterpreterTest
RTFSelfCallTreeCleanerTest
RottenTestsFinderTest

Filter...

Results:

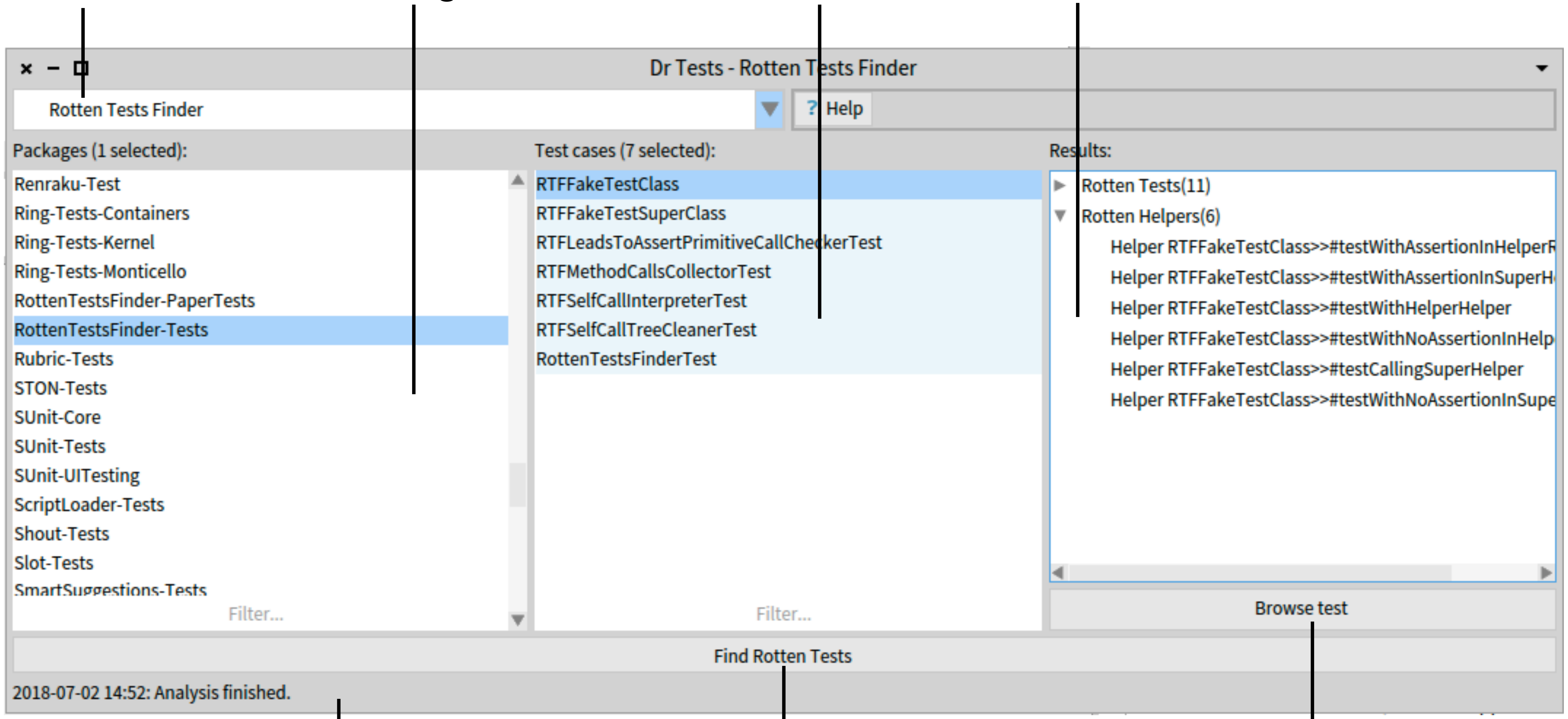
Errors(0)
Failures(1) Re-run
RTFFakeTestClass>>#testFailingButNotConsideredAsRot
Skipped tests(0)
Passed tests(17)

Browse test

Run Tests

2018-07-02 14:45: Tests finished.

Proposal: DrTests



Dr Tests - Rotten Tests Finder

Rotten Tests Finder ? Help

Packages (1 selected):

- Renraku-Test
- Ring-Tests-Containers
- Ring-Tests-Kernel
- Ring-Tests-Monticello
- RottenTestsFinder-PaperTests
- RottenTestsFinder-Tests
- Rubric-Tests
- STON-Tests
- SUnit-Core
- SUnit-Tests
- SUnit-UITesting
- ScriptLoader-Tests
- Shout-Tests
- Slot-Tests
- SmartSuggestions-Tests

Filter...

Test cases (7 selected):

- RTFFakeTestClass
- RTFFakeTestSuperClass
- RTFLeadsToAssertPrimitiveCallCheckerTest
- RTFMethodCallsCollectorTest
- RTFSelfCallInterpreterTest
- RTFSelfCallTreeCleanerTest
- RottenTestsFinderTest

Filter...

Results:

- ▶ Rotten Tests(11)
- ▼ Rotten Helpers(6)
  - Helper RTFFakeTestClass>>#testWithAssertionInHelperR
  - Helper RTFFakeTestClass>>#testWithAssertionInSuperH
  - Helper RTFFakeTestClass>>#testWithHelperHelper
  - Helper RTFFakeTestClass>>#testWithNoAssertionInHelp
  - Helper RTFFakeTestClass>>#testCallingSuperHelper
  - Helper RTFFakeTestClass>>#testWithNoAssertionInSupe

Browse test

Find Rotten Tests

2018-07-02 14:52: Analysis finished.