SUnit

Once upon a time there were three classes ...

- TestCase
- TestSuite
- TestResult

... and then there was a fourth

— TestResource

This talk is about:

- **(mainly) TestResources in SUnit 3.2**
- **(briefly) SUnit status**

TestResource is an optimisation

Make it run

- **Kent Beck's rules of optimisation**
  - — Rule 1: do it later
  - — Rule 2: see rule 1

Make it right

- **eXtreme Programming practices**
  - — Test-driven development
    - — "But my tests are too slow."
  - — Refactoring
    - — "So refactor your tests to be fast."

Make it fast

- **enter TestResource**
  - —and (somewhat later) explanations of it

Niall Ross

Problems with TestResource

XP style: "Do it later" / "You won't need it":

- **Every resource set up before any test is run**
- **If one resource of one test in a suite of 15,000+ fails …**
  - … the run does nothing – not what you want to see when next you look

XP Style: refactoring + "last make it fast"

```
MyTestCase>>setUp
    …
    self assert: databaseSession isOnline description: 'not online'.
```

Tests getting slow? Refactor to a TestResource.

```
MyTestResource(Object)>>doesNotUnderstand: #assert:description:
```

Niall Ross

Problems with TestResources

Resources can compete with other resources:

- **e.g. connect to one DB at a time, several DBs to test**
- **I coded the CompetingResource pattern:**
  - in SUnit 3.1 and earlier, not easy !
  - Stephane D and Martin K also had patterns – also not easy

Resources can rely on other resources:

- **Tests (and resources) can have ordered resources**

```
MyTestCase class>>resources
    ^Array wth: ConnectToDBResource with: AddTestDataToDBResource
```

- **resource setUp (tearDown) *not* in order (reverse order)**
- **resource setUp / tearDown *after* resource that needs it**

Niall Ross

What has changed in TestResource

Resources are made available just-in-time:

- **first test that needs it prompts set up**
- **later tests that need it see it has (or failed to) set up**
- **tearDown guaranteed at end of run; can be done anytime**
  - — resetting in a test's tearDown trades performance for test isolation

Resources understand #assert:… protocol

- **setUp and isAvailable run inside the handler**
  - — in end-run tearDown, #assert: is just better protocol for same behaviour

Resource-processing is ordered

- — a test's resources setUp in order and tearDown in reverse order
- — a resource's resources setUp before it and tearDown after it

Code changes: just-in-time resourcing

```
TestCase>>runCase
  self resources do:
        [:each |
        self assert: each isAvailable
             description: 'Unavailable resource ', each name,
                          ' requested by test ', self printString].
  [self setUp.
  self performTest] sunitEnsure: [self tearDown].

TestSuite>>run
  | result | result := TestResult new.
  self resources do:
        [ :each | each isAvailable ifFalse: [^each signalInitializationError]].
  [self run: result]
        sunitEnsure: [self resources reverseDo: [:each | each reset]].
  ^result
```

Niall
Ross

Code changes: 3-valued logic for ‘current’

```
TestResource class>>isAvailable
  current isNil ifTrue: [self makeAvailable].
  ^self isAlreadyAvailable

TestResource class>>makeAvailable
  | candidate |
  current := false.  "any object not nil and not an instance of me would do"
  self resources do:
          [:each |
          self assert: each isAvailable
                description: 'Unavailable resource ', each name,
                             ' requested by resource ', self name].
  candidate := self new.
  candidate isAvailable ifTrue: [current := candidate].

TestResource class>>isAlreadyAvailable
          ^current class == self
```

Niall Ross

Class changes

Once upon a time there were three classes ...

TestCase, TestSuite and TestResult

... and then there was a fourth ...

TestResource

... and now a fifth ...

TestAsserter : abstract superclass of

- TestCase
- TestResource
- any user-created TestCase delegate class

(... and that’s enough !)

Niall Ross

Any impact on Users ?

Logging

- **TestCase methods moved to the class-side**
  - #isLogging, and #failureLog (and #logFailure: is on both sides)

(So, who here overrides #isLogging or #failureLog ?)

Profiling

- **a test… method's time: no impact**
- **a test suite's overall time: no impact**
- **a test's time in #runCase: sometimes see resource time**
  - time moved from start of suite's #run to start of (some) tests' #runCase

Any objections, voice them now !

SUnit 3.2

Make your tests run

Make your tests right

Make your tests fast

(resources can help)

Thanks to Yuri Mironenko, Dale Henrichs, James Foster, Tim MacKinnon for helping me port to Squeak, Gemstone and Dolphin.

SUnit and Friends

SUnit: cross-dialect, backward-compatible, 3-5 classes

- Add-ons: SUnitXProcPatterns, SUnitResourcePatterns, etc.
- UIs: RBSUnitExtensions SUnitBrowser, Squeak TestRunner, etc.

SUnitToo: VW-specific, experimental, 11 classes

- SUnit-Bridge2SU2 maps SUnit tests to SUnitToo tests

Assessments: VW-specific, configurable, 40+ classes

- transparent bridges configurable for SUnit, SUnitToo, etc.

GemStone's test framework

…

SUnit wants ideas

SUnit will remain cross-dialect, backward-compatible, small