Unit Testing for Dummies
Writing Unit Tests

PUG UK & Ireland

Making Better Progress.....

Mike Fechner, Consultingwerk Ltd.
James Palmer, Consultingwerk Ltd.
james.palmer@consultingwerk.de

Mike Fechner

James Palmer

Consultingwerk Ltd.

Progress Partner
PREMIER SERVICE DELIVERY
#ProgressProud

- Independent IT consulting organization
- Focusing on **OpenEdge** and **related technology**
- Located in Cologne, Germany, subsidiaries in UK and Romania
- Customers in Europe, North America, Australia and South Africa
- Vendor of developer tools and consulting services
- 28 years of Progress experience (V5 … OE11)
- Specialized in GUI for .NET, Angular, OO, Software Architecture, Application Integration

Agenda

- **Introduction**
- A simple ABL Unit Test
- Structure of a Unit Test
- Unit Testing Tooling
- Writing testable code
- Mocking dependencies
- Dealing with Data

Introduction

- Developer of **SmartComponent Library** Framework for OpenEdge Developers
- Source code shipped to clients, 99% ABL code
- Used by over 25 customers
- Up to weekly releases (customers usually during development on a release not older than 3 month)
- Fully automated update of the framework DB at client
- Almost no regression bugs within last 10 years
- Can only keep up that pace due to automation

From a recent real world example

- Windows 10 Creators Upgrate (April 2017) breaks INPUT THROUGH statements from Progress 8.3 - OpenEdge 11.7
- Used in a method to verify email addresses (MX record lookup), manual test of that functionality not likely
- Jenkins Job alerted us around noon after the Windows update was applied to the build server
- Only two days later, discussions around the issue on StackOverflow, Progress Communities and later in PANS

Unit Tests saved the day! As we had a fix in place already!

From a recent real world example

- A pretty simple API got broken; caused by a Windows update
- No matter if it’s Progress’ fault or Microsoft – it’s a 3rd party
- We execute our Unit Tests on OpenEdge 10.2B, 11.3, 11.6 and 11.7
- We execute our Unit Tests on Windows 10 and Linux (VMware)
- Considering to add additional Windows Versions in VM’s because of the Easter 2017 experience

Introduction

- *“In computer programming, unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use.”*, Wikipedia
- A Unit should be considered the smallest testable component
- Unit Tests may be automated
- Automated Unit Tests simplify regression testing
- Write test once, execute for a life time

Agenda

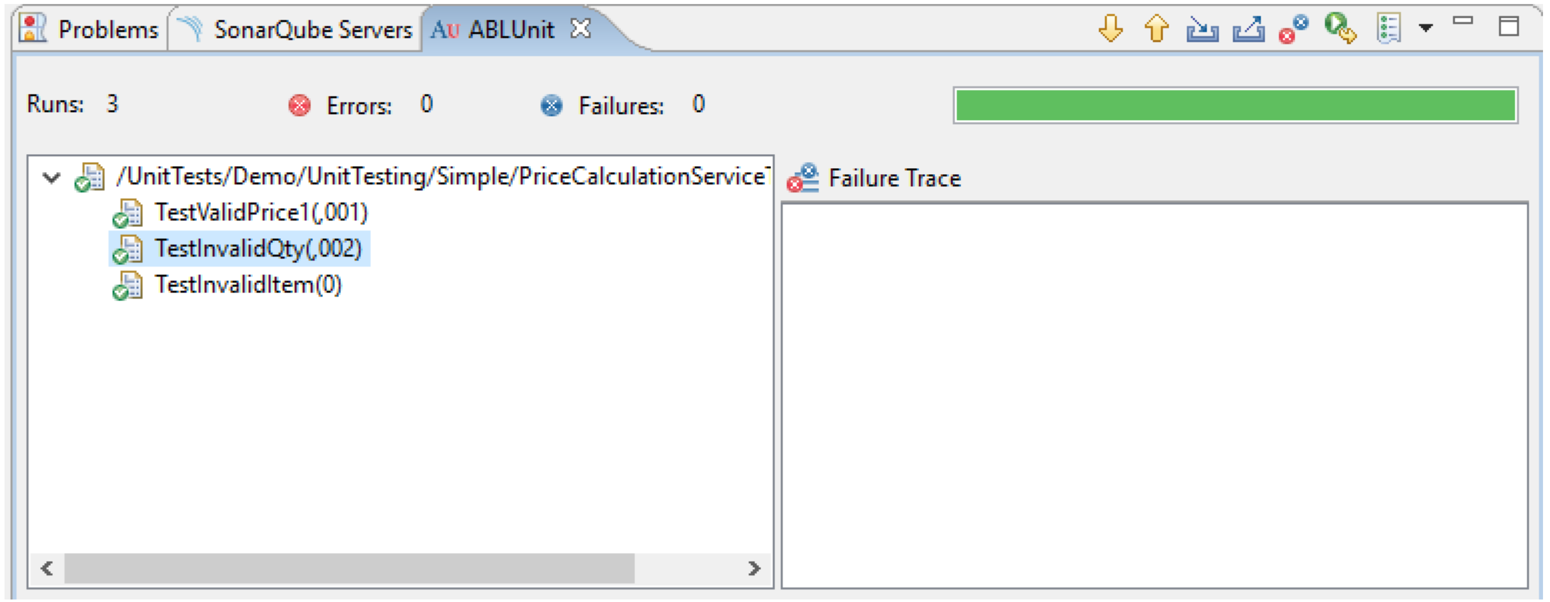
- Introduction
- **A simple ABL Unit Test**
- Structure of a Unit Test
- Unit Testing Tooling
- Writing testable code
- Mocking dependencies
- Dealing with Data

Demo

- Creating Unit Test Project
- Writing a simple test class

Demo

- Execute Unit Test in ABLUnit
- ABL Unit Launch Configuration in PDSOE
- ABLUnit View / Perspective
- Executing a single Unit Test Method



Runs: 3 Errors: 0 Failures: 0

/UnitTests/Demo/UnitTesting/Simple/PriceCalculationService
TestValidPrice1(,001)
TestInvalidQty(,002)
TestInvalidItem(0)

Agenda

- Introduction
- A simple ABL Unit Test
- **Structure of a Unit Test**
- Unit Testing Tooling
- Writing testable code
- Mocking dependencies
- Dealing with Data

Structure of a Unit Test

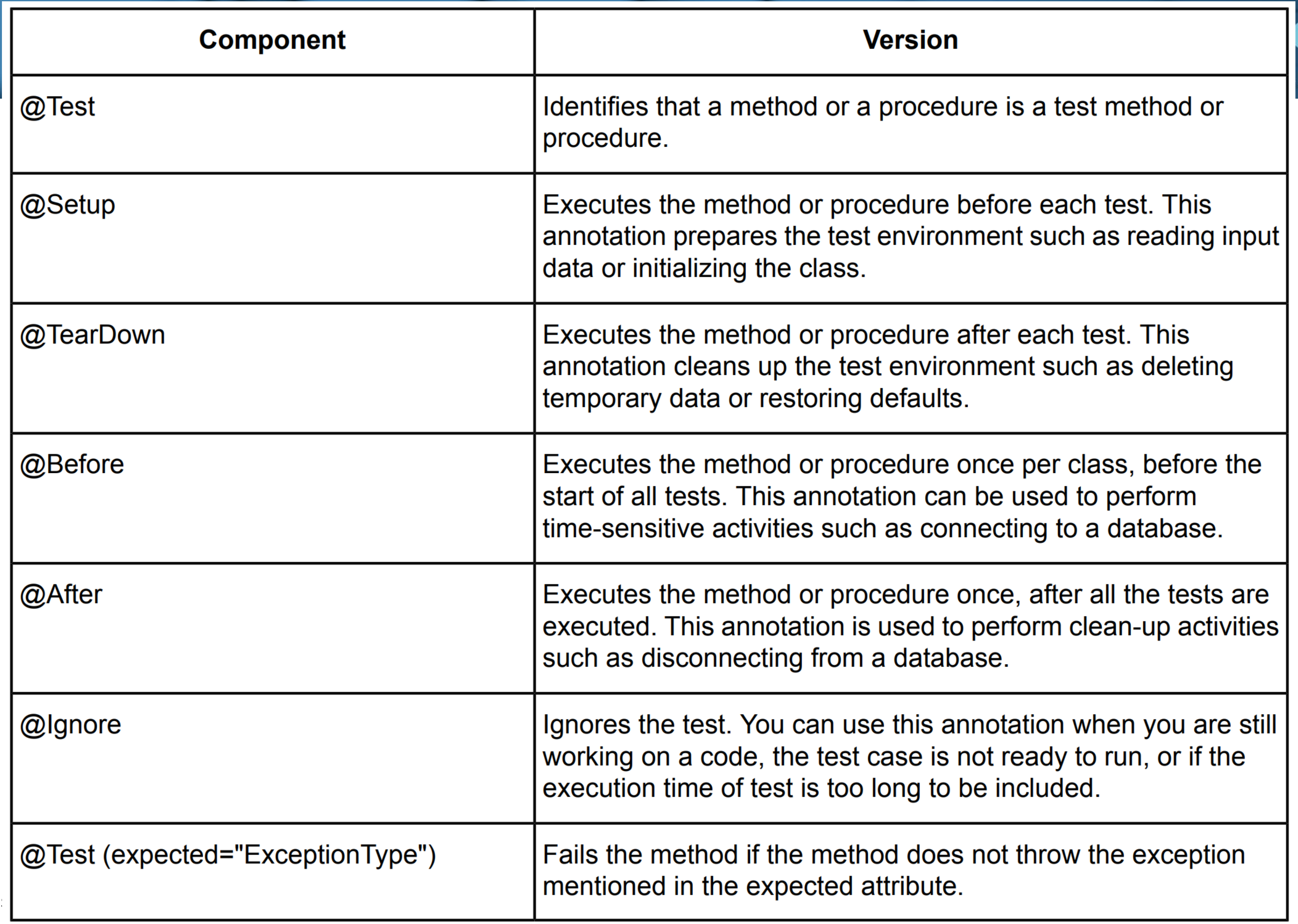
- (ABL) Unit Tests may be developed in procedures and in classes
- A Unit Test is a method or internal procedure which executes a piece of code and asserts the result of that piece of code
- Unit Tests may be included in the compilation unit which is tested
- Unit Tests may be placed in separate class or procedure files to keep them separated from deployed code (my preference)
- Unit Test classes and methods or procedures may not have parameters
- Unit Test methods or procedures are annotated with @Test.

Test Annotations

- (ABL) Unit Tests are annotation driven

- Documented in the Progress Documentation
- Well hidden in the *"Progress Developer Studio for OpenEdge Help"*
- *"Annotations supported with ABLUnit"*
- https://documentation.progress.com/output/OpenEdge117/pdfs/devstudio/devstudio.pdf

Te



Component	Version
@Test	Identifies that a method or a procedure is a test method or procedure.
@Setup	Executes the method or procedure before each test. This annotation prepares the test environment such as reading input data or initializing the class.
@TearDown	Executes the method or procedure after each test. This annotation cleans up the test environment such as deleting temporary data or restoring defaults.
@Before	Executes the method or procedure once per class, before the start of all tests. This annotation can be used to perform time-sensitive activities such as connecting to a database.
@After	Executes the method or procedure once, after all the tests are executed. This annotation is used to perform clean-up activities such as disconnecting from a database.
@Ignore	Ignores the test. You can use this annotation when you are still working on a code, the test case is not ready to run, or if the execution time of test is too long to be included.
@Test (expected="ExceptionType")	Fails the method if the method does not throw the exception mentioned in the expected attribute.

Initialization/cleanup annotations

- @Before and @After methods can be used to initialize and shut down framework components (or mocks of those) required to execute all unit test methods/procedures in test class/procedure

- @Setup and @TearDown methods can be used to initialize and cleanup for every test method in a test class
  - Ensure that every test has the same starting point, e.g. loading of data into temp-tables etc.

Tiny little ABLUnit bug …

- @Test (expected="ExceptionType").

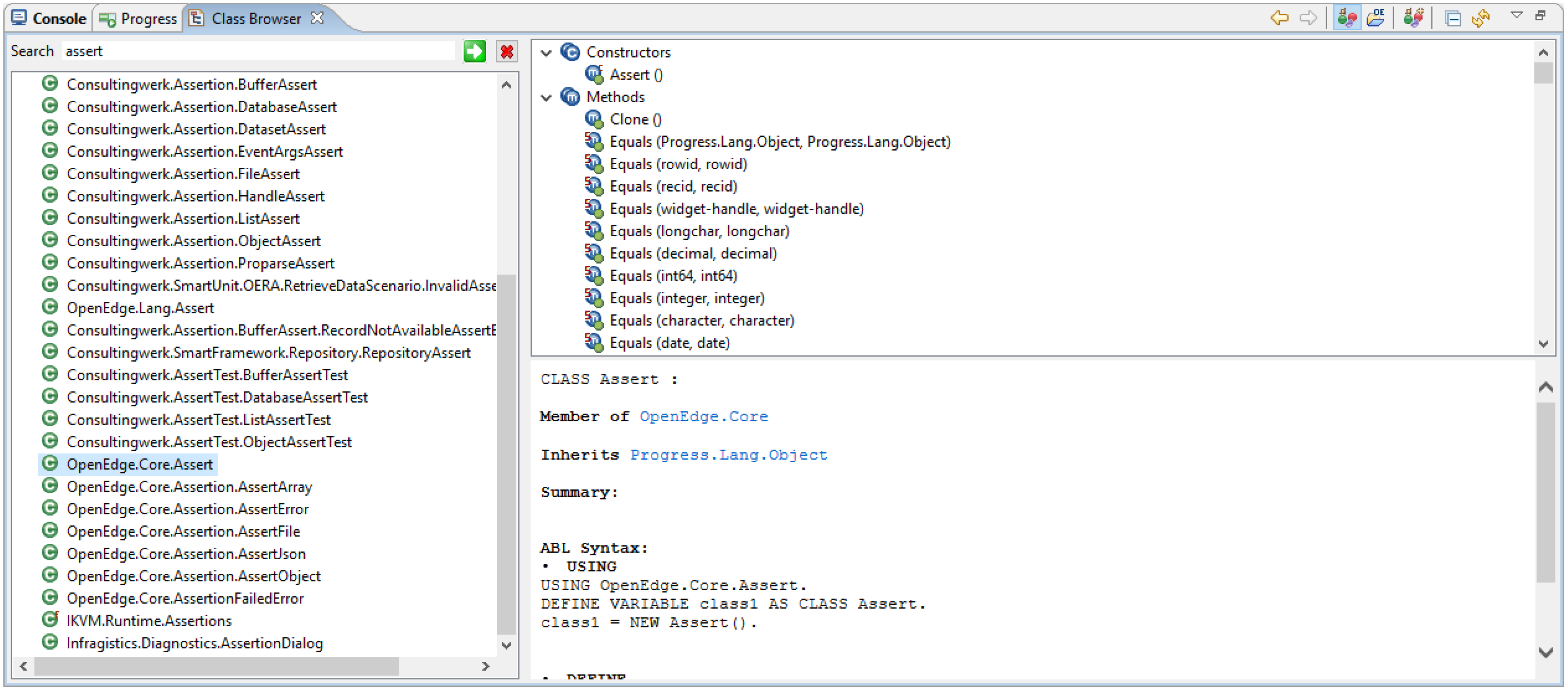
- Don't add a space before the period. ABLUnit will ignore the annotation parameter then
  - Logged and confirmed as a bug

Assert-Classes and methods

- Simple way to test a value received by the tested method
- STATIC methods
- A single method call that
  - Tests a value
  - THROW’s an error when the value does not match the expected value
  - Fire and forget

Assert-Classes and Methods

- OpenEdge.Core.Assert
- Consultingwerk.Assertions.*
- Roll your own



Search assert

Consultingwerk.Assertion.BufferAssert
Consultingwerk.Assertion.DatabaseAssert
Consultingwerk.Assertion.DatasetAssert
Consultingwerk.Assertion.EventArgsAssert
Consultingwerk.Assertion.FileAssert
Consultingwerk.Assertion.HandleAssert
Consultingwerk.Assertion.ListAssert
Consultingwerk.Assertion.ObjectAssert
Consultingwerk.Assertion.ProparseAssert
Consultingwerk.SmartUnit.OERA.RetrieveDataScenario.InvalidAsse
OpenEdge.Lang.Assert
Consultingwerk.Assertion.BufferAssert.RecordNotAvailableAssertE
Consultingwerk.SmartFramework.Repository.RepositoryAssert
Consultingwerk.AssertTest.BufferAssertTest
Consultingwerk.AssertTest.DatabaseAssertTest
Consultingwerk.AssertTest.ListAssertTest
Consultingwerk.AssertTest.ObjectAssertTest
OpenEdge.Core.Assert
OpenEdge.Core.Assertion.AssertArray
OpenEdge.Core.Assertion.AssertError
OpenEdge.Core.Assertion.AssertFile
OpenEdge.Core.Assertion.AssertJson
OpenEdge.Core.Assertion.AssertObject
OpenEdge.Core.AssertionFailedError
IKVM.Runtime.Assertions
Infragistics.Diagnostics.AssertionDialog

Constructors
- Assert ()

Methods
- Clone ()
- Equals (Progress.Lang.Object, Progress.Lang.Object)
- Equals (rowid, rowid)
- Equals (recid, recid)
- Equals (widget-handle, widget-handle)
- Equals (longchar, longchar)
- Equals (decimal, decimal)
- Equals (int64, int64)
- Equals (integer, integer)
- Equals (character, character)
- Equals (date, date)

```
CLASS Assert :

Member of OpenEdge.Core

Inherits Progress.Lang.Object

Summary:


ABL Syntax:
• USING
USING OpenEdge.Core.Assert.
DEFINE VARIABLE class1 AS CLASS Assert.
class1 = NEW Assert().
```

- Demo roll your own Assert

Consultingwerk
software architecture and development

Testing PROTECTED members

- When unit test is in a seperate class, it only has access to PUBLIC methods of the class to be tested
- Making internal methods PUBLIC for the purpose of testing is the wrong approach!
- Solution:
  - Unit Test class can inherit from class to be tested to access PROTECTED
  - (some) Unit Test methods may be placed inside the class to be tested to access PRIVATE members
  - A combination

- Demo Unit test of a protected class member

Agenda

- Introduction
- A simple ABL Unit Test
- Structure of a Unit Test
- **Unit Testing Tooling**
- Writing testable code
- Mocking dependencies
- Dealing with Data

Unit Testing Tooling

- #1 tool supporting Unit Testing: Structured Error Handling
  - Unit Tests rely heavily on solid error handling
  - Unit Testing tool can’t trace errors not thrown far enough
- ABLUnit OpenEdge’s Unit Testing tool integrated into PDSOE
- Proprietary ABL Unit Testing tools
  - ProUnit
  - OEUnit
  - ***SmartUnit (component of the SmartComponent Library)***
- All very similar but different in detail

JUnit legacy

- NUnit, JSUnit, ABLUnit, SmartUnit, …
- Most unit tests follow the JUnit conventions
- Usage of @Test. annotations (or similar)
- JUnit output file de facto standard
  - xml file capturing the result (success, error, messages, stack trace) of a single test or a test suite
  - Understood by a bunch of tools, including Jenkins CI
  - No formal definition though

Agenda

- Introduction
- A simple ABL Unit Test
- Structure of a Unit Test
- Unit Testing Tooling
- **Writing testable code**
- Mocking dependencies
- Dealing with Data

Object oriented or procedural?

- Procedures can be unit tested
- In fact, ABLUnit supports the execution of test-procedures as well
- OO-thinking however simplifies writing testable code
- Procedural code has tendency to be monolithic
- “Mocking” of dependencies requires patterns such as factories or dependency injection
  - In theory possible with procedures
  - More natural in object oriented programming

Writing testable code

- A huge financial report or invoice generation is barely testable in whole
- Large
- May call sub routines
- If it fails, what has been causing this?
  - A bug in code
  - False assumptions
  - Wrong data in DB?
- Output: A PDF file, how to assert this?

Writing testable code

- Break up financial report into a bunch of smaller components
- Test individual components
- Test report as a whole
- This allows to narrow down source of reported errors
- Separate report logic from output logic
  - Financial report should return temp-tables first
    - This can be tested
  - A separate module produces PDF output based on temp-table data
    - Testing difficult

Errors must be THROWN

- BLOCK-LEVEL ON ERROR UNDO, THROW almost mandatory
- Alternative Form of solid error handling
- Unit Testing tools don’t capture ** Customer record not on file (138) when written to stdout or a message box

Agenda

- Introduction
- A simple ABL Unit Test
- Structure of a Unit Test
- Unit Testing Tooling
- Writing testable code
- **Mocking dependencies**
- Dealing with Data

Mocking Dependencies

- Writing Unit Tests (for complex code) is a permanent fight against dependencies (and the bugs in them)
- If the PriceInfoService relies on the CustomerBusinessEntity, the ItemBusinessEntity, an InventoryService and the framework’s AuthorizationManager you’re always testing the integration of 5 components
- Who’s fault is it, when the test fails?
- How do we test extreme situations? Caused by unexpected data returned from one of the dependencies?

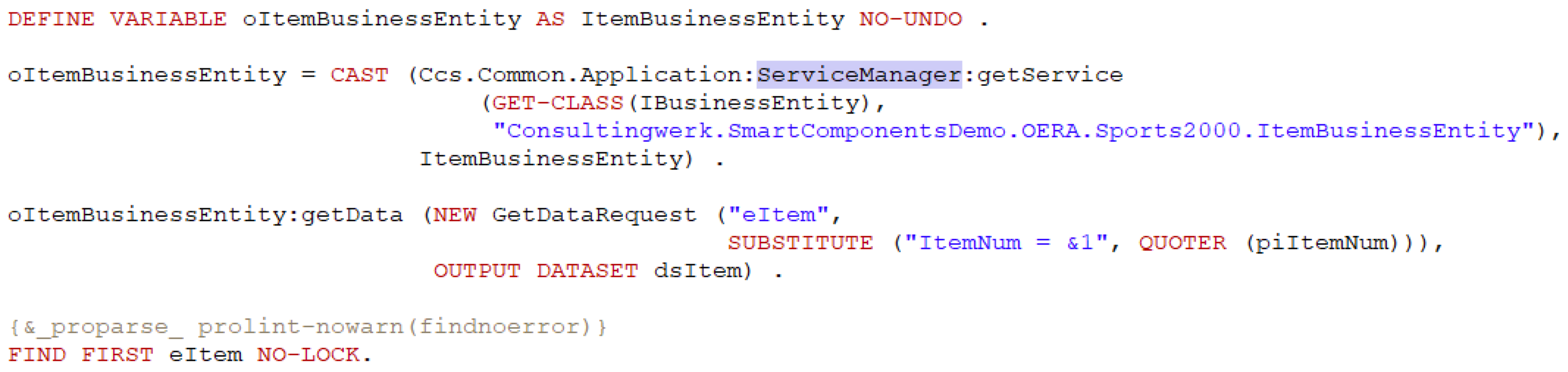
Mocking Dependencies - Wikipedia

- “In object-oriented programming, **mock objects** are simulated objects that mimic the behavior of real objects in controlled ways. A programmer typically creates a mock object to test the behavior of some other object, in much the same way that a car designer uses a crash test dummy to simulate the dynamic behavior of a human in vehicle impacts.”
- “In a unit test, mock objects can simulate the behavior of complex, real objects and are therefore useful when a real object is impractical or impossible to incorporate into a unit test.”

Mocking

- Requires abstraction of object construction
- PriceInfoService should not NEW CustomerBusinessEntity as this would disallow to mock this
- Rather rely on Dependency Injection or CCS Service Manager component (or similar) to provide CustomerBusinessEntity or a mock based on configuration
- Same technique applies to any other sort of dependent components

CCS Business Entity getData instead of FIND in DB



```
DEFINE VARIABLE oItemBusinessEntity AS ItemBusinessEntity NO-UNDO .

oItemBusinessEntity = CAST (Ccs.Common.Application:ServiceManager:getService
                               (GET-CLASS(IBusinessEntity),
                                "Consultingwerk.SmartComponentsDemo.OERA.Sports2000.ItemBusinessEntity"),
                           ItemBusinessEntity) .

oItemBusinessEntity:getData (NEW GetDataRequest ("eItem",
                                                 SUBSTITUTE ("ItemNum = &1", QUOTER (piItemNum))),
                            OUTPUT DATASET dsItem) .

{&_proparse_ prolint-nowarn(findnoerror)}
FIND FIRST eItem NO-LOCK.
```

Agenda

- Introduction
- A simple ABL Unit Test
- Structure of a Unit Test
- Unit Testing Tooling
- Writing testable code
- Mocking dependencies
- **Dealing with Data**

Dealing with Data

- We’re using ABL to develop database applications
- Application functionality highly dependent on data in a database
- That’s a resource that’s difficult to deal with …

Consultingwerk
software architecture and development

Don’t use a shared database for Unit Tests

- Your tests may rely on stock data or price data in the database
- A different developer may modify those records for his tests
- This can break your test

Don’t reuse your own database

- Your test sequence will include tests that modify data
- Maybe there is even a test to remove the item record that some other test depends on
  - Suddenly after adding this new test, a different test fails as the database contents are no longer the same

Solutions to the database dependency

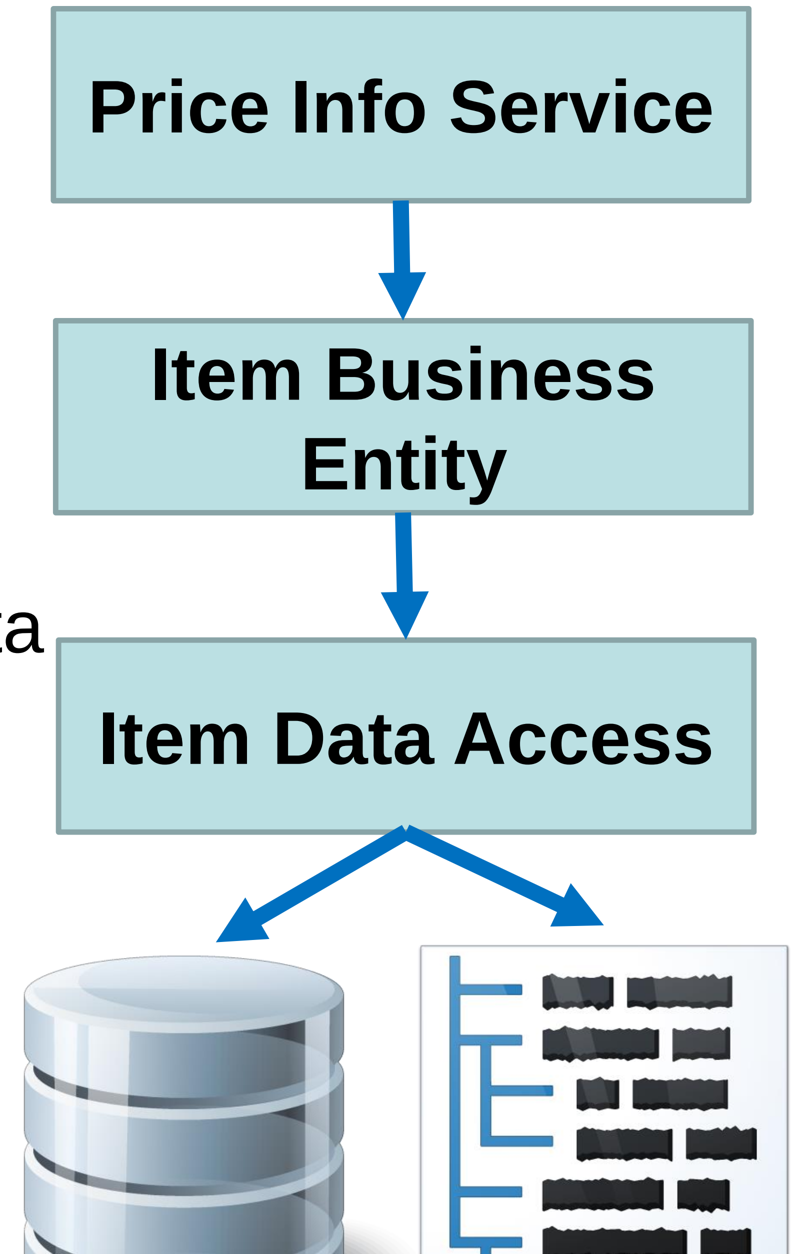
- Always restore a known database state from a backup
- Or rebuild a database for each test run from .d and .df
  - This may be easier when the database schema may change during a test sequence
- You may need to rebuild a database multiple times during a test sequence
- Produces lots of Disk I/O
- Disk I/O on one of the SSD’s of the build server if the bottleneck in our test environment (CPU and memory barely busy)

Transactions

- When used carefully database transactions can be a solution to test modifying or deleting records
  - Execute deletion of a record
  - Test that it’s really gone (CAN-FIND)
  - UNDO transaction in test-class
- May cause side-effects if the code to be tested relies on a specific transaction behavior influenced by the fact that there’s an outer transaction now

Mock the code that accesses the DB

- May follow OERA or CCS principles
- Data Access class should be the only code that ever access the database
- Not even the business entity should be able to know that the data access class is using data from an XML file instead



Questions