Consultingwerk
software architecture and development

ABL Unit Testing Part 2: Writing Unit Tests

James Palmer
Senior Consultant / Database Administrator

Agenda

- **Introduction**
- A simple ABL Unit Test
- Structure of a Unit Test
- Unit Testing Tooling
- Writing Testable Code
- Mocking Dependencies


Consultingwerk
software architecture and development

Consultingwerk Software Services Ltd.

- 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
- Specialized in GUI for .NET, Angular, OO, Software Architecture, Application Integration
- Experts in OpenEdge Application Modernization

James Palmer

- Senior Consultant / Database Administrator
- Working in the OpenEdge world since 2003
- Varied experience across a variety of industries and applications as a developer and more recently as a DBA
- Chairman of the UK & Ireland PUG, and speaker on a variety of topics both at EMEA PUG Challenge events, and smaller conferences in the UK and USA

Questions…

- Who was in Mike Fechner's Strategy talk earlier?
- Who uses Unit Testing already?
- Who is aware of Unit Testing?
- Who has never seen or heard about it?

A recent real world example

- April 2017 Windows Creators Update – breaks INPUT THROUGH statements
- Jenkins job alerts around noon after update applied
- We found a fix
- 2 days later discussions start on Forums etc.

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

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

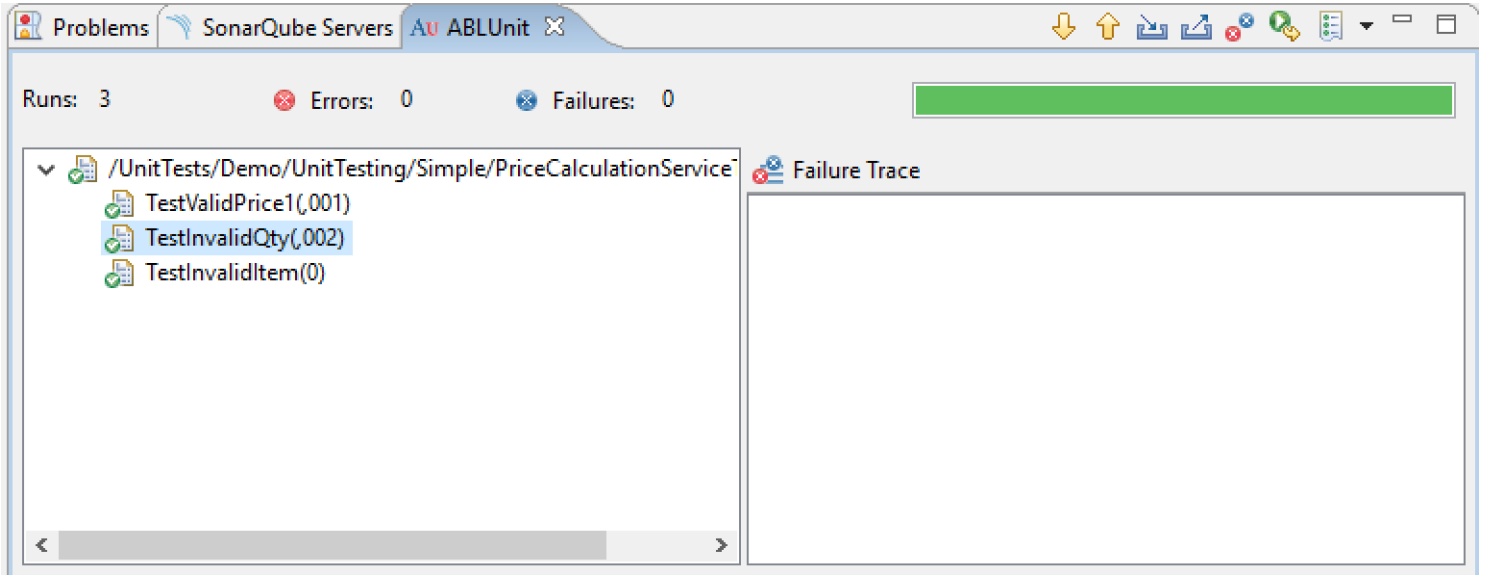
Demo – Division.p

- ABLUnit appeared in 11.4
- Enhanced in 11.5 and 11.6
- Prerequisite of 11.6 for my examples

- Creating Unit Test Project
- Writing a simple test class

Demo – PriceCalculationService.cls

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



Demo – FinanceReport.p

- Not limited to classes
- Although the OO strategy simplifies writing testable code
- Procedural code tends to be monolithic
- Dependency “mocking” can be difficult, but possible

Demo – MyDemoTestSuite.cls

- Test Suites to carry out multiple test classes/procedures in one go
- Can mix procedures and classes

Agenda

- Introduction
- A simple ABL Unit Test
- **Structure of a Unit Test**
- Unit Testing Tooling
- Writing Testable Code
- Mocking Dependencies

Structure of a Unit Test

- May be included in the compilation unit tested
- May be in a separate class or procedure
  - Separate from deployed code
  - Our preference where possible
- May not have parameters
- Annotations

Annotations

- All ABLUnit tests are annotation driven

- Documented, but well hidden!
- *“Progress Developer Studio for OpenEdge Help”*
- https://documentation.progress.com/output/OpenEdge117/pdfs/devstudio/devstudio.pdf

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.

Annotations

- @Before and @After – initialize and shut down framework components
- @Setup and @TearDown – Ensure each test has the same start point
- Small bug: No space before the .
  - ABLUnit ignores the annotation
  - Logged and confirmed as a bug

Assert – Classes and Methods

- Simple way to test a value received
- STATIC Methods
- A single method call that
  - Tests a value
  - THROWs an error when it doesn't match
  - Fire and forget
- OpenEdge.Core.Assert
- Consultingwerk.Assertions.*
- Roll your own
- Demo – TestCaseSensitive.cls

PROTECTED / PRIVATE

- Unit Test class only has access to PUBLIC members
- Making methods public for testing is BAD
- Solution:
  - Inheritance for PROTECTED
  - Place tests inside class for PRIVATE
  - Or a combination

- Demo – MyProtectedClass.cls

Agenda

- Introduction
- A simple ABL Unit Test
- Structure of a Unit Test
- **Unit Testing Tooling**
- Writing Testable Code
- Mocking Dependencies

Unit Testing Tooling

- Structured Error Handling
  - Can’t trace errors not thrown far enough
- ABLUnit
- Proprietary tools
  - ProUnit
  - OEUnit
  - SmartUnit
- Test results captured in XML
  - Jenkins CI etc.

Agenda

- Introduction
- A simple ABL Unit Test
- Structure of a Unit Test
- Unit Testing Tooling
- **Writing Testable Code**
- Mocking Dependencies

Writing testable code

- Object Oriented or Procedural
- Huge financial report?
  - PDF Output?
  - Dependencies?
  - Sub routines?
  - Failure caused by what?
  - Break into smaller components
  - Test each component
  - Separate report and output logic
- Error Handling! Again!

Agenda

- Introduction
- A simple ABL Unit Test
- Structure of a Unit Test
- Unit Testing Tooling
- Writing Testable Code
- **Mocking Dependencies**

Mocking Dependencies

- Constant fight against dependencies (and their bugs)
- PriceInfoService relies on 4 other services and Authorization
  - Whose fault is a failed test?
  - What if a dependency returns extreme data?
- What if the data in the database is wrong?

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
- PriceInfoService should NEW other services
- Rely on Dependency Injection, or CCS Service Manager component etc.
- Same techniques for other dependencies

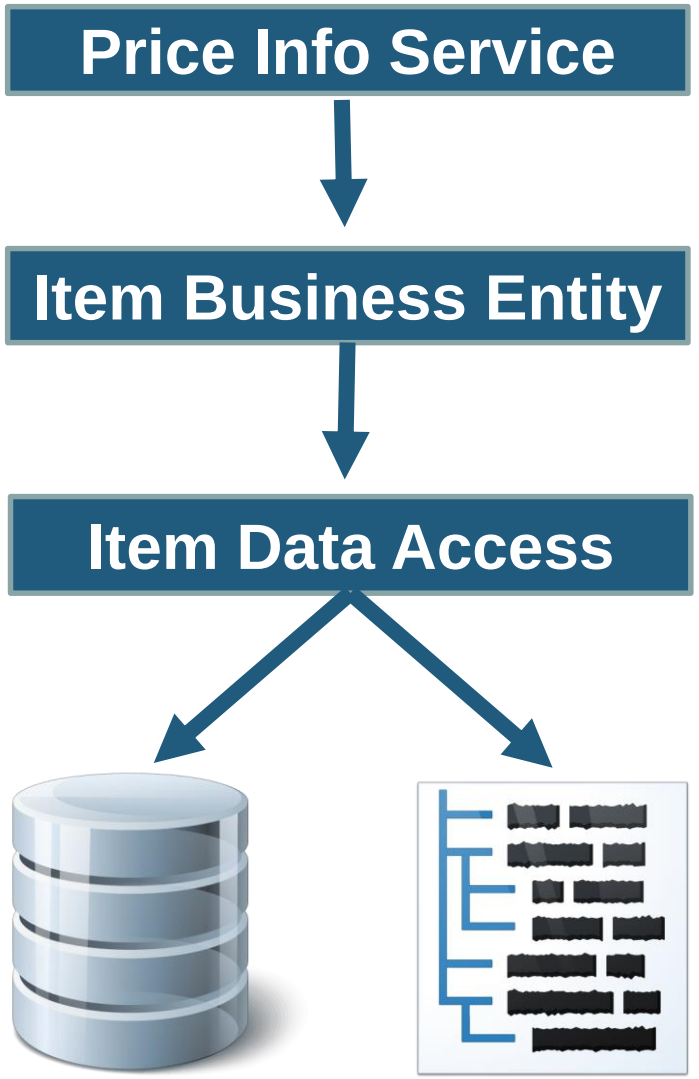
```
DEFINE VARIABLE oItemBusinessEntity AS ItemBusinessEntity NO-UNDO .

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.
```

What about data?

- Database application = Data. Data changes.
- Many methods.
- Mock the data access code

What about data?

- 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



Demo – PriceCalculationServiceMock.cls

- Database application = Data. Data changes.
- Many methods.
- Mock the data access code

Demo – CurrencyConversionService.cls

- Reliance on data that changes
- How to mock this dependency so result is consistent?
- Our solution is to have configurable services

Questions


Consultingwerk
software architecture and development