Consultingwerk
software architecture and development

ABL Unit Testing Part 1: Test Strategy

Mike Fechner
Director

Consultingwerk
software architecture and development

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

Mike Fechner

- Director, Lead Modernization Architect and Product Manager of the SmartComponent Library and WinKit
- Specialized on object oriented design, software architecture, desktop user interfaces and web technologies
- 28 years of Progress experience (V5 … OE11)
- Active member of the OpenEdge community
- Frequent speaker at OpenEdge related conferences around the world

Agenda

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

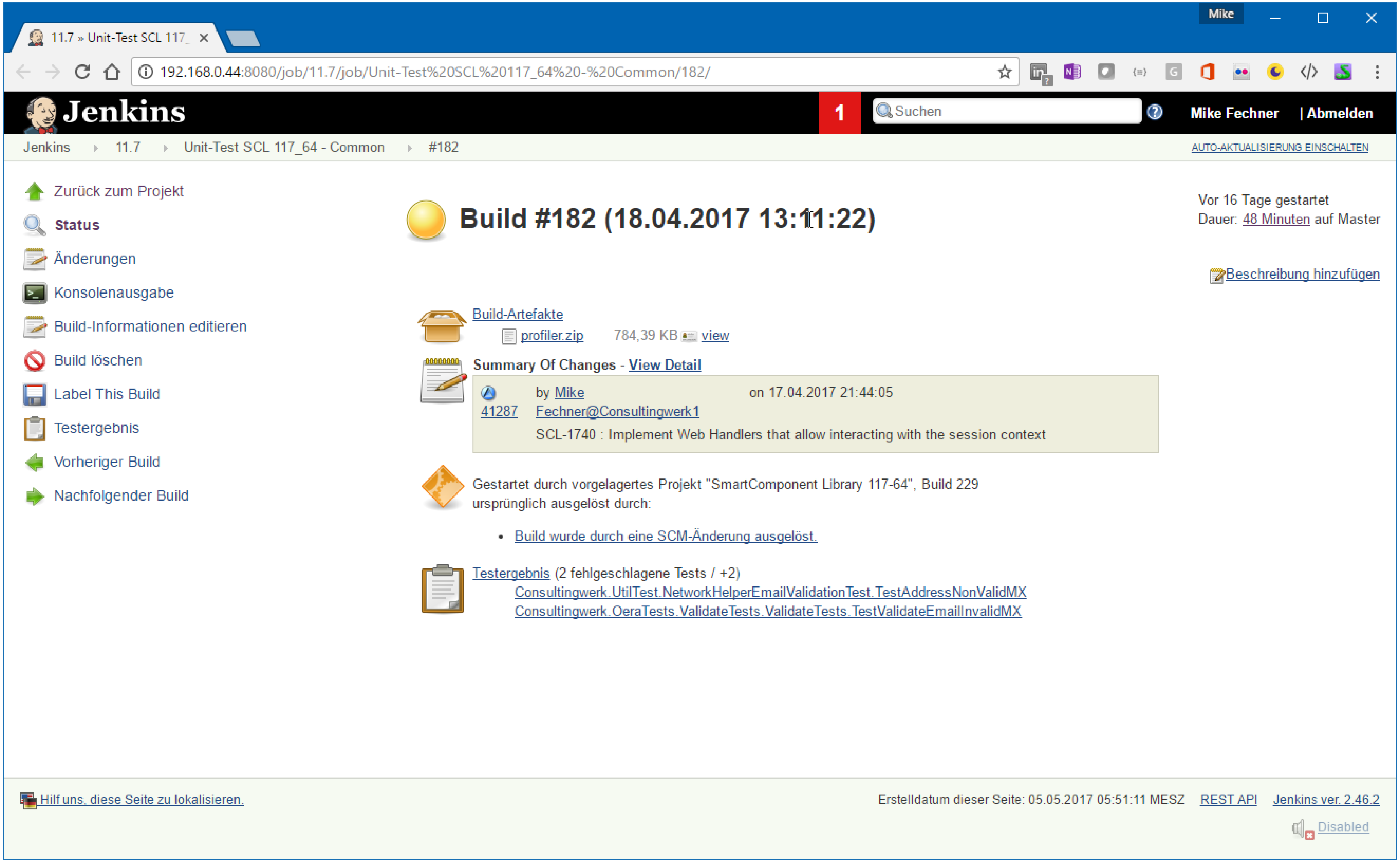
Introduction

- 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!


11.7 » Unit-Test SCL 117_
Mike
192.168.0.44:8080/job/11.7/job/Unit-Test%20SCL%20117_64%20-%20Common/182/
Jenkins
1
Suchen
Mike Fechner
| Abmelden
Jenkins › 11.7 › Unit-Test SCL 117_64 - Common › #182
AUTO-AKTUALISIERUNG EINSCHALTEN
Zurück zum Projekt
Status
Änderungen
Konsolenausgabe
Build-Informationen editieren
Build löschen
Label This Build
Testergebnis
Vorheriger Build
Nachfolgender Build

Build #182 (18.04.2017 13:11:22)

Vor 16 Tage gestartet
Dauer: 48 Minuten auf Master

Beschreibung hinzufügen

Build-Artefakte
profiler.zip 784,39 KB view

Summary Of Changes - View Detail

41287	by Mike Fechner@Consultingwerk1	on 17.04.2017 21:44:05
	SCL-1740 : Implement Web Handlers that allow interacting with the session context	

Gestartet durch vorgelagertes Projekt "SmartComponent Library 117-64", Build 229
ursprünglich ausgelöst durch:

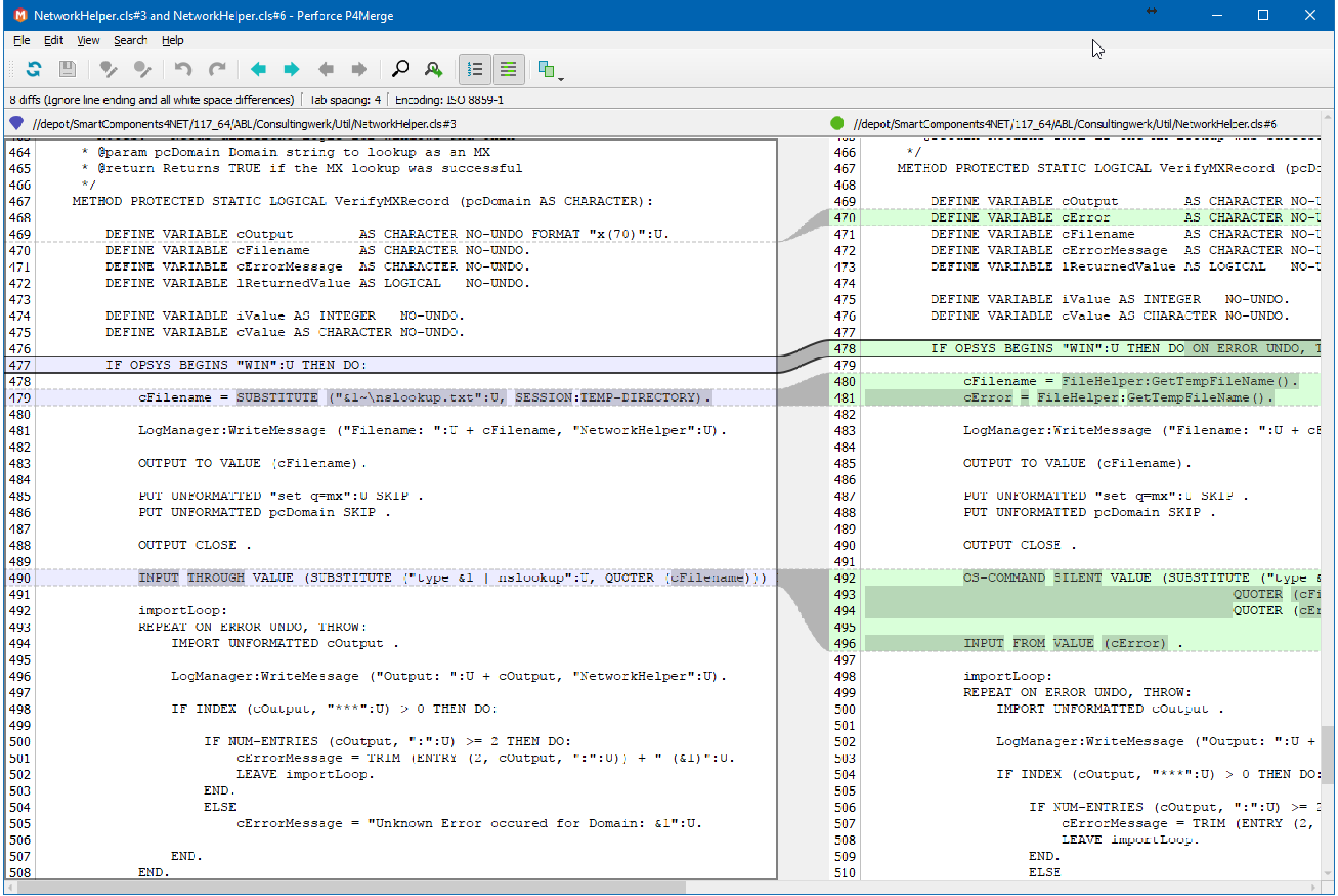
- Build wurde durch eine SCM-Änderung ausgelöst.

Testergebnis (2 fehlgeschlagene Tests / +2)
Consultingwerk.UtilTest.NetworkHelperEmailValidationTest.TestAddressNonValidMX
Consultingwerk.OeraTests.ValidateTests.ValidateTests.TestValidateEmailInvalidMX

Hilf uns, diese Seite zu lokalisieren.

Erstelldatum dieser Seite: 05.05.2017 05:51:11 MESZ REST API Jenkins ver. 2.46.2

Disabled


NetworkHelper.cls#3 and NetworkHelper.cls#6 - Perforce P4Merge
File
Edit
View
Search
Help

8 diffs (Ignore line ending and all white space differences) | Tab spacing: 4 | Encoding: ISO 8859-1

//depot/SmartComponents4NET/117_64/ABL/Consultingwerk/Util/NetworkHelper.cls#3

```
     * @param pcDomain Domain string to lookup as an MX
     * @return Returns TRUE if the MX lookup was successful
     */
    METHOD PROTECTED STATIC LOGICAL VerifyMXRecord (pcDomain AS CHARACTER):

        DEFINE VARIABLE cOutput        AS CHARACTER NO-UNDO FORMAT "x(70)":U.
        DEFINE VARIABLE cFilename      AS CHARACTER NO-UNDO.
        DEFINE VARIABLE cErrorMessage  AS CHARACTER NO-UNDO.
        DEFINE VARIABLE lReturnedValue AS LOGICAL   NO-UNDO.

        DEFINE VARIABLE iValue AS INTEGER   NO-UNDO.
        DEFINE VARIABLE cValue AS CHARACTER NO-UNDO.

        IF OPSYS BEGINS "WIN":U THEN DO:

            cFilename = SUBSTITUTE ("&1~\nslookup.txt":U, SESSION:TEMP-DIRECTORY).

            LogManager:WriteMessage ("Filename: ":U + cFilename, "NetworkHelper":U).

            OUTPUT TO VALUE (cFilename).

            PUT UNFORMATTED "set q=mx":U SKIP .
            PUT UNFORMATTED pcDomain SKIP .

            OUTPUT CLOSE .

            INPUT THROUGH VALUE (SUBSTITUTE ("type &1 | nslookup":U, QUOTER (cFilename)))

            importLoop:
            REPEAT ON ERROR UNDO, THROW:
                IMPORT UNFORMATTED cOutput .

                LogManager:WriteMessage ("Output: ":U + cOutput, "NetworkHelper":U).

                IF INDEX (cOutput, "***":U) > 0 THEN DO:

                    IF NUM-ENTRIES (cOutput, ":":U) >= 2 THEN DO:
                        cErrorMessage = TRIM (ENTRY (2, cOutput, ":":U)) + " (&1)":U.
                        LEAVE importLoop.
                    END.
                    ELSE
                        cErrorMessage = "Unknown Error occured for Domain: &1":U.

                END.
            END.
```

//depot/SmartComponents4NET/117_64/ABL/Consultingwerk/Util/NetworkHelper.cls#6

```
     */
    METHOD PROTECTED STATIC LOGICAL VerifyMXRecord (pcDc

        DEFINE VARIABLE cOutput        AS CHARACTER NO-U
        DEFINE VARIABLE cError         AS CHARACTER NO-U
        DEFINE VARIABLE cFilename      AS CHARACTER NO-U
        DEFINE VARIABLE cErrorMessage  AS CHARACTER NO-U
        DEFINE VARIABLE lReturnedValue AS LOGICAL   NO-U

        DEFINE VARIABLE iValue AS INTEGER   NO-UNDO.
        DEFINE VARIABLE cValue AS CHARACTER NO-UNDO.

        IF OPSYS BEGINS "WIN":U THEN DO ON ERROR UNDO, T

            cFilename = FileHelper:GetTempFileName().
            cError = FileHelper:GetTempFileName().

            LogManager:WriteMessage ("Filename: ":U + cF

            OUTPUT TO VALUE (cFilename).

            PUT UNFORMATTED "set q=mx":U SKIP .
            PUT UNFORMATTED pcDomain SKIP .

            OUTPUT CLOSE .

            OS-COMMAND SILENT VALUE (SUBSTITUTE ("type &
                                             QUOTER (cFi
                                             QUOTER (cEr

            INPUT FROM VALUE (cError) .

            importLoop:
            REPEAT ON ERROR UNDO, THROW:
                IMPORT UNFORMATTED cOutput .

                LogManager:WriteMessage ("Output: ":U +

                IF INDEX (cOutput, "***":U) > 0 THEN DO:

                    IF NUM-ENTRIES (cOutput, ":":U) >= 2
                        cErrorMessage = TRIM (ENTRY (2,
                        LEAVE importLoop.
                    END.
                    ELSE
```

From a recent real world example

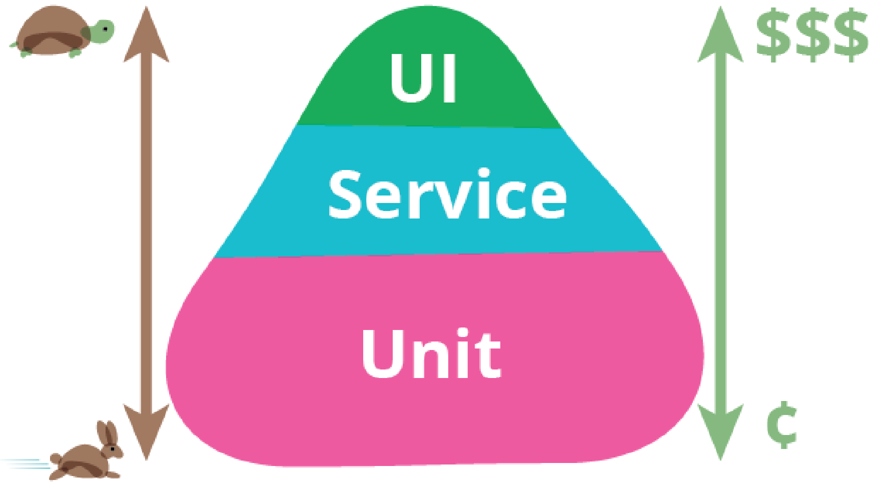
- A pretty simple API got broken; caused by a Windows update
- No matter if it’s Progress’ fault or Microsoft – it’s a 3rd party
- We execute our Unit Tests on OpenEdge 10.2B, 11.3, 11.6 and 11.7
- We execute our Unit Tests on Windows 10 and Linux (VMware)
- Considering to add additional Windows Versions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

The test pyramid

- Symbolizes different kind of tests that can be used to automate testing a (layered) application
- Unit Tests are relatively simple (cheap) to program, there should be lots of them
- API/Service Tests are more complex to write
- UI Tests are the most expensive to write and may require humans to execute them, may require frequent changes as the application evolves
- https://martinfowler.com/bliki/TestPyramid.html

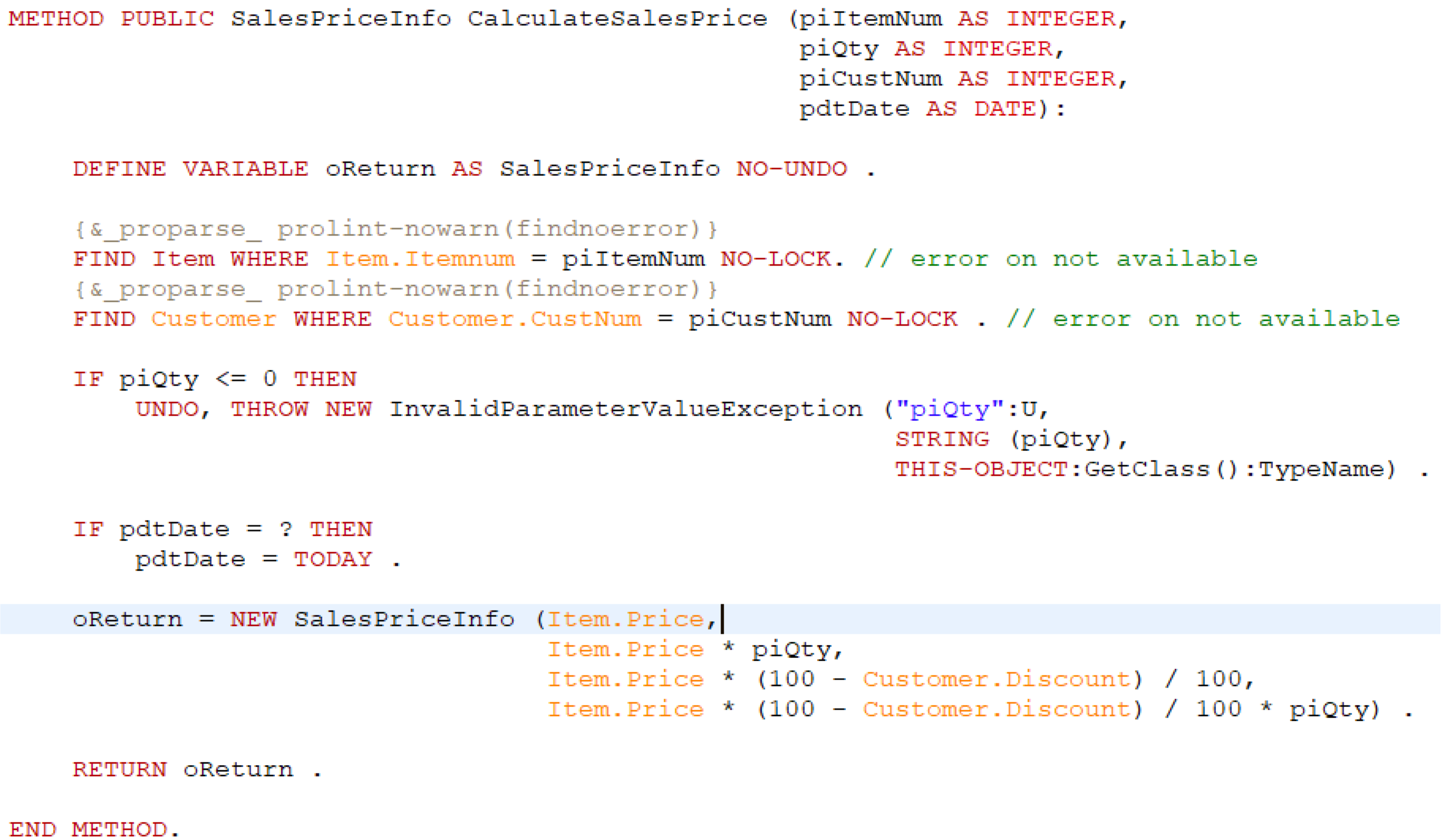


A customer’s testing stack for a web application

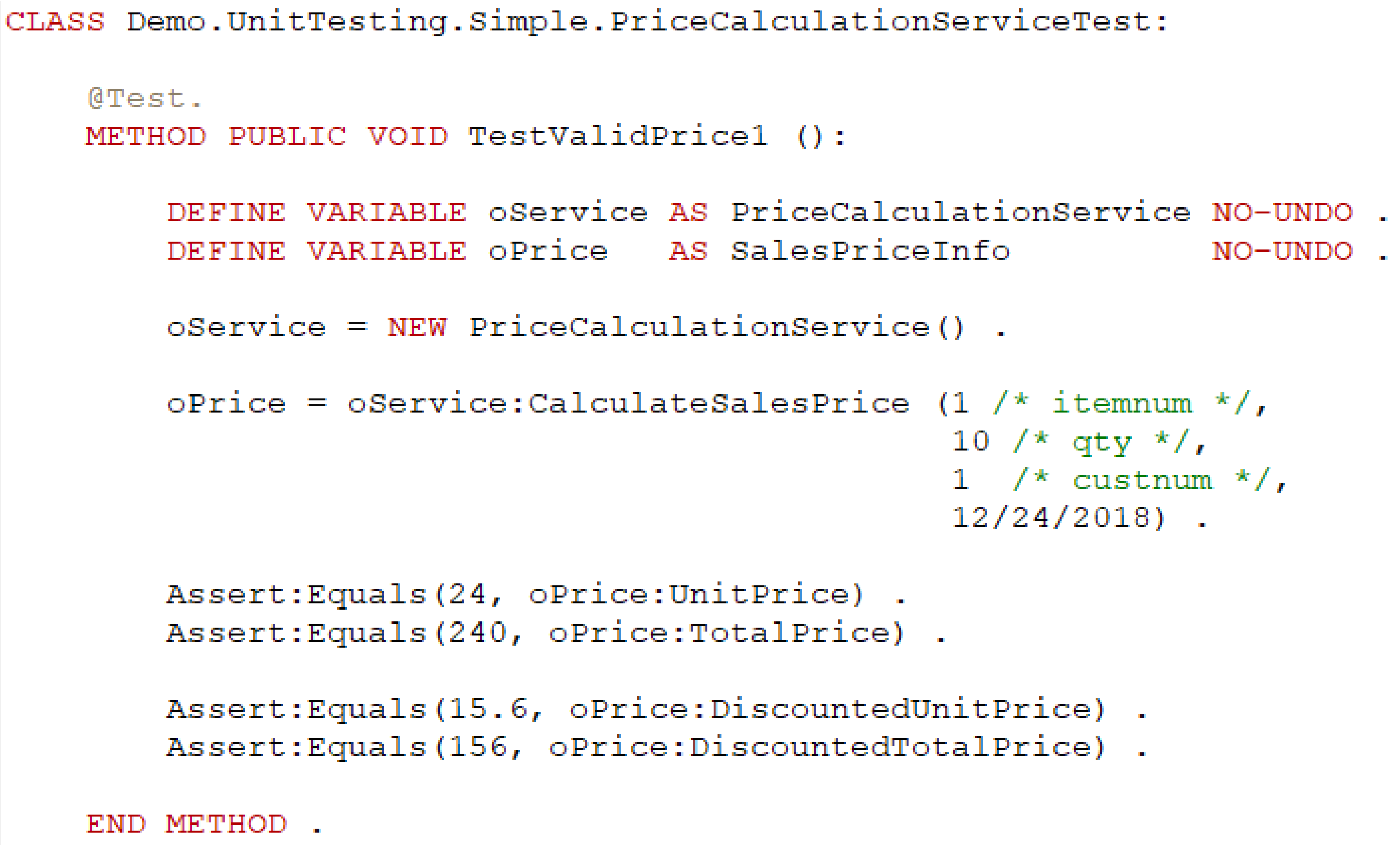
- Technology in use JavaScript, PASOE, Web Handlers for REST, OERA
- Browser UI Tests: Selenium (https://www.seleniumhq.org/)
- REST API’s
  - SOAP UI (https://www.soapui.org/), including load scripts
  - NUnit (.NET Unit Testing) as the test manager knows this well, and C# allows more complex test logic or sequences
- Backend Unit Test: ABLUnit and SmartUnit
- JavaScript Unit Testing: Soon to be adding JSUnit to the mix

Agenda

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

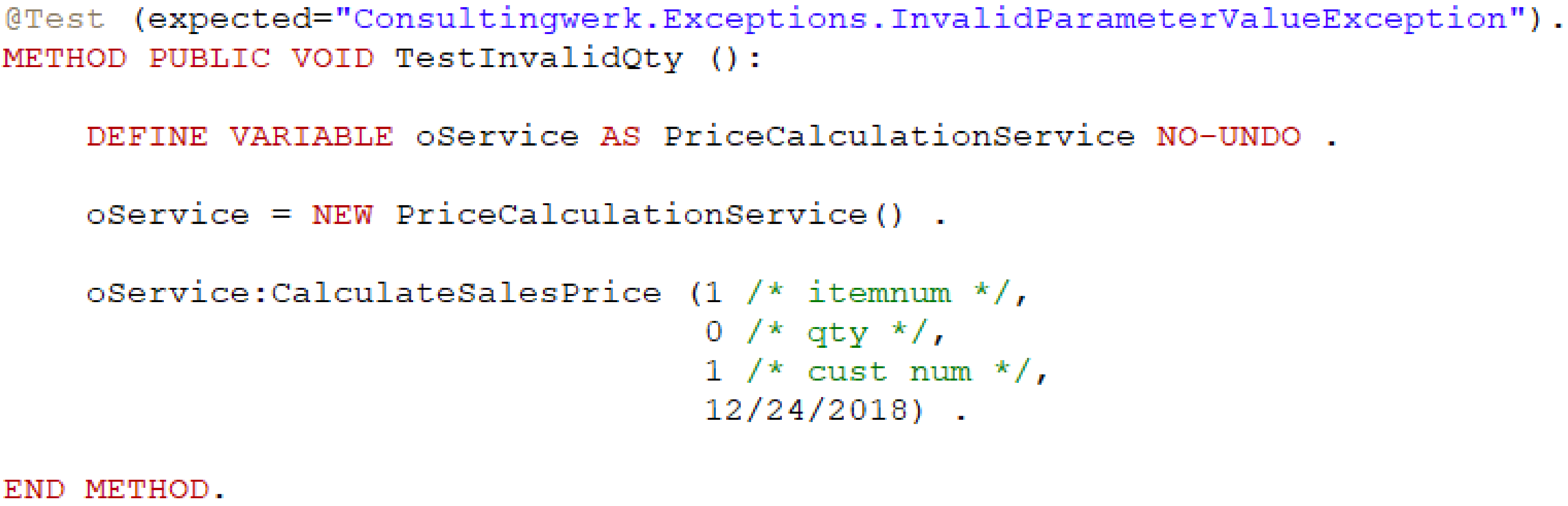


```
METHOD PUBLIC SalesPriceInfo CalculateSalesPrice (piItemNum AS INTEGER,
                                                  piQty AS INTEGER,
                                                  piCustNum AS INTEGER,
                                                  pdtDate AS DATE):

    DEFINE VARIABLE oReturn AS SalesPriceInfo NO-UNDO .

    {&_proparse_ prolint-nowarn(findnoerror)}
    FIND Item WHERE Item.Itemnum = piItemNum NO-LOCK. // error on not available
    {&_proparse_ prolint-nowarn(findnoerror)}
    FIND Customer WHERE Customer.CustNum = piCustNum NO-LOCK . // error on not available

    IF piQty <= 0 THEN
        UNDO, THROW NEW InvalidParameterValueException ("piQty":U,
                                                        STRING (piQty),
                                                        THIS-OBJECT:GetClass():TypeName) .

    IF pdtDate = ? THEN
        pdtDate = TODAY .

    oReturn = NEW SalesPriceInfo (Item.Price,
                                  Item.Price * piQty,
                                  Item.Price * (100 - Customer.Discount) / 100,
                                  Item.Price * (100 - Customer.Discount) / 100 * piQty) .

    RETURN oReturn .

END METHOD.
```



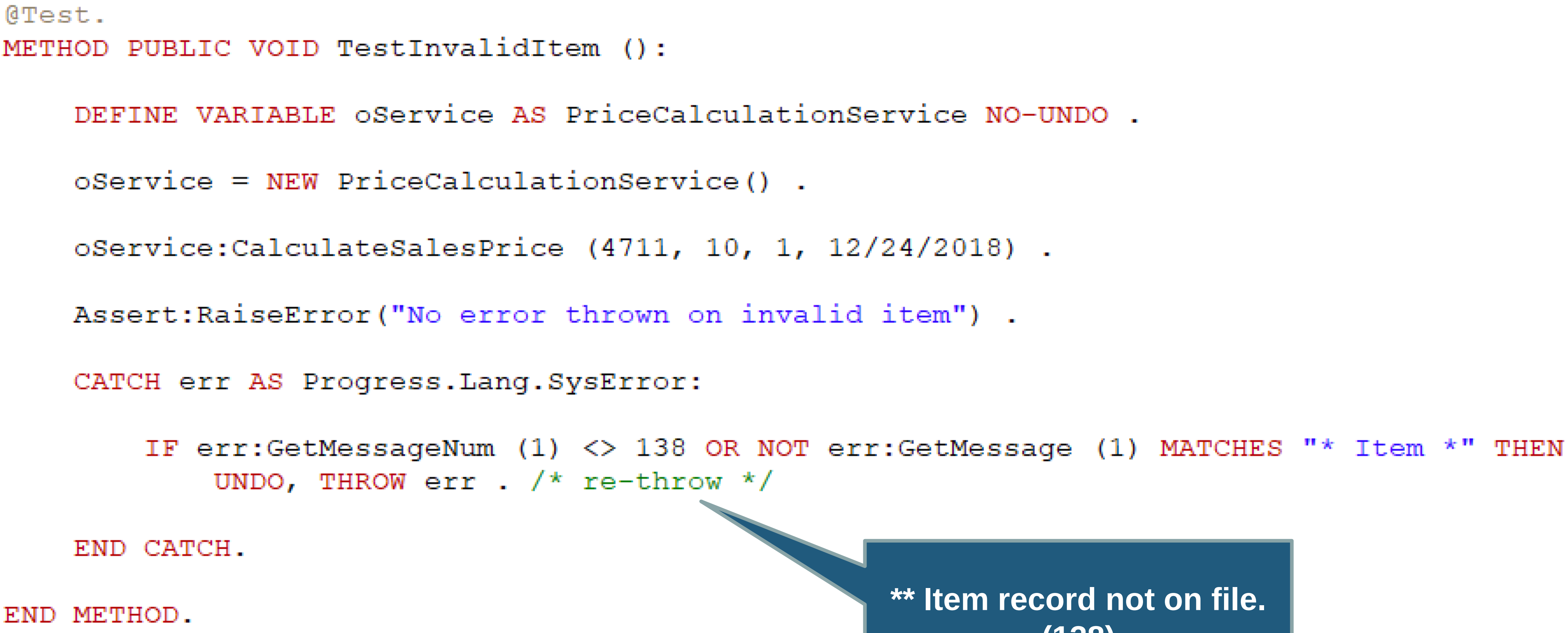
```
CLASS Demo.UnitTesting.Simple.PriceCalculationServiceTest:

    @Test.
    METHOD PUBLIC VOID TestValidPrice1 ():

        DEFINE VARIABLE oService AS PriceCalculationService NO-UNDO .
        DEFINE VARIABLE oPrice   AS SalesPriceInfo          NO-UNDO .

        oService = NEW PriceCalculationService() .

        oPrice = oService:CalculateSalesPrice (1 /* itemnum */,
                                               10 /* qty */,
                                               1  /* custnum */,
                                               12/24/2018) .

        Assert:Equals(24, oPrice:UnitPrice) .
        Assert:Equals(240, oPrice:TotalPrice) .

        Assert:Equals(15.6, oPrice:DiscountedUnitPrice) .
        Assert:Equals(156, oPrice:DiscountedTotalPrice) .

    END METHOD .
```

Test for a specific exception to be thrown



```
@Test (expected="Consultingwerk.Exceptions.InvalidParameterValueException").
METHOD PUBLIC VOID TestInvalidQty ():

    DEFINE VARIABLE oService AS PriceCalculationService NO-UNDO .

    oService = NEW PriceCalculationService() .

    oService:CalculateSalesPrice (1 /* itemnum */,
                                  0 /* qty */,
                                  1 /* cust num */,
                                  12/24/2018) .

END METHOD.
```

Expect a very specific error from a method

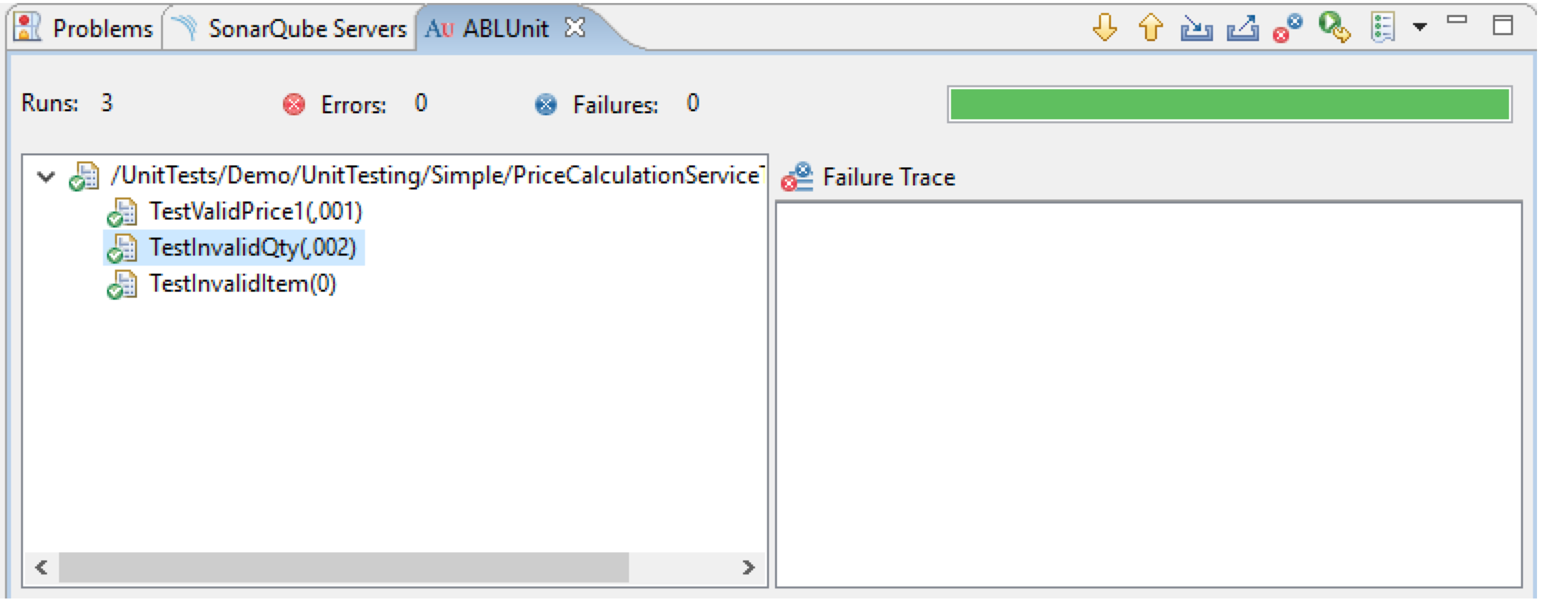


```
@Test.
METHOD PUBLIC VOID TestInvalidItem ():

    DEFINE VARIABLE oService AS PriceCalculationService NO-UNDO .

    oService = NEW PriceCalculationService() .

    oService:CalculateSalesPrice (4711, 10, 1, 12/24/2018) .

    Assert:RaiseError("No error thrown on invalid item") .

    CATCH err AS Progress.Lang.SysError:

        IF err:GetMessageNum (1) <> 138 OR NOT err:GetMessage (1) MATCHES "* Item *" THEN
            UNDO, THROW err . /* re-throw */

    END CATCH.

END METHOD.
```

** Item record not on file. (138)

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
- Advanced Unit Testing Features

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.

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.

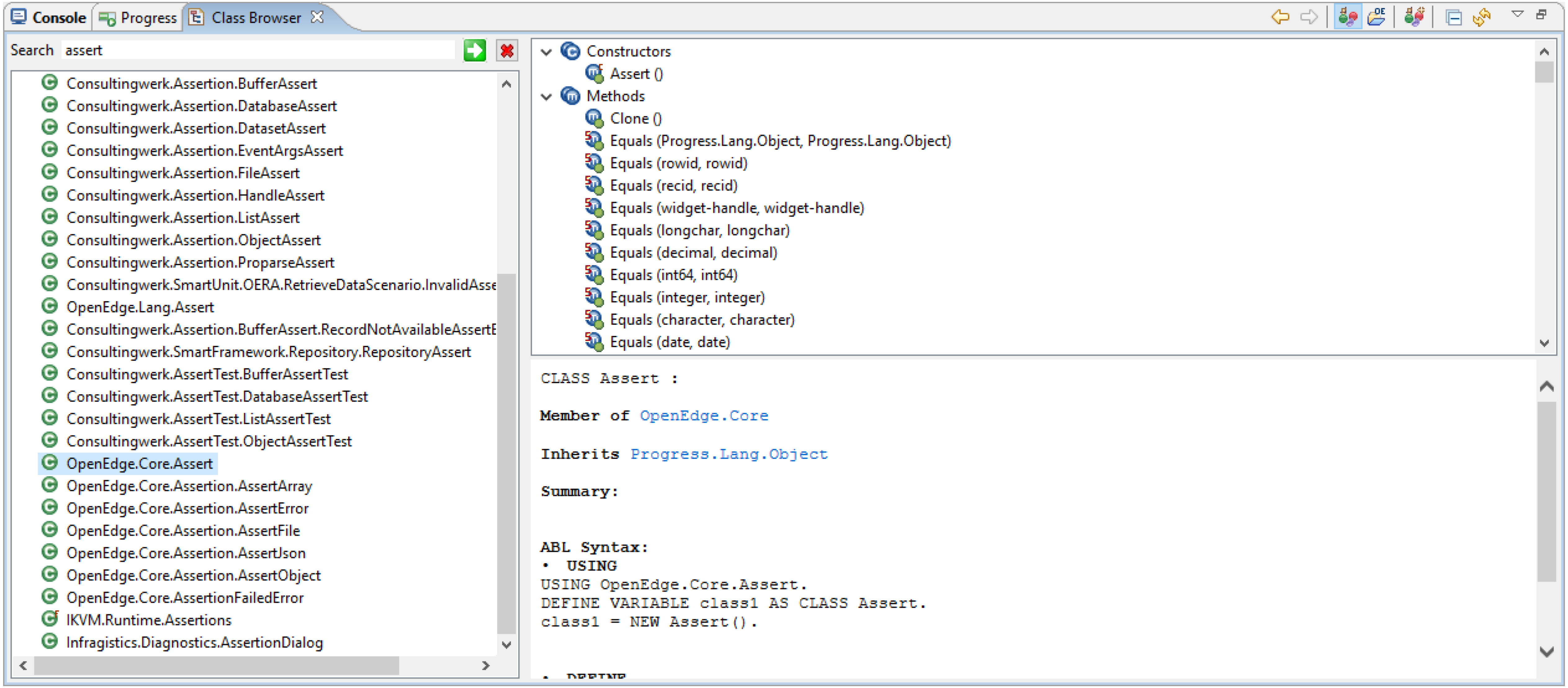
Assert-Classes and methods

- 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

Consultingwerk.Assertion.Assert:EqualsCaseSensitive
 (cReturnValue, “This is the expected value”) .

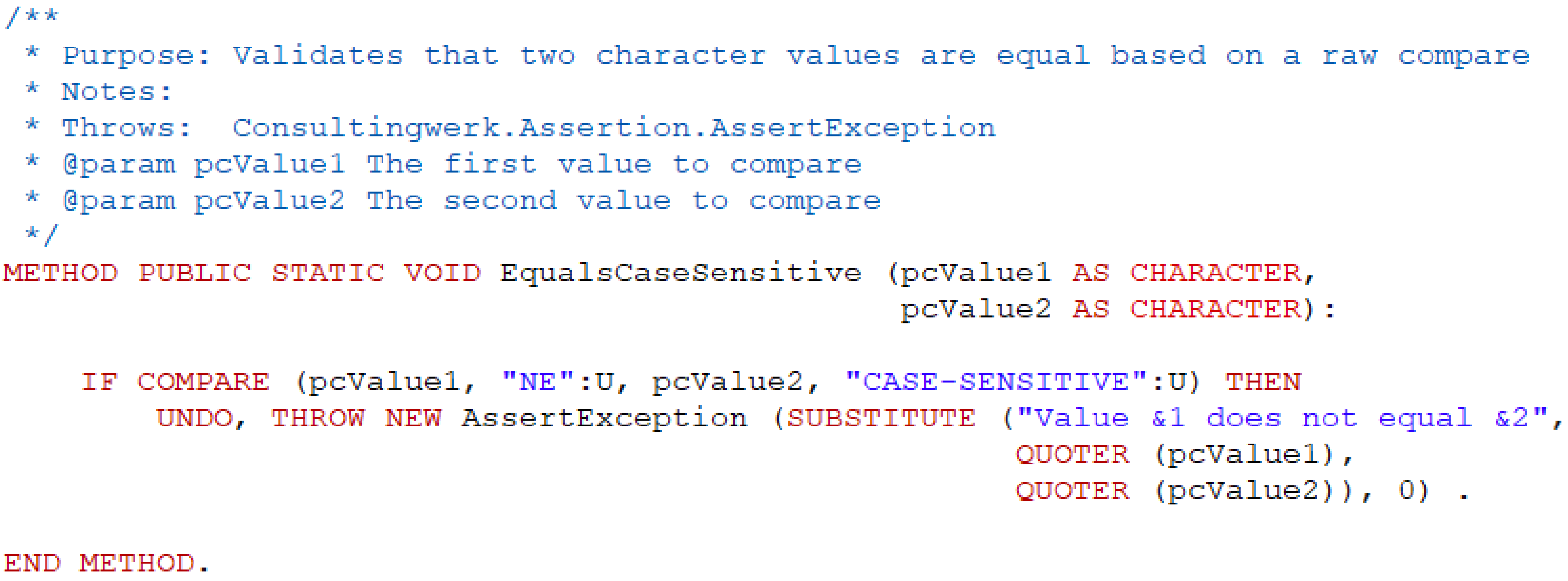


Search: assert

- 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().


• DEFINE
```



```
/**
 * Purpose: Validates that two character values are equal based on a raw compare
 * Notes:
 * Throws:  Consultingwerk.Assertion.AssertException
 * @param pcValue1 The first value to compare
 * @param pcValue2 The second value to compare
 */
METHOD PUBLIC STATIC VOID EqualsCaseSensitive (pcValue1 AS CHARACTER,
                                               pcValue2 AS CHARACTER):

    IF COMPARE (pcValue1, "NE":U, pcValue2, "CASE-SENSITIVE":U) THEN
        UNDO, THROW NEW AssertException (SUBSTITUTE ("Value &1 does not equal &2",
                                                     QUOTER (pcValue1),
                                                     QUOTER (pcValue2)), 0) .

END METHOD.
```

Agenda

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

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

JUnit output file

- results.xml produced by ABLUnit and similar tools
- Visualized by ABLUnit View
- Visualized / trended by Jenkins CI
- Visualized by ANT’s JUnit task (produces html output) or similar tools
- Alternatives like junit-viewer https://www.npmjs.com/package/junit-viewer

ANT

- Apache Build Scripting Language
- XML based batch file, OS-independent
- ANT-File may contain multiple targets (sub routines)
- Sub routines may have dependencies to each other
- Macros
- Error-Handling & Conditional execution
- Properties/Variables/Parameters

<APACHE ANT>

ANT

- Originally a Java-Build System
- Compiles Java-Code, executes JUnit Tests, etc.
- Other built in features (among many others):
  - File manipulations, copy, delete, …
  - ZIP, UNZIP
  - SCM Interaction
- https://ant.apache.org/manual/tasksoverview.html
- Extensible via plug-ins (offering further ANT Tasks)

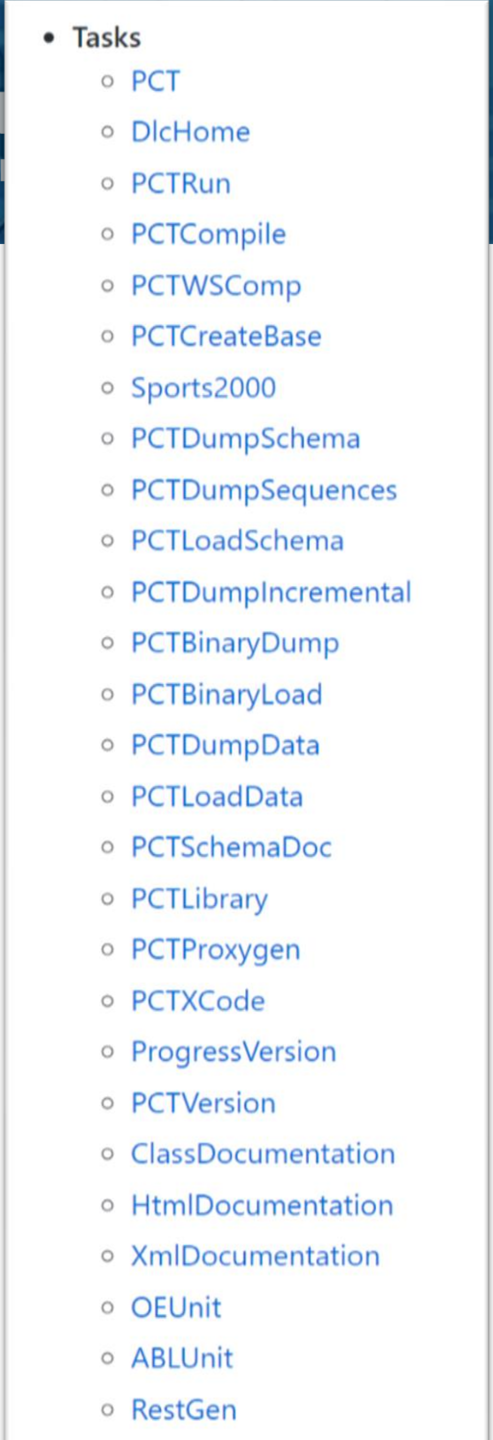
ANT

- ANT supports Unit Test execution
- ABLUnit Task delivered by PSC
- ABLUnit Task in PCT
- PCTRun to execute your own unit tests
- ANT scripts may be executed as part of a build pipeline, nightly builds, after every source code commit

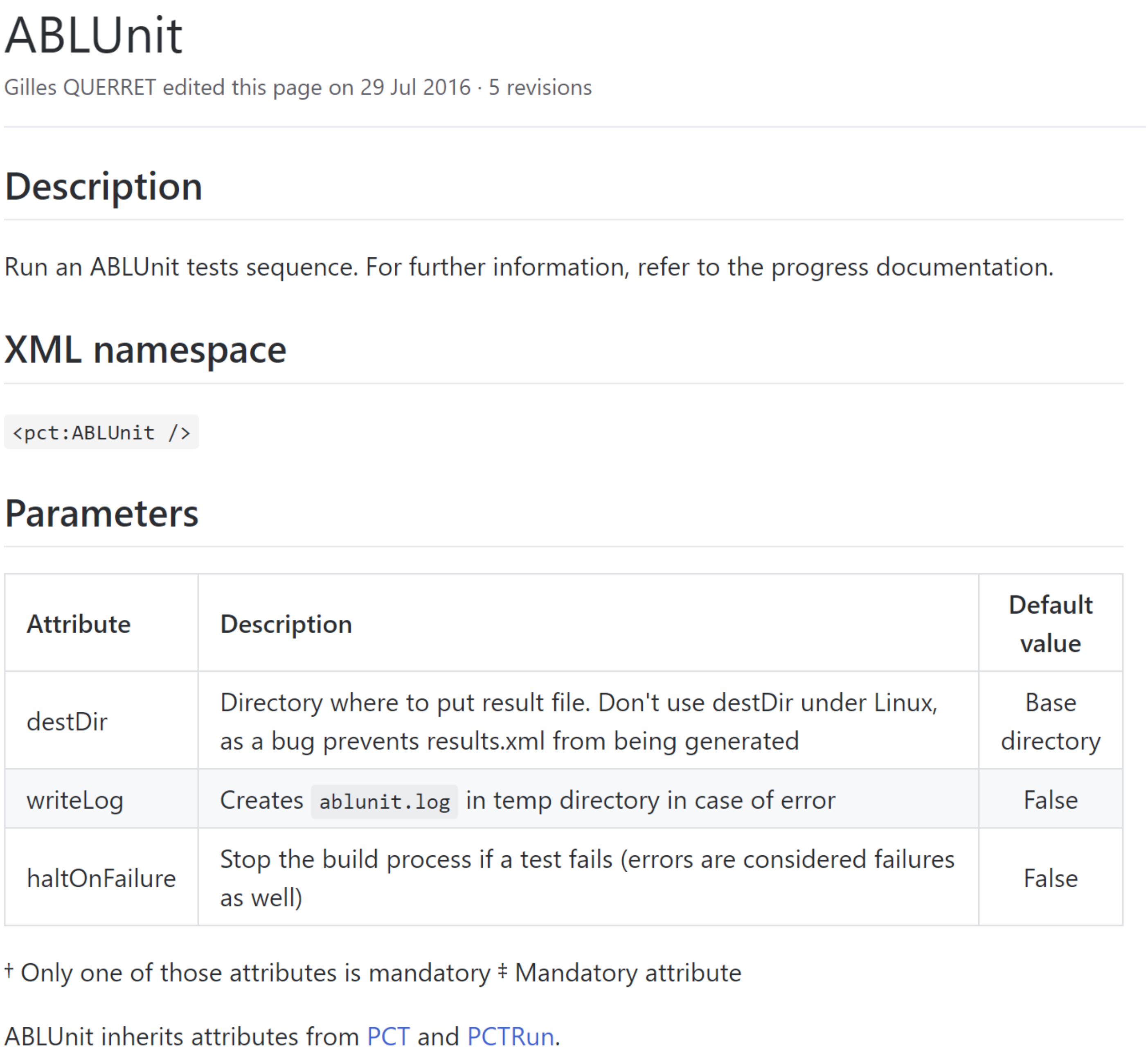
```
<target name="runtests">

    <ABLUnit destDir="Demo/UnitTesting/Simple" dlcHome="${progress.DLC}">
        <fileset dir="Demo/UnitTesting/Simple" includes="**/*.cls" />
        <propath>
            <pathelement path="." />
            <pathelement path="../ABL" />
        </propath>

        <DBConnection dbName="sports2000" dbDir="c:/Work/SmartComponents4NET/117_64/DB/sports2000" singleUser="true">
            <PCTAlias name="dictdb" />
        </DBConnection>

    </ABLUnit>

    <exec executable="c:\Users\${env.USERNAME}\AppData\Roaming\npm\junit-viewer.cmd" dir="Demo/UnitTesting/Simple">
        <arg value="--results=."/>
        <arg value="--save=results.html"/>
    </exec>

    <exec executable="c:\Windows\System32\cmd.exe" dir="Demo/UnitTesting/Simple">
        <arg value="/c"/>
        <arg value="start"/>
        <arg value="results.html"/>
    </exec>

</target>
```

PCT

- https://github.com/Riverside-Software/pct
- ANT tasks for OpenEdge
- Progress Compiler Tools
- open-source
- „Support“ via Github Issue-Tracking



- Tasks
  - PCT
  - DlcHome
  - PCTRun
  - PCTCompile
  - PCTWSComp
  - PCTCreateBase
  - Sports2000
  - PCTDumpSchema
  - PCTDumpSequences
  - PCTLoadSchema
  - PCTDumpIncremental
  - PCTBinaryDump
  - PCTBinaryLoad
  - PCTDumpData
  - PCTLoadData
  - PCTSchemaDoc
  - PCTLibrary
  - PCTProxygen
  - PCTXCode
  - ProgressVersion
  - PCTVersion
  - ClassDocumentation
  - HtmlDocumentation
  - XmlDocumentation
  - OEUnit
  - ABLUnit
  - RestGen



ABLUnit

Gilles QUERRET edited this page on 29 Jul 2016 · 5 revisions

Description

Run an ABLUnit tests sequence. For further information, refer to the progress documentation.

XML namespace

```
<pct:ABLUnit />
```

Parameters

Attribute	Description	Default value
destDir	Directory where to put result file. Don't use destDir under Linux, as a bug prevents results.xml from being generated	Base directory
writeLog	Creates `ablunit.log` in temp directory in case of error	False
haltOnFailure	Stop the build process if a test fails (errors are considered failures as well)	False

† Only one of those attributes is mandatory ‡ Mandatory attribute

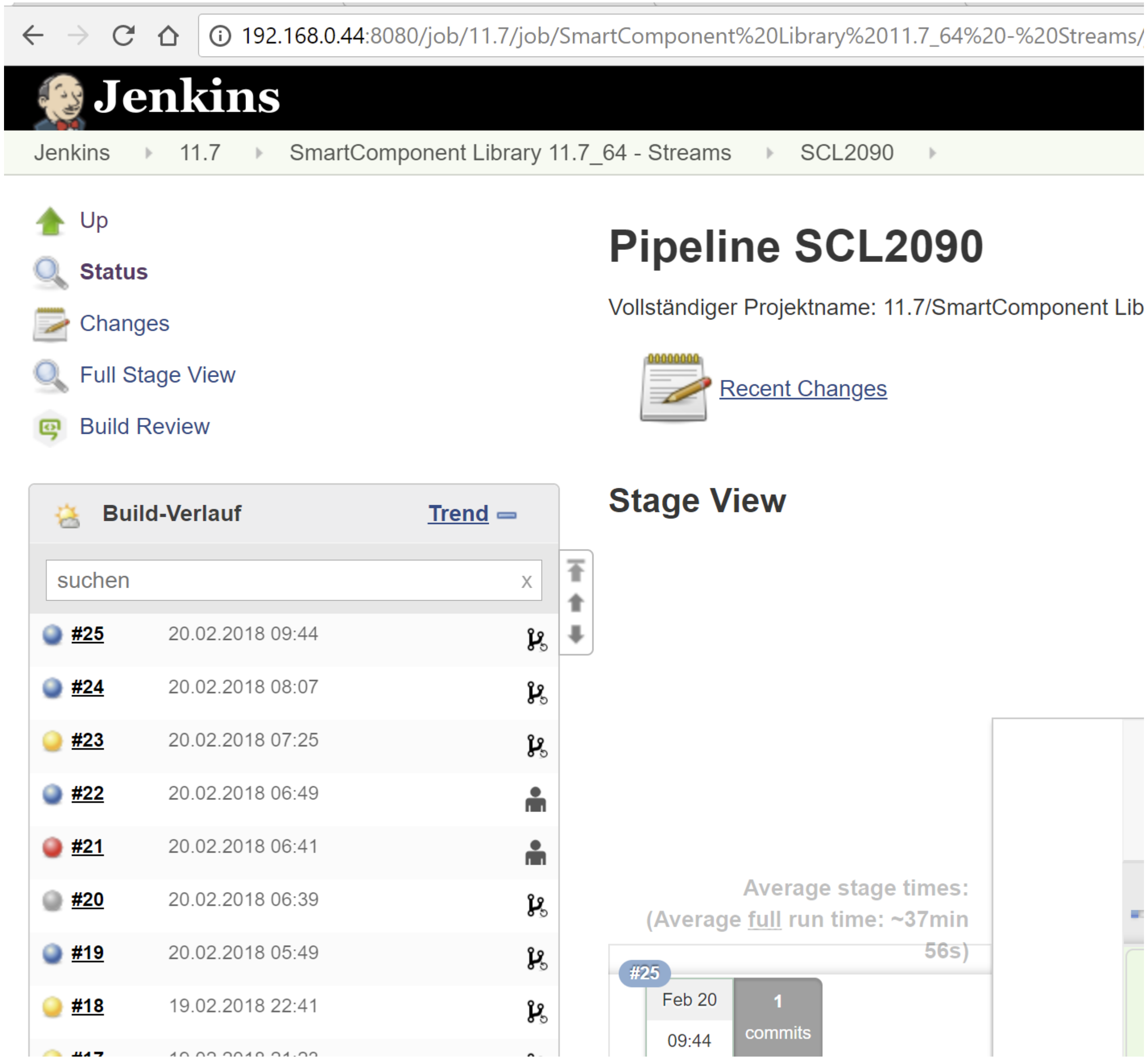
ABLUnit inherits attributes from PCT and PCTRun.

Jenkins CI Server

- Continuous Integration – permanent merging of various changes
- Forked from Hudson CI
- Standard tool for centralized execution of build and test jobs
- Controlled environment for the execution of (Build or Test) „Jobs“
- Visualization of success or failure of jobs, visualization of Unit Test results
- Emails on failure or other events

Jenkins CI Server

- Executes ANT scripts (and other scripts)
- Imports JUnit result files
- Provides trending on stability of software project
- Can propagate build artefacts based on test results



192.168.0.44:8080/job/11.7/job/SmartComponent%20Library%2011.7_64%20-%20Streams/

Jenkins

Jenkins › 11.7 › SmartComponent Library 11.7_64 - Streams › SCL2090 ›

- Up
- **Status**
- Changes
- Full Stage View
- Build Review

Build-Verlauf Trend

suchen x

#25	20.02.2018 09:44
#24	20.02.2018 08:07
#23	20.02.2018 07:25
#22	20.02.2018 06:49
#21	20.02.2018 06:41
#20	20.02.2018 06:39
#19	20.02.2018 05:49
#18	19.02.2018 22:41

Pipeline SCL2090

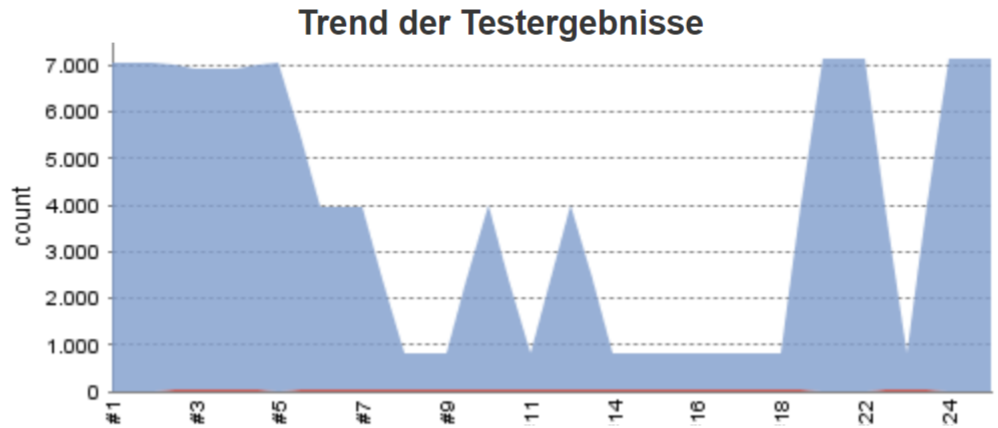
Vollständiger Projektname: 11.7/SmartComponent Lib

Recent Changes

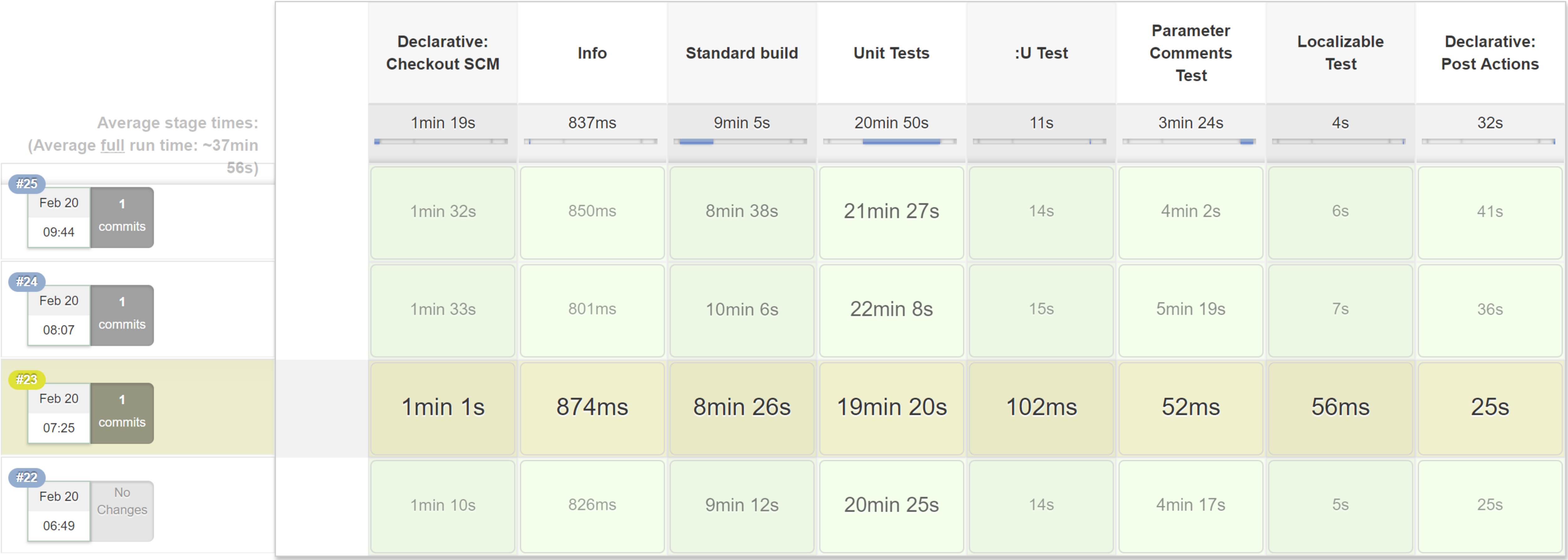
Stage View

Average stage times:
(Average full run time: ~37min 56s)

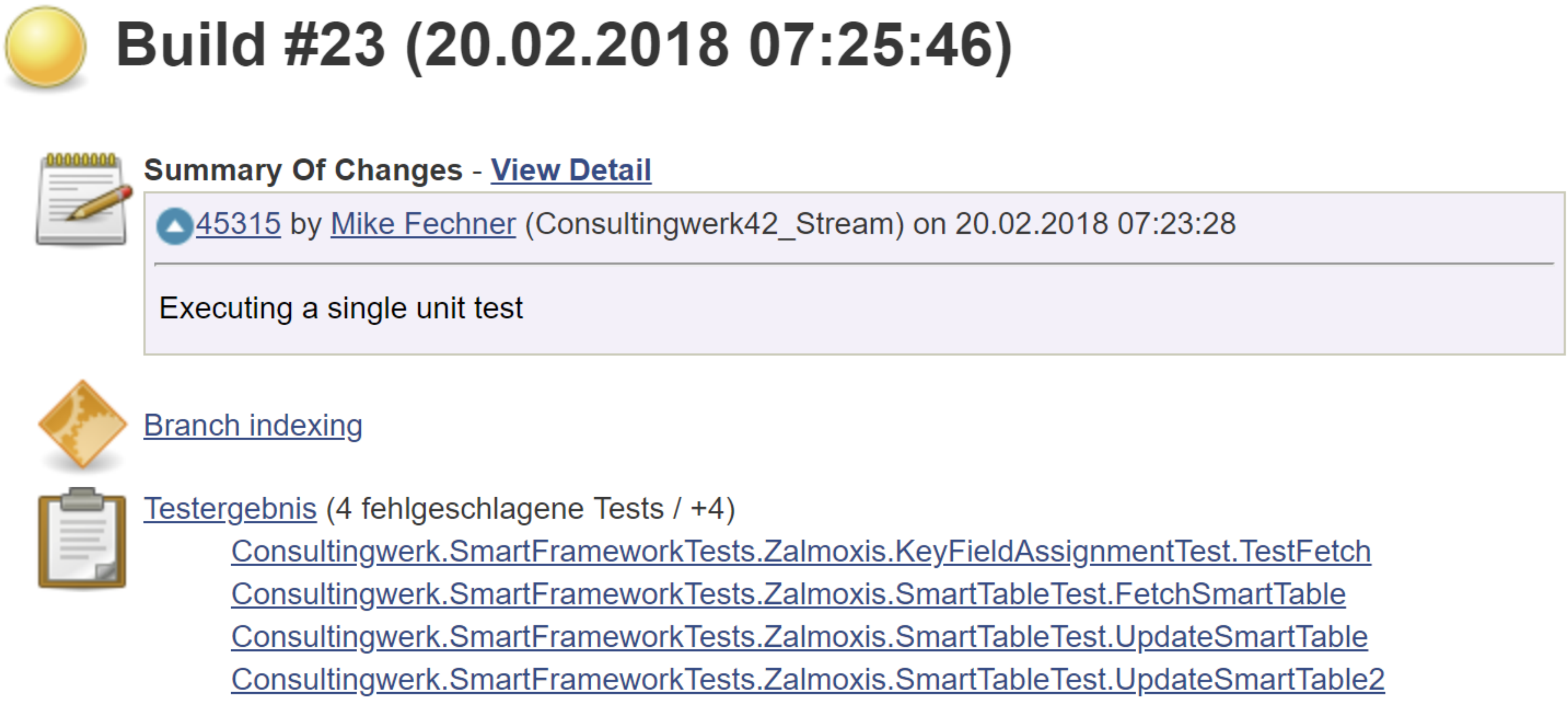
#25 Feb 20 09:44 1 commits



Trend der Testergebnisse



	Declarative: Checkout SCM	Info	Standard build	Unit Tests	:U Test	Parameter Comments Test	Localizable Test	Declarative: Post Actions
Average stage times: (Average full run time: ~37min 56s)	1min 19s	837ms	9min 5s	20min 50s	11s	3min 24s	4s	32s
#25 Feb 20 09:44 1 commits	1min 32s	850ms	8min 38s	21min 27s	14s	4min 2s	6s	41s
#24 Feb 20 08:07 1 commits	1min 33s	801ms	10min 6s	22min 8s	15s	5min 19s	7s	36s
#23 Feb 20 07:25 1 commits	1min 1s	874ms	8min 26s	19min 20s	102ms	52ms	56ms	25s
#22 Feb 20 06:49 No Changes	1min 10s	826ms	9min 12s	20min 25s	14s	4min 17s	5s	25s



Build #23 (20.02.2018 07:25:46)

Summary Of Changes - **View Detail**

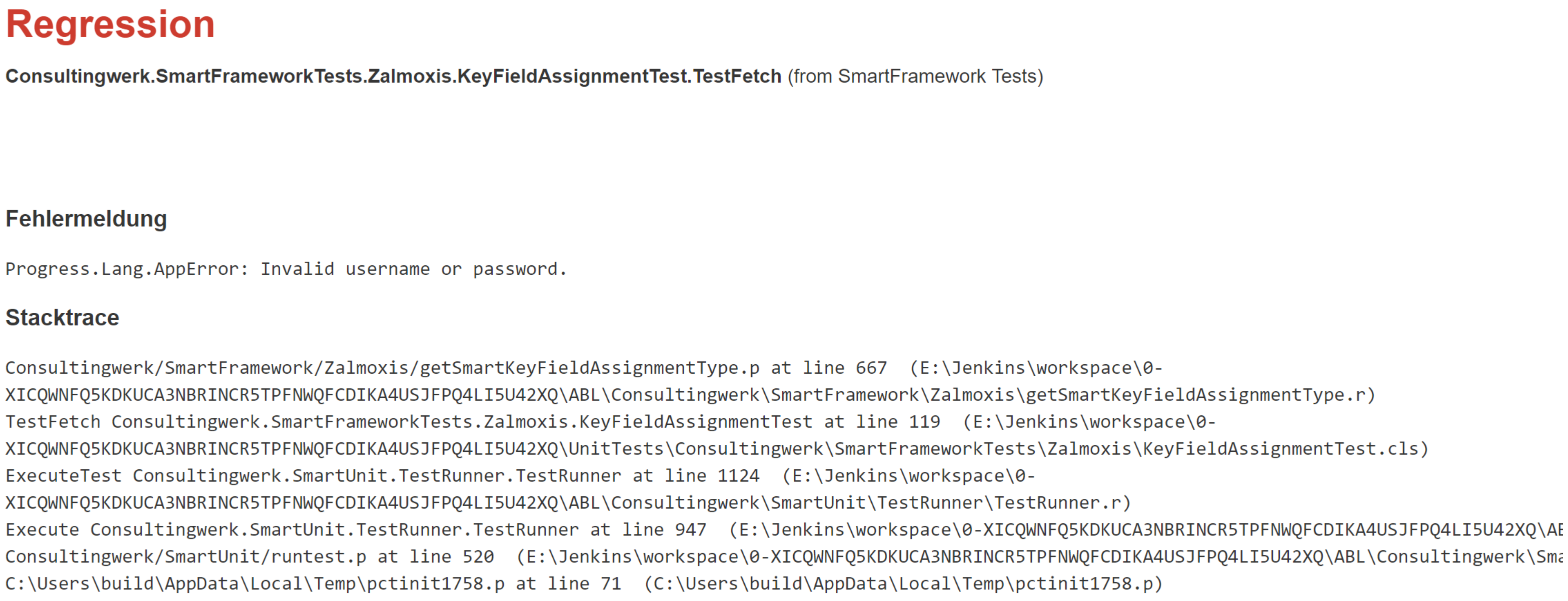
45315 by Mike Fechner (Consultingwerk42_Stream) on 20.02.2018 07:23:28

Executing a single unit test

Branch indexing

Testergebnis (4 fehlgeschlagene Tests / +4)

- Consultingwerk.SmartFrameworkTests.Zalmoxis.KeyFieldAssignmentTest.TestFetch
- Consultingwerk.SmartFrameworkTests.Zalmoxis.SmartTableTest.FetchSmartTable
- Consultingwerk.SmartFrameworkTests.Zalmoxis.SmartTableTest.UpdateSmartTable
- Consultingwerk.SmartFrameworkTests.Zalmoxis.SmartTableTest.UpdateSmartTable2



Regression

Consultingwerk.SmartFrameworkTests.Zalmoxis.KeyFieldAssignmentTest.TestFetch (from SmartFramework Tests)

Fehlermeldung

```
Progress.Lang.AppError: Invalid username or password.
```

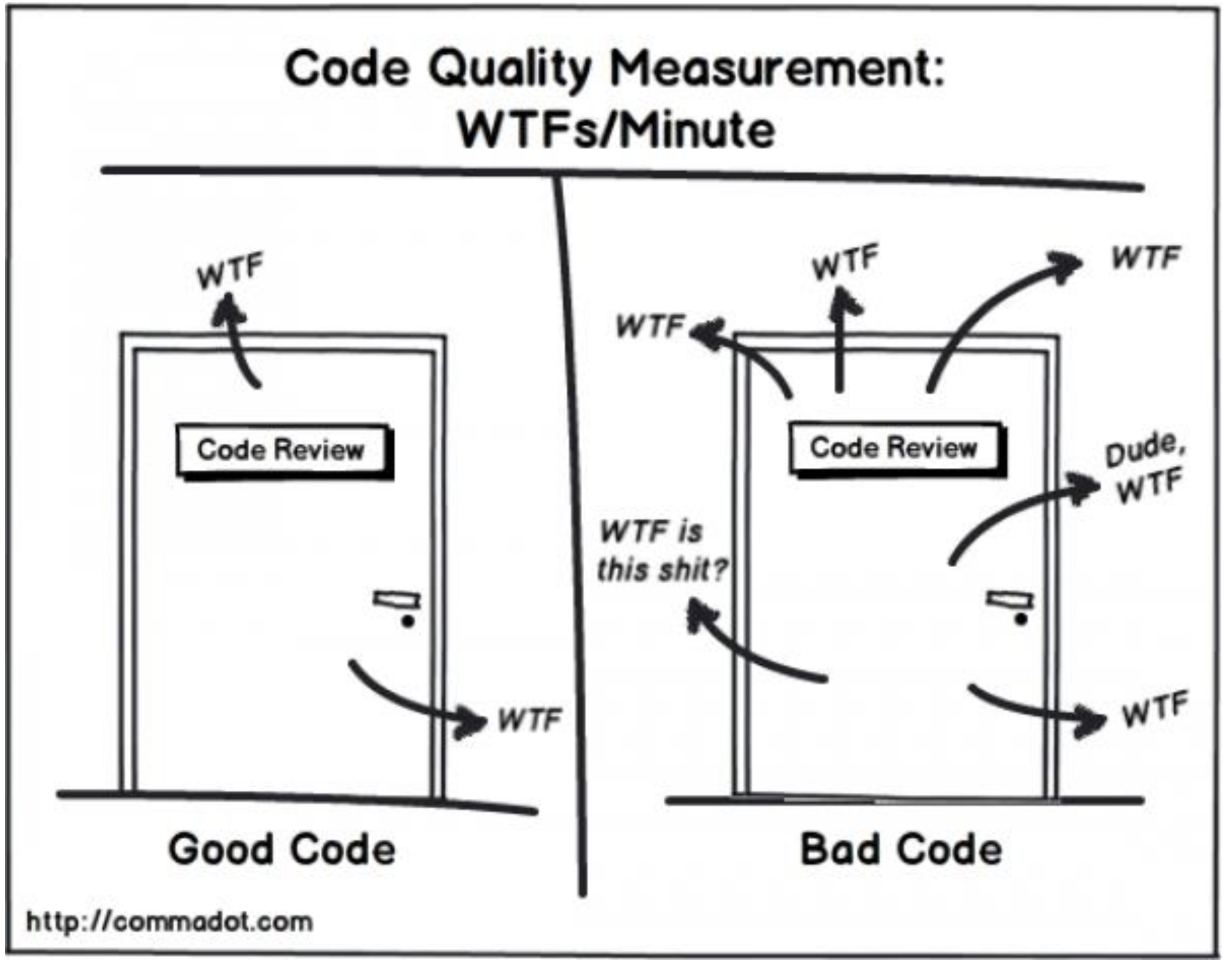
Stacktrace

```
Consultingwerk/SmartFramework/Zalmoxis/getSmartKeyFieldAssignmentType.p at line 667  (E:\Jenkins\workspace\0-
XICQWNFQ5KDKUCA3NBRINCR5TPFNWQFCDIKA4USJFPQ4LI5U42XQ\ABL\Consultingwerk\SmartFramework\Zalmoxis\getSmartKeyFieldAssignmentType.r)
TestFetch Consultingwerk.SmartFrameworkTests.Zalmoxis.KeyFieldAssignmentTest at line 119  (E:\Jenkins\workspace\0-
XICQWNFQ5KDKUCA3NBRINCR5TPFNWQFCDIKA4USJFPQ4LI5U42XQ\UnitTests\Consultingwerk\SmartFrameworkTests\Zalmoxis\KeyFieldAssignmentTest.cls)
ExecuteTest Consultingwerk.SmartUnit.TestRunner.TestRunner at line 1124  (E:\Jenkins\workspace\0-
XICQWNFQ5KDKUCA3NBRINCR5TPFNWQFCDIKA4USJFPQ4LI5U42XQ\ABL\Consultingwerk\SmartUnit\TestRunner\TestRunner.r)
Execute Consultingwerk.SmartUnit.TestRunner.TestRunner at line 947  (E:\Jenkins\workspace\0-XICQWNFQ5KDKUCA3NBRINCR5TPFNWQFCDIKA4USJFPQ4LI5U42XQ\AB
Consultingwerk/SmartUnit/runtest.p at line 520  (E:\Jenkins\workspace\0-XICQWNFQ5KDKUCA3NBRINCR5TPFNWQFCDIKA4USJFPQ4LI5U42XQ\ABL\Consultingwerk\Sma
C:\Users\build\AppData\Local\Temp\pctinit1758.p at line 71  (C:\Users\build\AppData\Local\Temp\pctinit1758.p)
```

Measuring your Unit Test Coverage

- Unit Test Coverage: % of lines of code which are executed during unit tests
- There are only two kinds of people that know there Unit Test Coverage:
  - Those that don’t use Unit Tests at all
  - Those that measure Unit Test Coverage using SonarSource

SonarSource: Code Quality measuring



SonarQube by SonarSource

- Commonly used Lint tool
- Support for various programming languages via plug-ins
- Java, JavaScript, C#, HTML, XML, CSS, …
- OpenEdge Plugin developed by Riverside Software (Gilles Querret)
  - engine open source
  - rules commercial
- Available since 2016, permanently new features added

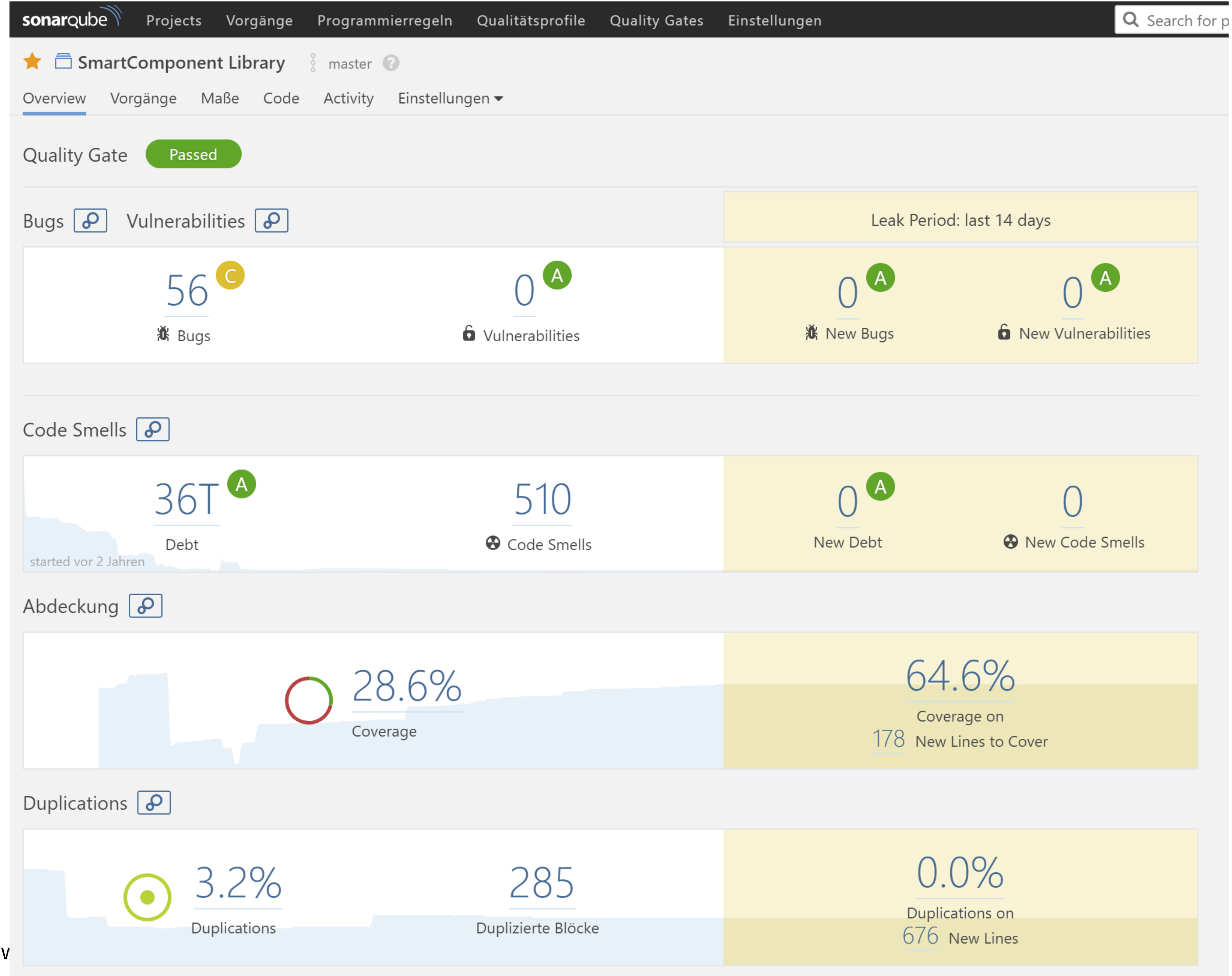
SonarQube by SonarSource

- Locates problems or potential bugs
- Violation of coding-standards
- Code duplication
- **Unit-Test coverage**

- Web-Dashboard
- CLI Utility (HTML or XML Reports)
- Eclipse Integration

Demo

- Sonar Lint Plugin into Progress Developer Studio



SmartComponent Library master

Quality Gate Passed

Bugs Vulnerabilities

Leak Period: last 14 days

56 C Bugs

0 A Vulnerabilities

0 A New Bugs

0 A New Vulnerabilities

Code Smells

36T A Debt

started vor 2 Jahren

510 Code Smells

0 A New Debt

0 New Code Smells

Abdeckung

28.6% Coverage

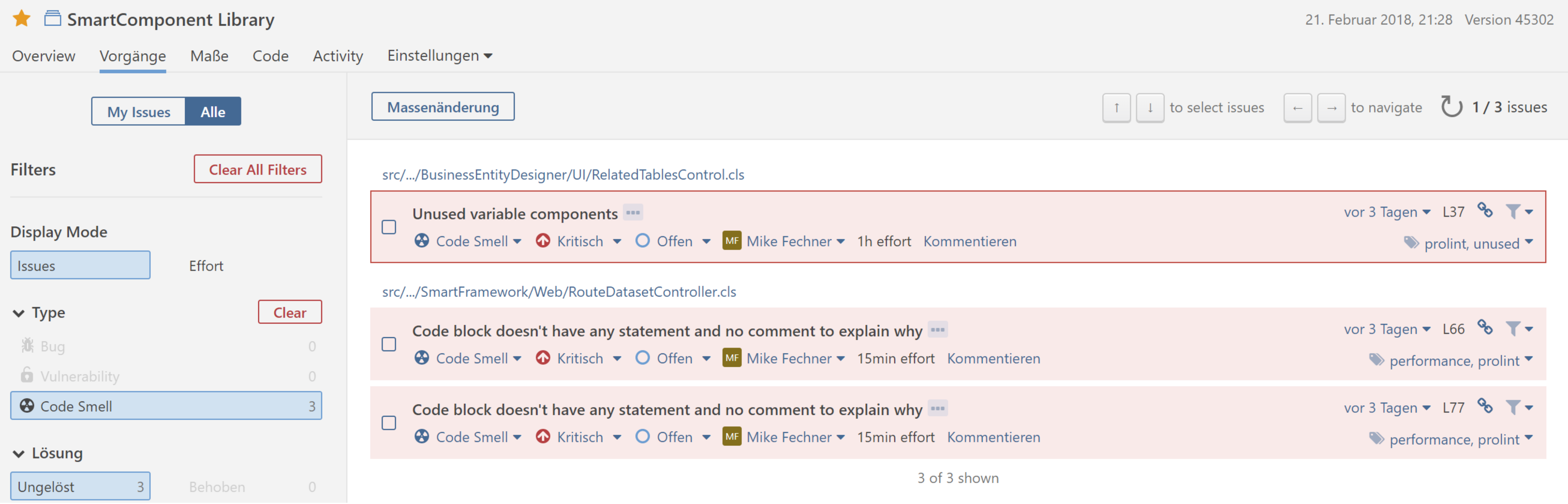
64.6% Coverage on 178 New Lines to Cover

Duplications

3.2% Duplications

285 Duplizierte Blöcke

0.0% Duplications on 676 New Lines



★ SmartComponent Library

21. Februar 2018, 21:28 Version 45302

Overview | Vorgänge | Maße | Code | Activity | Einstellungen ▾

My Issues | Alle

Massenänderung

↑ ↓ to select issues ← → to navigate ↻ 1 / 3 issues

Filters Clear All Filters

Display Mode

Issues | Effort

Type Clear

Bug	0
Vulnerability	0
Code Smell	3

Lösung

Ungelöst	3	Behoben	0

src/.../BusinessEntityDesigner/UI/RelatedTablesControl.cls

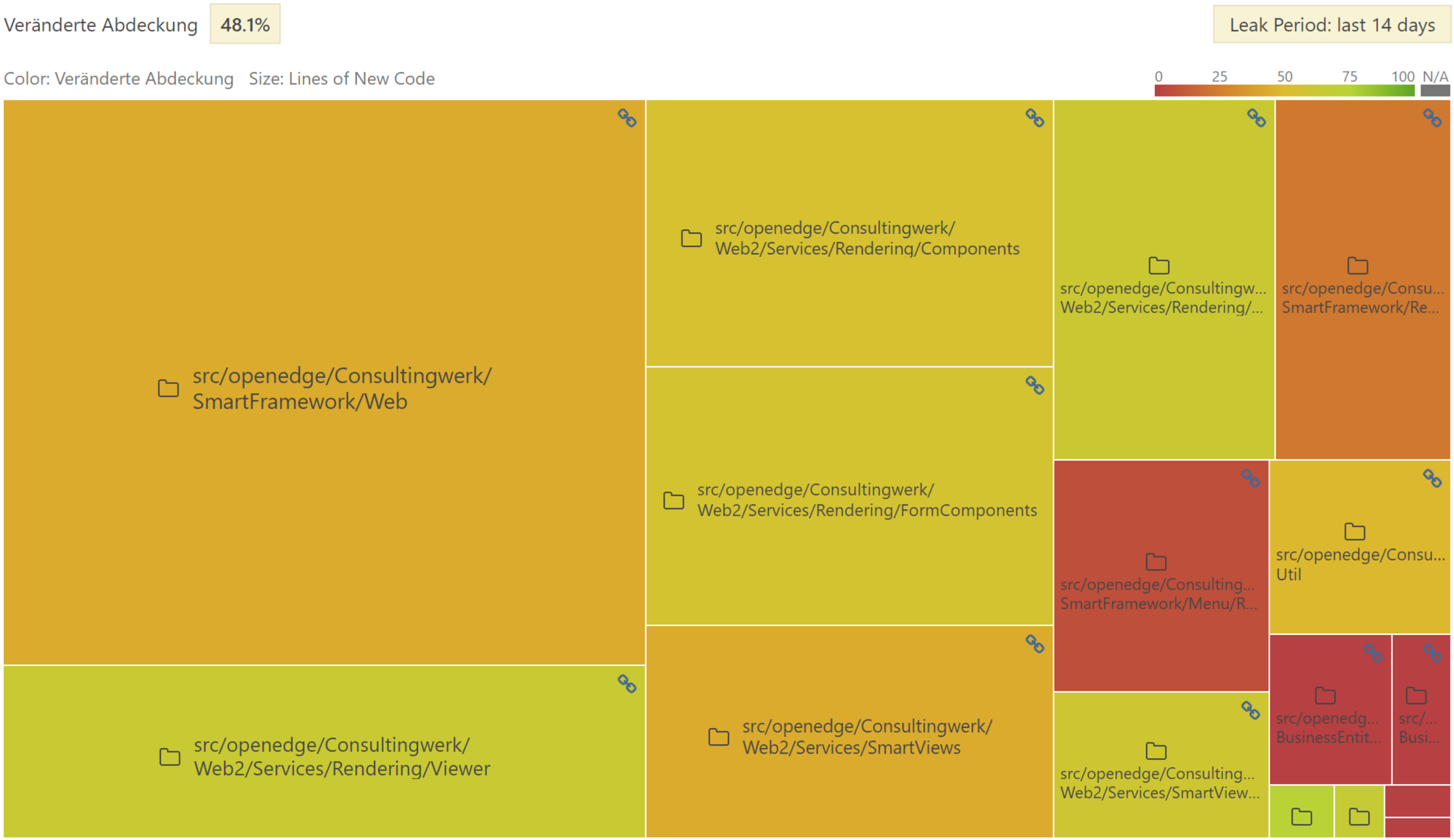
□ Unused variable components ···
Code Smell ▾ Kritisch ▾ ○ Offen ▾ MF Mike Fechner ▾ 1h effort Kommentieren
vor 3 Tagen ▾ L37 — prolint, unused ▾

src/.../SmartFramework/Web/RouteDatasetController.cls

□ Code block doesn't have any statement and no comment to explain why ···
Code Smell ▾ Kritisch ▾ ○ Offen ▾ MF Mike Fechner ▾ 15min effort Kommentieren
vor 3 Tagen ▾ L66 — performance, prolint ▾

□ Code block doesn't have any statement and no comment to explain why ···
Code Smell ▾ Kritisch ▾ ○ Offen ▾ MF Mike Fechner ▾ 15min effort Kommentieren
vor 3 Tagen ▾ L77 — performance, prolint ▾

3 of 3 shown

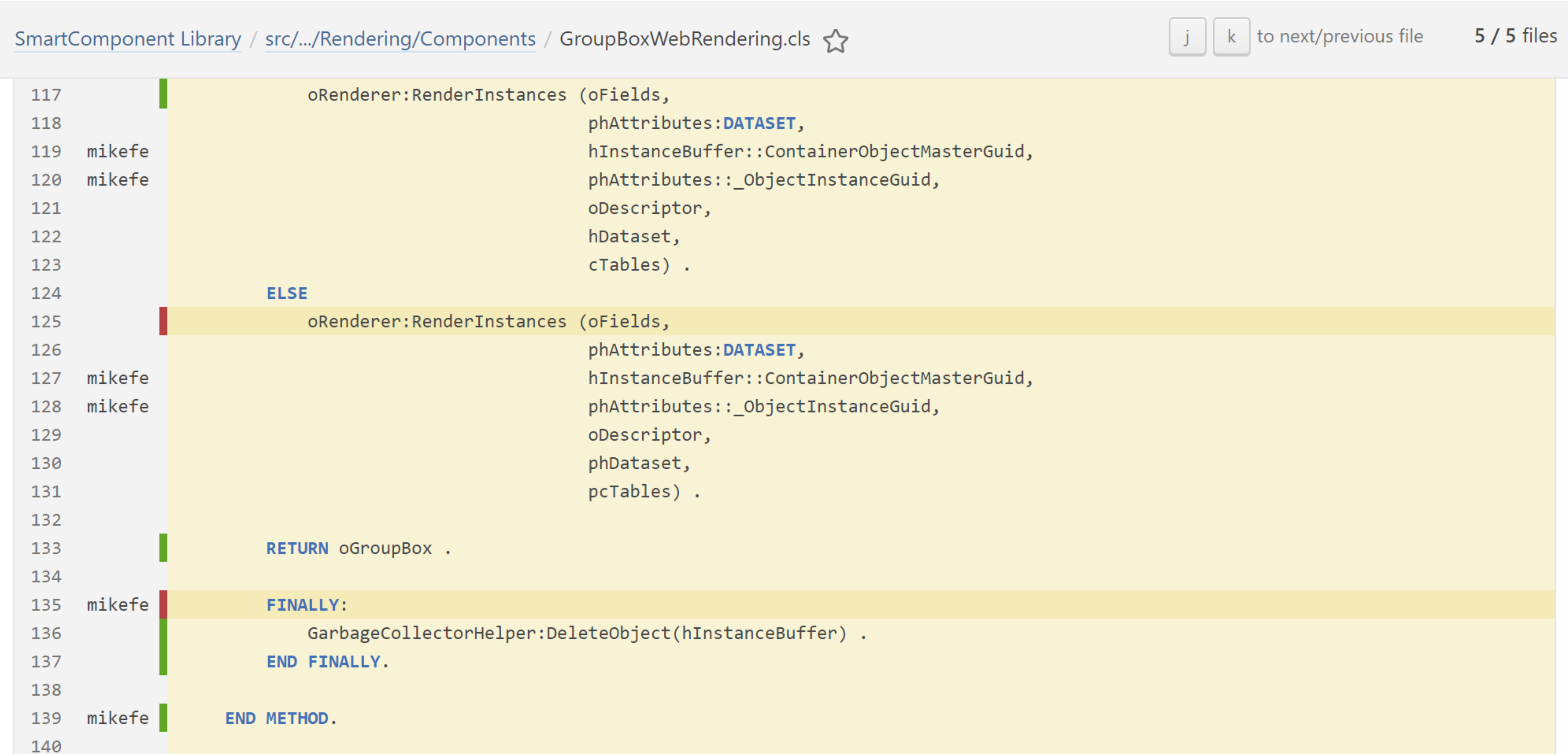


Veränderte Abdeckung 48.1%

Leak Period: last 14 days

Color: Veränderte Abdeckung Size: Lines of New Code

0
25
50
75
100
N/A
src/openedge/Consultingwerk/SmartFramework/Web
src/openedge/Consultingwerk/Web2/Services/Rendering/Viewer
src/openedge/Consultingwerk/Web2/Services/Rendering/Components
src/openedge/Consultingwerk/Web2/Services/Rendering/FormComponents
src/openedge/Consultingwerk/Web2/Services/SmartViews
src/openedge/Consultingw... Web2/Services/Rendering/...
src/openedge/Consu... SmartFramework/Re...
src/openedge/Consulting... SmartFramework/Menu/R...
src/openedge/Consu... Util
src/openedge/Consulting... Web2/Services/SmartView...
src/openedg... BusinessEntit...
src/... Busi...



SmartComponent Library / src/.../Rendering/Components / GroupBoxWebRendering.cls ☆

```
                    oRenderer:RenderInstances (oFields,
                                               phAttributes:DATASET,
        mikefe                                 hInstanceBuffer::ContainerObjectMasterGuid,
        mikefe                                 phAttributes::_ObjectInstanceGuid,
                                               oDescriptor,
                                               hDataset,
                                               cTables) .
                ELSE
                    oRenderer:RenderInstances (oFields,
                                               phAttributes:DATASET,
        mikefe                                 hInstanceBuffer::ContainerObjectMasterGuid,
        mikefe                                 phAttributes::_ObjectInstanceGuid,
                                               oDescriptor,
                                               phDataset,
                                               pcTables) .

                RETURN oGroupBox .

        mikefe  FINALLY:
                    GarbageCollectorHelper:DeleteObject(hInstanceBuffer) .
                END FINALLY.

        mikefe  END METHOD.

```

Agenda

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

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

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

Agenda

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

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?

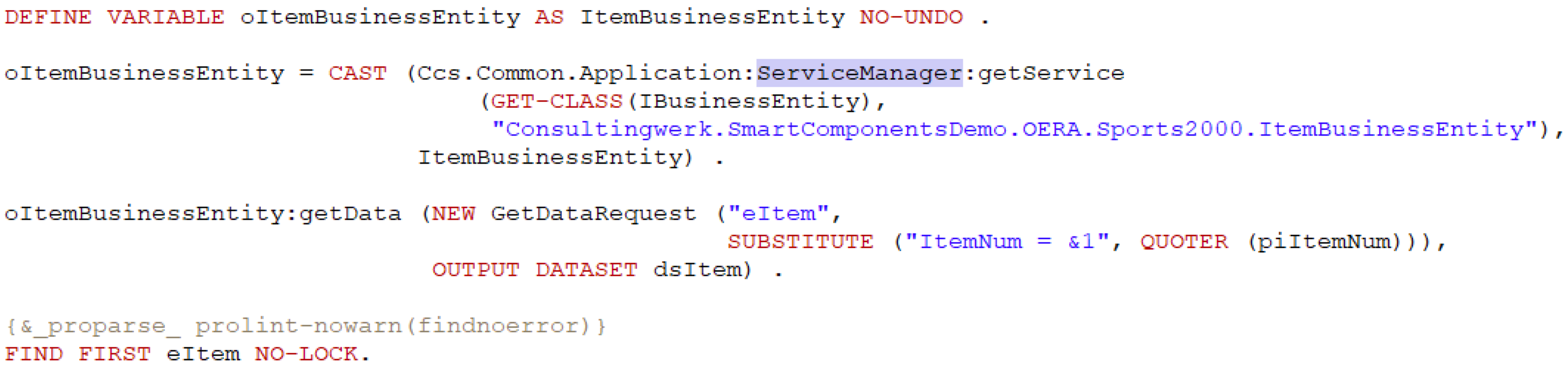
Mocking Dependencies - Wikipedia

- “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**
- Advanced Unit Testing Features

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 …

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

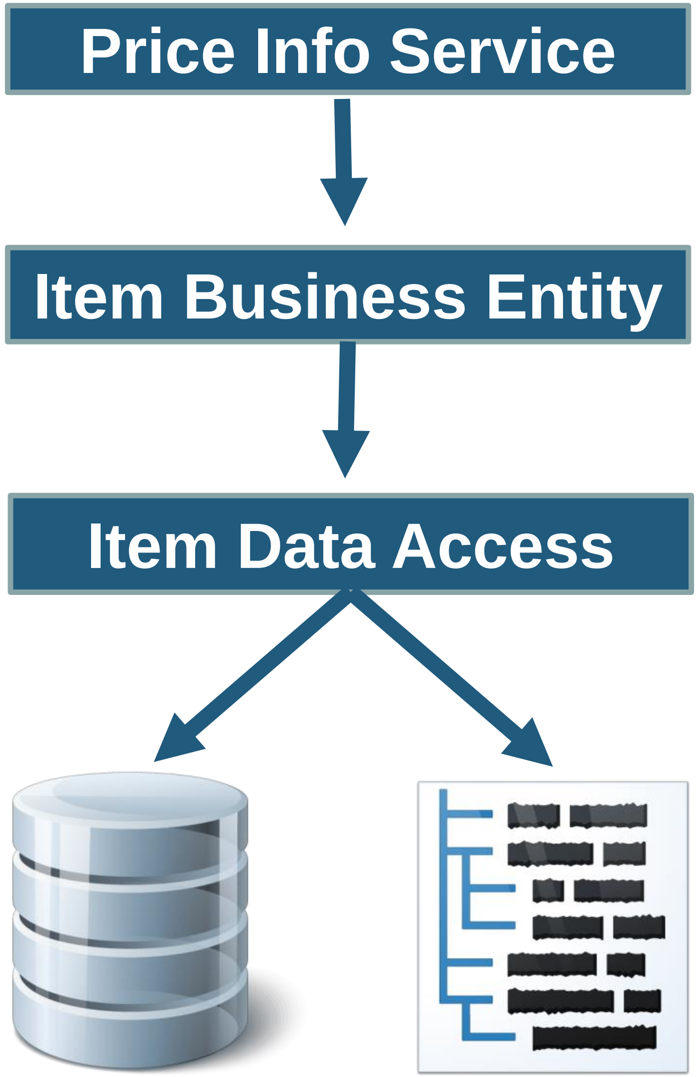
- Always restore a known database state from a backup
- Or rebuild a database for each test run from .d and .df
  - This may be easier when the database schema may change during a test sequence
- You may need to rebuild a database multiple times during a test sequence
- Produces lots of Disk I/O
- Disk 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



Agenda

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

Scenario driven Unit Tests

- Many Unit Tests are alike
- Testing read functionality of Business Entitiy a very repeating tasks
- Should test for runtime performance characteristics
  - Runtime (subject to system performance fluctuations)
  - Records accessed in database
- Should test for values (e.g. calculated values)
- Tests can be expressed as scenario instead of code

SmartUnit Feature

- Unit Test tool of the SmartComponent Library
- https://documentation.consultingwerkcloud.com/display/SCL/Scenario+based+Unit+Tests+for+Business+Entity+FetchData+%28read%29+operations

Markup Driven Assertions

- Read Operations
  - NumResults
  - CanFind (allows to find for Unique Key + Calculated Field value)
  - CanNotFind
  - MaxRuntime (may fail, when test server is busy)
  - MaxReads (in the database)
- Update Operations
  - Expected validation messages or similar output

Questions

Consultingwerk

software architecture and development