Consultingwerk
software architecture and development

ABL Unit Testing Mocking

Mike Fechner
Director

Consultingwerk

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

Agenda

- **Introduction**
- A simple ABL 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 40 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 (test) automation

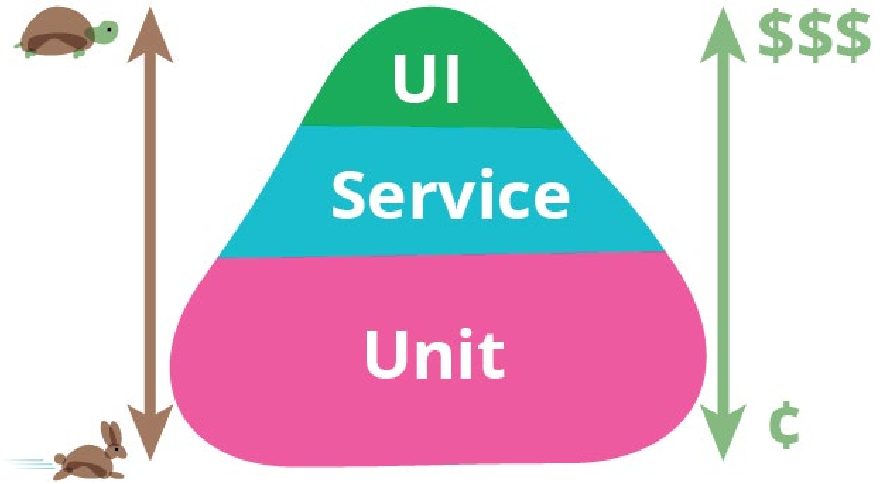
Introduction

- *“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

Consultingwerk
software architecture and development

The test pyramid

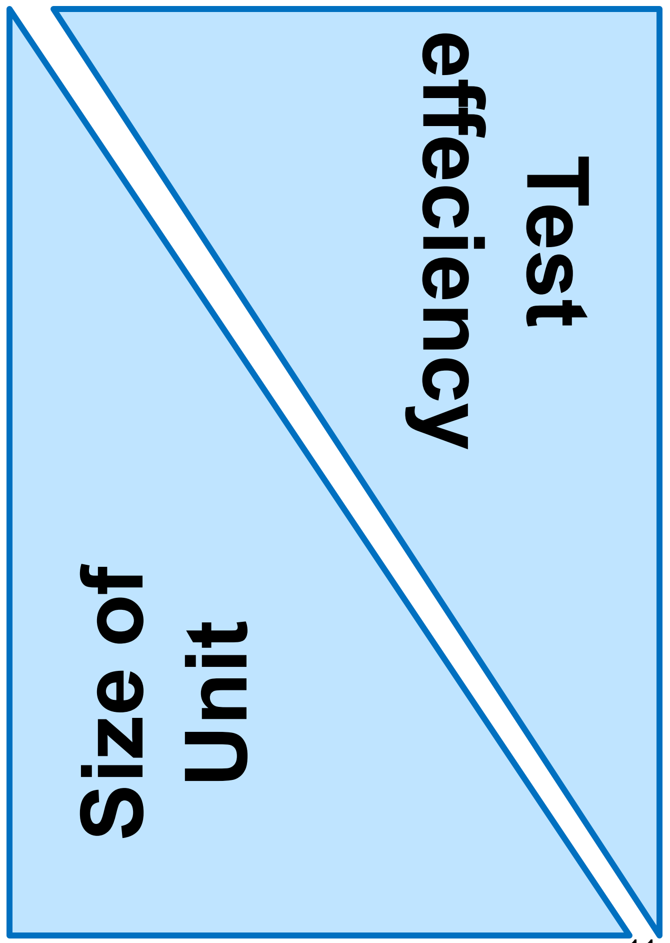
- Symbolizes different kind of tests that can be used to automate testing a (layered) application
- Unit Tests are relatively simple (cheap) to program, there should be lots of them
- API/Service Tests are more complex to write
- UI Tests are the most expensive to write and may require humans to execute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



Unit Tests

- In automated unit tests, a testable unit should be as small as possible

- A package?
- A screen?
- A class hierarchy?
- A class?
- A method?



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**
- 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.
```

Consultingwerk
software architecture and development

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)

Agenda

- Introduction
- A simple ABL 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

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

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

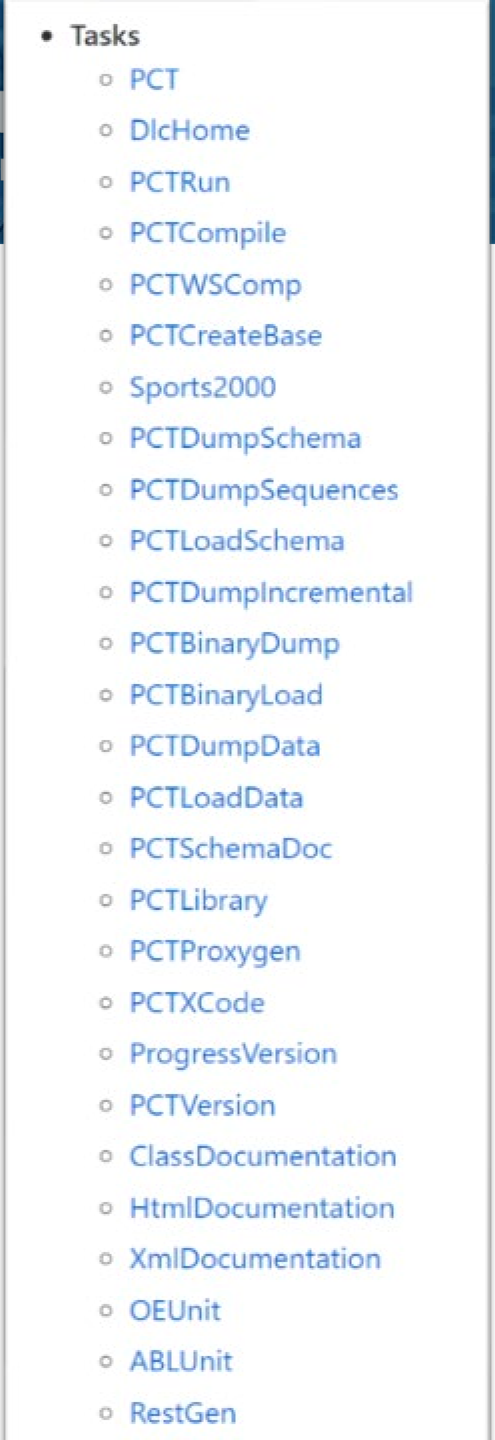
- ANT supports Unit Test execution
- ABLUnit Task delivered by PSC
- ABLUnit Task in PCT
- PCTRun to execute your own unit tests
- ANT scripts may be executed as part of a build pipeline, nightly builds, after every source code commit

```
<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

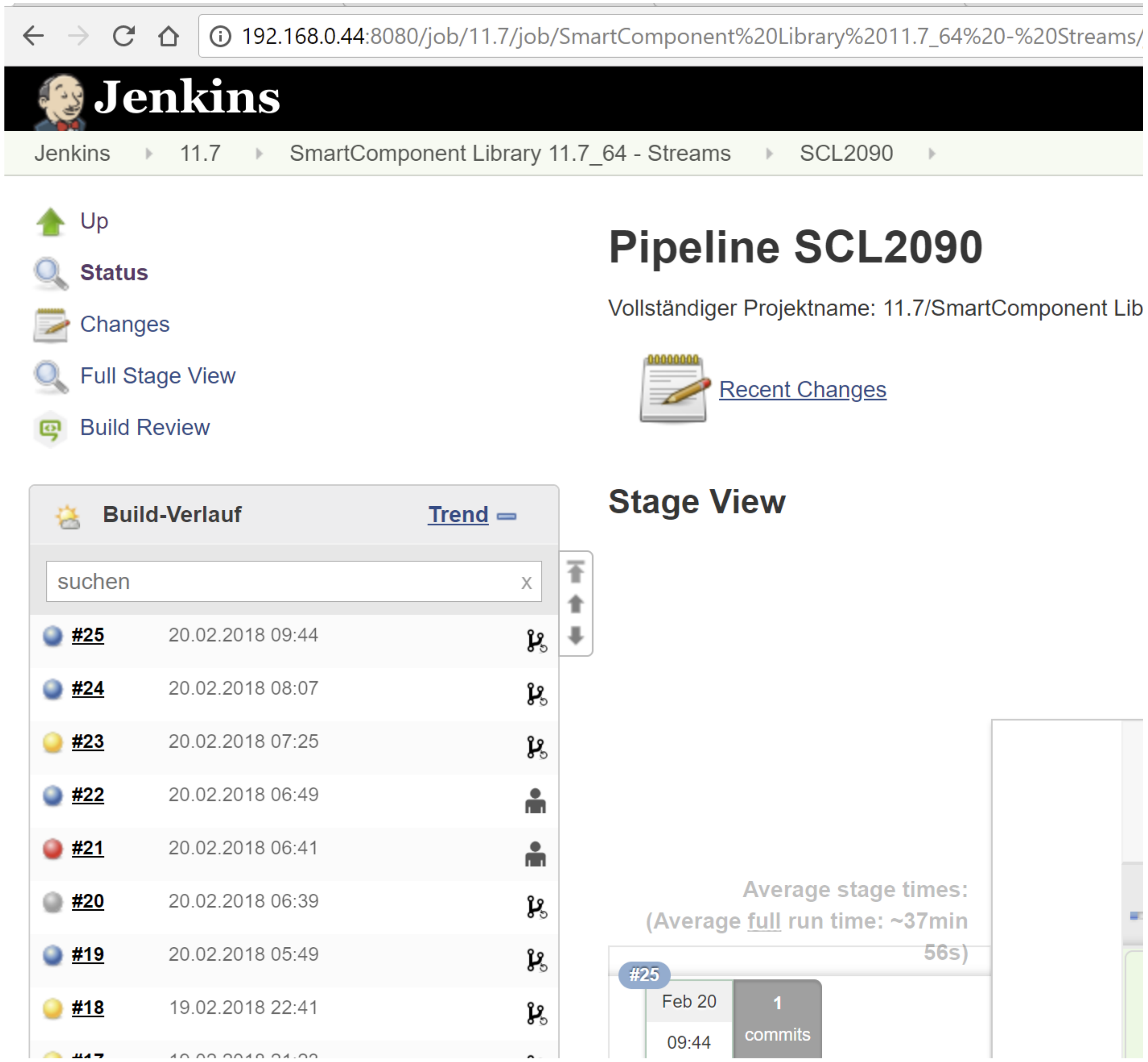
Consultingwerk
software architecture and development

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

Pipeline SCL2090

Vollständiger Projektname: 11.7/SmartComponent Lib

Recent Changes

Stage View

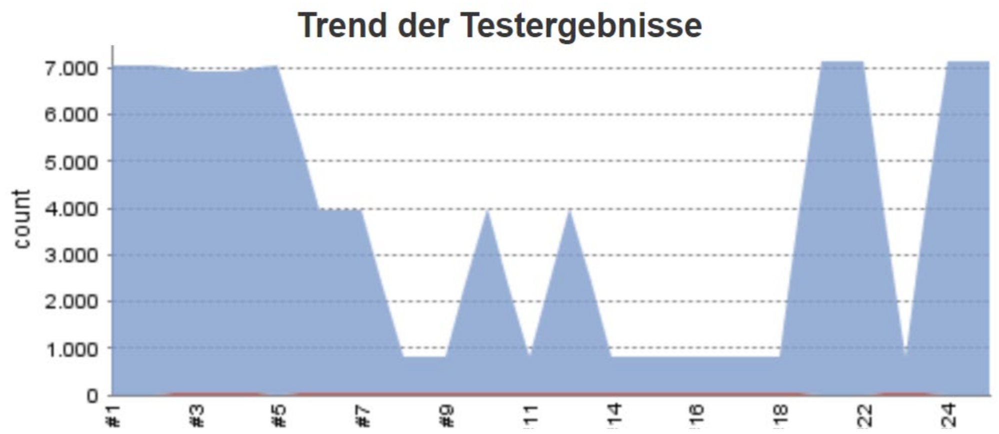
Build-Verlauf Trend

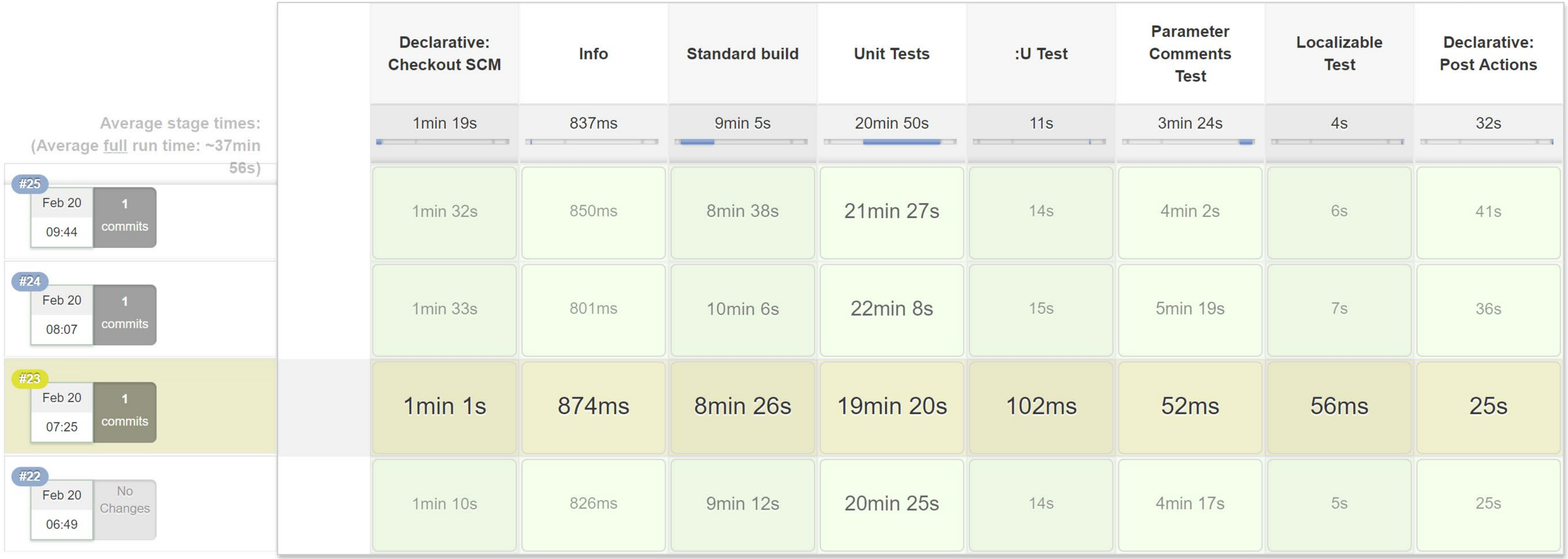
suchen x

●	#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

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

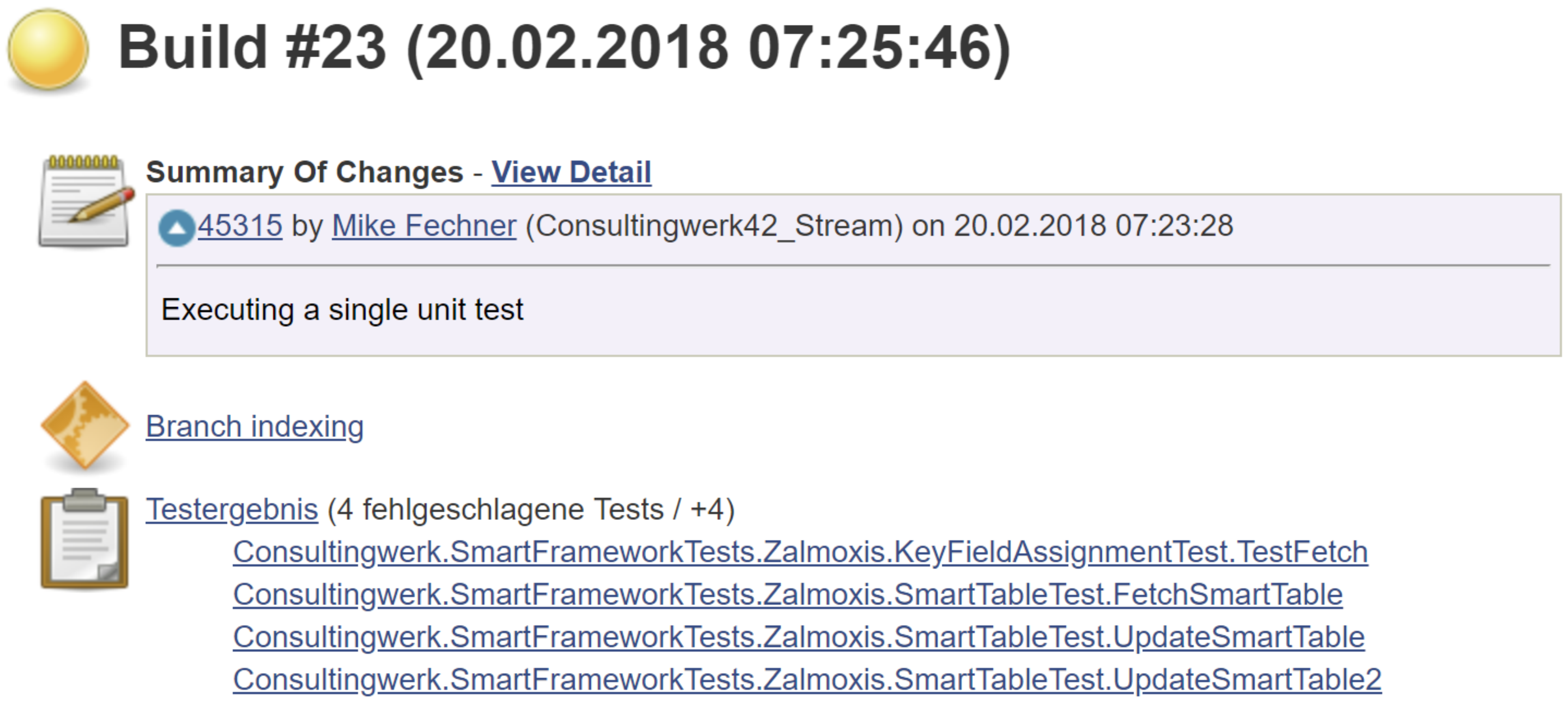
#25
Feb 20
09:44
1 commits





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

	Declarative: Checkout SCM	Info	Standard build	Unit Tests	:U Test	Parameter Comments Test	Localizable Test	Declarative: Post Actions
	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

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 their Unit Test Coverage:
  - Those that don’t use Unit Tests at all
  - Those that measure Unit Test Coverage using SonarSource

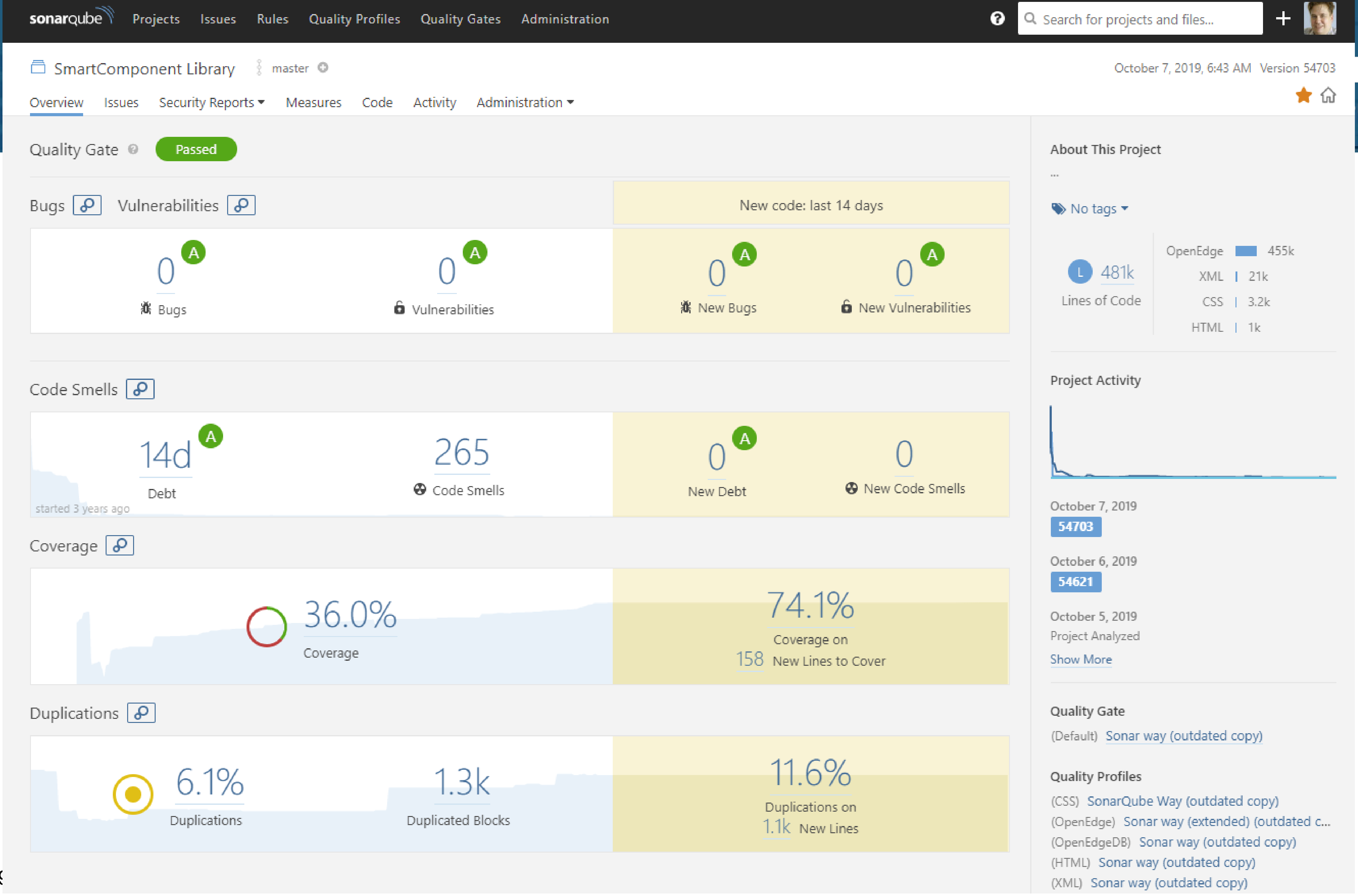
SonarQube by SonarSource

- 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
  - CABL included in OpenEdge 12.0 with small set of rules
- 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


sonarqube Projects Issues Rules Quality Profiles Quality Gates Administration

Search for projects and files...

SmartComponent Library master

October 7, 2019, 6:43 AM Version 54703

Overview Issues Security Reports Measures Code Activity Administration

Quality Gate Passed

Bugs Vulnerabilities

New code: last 14 days

0 A Bugs	0 A Vulnerabilities	0 A New Bugs	0 A New Vulnerabilities

Code Smells

14d A Debt	265 Code Smells	0 A New Debt	0 New Code Smells

started 3 years ago

Coverage

36.0% Coverage

74.1% Coverage on 158 New Lines to Cover

Duplications

6.1% Duplications

1.3k Duplicated Blocks

11.6% Duplications on 1.1k New Lines

About This Project

...

No tags

L 481k Lines of Code

OpenEdge	455k
XML	21k
CSS	3.2k
HTML	1k

Project Activity

October 7, 2019
54703

October 6, 2019
54621

October 5, 2019
Project Analyzed

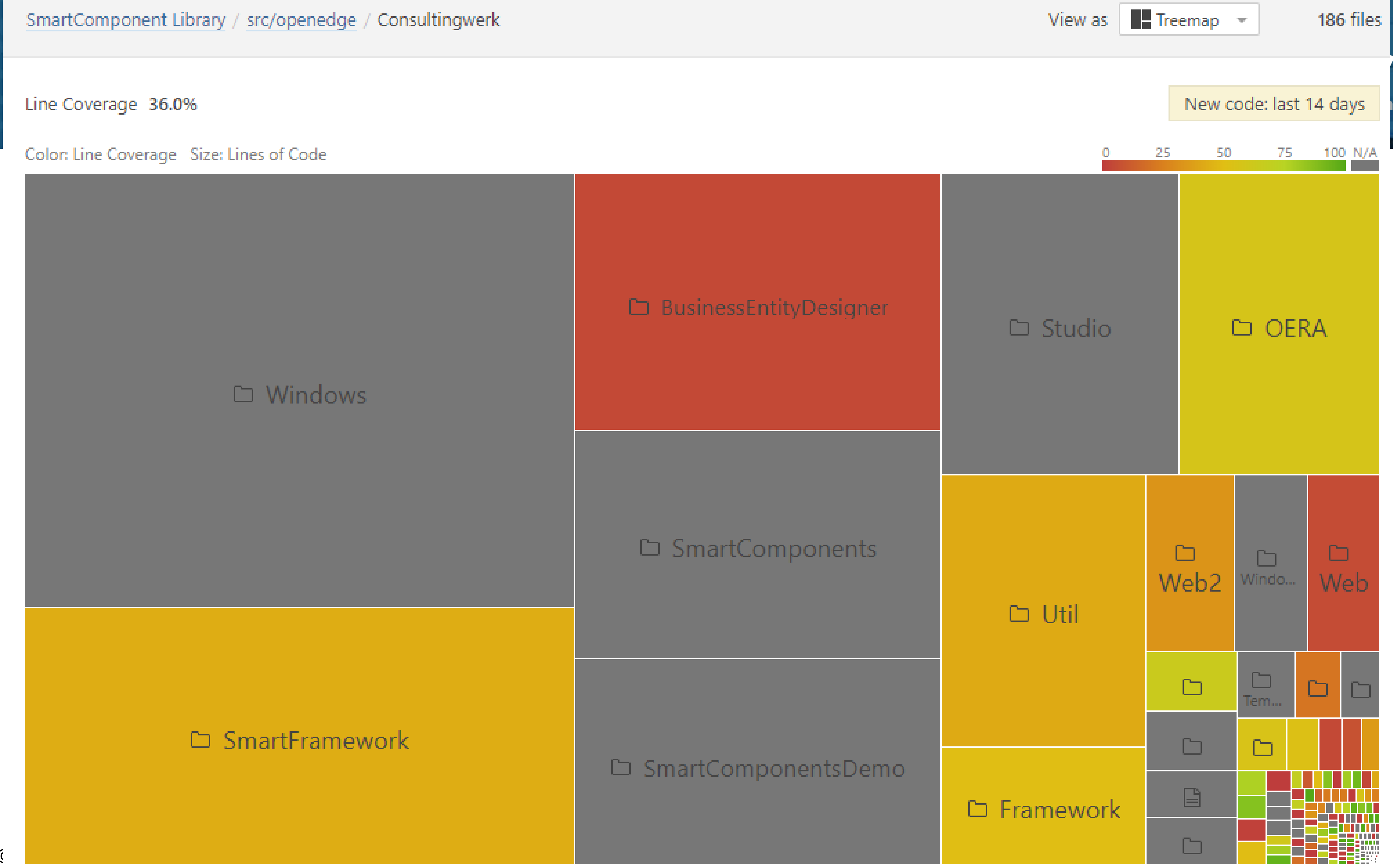
Show More

Quality Gate

(Default) Sonar way (outdated copy)

Quality Profiles

(CSS) SonarQube Way (outdated copy)
(OpenEdge) Sonar way (extended) (outdated c...
(OpenEdgeDB) Sonar way (outdated copy)
(HTML) Sonar way (outdated copy)
(XML) Sonar way (outdated copy)



SmartComponent Library / src/openedge / Consultingwerk

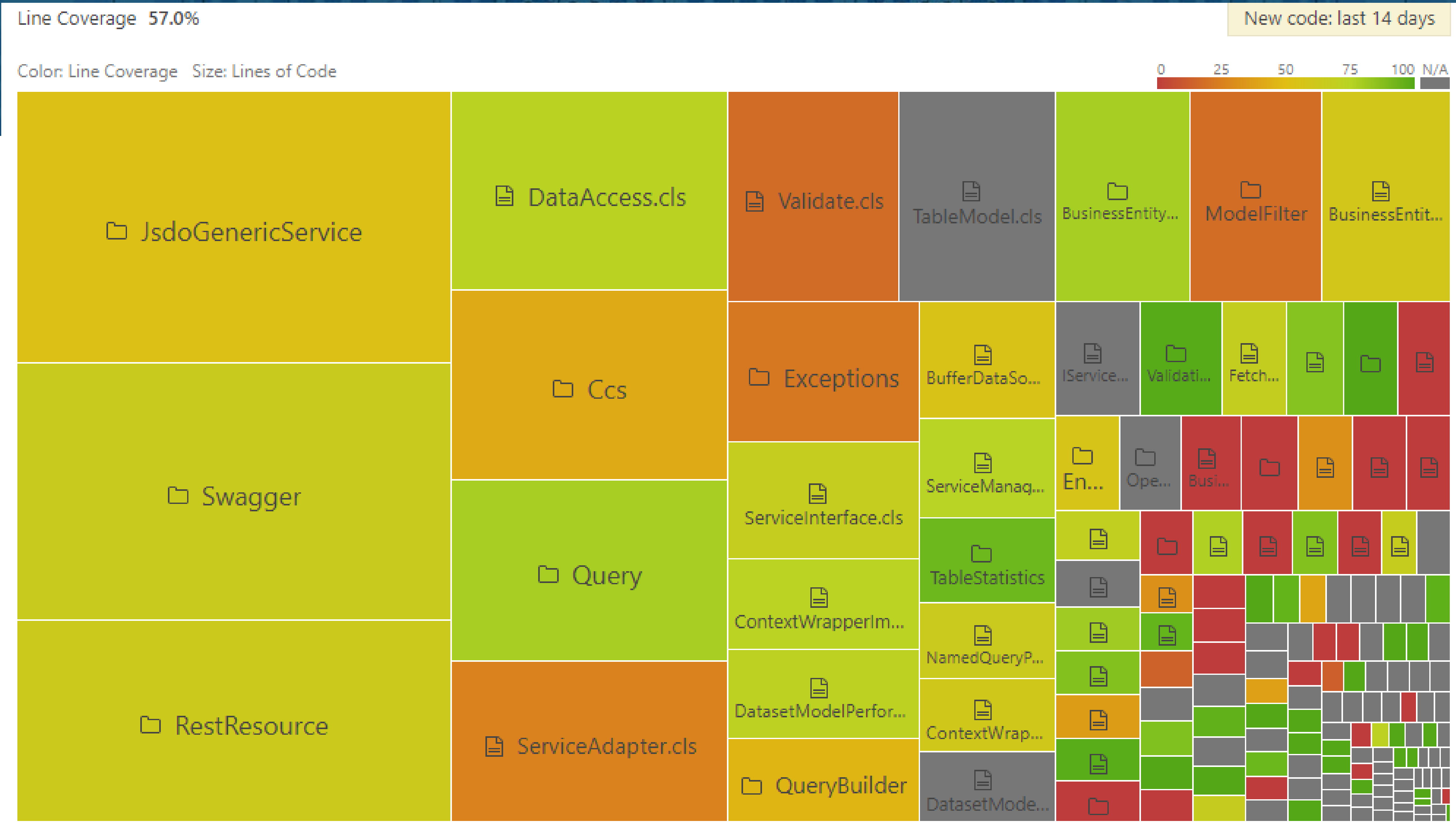
View as Treemap

186 files

Line Coverage 36.0%

New code: last 14 days

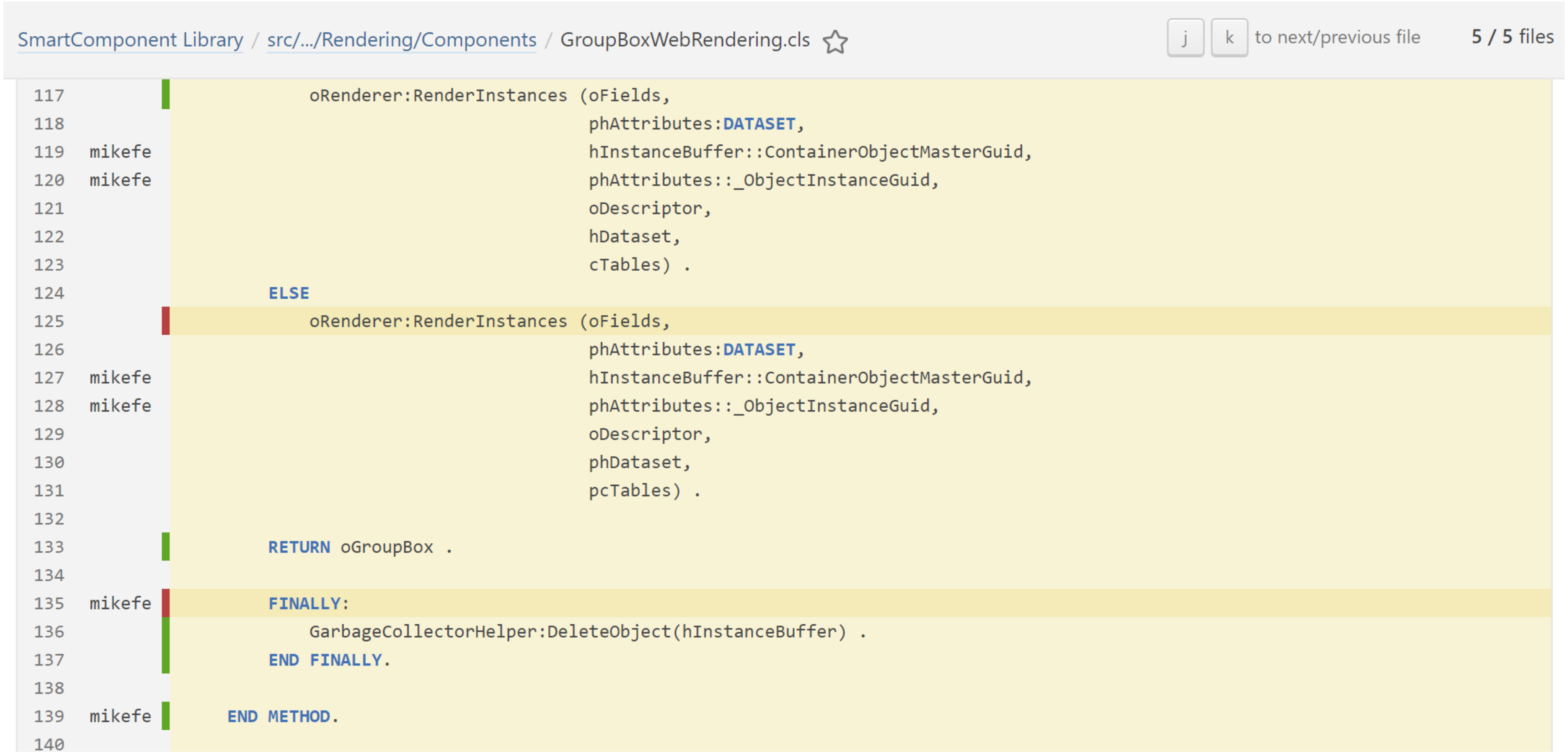
Color: Line Coverage Size: Lines of Code



Line Coverage 57.0%

New code: last 14 days

Color: Line Coverage Size: Lines of Code



SmartComponent Library / src/.../Rendering/Components / GroupBoxWebRendering.cls
j k to next/previous file 5 / 5 files

```
              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
- 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

Sample

- Unit-Test of an PASOE Web Handler based on test inheriting from the code to be tested

Consultingwerk
software architecture and development

```
@Test.
CLASS Consultingwerk.SmartFrameworkTests.Repository.SCL2090.FetchFormTest
    INHERITS SmartFormWebHandler:

 @Test.
 METHOD PUBLIC VOID TestFormByObjectName1 ():

     DEFINE VARIABLE oRequest AS MockWebRequest NO-UNDO .
     DEFINE VARIABLE oJson    AS JsonObject     NO-UNDO .

     oRequest = NEW MockWebRequest(?, "CustomerForm", ?, ?) .

     THIS-OBJECT:HandleGet(oRequest) .

     oJson = CAST (oResponse:Entity, JsonObject) .

     Assert:EqualsTrue(oJson:Has ("columns")) .
     Assert:GT(oJson:GetJsonArray ("columns"):Length, 0) .

 END METHOD.

/**
 * Purpose: Required to make assertion web output
 * Notes:
 * @param poResponse The http response generated by the web handler
 */
METHOD PROTECTED OVERRIDE VOID WriteResponse (poResponse AS OpenEdge.Net.HTTP.IHttpResponse):

    oResponse = CAST (poResponse, OpenEdge.Web.WebResponse) .

END METHOD.
```

Agenda

- Introduction
- A simple ABL 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.”

Consultingwerk
software architecture and development

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

Mocking of dependencies (code)

- Object to be tested may depend on:
- Framework Services, e.g. Token Security Service
- Application Services (Domain services), e.g. Currency Conversion Service
- Complex parameter object (e.g. WebRequest in previous slide)

Techniques for Mocking dependencies (code)

- Real world dependencies and mock objects are expected to implement same interface

- Constructor or Property Injection
- Service Locator, Service Manager in CCS, Service Container in .NET and SmartComponent Library (https://github.com/progress/ccs)

- See discussion at: https://www.martinfowler.com/articles/injection.html
- I personally have a preference for the service locator. This makes dependencies an implementation detail, not part of a contract

Sample mocking ITokenSecurityService

- Reasons for mocking the ITokenSecurityService
- Need to test **restricted and unrestricted authorization** of a critical business function – to achieve 100% test coverage
- Simpler than providing a Framework configuration with and without restriction for a certain functionality

- Using Constructor injection
- Using Overloading Service Container of the SmartComponent Library

Demo

- Using to inversion of control patterns to mock ITokenSecurityService

- Using service manager based inversion of control to mock Currency Calculation Service used in value object (Domain-driven design)

Agenda

- Introduction
- A simple ABL 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

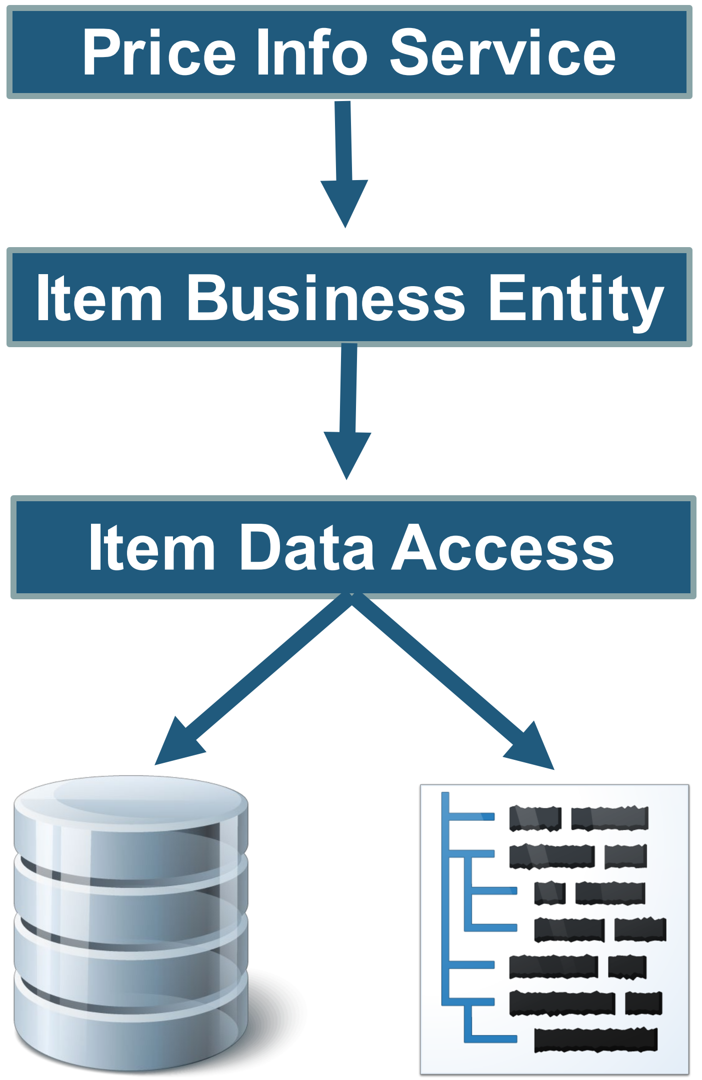
- Always restore a known database state from a backup
- Or rebuild a database for each test run from .d and .df
  - This may be easier when the database schema may change during a test sequence
- You may need to rebuild a database multiple times during a test sequence
- Produces lots of Disk I/O
- Disk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



Demo

- Mocking the Data Access using an XML file

Consultingwerk
software architecture and development

Agenda

- Introduction
- A simple ABL 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