ALIGNED

Aligned, Quality-centric, Software and Data Engineering

H2020 – 644055

D4.4 – Automated data testing & verification framework

Project Number:	644055
Project Title:	ALIGNED
Document Type: (Deliverable/Internal):	Deliverable
Deliverable Number:	D4.4
Deliverable Type (R/DEM/DEC/OTHER):	DEM- Demonstrator
Workpackages Contributing:	WP4
Dissemination Level (PU/CO):	PU - Public
Contractual Delivery Date:	July 2016
Actual Delivery Date:	July 2016
Version:	V1.01
Editor(s)/Lead Authors:	Dimitris Kontokostas, Christian Dirschl
With contributions from:	Katja Eck, Markus Freudenbeurg, Andreas Koller
Reviewers(s):	Rob Brennan

Abstract:

This deliverable briefly describes our framework for bringing data engineering test and verification tools closer to software engineers by integrating them into familiar processes and tools.

Keyword List:

Data Quality, RDFUnit, JUnit, maven, Testing

History

Version	Date	Reason	Revised by
0.1	17/07/2016	First Draft	Dimitris Kontokostas
0.2	18/07/2016	Full Internal Draft for Review	Markus Freudenberg
0.3	20/07/2016	Review Revision	Rob Brennan
1.0	21/07/2016	Release Candidate	Rob Brennan
1.01	22/07/2016	Final Version	Rob Brennan

Author List

Organisation	Name	Contact Information
UL	Kontokostas Dimitris	kontokostas@informatik.uni-leipzig.de
WKD	Christian Dirschl	Christian.Dirschl@wolterskluwer.com
UL	Markus Freudenberg	freudenberg@informatik.uni-leipzig.de
TCD	Rob Brennan	rob.brennan@cs.tcd.ie

Executive Summary

This deliverable describes new automated tools and methods for test-driven data engineering using UL’s RDFUnit tool and for integrating RDFUnit into software engineering test tools and workflows. RDFUnit consumes RDF-based system specifications (schemas) or ALIGNED meta-models (especially RUT-based test case definitions) and automatically generates and executes test cases for verification of data consistency, completeness and coverage. The prototype implementation is integrated with Maven and JUnit and thus, it is part of of the software build reporting and documentation process. A demo video[1] has been prepared to accompany this deliverable and the source code is available on github.

[1] https://youtu.be/T0enhLikN54

Contents

List of Figures

1 Introduction

This deliverable enables testing and verification of RDF data in familiar Software Engineering processes, tools and infrastructures. It targets software engineers, by making it easier for them to use flexible data engineering testing tools like RDFUnit. It complements the conceptual modelling and support for unified governance of software and data engineering provided by D5.4 Test Case Linking and D5.2 Unified Governance Tools. During Phase 2 of ALIGNED the target use case for deployment of this tool (and hence source of most requirements) is JURION. D2.5 (Use Cases and Requirements) documents the automated data testing & verification framework requirements as established at the start of ALIGNED Phase 2.

In the first iteration of this deliverable we have accomplished the integration of UL's data testing framework RDFUnit into JUnit[2], a very popular Java Unit Testing framework. Choosing JUnit as an integration point was a strategic decision, since JUnit is widely adopted by software engineers. In addition:

- It can be run by Java and all JVM-based programming languages (e.g. Scala)
- It is integrated in popular build tools like Maven[3], Ant[4], SBT[5] or Gradle[6]
- It is integrated in popular Continuous Integration (CI) platforms like Bamboo[7], Jenkins[8] and Travis-CI[9], using the aforementioned build tools.
- JUnit tools and tool-chains are extensively deployed in the live JURION software and data engineering environment.

The automated data testing & verification framework described here will be included in the integrated tool-set (Phase 2 trial platform) for the JURION use case as part of D5.5 Integrated ALIGNED Tools in month 19. The framework performance will be reported on D6.5 Phase 3 Trial Reports and this will feed into D2.6 Phase 3 Use Cases and Requirements and D2.9 ALIGNED Methods Specification. An enhanced, second iteration of this framework will be produced in Phase 3 of the project as D4.7.

The contents of this overview of the demonstrator are as follows: Section 2 describes the location of the accompanying demo video and open source code. In Section 3 we provide a brief technical overview of the framework and the three means provided for automated data testing & verification through common software engineering processes: a) JUnit integration using Java annotations (Section 3.1), b) JUnit integration using the JUnit XML

[2] http://junit.org/
[3] https://maven.apache.org/
[4] http://ant.apache.org/
[5] http://www.scala-sbt.org/
[6] https://gradle.org/
[7] https://www.atlassian.com/software/bamboo
[8] https://jenkins.io/
[9] https://travis-ci.org/

Schema (Section 3.2) and c) Custom integration as a java library (Section 3.3). In Section 4 we provide our future plans and conclude.

2 Accompanying Demonstrator Video and Open Source Code

A short screencast video has been developed by ALIGNED which showcases the automated data testing & verification framework described in this deliverable and will be used to promote the project results. The screencast shows the prototype implementation in action. It is available on YouTube[10] and is linked through the project website[11].

The RDFUnit automated data testing & verification framework, including the extensions developed for this demonstrator is available as open source code and can be downloaded through the ALIGNED project website[12] and github[13].

3 Technical Overview

RDFUnit is an RDF validation framework inspired by test-driven software development. The test case definition language of RDFUnit is SPARQL, which is convenient to directly query for identifying violations. For rapid test case instantiation, a pattern-based SPARQL-Template engine is supported where the user can easily bind variables into patterns. RDFUnit has a Test Auto Generator (TAG) component. TAG searches for schema information and automatically instantiates new test cases. Schema information can be in the form of RDFS or OWL axioms that RDFUnit translates into SPARQL under Closed World Assumption (CWA) and Unique Name Assumption (UNA). Other schema languages such as SHACL[14], IBM Resource Shapes[15] or Description Set Profiles[16] are also supported. For a full overview of RDFUnit's data testing and verification capabilities see [1] and [2].

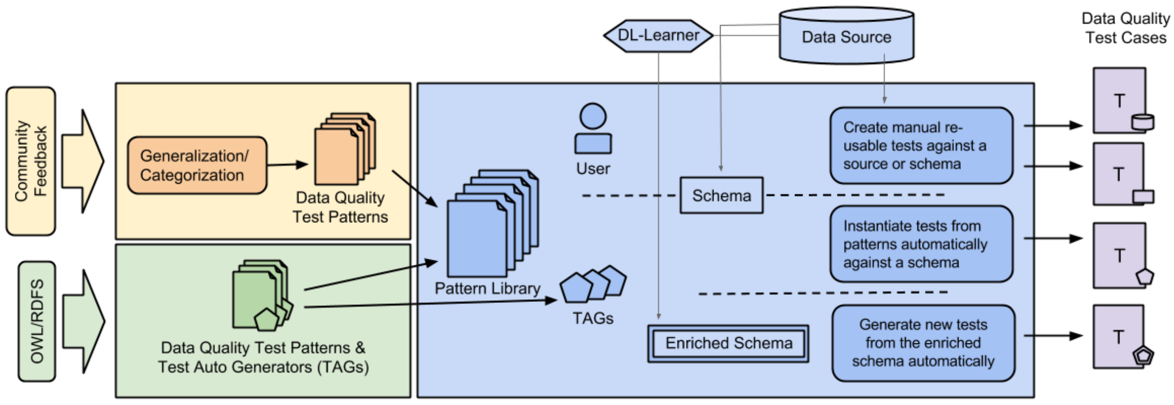

DL-Learner
Data Source
Data Quality Test Cases
Community Feedback
Generalization/ Categorization
Data Quality Test Patterns
User
Schema
Create manual re-usable tests against a source or schema
Instantiate tests from patterns automatically against a schema
Generate new tests from the enriched schema automatically
Pattern Library
TAGs
Enriched Schema
OWL/RDFS
Data Quality Test Patterns & Test Auto Generators (TAGs)
T
T
T
T

Figure 1 RDFUnit architecture

[10] https://youtu.be/T0enhLikN54
[11] http://aligned-project.eu/training/
[12] http://aligned-project.eu/tools/
[13] https://github.com/AKSW/RDFUnit/
[14] https://www.w3.org/TR/shacl/
[15] https://www.w3.org/Submission/shapes/
[16] http://dublincore.org/documents/dc-dsp/

The following subsections describe the three different ways we have developed in ALIGNED for integrating RDFUnit-based data testing and verification into software engineering workflows. We provide three different ways to cover a wider range of use cases. The JUnit runner with annotations (Section 3.1) provides a very easy and well integrated with option but does not give room for flexibility beyond testing an input dataset to a fixed schema. The JUnit XML Report (section 3.2) gives room for greater flexibility by utilizing the complete RDFUnit command line options. Finally, the Custom maven-based integration (Section 3.3) gives the software engineers a way to fine-tune the way they want to automate their data testing & verification options.

3.1 JUnit Runner integration with Java annotations

JUnit allows other testing frameworks to extend JUnit with custom Runners[17] tailored for specific testing. We implemented a custom JUnit Runner, called RdfUnitJunitRunner[18], which can be used to define JUnit tests for validating RDF datasets against a schema, by adding Java annotations to a JUnit test.

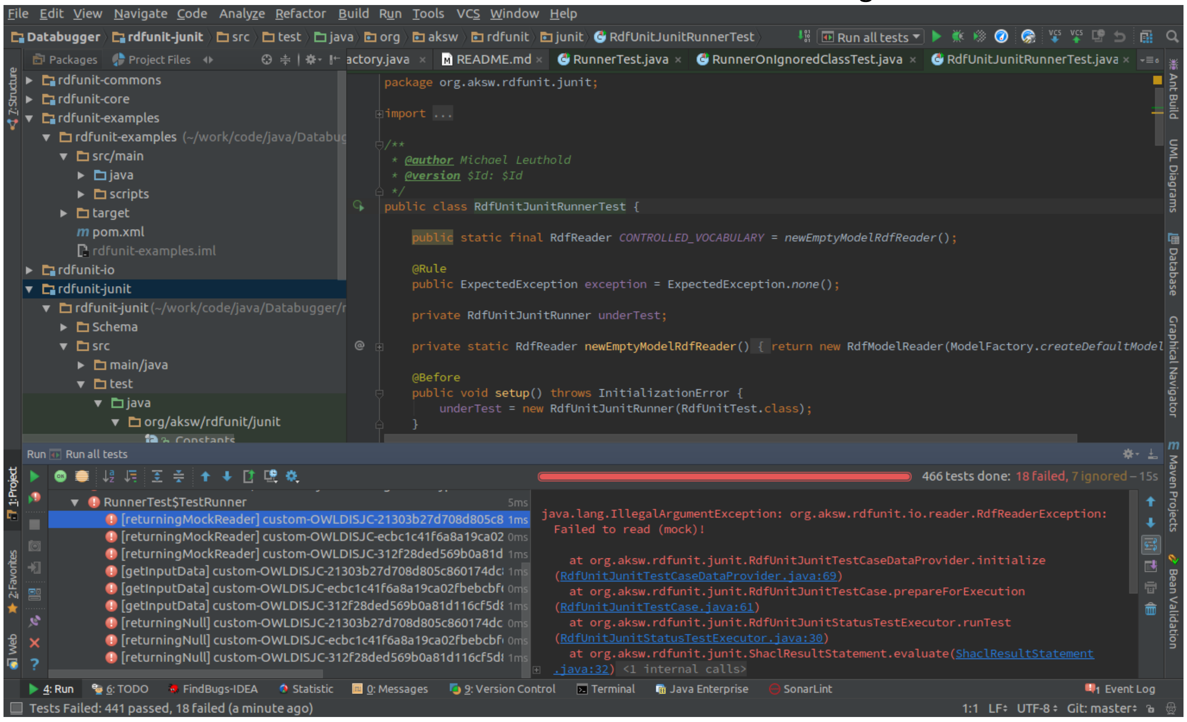
An example RDFUnit / JUnit test is the following:

```
@RunWith(RdfUnitJunitRunner.class)
@Schema(uri = "schema.ttl")
public static class TestRunner {


    @TestInput
    public RDFReader getInputData() {
          return new RdfModelReader(
          RdfReaderFactory.createResourceReader(
               "/inputmodels/data.ttl" ).read()); }

}
```

Where *data.ttl* is a rdf data file (using the **@TestInput** annotation) tested by a JUnit test against *schema.ttl* (using the **@Schema** annotation).

For every automatically generated RDFUnit test a separate JUnit test is generated, that validates the input dataset for a specific violation. The reporting of validation errors is integrated with JUnit reports, thus providing the means to display them through IDEs like IntelliJ or with build tools like Maven.

[17] https://github.com/junit-team/junit4/wiki/test-runners
[18] https://github.com/AKSW/RDFUnit/tree/master/rdfunit-junit



```
package org.aksw.rdfunit.junit;

import ...

/**
 * @author Michael Leuthold
 * @version $Id: $Id
 */
public class RdfUnitJunitRunnerTest {

    public static final RdfReader CONTROLLED_VOCABULARY = newEmptyModelRdfReader();

    @Rule
    public ExpectedException exception = ExpectedException.none();

    private RdfUnitJunitRunner underTest;

    private static RdfReader newEmptyModelRdfReader() { return new RdfModelReader(ModelFactory.createDefaultModel

    @Before
    public void setup() throws InitializationError {
        underTest = new RdfUnitJunitRunner(RdfUnitTest.class);
    }
```

```
466 tests done: 18 failed, 7 ignored – 15s

RunnerTest$TestRunner                                                   5ms
  [returningMockReader] custom-OWLDISJC-21303b27d708d805c8              1ms
  [returningMockReader] custom-OWLDISJC-ecbc1c41f6a8a19ca02             0ms
  [returningMockReader] custom-OWLDISJC-312f28ded569b0a81d              1ms
  [getInputData] custom-OWLDISJC-21303b27d708d805c860174dc              1ms
  [getInputData] custom-OWLDISJC-ecbc1c41f6a8a19ca02fbebcbf             0ms
  [getInputData] custom-OWLDISJC-312f28ded569b0a81d116cf5d              1ms
  [returningNull] custom-OWLDISJC-21303b27d708d805c860174dc             0ms
  [returningNull] custom-OWLDISJC-ecbc1c41f6a8a19ca02fbebcbf            0ms
  [returningNull] custom-OWLDISJC-312f28ded569b0a81d116cf5d             1ms
```

```
java.lang.IllegalArgumentException: org.aksw.rdfunit.io.reader.RdfReaderException:
  Failed to read (mock)!

    at org.aksw.rdfunit.junit.RdfUnitJunitTestCaseDataProvider.initialize
  (RdfUnitJunitTestCaseDataProvider.java:69)
    at org.aksw.rdfunit.junit.RdfUnitJunitTestCase.prepareForExecution
  (RdfUnitJunitTestCase.java:61)
    at org.aksw.rdfunit.junit.RdfUnitJunitStatusTestExecutor.runTest
  (RdfUnitJunitStatusTestExecutor.java:30)
    at org.aksw.rdfunit.junit.ShaclResultStatement.evaluate(ShaclResultStatement
  .java:32) <1 internal calls>
```

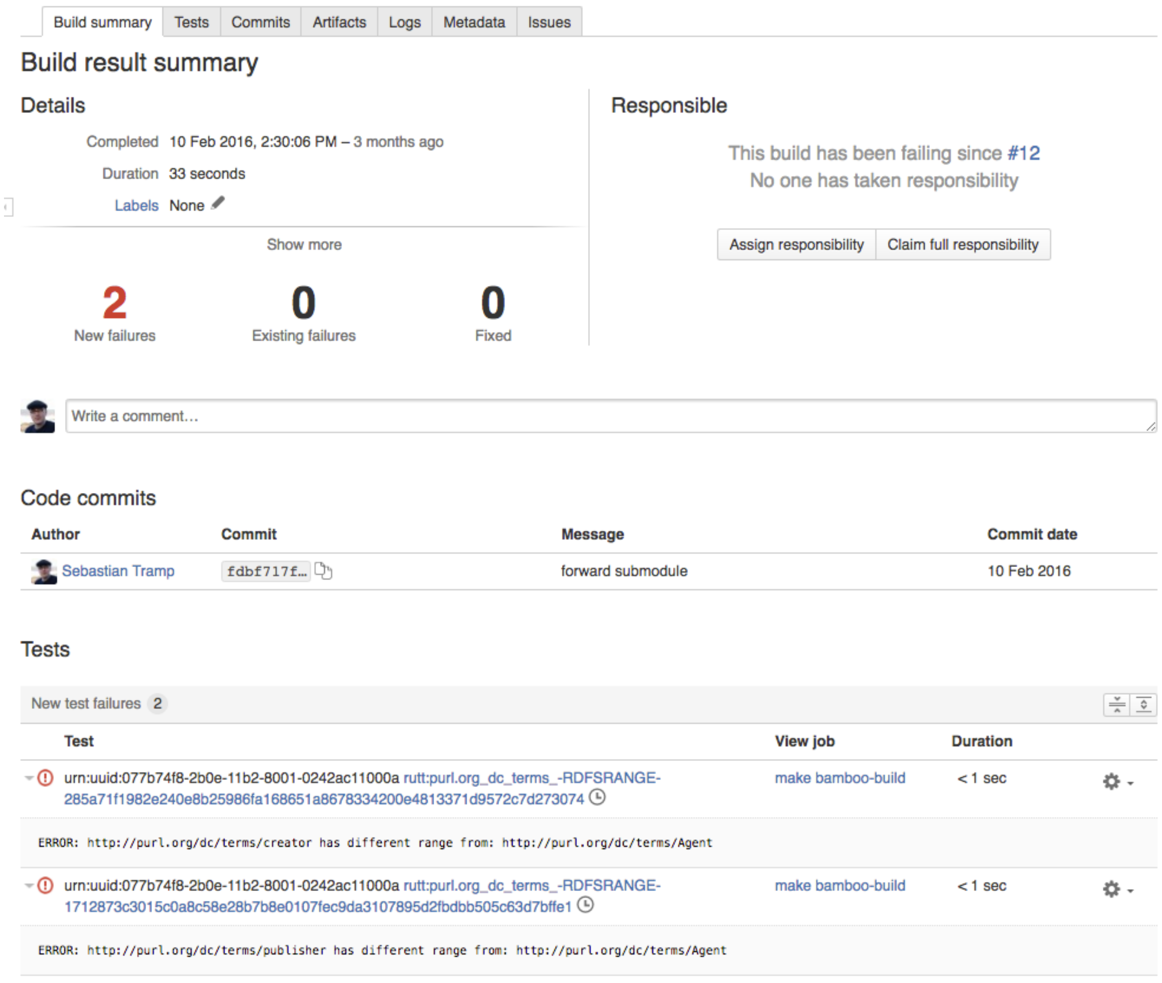
```
Tests Failed: 441 passed, 18 failed (a minute ago)
```

Figure 2 RDFUnit report from the IntelliJ IDE

3.2 JUnit XML report-based integration

JUnit uses a specific XML schema to communicate the test results to IDEs or build tools. For cases when defining a RDFUnit/JUnit test is not an option (i.e. the files are not accessible from the build system with Java code), we provide an alternative by converting the RDFUnit results to the JUnit XML Schema. In these cases, developers can run RDFUnit as a command line tool or through custom code, expecting validation results in the JUnit XML Schema. Build systems, such as Bamboo, can then be configured to look at specific locations for such XML files and report the RDFUnit validation results with the existing unit test error reporting tools.[19]

[19] https://github.com/AKSW/RDFUnit/wiki/Using-RDFunit-with-Bamboo


Build summary | Tests | Commits | Artifacts | Logs | Metadata | Issues

Build result summary

Details

Completed	10 Feb 2016, 2:30:06 PM – 3 months ago
Duration	33 seconds
Labels	None

Show more

2 New failures · 0 Existing failures · 0 Fixed

Responsible

This build has been failing since #12
No one has taken responsibility

Assign responsibility | Claim full responsibility

Write a comment…

Code commits

Author	Commit	Message	Commit date
Sebastian Tramp	fdbf717f…	forward submodule	10 Feb 2016

Tests

New test failures 2

Test	View job	Duration
urn:uuid:077b74f8-2b0e-11b2-8001-0242ac11000a rutt:purl.org_dc_terms_-RDFSRANGE-285a71f1982e240e8b25986fa168651a8678334200e4813371d9572c7d273074	make bamboo-build	< 1 sec

```
ERROR: http://purl.org/dc/terms/creator has different range from: http://purl.org/dc/terms/Agent
```

Test	View job	Duration
urn:uuid:077b74f8-2b0e-11b2-8001-0242ac11000a rutt:purl.org_dc_terms_-RDFSRANGE-1712873c3015c0a8c58e28b7b8e0107fec9da3107895d2fbdbb505c63d7bffe1	make bamboo-build	< 1 sec

```
ERROR: http://purl.org/dc/terms/publisher has different range from: http://purl.org/dc/terms/Agent
```

Figure 3 Example Bamboo overview from an RDFUnit JUnit XML report

3.3 Custom Apache Maven-based integration

When the input data or schema graph are not simple input files, but generated through custom procedures, the aforementioned methods are not easy to apply. For those cases RDFUnit can be used as a Java library, fine-tuned for custom input or more sophisticated Jenkin reports. This was the case with the JURION demo (deliverable D4.1) where RDFUnit was used to validate if the output of specific XSLT scripts adhered to the JURION Schema. All results were archived to a triple store for post-processing analysis (see Image 3). In addition to deliverable D4.1, a research paper was also published [2] that describes this use case in detail.

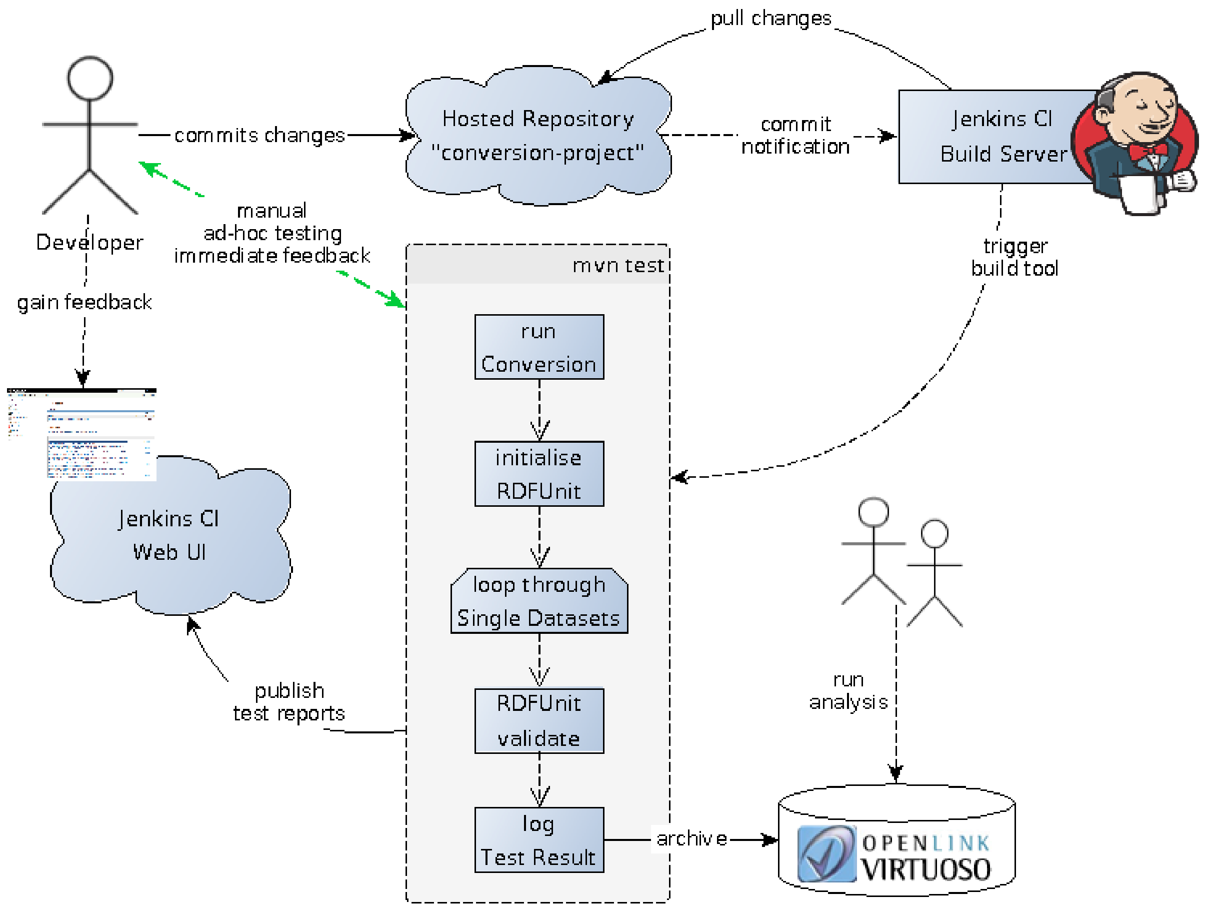


Figure 4 Custom JUnit integration with RDFUnit as a library for JURION Use Case in ALIGNED

4 Next Steps

The automated data testing & verification framework described here will be included in the JURION use case trail platform as part of D5.5 and evaluated in the ALIGNED Phase 2 trials.

A new version of the framework is due in July 2017 as D4.7. We see D4.4 as a good demonstration of software and data engineering aligning demonstration that we will try to adopt in other technical deliverables. A good candidate is deliverable D5.8, *Test case Dependencies & Linking tools & Methods*, where we will explore ways to converge the results of this work and automate test case linking through data verification. Another area we want to focus on is the data test coverage report. This is a required feature from the JURION use case, making the violation reports even more useful for the software engineers.

References

[1] D. Kontokostas, P. Westphal, S. Auer, S. Hellmann, J. Lehmann, R. Cornelissen, and A. Zaveri. Test-driven Evaluation of Linked Data Quality, Proc. 23rd International Conference on World Wide Web, page 747--758., (2014) DOI 10.1145/2566486.2568002

[2] Dimitris Kontokostas, Christian Mader, Michael Leuthold, Christian Dirschl, Katja Eck, Jens Lehmann and Sebastian Hellmann. Semantically Enhanced Quality Assurance in the JURION Business Use Case. ESWC 2016, Crete. Available at: http://link.springer.com/chapter/10.1007/978-3-319-34129-3_40