



Wf4Ever: Advanced Workflow Preservation Technologies for Enhanced Science

STREP FP7-ICT-2007-6 270192

Objective: ICT-2009.4.1b — “Advanced preservation scenarios”

D4.3: Integrity and Authenticity software evaluation

Deliverable Co-ordinator: Graham Klyne

Deliverable Co-ordinating Institution: University of Oxford

Other Authors: Graham Klyne, Esteban García-Cuesta, and José Manuel Gómez-Pérez

This deliverable evaluates the main WP4 software outputs: the checklist and stability services.

Document Identifier:	Wf4Ever/2013/D4.3/0.1	Date due:	November 30, 2013
Class Deliverable:	Wf4Ever FP7-ICT-2007-6 270192	Submission date:	November 29, 2013
Project start date	December 1, 2010	Version:	0.1
Project duration:	3 years	State:	Draft
		Distribution:	Public

Wf4Ever Consortium

This document is part of the Wf4Ever research project funded by the IST Programme of the Commission of the European Communities by the grant number FP7-ICT-2007-6 270192. The following partners are involved in the project:

Intelligent Software Components S.A. (ISOCO) – Coordinator Edificio Testa, Avda. del Partenón 16-18, 1 ^o , 7 ^a Campo de las Naciones, 28042 Madrid Spain Contact person: Jose Manuel Gómez Pérez E-mail address: jmgomez@isoco.com	University of Manchester (UNIMAN) School of Computer Science Oxford Road, Manchester M13 9PL United Kingdom Contact person: Carole Goble E-mail address: carole.goble@manchester.ac.uk
Universidad Politécnica de Madrid (UPM) Departamento de Inteligencia Artificial, Facultad de Informática. 28660 Boadilla del Monte. Madrid Spain Contact person: Oscar Corcho E-mail address: ocorcho@fi.upm.es	Instytut Chemii Bioorganicznej PAN - Poznan Supercomputing and Netowrking Center (PSNC) Network Services Department Ul Z. Noskowskiego 12-14 61704 Poznań Poland Contact person: Raul Palma E-mail address: rpalma@man.poznan.pl
University of Oxford (OXF) Department of Zoology South Parks Road, Oxford OX1 3PS United Kingdom Contact person: Jun Zhao, David De Roure E-mail address: jun.zhao@zoo.ox.ac.uk david.deroure@oerc.ox.ac.uk	Instituto de Astrofísica de Andalucía (IAA) Dpto. Astronomía Extragaláctica. Glorieta de la Astronomía s/n, 18008 Granada Spain Contact person: Lourdes Verdes-Montenegro E-mail address: lourdes@iaa.es
Leiden University Medical Centre (LUMC) Department of Human Genetics Albinusdreef 2, 2333 ZA Leiden The Netherlands Contact person: Marco Roos E-mail address: M.Roos1@uva.nl	

Work package participants

The following partners have taken an active part in the work leading to the elaboration of this document, even if they might not have directly contributed to the writing of this document or its parts:

- iSOCO
- OXF
- PSNC
- UNIMAN
- UPM

Change Log

Version	Date	Amended by	Changes
0.1	2013-10-08	Graham Klyne	Initial outline and draft of checklist service evaluation
0.2	2013-10-09	Graham Klyne	Implemented Markdown conversion to PDF via LaTeX, using LaTeX templates from D2.2v2. Using BibTeX references. Fixed some typos and improved prose.

Executive Summary

This deliverable describes the evaluation of the primary software developments for the Wf4Ever reference architecture supporting the Integrity and Authenticity package goals. The checklist service provides a point-in-time of the conformance of a Research Object with some checklist of requirements. The stability service analyzes the continuing health of a Research Object over time in the face of changes applied to the RO itself, and changes in the external environment within which it is situated.

This deliverable should be read in tandem with D4.2v2 (Design, implementation and deployment of Workflow Integrity and Authenticity Maintenance components – Phase II) in order to provide a complete picture of the state of the Wf4Ever Phase II components.

Contents

1	Introduction	6
2	Evaluation framework	7
3	Checklist service	8
3.1	Component description	8
3.2	Usability	8
3.2.1	User perspective	9
3.2.2	User-developer perspective	11
3.2.3	Developer perspective	12
3.3	Benchmarking	13
3.3.1	Capabilities	13
3.3.2	Performance	15
3.4	Sustainability and maintainability	18
3.4.1	Survey	20
3.5	Summary of evaluation for checklist service	26
4	Stability service	27
4.1	Component description	27
4.2	Usability study	27
4.2.1	User perspective	27
4.2.2	User-developer perspective	29
4.2.3	Developer perspective	30
4.3	Benchmarking	31
4.3.1	Capabilities	31
4.3.2	Performance	31
4.4	Sustainability and maintainability	31
4.4.1	Survey	31
4.5	Summary of evaluation for stability service	38
5	Conclusions	38
	Bibliography	39

1 Introduction

The Integrity and Authenticity (I&A) work package is concerned with supporting longevity of workflows; i.e. that workflows created today continue to be usable and useful in years to come. It emphasises proactive detection and prevention of undesired degradation in performance.

This evaluation assesses the performance of two main tools created to proactively evaluate and monitor the health of Research Objects, and in particular those containing workflows. The focus of our work has been informed by an analysis of the causes of workflow decay [9], which highlighted the role of researchers in maintaining and adapting workflows to survive in a changing environment, and the importance of supporting documentation for human consumption as well as computational tools for analysis and preservation - that is, we need to treat this as an exercise in *conservation* of function rather than merely *preservation* of bits.

This distinction between conservation and preservation is perhaps best illustrated by studies of biological species and ecosystems. Historical study of species was primarily concerned with collecting and preserving specimens in museums. Current activities also focus on conservation: understanding the dynamics of interaction between species and their environment, and making selected interventions that help species to survive as living creatures. And so it is with workflows: their utility stems from their dynamic behaviour as executing processes, interacting with external datasets and services to produce scientifically useful results.

For example, one set of pre-existing bioinformatic workflows we studied, Analysis of KEGG workflows decay¹, made use of genetic pathway information from services provided by Kyoto Encyclopedia of Genes and Genomes (KEGG)². In 2012, KEGG announced that they were upgrading their service to use a new REST interface, and discontinuing the previously offered Web Service offerings. Clearly, preserved workflows that use these services would cease to function as designed, and would need to be updated to use the new services offered if they were to continue to be runnable. This kind of intervention is what we would describe as a “conservation” action, be it applied manually or automatically. Part of our focus in this work has been to detect the need for such intervention, a clearly needed first step to applying it by any means. Separate and ongoing research [3] is concerned with automatic characterization and eventual application of a required intervention, but such work is not covered here.

Until such time as all conservation actions can be automatically determined and applied, we also need to support researchers who are faced with decayed workflows. Our project colleagues have analyzed their workflow-based research activities and come up with a set of requirements³ and best practices in workflow design [4], which are aimed at providing such support. Another part of our focus has been to analyze Workflow-centric Research Objects to report on the extent to which such practices have been followed (e.g., checking that the description of a workflow’s intended purpose and a design sketch have been provided).

The tools we have developed within the Wf4Ever project to support these goals are:

- a checklist evaluation service, used to provide a point-in-time evaluation of the status of a Research Object with respect to decay detection and its conformance to community best practices.
- a stability evaluation service, which builds upon the checklist service to provide a view of how the Research Object, considered as a evolving entity in a dynamic environment, continues to maintain these qualities over time.

These evaluations are designed to guide researchers to providing sufficient supporting information to allow future researchers to perform any conservation interventions that may be needed (is it good enough?), to provide alerting information when interventions are needed (something has broken), and to provide some indication whether conservation is a viable option for a given workflow (is it suitable for reuse?).

¹<http://www.wf4ever-project.org/wiki/display/docs/Showcase+100.+Analysis+of+KEGG+workflows+decay>

²<http://www.genome.jp/kegg/>

³<http://wf4ever.github.io/Requirements/docs/output/UserRequirements-all.html>

As anticipated in the project proposal, provenance is a thread that runs through many of the I&A assessments. Provenance not an end in itself, but is used in the evaluation of features that have been identified as useful for workflow conservation (e.g. does a workflow have example inputs and outputs that can be used to verify its correct operation?). As such, the evaluations that follow don't specifically mention provenance, but evaluation some of the identified requirements does depend on availability of provenance information. Much of our work on provenance has been in the form of contributions to the W3C Provenance Working Group⁴, and as such has been validated through the normal processes of open standards review and adoption: acceptance as a full W3C Recommendation entails community consensus and successful adoption of the provenance model and ontology.

2 Evaluation framework

Our evaluation of the I&A tools is structured around the guidelines provided by the UK's Software Sustainability Institute (SSI): Software evaluation guide [6], Software Evaluation: Tutorial-based Assessment [7] and Software Evaluation: Criteria-based Assessment [5].

The SSI evaluation guides target three kinds of user: end-users (i.e., non-developers who will employ the software), developer-users (i.e., developers who will use APIs and other middleware facilities provided in the creation of some wider application) and developers (i.e., those tasked with enhancing or repairing the software).

The SSI guides further describe two software evaluation approaches. Tutorial-based assessment is based on specific tasks to be accomplished by a user, who reports on the experience running those tasks, and Criteria-based assessment on a number of criteria which may be checked depending on the quality and status of the software. There is a fair degree of overlap between the two evaluation approaches in the topics that they are intended to cover, which we have tried to minimize in the adaptation of the SSI topic coverage reported here.

For user-facing software components, usability should be described. The WP4 components are mostly middleware fronted by other parts of the system (such as myExperiment, RO Portal, etc.), and as such are not involved in the same level of direct user-facing deployment as those other components. The main focus of our usability evaluation is therefore on the developer-user aspects, and the ease and effectiveness of integration of our components with other parts of the Wf4Ever reference implementation.

For developers, usability focuses on the ease of accessing, building, understanding, enhancing and testing the code. There is a strong connection between developer usability and overall sustainability and maintainability of a software product, reflected by some overlap in the topics assessed.

A third dimension to be included in the evaluation is some benchmarking of the components in order to assess indicators of functional capability and performance.

Finally, features of the software base and its management structure are surveyed to get a sense of its sustainability and maintainability, and what further activities might be needed to make the software into a fully fledged, sustainable product. Many of the features covered here relate to the creation and sustenance of a community of users and developers for the product.

This framework is reflected in the structure of the sections that follow, which deal with evaluation of the checklist service and stability service respectively. The overall evaluation reporting structure for each is as follows:

1. Component description, sufficient to place the evaluation within the overall context of the Wf4Ever reference architecture.

⁴http://www.w3.org/2011/prov/wiki/Main_Page

2. Usability study, reporting on applicable usability and effectiveness of the component from:

- end-user (researcher) perspective
- developer-user and system integration perspective
- developer perspective, for enhancing and repairing the software

The reporting is based on considerations covered by the SSI Tutorial-based Assessment [7], adapted to draw upon our ongoing evaluation efforts, reflecting the agile nature of our development. The survey sub-sections draw particularly from the SSI guide.

3. Benchmarking: an assessment of functional and performance characteristics of the software with respect to the tasks it is expected to perform.

4. Sustainability and maintainability study, based on the SSI Criteria-based Assessment [5].

3 Checklist service

3.1 Component description

The checklist service takes an RO, a Minim checklist description and other parameters, and on the basis of these performs an evaluation of the RO or specified resource and returns a result indicating how well the requirements of the checklist were satisfied.

See also:

- Wf4Ever: Design, implementation and deployment of Workflow Integrity and Authenticity Maintenance components - Phase II [2]
- RO decay detection using checklists⁵
- RO checklist evaluation API⁶
- Checklist traffic light API⁷

3.2 Usability

The usability survey references these project resources:

- Checklist service README file⁸
- RO Manager project in Github⁹
- RO Manager software distribution at PyPI¹⁰
- RO Manager FAQ¹¹
- Wf4Ever sandbox configuration information¹²

⁵<http://www.wf4ever-project.org/wiki/display/docs/RO+decay+detection+using+checklists>

⁶<http://www.wf4ever-project.org/wiki/display/docs/RO+checklist+evaluation+API>

⁷<http://www.wf4ever-project.org/wiki/display/docs/Checklist+traffic+light+API>

⁸<https://github.com/wf4ever/ro-manager/blob/master/src/roweb/README.md> (README file for checklist service software)

⁹<https://github.com/wf4ever/ro-manager>

¹⁰<https://pypi.python.org/pypi/ro-manager>

¹¹<http://www.wf4ever-project.org/wiki/display/docs/RO+Manager+FAQ>

¹²<http://www.wf4ever-project.org/wiki/display/docs/Sandbox+configuration>

3.2.1 User perspective

This component is not used directly by an end user. Rather, it is called by other user-facing Wf4Ever components to provide information about how well RO meets some designated criteria. As such, it has not been subjected to a separate usability study.

@@TODO: cover visualization or ensure it is covered in other deliverables. Cross-ref?

Survey

General usability:

- Visibility of system status. Does it give users appropriate feedback within reasonable time?
 - N/A
- Match between system and the real world. Does it speak the user's language and make information appear in a natural and logical order? Are implementation-specific details hidden from the user?
 - N/A
- User control and freedom. Does it provide clearly marked exits, undo and redo?
 - N/A
- Consistency and standards. Is it consistent within the software, with other similar packages and with platform conventions?
 - N/A
- Error prevention. Does it prevent errors in the first place or help users avoid making them?
 - N/A
- Recognition rather than recall. Does it make objects, actions and options visible and reduce the amount of information a user has to remember?
 - N/A
- Flexibility and efficiency of use. Does it offer short-cuts and support macros for frequently- done action sequences?
 - N/A
- Aesthetic and minimalist design. Does it avoid showing irrelevant or rarely-needed information?
 - N/A
- Help users recognize, diagnose, and recover from errors. Does it make errors clear, comprehensible, precise and suggest solutions if possible?
 - N/A
- Help and documentation. Does it provide concise, accurate, clear, easily-searchable task- oriented doc centred around concrete lists of steps?
 - N/A
- Robustness. Are responses sensible when instructions are not followed; e.g., wrong commands, illegal values, etc.?
 - N/A
- Obviousness. Can tasks be accomplished without any consultation of user documents or other material?
 - N/A

Release packaging:

- Are the binary releases packaged for immediate use in a suitable archive format?

- Yes: installation via PyPI ¹³
- See also the README file
- Is it clear how to get the software from the web site? Does it have version numbers?
 - Yes
- Is it clear from the web site or user doc what other packages are required?
 - Yes
- Is it clear what the licencing and copyright is on the web site?
 - Yes
- How to get started. Is there a README, FAQ?
 - Yes

User documentation:

- Are there relevant user documents?
 - Yes. See the README file and linked documents.
- Is the user documentation accurate?
 - It is believed to be
- Does it partition user, user-developer and developer information or mix it all together?
 - Some partitioning, but not rigorous
- Is the user doc online?
 - Yes (github and Wf4Ever wiki)
- Are there any supporting tutorials?
 - No
- Do these list the versions they apply to?
 - N/A
- Is it task-oriented, structured around helping users achieve their tasks?
 - N/A

Help and support:

- Is there a list of known bugs and issues, or a bug/issue tracker?
 - Sort-of: <https://github.com/wf4ever/ro-manager/issues>, <https://github.com/wf4ever/ro-manager/blob/master/TODOTxt>
- Is it clear how to ask for help e.g. where to e-mail or how to enter bugs/issues
 - No.
- Are there e-mail list archives or forums?
 - No
- If so, is there evidence of use?
 - N/A
- Are they searchable?
 - N/A
- Is there a bug/issue tracker?
 - Yes (but not much used).
 - <https://github.com/wf4ever/ro-manager/issues>
 - <https://github.com/wf4ever/ro-manager/blob/master/TODOTxt>
 - <https://jira.man.poznan.pl/jira/issues> (not public)
- If so, there evidence of use?
 - Some, but not consistent

¹³<https://pypi.python.org/pypi/ro-manager>

- Does it seem that bugs and issues are resolved or, at least, looked at?
 - Some
- Is it clear what quality of service a user expect in terms of support e.g. best effort, reasonable effort, reply in 24 hours etc.?
 - No
- Is it clear how to contribute bugs, issues, corrections (e.g. in tutorials or user doc) or ideas?
 - No, but use of Github offers a generic route.

3.2.2 User-developer perspective

The checklist provides a simple REST API¹⁴ which is used by other component developers to perform a required evaluation. There is also a “traffic light API”¹⁵ that is very similar in style. The REST API has proved quite easy to use, and has been successfully integrated with minimal additional guidance from the checklist service developer by other components of the Wf4Ever project:

1. Showcase 47 (<http://www.wf4ever-project.org/wiki/pages/viewpage.action?pagelId=3506198>): in this activity, the checklist service (developed by Oxford) was accessed by an early prototype quality display service (developed separately by iSOCO). This was our first attempt to integrate Integrity and Authenticity components developed by different project members, and showed that the REST style adopted could facilitate integration of software components.
2. myExperiment (<http://alpha2.myexperiment.org/>): the checklist display has been integrated into the display of a myExperiment PACK.
3. RO Portal (<http://sandbox.wf4ever-project.org/portal/>): the checklist display has been integrated into the RO portal display of a Research Object.

To perform a checklist evaluation, a checklist description conforming to the Minim model¹⁶ must be created if one does not already exist. This requires some knowledge of RDF and SPARQL. Originally, checklists were created by hand-editing RDF, which in practice meant they were initially coded by the checklist software developer. Once an initial checklist had been created, other developers were generally able to make modest changes to the checklist (based on experiences from setting up software demonstrations, e.g. <http://www.wf4ever-project.org/wiki/display/docs/135.+TIMBUS+Demo+preparation>).

More recently, based in part on input from a new project member, we have designed a spreadsheet based format for creating checklists, and created a tool mkminim¹⁷ for converting this to RDF for consumption by the checklist evaluation service. At the time of writing, we have not conducted a formal usability study of this tool and the associated spreadsheet format.

As part of another approach to mitigating the possible difficulty of creating checklist descriptions, we have created an initial collection of example and skeleton checklists¹⁸, which may be used as a starting point for creating new checklist definitions.

Survey How easy is it to set up development environment to write code that uses the software or service? This may involve getting the source code of the software but for online services, it might not.

- Is it clear what third-party tools and software you need, which versions you need, where to get these and how to set them up?
 - Yes. See the README file and linked documents.

¹⁴<http://www.wf4ever-project.org/wiki/display/docs/RO+checklist+evaluation+API>

¹⁵<http://www.wf4ever-project.org/wiki/display/docs/Checklist+traffic+light+API>

¹⁶<https://github.com/wf4ever/ro-manager/blob/master/src/roweb>

¹⁷<https://github.com/wf4ever/ro-manager/blob/master/src/checklist/mkminim.md>

¹⁸<https://github.com/wf4ever/ro-catalogue/tree/master/minim>

- Are there tutorials available for user-developers?
 - No
- If so, Do these list the versions they apply to? Are they accurate, and understandable?
 - N/A
- Is there example code that can be compiled, customised and used?
 - Yes: see <https://github.com/wf4ever/ro-manager/tree/develop/src/roweb/samples>; these samples are shell scripts that show how to use the HTTP API via CURL. The logic should be easily transplanted to any languages that provides a suitable HTTP client library.
- How accurate, understandable and complete is the API documentation? Does it provide examples of use?
 - Believed to be complete, but examples are not directly runnable:
 - <http://www.wf4ever-project.org/wiki/display/docs/RO+checklist+evaluation+API>
 - <http://www.wf4ever-project.org/wiki/display/docs/Checklist+traffic+light+API>
- For services, is there information about quality of service? e.g. number of requests that can be run in a specific time period. How do user-developers find out when services might be down etc.
 - No.
- For services, is there information about copyright and licencing as to how the services can be used? e.g. for non-commercial purposes only, does the project have to be credited etc. Is there information on how any data can be used, who owns it etc.?
 - No.
- Is the copyright and licencing of the software and third-party dependencies clear and documented so you can understand the implications on extensions you write?
 - N/A

3.2.3 Developer perspective

The checklist evaluation software has been developed as part of the RO Manager software suite, and makes use of many of the same components. Source code is written in Python, and development has followed a test-led development practice, an effect of which is that there are many examples of the code function for other developers for study.

Development of RO Manager has been led by a single programmer at Oxford, but part of the suite, handling exchange of Research Objects with RODL, has been written by developers at Poznan. This gives us some confidence to claim that the code base is accessible and usable by developers who wish to enhance and/or fix the software.

Much of the developer-targeted documentation is included alongside the source code in Github, as text or Markdown files that can be viewed while browsing the source repository, or in a text editor while modifying the code.

Survey

- How easy is it to set up development environment to change the software?
 - Fairly easy - standard Python environment
 - NOTE: not tested under Windows.
 - See the README file and linked documents.
- How easy is it to access to up-to-date versions of the source code that reflect changes made since the last release? i.e. access to the source code repository.
 - <https://github.com/wf4ever/ro-manager>

- How easy is it to understand the structure of the source code repository? Is there information that relates the structure of the source code to the software's architecture?
 - See the README file and linked documents.
- Is it clear what third-party tools and software you need, which versions you need, where to get these and how to set them up?
 - See the README file and linked documents.
- How easy is it to compile the code?
 - Standard Python environment: no separate compilation needed.
- How easy is it to build a release bundle or deploy a service?
 - Easy (see <https://github.com/wf4ever/ro-manager/blob/master/NOTES.txt>)
- How easy is it to validate changes you've made? This includes building the software, getting, building and running tests.
 - Fairly easy: see the README file.
- Is there design documentation available? How accurate and understandable is it?
 - Some, not comprehensive. See <http://repo.wf4ever-project.org/Content/51/D4.2v2.pdf> and [README file][checklist-service-readme].
 - The LISC2013 workshop paper describes the minim model design and rationale.
- Are there tutorials available for developers?
 - No
- If so, are they accurate, and understandable?
 - N/A
- How readable is the source code? Well-laid out with good use of white-space and indentation?
 - Yes
- How accurate or comprehensive is the source code commenting? Does it focus on why the code is as it is?
 - Yes - comments generally explain rather than repeat code.

3.3 Benchmarking

3.3.1 Capabilities

The checklist evaluation service was developed using an agile approach. It was originally designed to support quality requirements articulated by our bioinformatics and astrophysics scientific research partners^{19 20}, and we focused initially on addressing those without for concern for other requirements that might come later. This allowed us to get up-and-running quickly with something that researchers could comment upon.

Specifically, our initial focus was on “completeness”²¹ - providing a report on whether or not expected or required components were present in the experimental context described by a Research Object. This focus naturally led us to using a checklist-based evaluation model, building from earlier work at Manchester by Matt Gamble [1], and resonating with other work such as JERM²². Continuing our user-led approach, we then shifted our focus to detecting and reporting on causes of workflow decay [9], which required us to look at issues such as “liveness” of workflow resources^{23 24}.

This user-led, agile approach is all very well for addressing our project's internal goals, but did not help us to see how our approach compares with other work in the field. Accordingly, we looked at a number of quality

¹⁹<http://wf4ever.github.io/Requirements/docs/output/UserRequirements-all.html>

²⁰<http://www.wf4ever-project.org/wiki/display/docs/Review+and+publication+with+ROs>

²¹<http://www.wf4ever-project.org/wiki/display/docs/Integrity+and+Authenticity+component>

²²<http://www.sysmo-db.org/jerm>

²³<http://www.wf4ever-project.org/wiki/display/docs/Checklists+discussion+in+Manchester>

²⁴<http://www.wf4ever-project.org/wiki/display/docs/RO+decay+detection+using+checklists>

evaluation activities, with a view to understanding whether and how the capabilities of our checklist-based tool were capable of reproducing that work:

- Detecting observed causes of decay in workflows using KEGG services.
- Evaluation of completeness of ChemBox chemical descriptions extracted from Wikipedia.
- Evaluating the quality of SKOS vocabularies.

This led to some refactoring and refinement of our model ²⁵, but our general conclusion was that the checklist approach, possibly coupled with a few specialized services, was very capable of addressing a wide range of quality assessment requirements.

Parts of our investigation are described in the project wiki ²⁶, and raw notes of the checklist item analyses are captured in Github ²⁷.

Detecting causes of workflow decay This evaluation was undertaken to confirm that the tool was indeed capable of supporting our goal of detecting workflow decay.

In 2012 it was announced ²⁸ [http://www.kegg.jp/kegg/rest/] that the SOAP-based web service for KEGG would be discontinued, to be replaced by a new REST-style service. This presented us with an ideal opportunity to validate an aspect of our workflow decay detection capabilities.

Prior to the KEGG SOAP service shutdown, a number of Taverna workflows using the KEGG service were located in myExperiment ²⁸, and were evaluated to see if they were still usable for the purpose of collecting workflow run provenance ²⁹. Of 92 potential candidates, about 47 were found to be runnable ³⁰, and workflow run provenance data was collected for these ^{31 32}.

Following the shutdown, we created research objects for the identified Taverna workflows using the Wf4Ever Wf-RO service ³³. The overall process and results of this exercise are described in the project wiki ³⁴.

In the final analysis we were able to correctly detect decay for all the workflows that could be converted to fully-described workflow Research Objects, including a workflowdescription using the WFDesc vocabulary ³⁵. All of the problems reported were due to some failure of the workflow conversion process. This shows the checklist service is capable of performing the required decay detection, *provided* that the necessary information for analysis is provided in the Research Object.

Completeness of chemical descriptions This evaluation was based on earlier work by Matt Gamble to validate his MIM model [1]. We wished to test whether our Minim model design could match all the capabilities of an alternative checklist model design based on SPIN ³⁶ and some elements of OWL.

We surveyed and categorized all the information reporting requirements used in the MIM evaluation ³⁷. As many of the requirements were very similar, we selected a representative example of each pattern encountered and implemented it as a Minim checklist item ³⁸.

²⁵ <https://github.com/wf4ever/ro-manager/blob/master/Minim/minim-revised.md>

²⁶ <http://www.wf4ever-project.org/wiki/display/docs/Showcase+128+-+Evaluate+checklist+toolkit>

²⁷ <https://github.com/wf4ever/ro-catalogue/blob/master/v0.1/minim-evaluation/checklist-item-survey.md>

²⁸ <http://www.myexperiment.org>

²⁹ <http://www.wf4ever-project.org/wiki/display/docs/show+88.+provenance+collection+for+Kegg+workflows+in+myexperiment>

³⁰ https://docs.google.com/spreadsheet/ccc?key=0Ahxrga9AQHb_dFBQYnNKb25oMmN3Q1VpNjJkS296WGc#gid=0

³¹ <http://www.wf4ever-project.org/wiki/display/docs/Provenance+corpus>

³² <https://github.com/wf4ever/provenance-corpus>

³³ <http://www.wf4ever-project.org/wiki/display/docs/Wf-RO+transformation+service>

³⁴ <http://www.wf4ever-project.org/wiki/display/docs/Showcase+100.+Analysis+of+KEGG+workflows+decay>

³⁵ <http://wf4ever.github.io/ro/#wfdesc>

³⁶ <http://spinrdf.org>

³⁷ <https://github.com/wf4ever/ro-catalogue/blob/master/v0.1/minim-evaluation/chemmim-summary.md>

³⁸ <https://github.com/wf4ever/ro-catalogue/blob/master/v0.1/minim-evaluation/chembox-minim-samples.ttl>

Using this, we were able to demonstrate tests for each of the patterns identified in the MIM “ChemMIM” description, with one exception.

The exception was “MolecularFormula”, which contained a very complex SPIN query that could not be converted to SPARQL by available tools. This test was using a very complex query pattern to validate the form of a molecular formula obtained from the ChemBox data. The fact that SPIN is substantially based on SPARQL queries gives some confidence that the requirement could be handled if the query could be extracted. But for practical use, it would probably be easier to use the rule or query test extension points of the Minim model to introduce a new element to validate the chemical formula. It is conceivable that the chemical formula validation could be handled using regular expressions (regexes), which are supported in SPARQL filters, but we did not confirm this.

SKOS vocabulary quality To further test our checklist capabilities, we chose a quality evaluation exercise that came from a field very different than our scientific data and method validation work. This was an evaluation of SKOS vocabulary quality by Mader, *et. al.* [8]. This work has identified a number of common quality problems in SKOS vocabularies, drawn from a number of expert sources, and then implemented a tool to detect these problems in SKOS vocabularies published on the web ³⁹.

We analyzed the qSKOS catalogue of quality issues ⁴⁰, drafted Minim checklist items for those we could detect ⁴¹ and identified those which we could describe and using the Minim checklist model as-is, and those which were beyond the current Minim capabilities ⁴².

We did not complete testing the qSKOS checklist items for lack of time.

Our analysis indicated that about half of the qSKOS-documented quality problems could be detected by our Minim-based approach, without further modification. Detecting most of others would need just three generic extensions to the Minim model, coupled with some additional reference resources (e.g. a list of valid ISO country codes available as RDF):

- A way to compare result sets from different queries
- A way to query an external service or resource
- A way to filter query results by URI namespace (we now think this can be done using SPARQL)

There are two qSKOS tests that appear difficult to achieve without some special-purpose extension to the Minim model: “Weakly Connected Components” and “Cyclic Hierarchical Relations”. Both of these involve detecting properties of the RDF graph of the vocabulary being evaluated, and both would probably need some special purpose graph analysis implementation. The Minim model might be extended generically to accommodate these by providing a generic web service interface to invoke a separately implemented analysis and incorporate the result into the checklist evaluation. Alternatively, separate analyses might be implemented that add annotations to the RO (similar to annotations that a manual review has been performed): these could be detected by the Minim-based evaluation as it stands.

3.3.2 Performance

The original design of the checklist evaluation assumed that the scale of RO annotation data involved would be sufficiently low that performance and scalability would not be a concern. For most of our work with checklists, this has been the case, but two issues have arisen in our testing that cause us to question this assumption:

³⁹<https://github.com/cmader/qSKOS>

⁴⁰<https://github.com/cmader/qSKOS/wiki/Quality-Issues>

⁴¹<https://github.com/wf4ever/ro-manager/blob/master/minim-eval/src/iaeval/test/test-qskos/Minim-qskos.ttl>

⁴²<https://github.com/wf4ever/ro-catalogue/blob/master/v0.1/minim-evaluation/qskos-summary.md>

1. When performing the initial ChemBox evaluations⁴³, there were a total of about 7,500 individual chemical descriptions to evaluate. The overhead for creating a web-accessible RO with the available tools was sufficiently great that, rather than creating 7500 separate ROs for each chemical, we opted to create a single RO containing data about all 7500 chemicals. This was used as the basis for all evaluations. Unfortunately, the single merged RO contained about 28Mb of aggregated chemical description data, and evaluation took about 100-200 seconds for each chemical. A cursory investigation suggested that most of the time was spent reading and parsing the RDF annotation data.

We subsequently improved the performance of the ChemBox evaluations by about 2 orders of magnitude by using lightweight “Overlay ROs” created on-the-fly for each chemical⁴⁴.

2. When processing a music information retrieval workflow RO with full Taverna-generated workflow run descriptions⁴⁵, containing several megabytes of detailed provenance information, the checklist evaluation was taking about 20 seconds to complete. Again, the bottleneck appeared to be the time taken to load and parse the RDF data.

Apart from these cases, all indications have been that checklist evaluation performance is more than adequate once the RO annotation data has been loaded and parsed. We therefore undertook some benchmarking to test whether the performance issues noted were indeed due to RDF load and parse times. Files used for this benchmarking exercise are in GitHub⁴⁶.

Checklist evaluation involves the steps shown in figure 1. To perform the benchmarking, we created some synthetic ROs with varying numbers and sizes of annotation files, with aggregate disk usage up to 80Mbytes. We also created a trivial checklist whose actual evaluation time was designed to be minimal on any RO. We used RO Manager and locally stored ROs to avoid having RO server response and network transfer times contribute to the results (steps 1, 2 and 6).

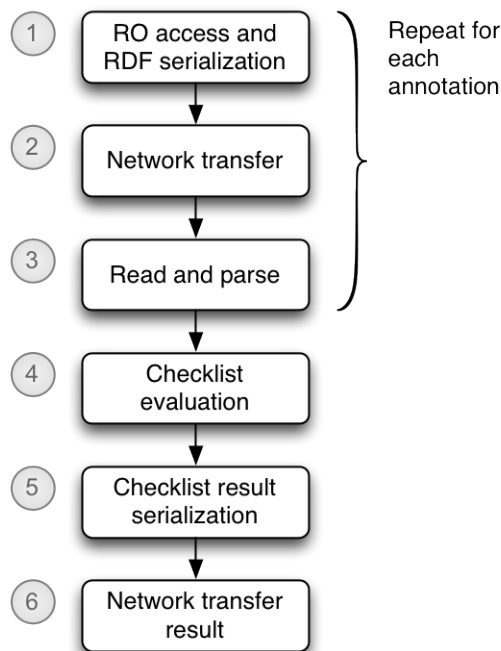


Figure 1: Checklist processing model

⁴³<https://github.com/wf4ever/ro-catalogue/blob/master/v0.1/minim-evaluation/chemmim-summary.md>

⁴⁴https://github.com/wf4ever/ro-catalogue/blob/master/v0.1/minim-evaluation/chembox_evaluate_rov.md

⁴⁵<http://www.wf4ever-project.org/wiki/display/docs/135.+TIMBUS+Demo+preparation>

⁴⁶<https://github.com/wf4ever/ro-manager/tree/develop/src/iaeval/benchmark>

The first tests performed were to confirm that the trivial checklist evaluation was indeed not taking significant time, by modifying the checklist evaluation code to short-circuit the actual evaluation logic, but to perform all other steps. This effectively removed step 4 in figure 1, so the resulting timings would be for steps 3 and 5. Step 5 is assumed to be a small constant time, as the size of the generated checklist evaluation result depends on the checklist used but not on the size of the RO (the same trivial checklist was used for all runs). These timings were compared with an unmodified checklist evaluation code (performing steps 3, 4 and 5):

Sample timings with checklist evaluation included:

```
Starting evaluation of benchmark_1_10: Mon  7 Oct 2013 16:01:41 BST
      completed benchmark_1_10: Mon  7 Oct 2013 16:01:43 BST
      elapsed time for benchmark_1_10: 2s
Starting evaluation of benchmark_10_10: Mon  7 Oct 2013 16:01:43 BST
      completed benchmark_10_10: Mon  7 Oct 2013 16:01:44 BST
      elapsed time for benchmark_10_10: 1s
Starting evaluation of benchmark_100_10: Mon  7 Oct 2013 16:01:44 BST
      completed benchmark_100_10: Mon  7 Oct 2013 16:01:50 BST
      elapsed time for benchmark_100_10: 6s
Starting evaluation of benchmark_1000_10: Mon  7 Oct 2013 16:01:50 BST
      completed benchmark_1000_10: Mon  7 Oct 2013 16:06:46 BST
      elapsed time for benchmark_1000_10: 296s
Starting evaluation of benchmark_1_1000: Mon  7 Oct 2013 16:06:46 BST
      completed benchmark_1_1000: Mon  7 Oct 2013 16:06:48 BST
      elapsed time for benchmark_1_1000: 2s
Starting evaluation of benchmark_10_1000: Mon  7 Oct 2013 16:06:48 BST
      completed benchmark_10_1000: Mon  7 Oct 2013 16:07:10 BST
      elapsed time for benchmark_10_1000: 22s
```

Sample timings with checklist evaluation excluded:

```
Starting evaluation of benchmark_1_10: Mon  7 Oct 2013 15:53:44 BST
      completed benchmark_1_10: Mon  7 Oct 2013 15:53:45 BST
      elapsed time for benchmark_1_10: 1s
Starting evaluation of benchmark_10_10: Mon  7 Oct 2013 15:53:45 BST
      completed benchmark_10_10: Mon  7 Oct 2013 15:53:45 BST
      elapsed time for benchmark_10_10: 0s
Starting evaluation of benchmark_100_10: Mon  7 Oct 2013 15:53:45 BST
      completed benchmark_100_10: Mon  7 Oct 2013 15:53:50 BST
      elapsed time for benchmark_100_10: 5s
Starting evaluation of benchmark_1000_10: Mon  7 Oct 2013 15:53:50 BST
      completed benchmark_1000_10: Mon  7 Oct 2013 15:58:56 BST
      elapsed time for benchmark_1000_10: 306s
Starting evaluation of benchmark_1_1000: Mon  7 Oct 2013 15:58:56 BST
      completed benchmark_1_1000: Mon  7 Oct 2013 15:58:57 BST
      elapsed time for benchmark_1_1000: 1s
Starting evaluation of benchmark_10_1000: Mon  7 Oct 2013 15:58:57 BST
      completed benchmark_10_1000: Mon  7 Oct 2013 15:59:13 BST
      elapsed time for benchmark_10_1000: 16s
```

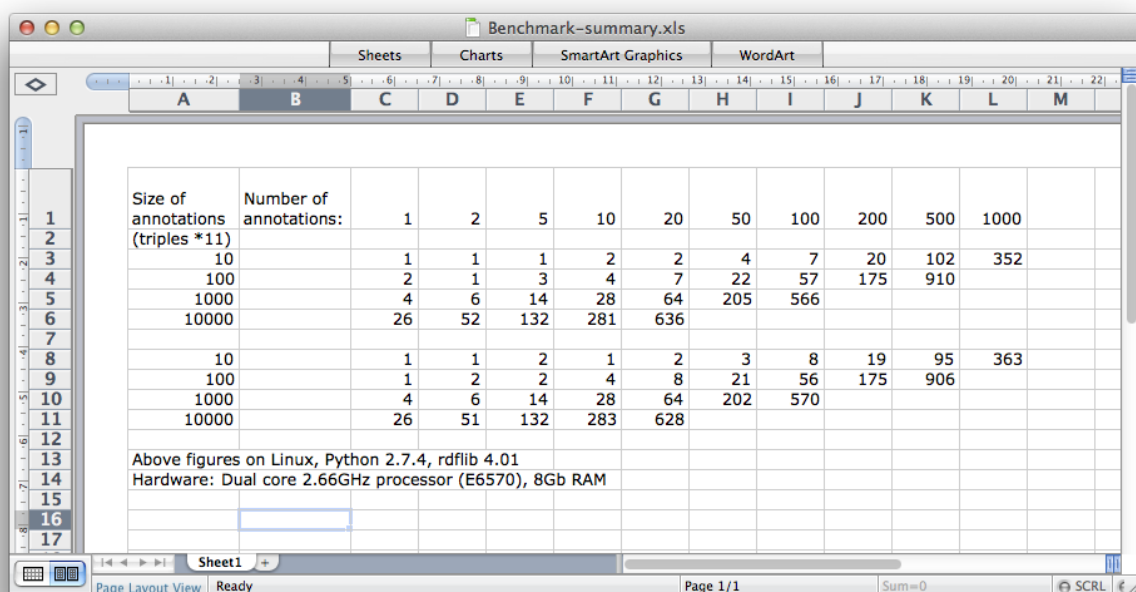
(The above figures were obtained running on a MacBook Pro, with a 2.6GHz Intel Core i7 CPU and 16Gb RAM, using Python 2.7.2 and rdflib 4.01.)

Each benchmark RO used has a name of the form benchmark_NNN_SSS, where NNN is the number of separate annotation files, and SSS is the size of each file, measured as a count of “annotation groups”, where each such

group consists of 11 RDF triples. Thus, benchmark_100_10 has 100 distinct annotations, each containing 10 annotation groups, or about 110 RDF triples. Each run was performed with freshly-generated ROs, so there should be no disk cache warming effects between runs.

Some difference between the timings can be seen, but this was generally within the range of run-to-run variation. The main observation here is that there is no large and consistent variation between run times with and without the checklist evaluation step 4 in figure 1 included. Accordingly, all remaining benchmarking is performed with the standard checklist evaluation utility, including the evaluation step, with a view to understanding how performance varies with size and number of annotations.

An expanded set of tests was performed to get a picture of how annotation count and size affect overall checklist evaluation performance (or, more specifically, the RO annotation load and parse time). The recorded results of two successive runs are in a spreadsheet Benchmark-summary.xls, in Github with the benchmark script and supporting code. For ease of reference, a summary of results is shown in figure 2.



Size of annotations (triples *11)	Number of annotations:	1	2	5	10	20	50	100	200	500	1000
10	1	1	1	1	2	2	4	7	20	102	352
100	2	1	1	3	4	7	22	57	175	910	
1000	4	6	14	28	64	205	566				
10000	26	52	132	281	636						
10	1	1	2	1	2	3	8	19	95	363	
100	1	2	2	4	8	21	56	175	906		
1000	4	6	14	28	64	202	570				
10000	26	51	132	283	628						

Above figures on Linux, Python 2.7.4, rdflib 4.01
Hardware: Dual core 2.66GHz processor (E6570), 8Gb RAM

Figure 2: Summary of benchmark timings

Viewing these as a series of plots of running time vs number of annotations (figure 3), we clearly see that the performance is super-linear with respect to the number of distinct annotations.

Viewing these as a series of plots of running time vs size of annotations (figure 4), it appears that the performance is linear with respect to the size of the annotations (also borne out by examination of the numbers).

3.4 Sustainability and maintainability

The checklist evaluation software has been developed as part of the RO Manager software suite, and makes use of many of the same components. This commonality means that fixes developed for one can benefit the other. The total amount of specific checklist evaluation code⁴⁷ is therefore quite small. The checklist service

⁴⁷<https://github.com/wf4ever/ro-manager/tree/master/src/iaeval>

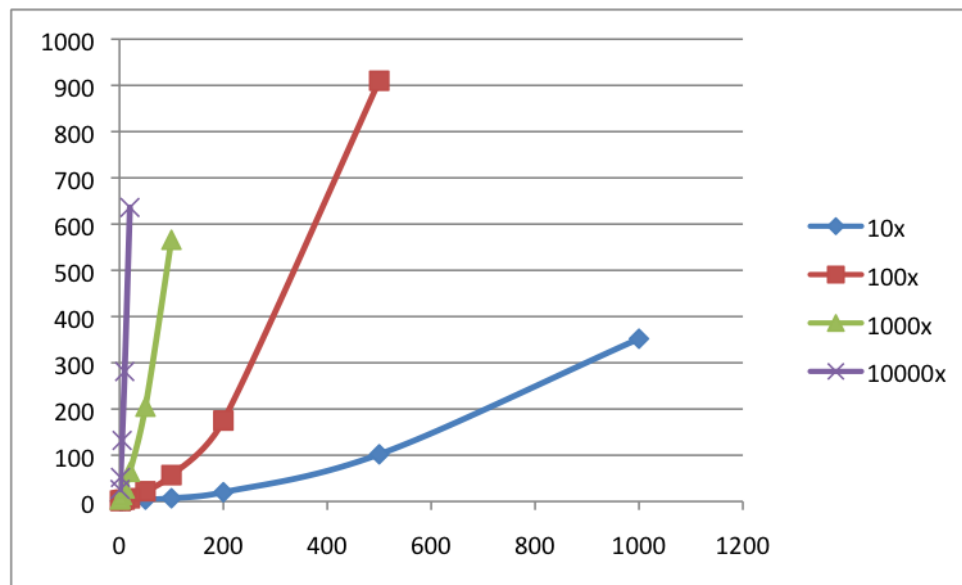


Figure 3: Timings vs number of annotations for different annotation sizes

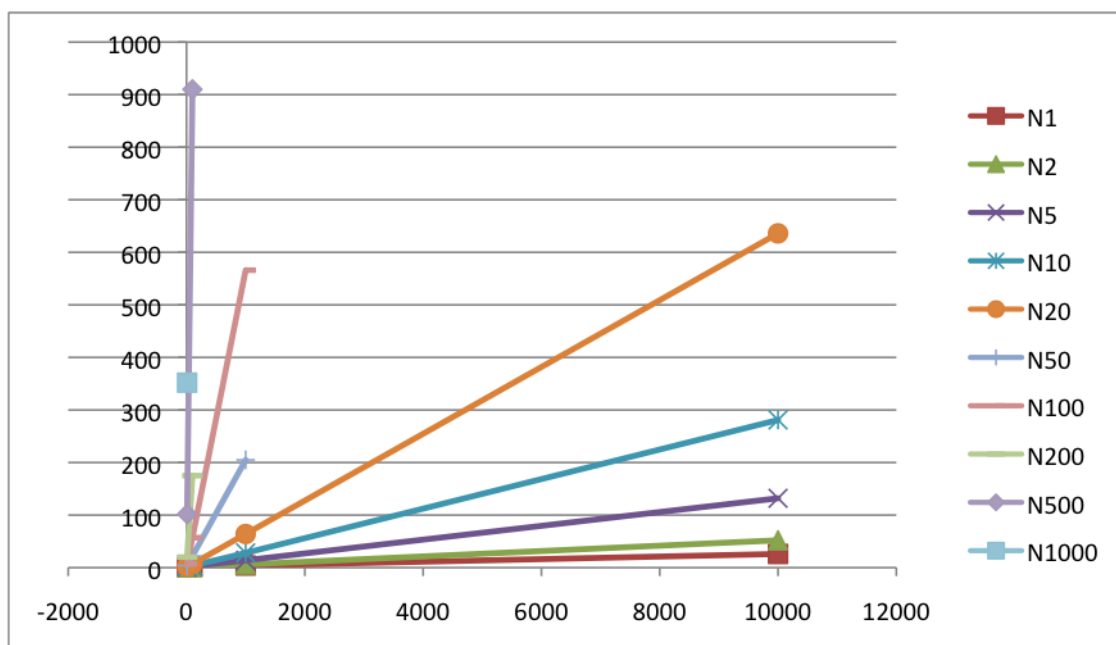


Figure 4: Timings vs size of annotations for different numbers of annotations

is built upon the Pyramid web application framework ⁴⁸, but the amount of dependent code is quite small and porting to the Django framework ⁴⁹ has been considered.

The RO Manager software is packaged and distributed via PyPI ⁵⁰, and the standard Python installation utilities are used for installation (pip, or easy_install). These locate and install most external dependencies. One dependency of the checklist service that is not automatically installed by the RO Manager package is the Pyramid web framework, which itself can be installed with all its dependencies by a single command. Further information about installing and deploying the checklist service can be found in the checklist service software README file ⁵¹.

The checklist evaluation software has been designed and developed by a single programmer, with maintainability and sustainability efforts focused on technical features (unit and integration tests), and to date very little effort has been committed to developer-community building.

3.4.1 Survey

Identity:

To what extent is the identity of the project/software clear and unique both within its application domain and generally?

- Project/software has its own domain name.
 - No, but it is under the umbrella of the Wf4Ever project ⁵² which does
- Project/software has a logo.
 - No, but Wf4Ever project does.
- Project/software has a distinct name within its application area. A search by Google on the name plus keywords from the application area throws up the project web site in the first page of matches.
 - Yes (“RO Manager”: 3rd and 9th hits as of 20131001. Adding “wf4ever” provides only relevant hits on the first page.)
- Project/software has a distinct name regardless of its application area. A search by Google on the name plus keywords from the application area throws up the project web site in the first page of matches.
 - Yes (“RO Manager”: 3rd and 9th hits as of 20131001. Adding “wf4ever” provides only relevant hits on the first page.)
- Project/software name does not throw up embarrassing “did you mean...” hits on Google.
 - OK (Unless “Did you mean: ROM Manager” is considered embarrassing.)
- Project/software name does not violate an existing trade-mark.
 - Not obviously
- Project/software name is trade-marked.
 - No

Copyright:

To what extent is it clear who wrote the software and owns its copyright?

- Web site states copyright.
 - Yes
- Web site states who developed/develops the software, funders etc.
 - Yes: acknowledgement link to Wf4Ever project front page.

⁴⁸<http://docs.pylonsproject.org/projects/pyramid/en/latest/>

⁴⁹<https://www.djangoproject.com>

⁵⁰<https://pypi.python.org/pypi/ro-manager>

⁵¹<https://github.com/wf4ever/ro-manager/blob/master/src/roweb/README.md> (README file for checklist service software)

⁵²<http://www.wf4ever-project.org>

- If there are multiple web sites then these all state exactly the same copyright, licencing and authorship.
 - N/A
- Each source code file has a copyright statement.
 - No (@@TODO: during ISWC travels)
- If supported by the language, each source code file has a copyright statement embedded within a constant.
 - No

Licencing:

Has an appropriate licence been adopted?

- Web site states licence.
 - Yes
- Software (source and binaries) has a licence.
 - Yes
- Software has an open source licence.
 - Yes (MIT)
- Software has an Open Software Initiative (OSI) recognised licence (<http://www.opensource.org/>).
 - Yes (<http://opensource.org/licenses/MIT>)
- Each source code file has a licence header.
 - No

Governance:

To what extent does the project make its management, or how its software development is managed, transparent?

- Project has defined a governance policy.
 - No
- Governance policy is publicly available.
 - N/A

Community:

To what extent does/will an active user community exist for this product?

- Web site has statement of number of users/developers/members.
 - No
- Web site has success stories.
 - No
- Web site has quotes from satisfied users.
 - No
- Web site has list of important partners or collaborators.
 - No (but see Wf4Ever web site)
- Web site has list of the project's publications.
 - No (but see Wf4Ever web site)
- Web site has list of third-party publications that cite the software.
 - No
- Web site has list of software that uses/bundles this software.
 - No
- Users are requested to cite the project if publishing papers based on results derived from the software.
 - No

- Users are required to cite a boilerplate citation if publishing papers based on results derived from the software.
 - No
- Users exist who are not members of the project.
 - No
- Developers exist who are not members of the project.
 - No

Availability:

To what extent is the software available? (The SSI evaluation guide uses “accessible” for this quality, but that has been changed here to avoid confusion with other uses of “accessibility”)

- Binary distributions are available (whether for free, payment, registration).
 - Yes (PyPI distribution)
- Binary distributions are freely available.
 - Yes
- Binary distributions are available without the need for any registration or authorisation of access by the project.
 - Yes
- Source distributions are available (whether for free, payment, registration).
 - Yes (PyPI distribution and Github)
- Source distributions are freely available.
 - Yes
- Source distributions are available without the need for any registration or authorisation of access by the project.
 - Yes
- Access to source code repository is available (whether for free, payment, registration).
 - Yes
- Anonymous read-only access to source code repository.
 - Yes
- Ability to browse source code repository online.
 - Yes
- Repository is hosted externally to a single organisation/institution in a sustainable third- party repository (e.g. SourceForge, GoogleCode, LaunchPad, GitHub) which will live beyond the lifetime of any current funding line.
 - Yes (<https://github.com/wf4ever/ro-manager>)
- Downloads page shows evidence of regular releases (e.g. six monthly, bi-weekly, etc.).
 - Yes, but not to a fixed schedule ⁵³.

Testability:

How straightforward is it to test the software to verify modifications?

- Project has unit tests.
 - Yes
- Project has integration tests.
 - Partial
- For GUIs, project uses automated GUI test frameworks.
 - N/A
- Project has scripts for testing scenarios that have not been automated (e.g. for testing GUIs).

⁵³<https://pypi.python.org/pypi/ro-manager>

- No
- Project recommends tools to check conformance to coding standards.
 - No
- Project has automated tests to check conformance to coding standards.
 - No
- Project recommends tools to check test coverage.
 - No
- Project has automated tests to check test coverage.
 - No
- A minimum test coverage level that must be met has been defined.
 - No
- There is an automated test for this minimum test coverage level.
 - N/A
- Tests are automatically run nightly.
 - No
- Continuous integration is supported – tests are automatically run whenever the source code changes.
 - No
- Test results are visible to all developers/members.
 - N/A
- Test results are visible publicly.
 - N/A
- Test results are e-mailed to a mailing list.
 - N/A
- This e-mailing list can be subscribed to by anyone.
 - N/A
- Project specifies how to set up external resources e.g. FTP servers, databases for tests.
 - No (some tests access RODL)
- Tests create their own files, database tables etc.
 - Many do

Portability:

To what extent can the software be used on other platforms?

- Application can be built on and run under Windows.
 - Yes in theory; not tested
- Application can be built on and run under Windows 7.
 - Yes in theory; not tested
- Application can be built on and run under Windows XP.
 - Yes in theory; not tested
- Application can be built on and run under Windows Vista.
 - Yes in theory; not tested
- Application can be built on and run under UNIX/Linux.
 - Yes
- Application can be built on and run under Solaris.
 - Not tested
- Application can be built on and run under RedHat.
 - Yes in theory; not tested
- Application can be built on and run under Debian.
 - Yes
- Application can be built on and run under Fedora.

- Yes in theory; not tested
- Application can be built on and run under Ubuntu.
 - Yes
- Application can be built on and run under MacOSX.
 - Yes
- Browser applications run under Internet Explorer.
 - N/A
- Browser applications run under Mozilla Firefox.
 - N/A
- Browser applications run under Google Chrome.
 - N/A
- Browser applications run under Opera.
 - N/A
- Browser applications run under Safari.
 - N/A

Supportability:

To what extent will the product be supported currently and in the future?

- Web site has page describing how to get support.
 - No (but common Github tools apply)
- User doc has page describing how to get support.
 - No
- Software describes how to get support (in a README for command-line tools or a Help=>About window in a GUI).
 - No
- Above pages/windows/files describe, or link to, a description of “how to ask for help” e.g. cite version number, send transcript, error logs etc.
 - No
- Project has an e-mail address.
 - No (but see Wf4ever project)
- Project e-mail address has project domain name.
 - N/A
- E-mails are read by more than one person.
 - N/A
- E-mails are archived.
 - N/A
- E-mail archives are publicly readable.
 - N/A
- E-mail archives are searchable.
 - N/A
- Project has a ticketing system.
 - Yes (Github; also JIRA for Wf4Ever internal use)
- Ticketing system is publicly readable.
 - Yes
- Ticketing system is searchable.
 - Yes
- Web site has site map or index.
 - Yes, via RO Manager page ⁵⁴, which links to the checklist service README file.

⁵⁴<https://github.com/wf4ever/ro-manager>

- Web site has search facility.
 - No
- Project resources are hosted externally to a single organisation/institution in a sustainable e-mail archives or ticketing system shows that queries are responded to within a week (not necessarily fixed, but at least looked at and a decision taken as to their priority).
 - Source code, web pages and developer facilities are hosted in GitHub; some supporting material is on Wf4Ever project servers.
 - See also Wf4ever project
- If there is a blog, is it is regularly used.
 - N/A
 - See also Wf4ever project
- E-mail lists or forums, if present, have regular posts.
 - N/A

Analysability:

How straightforward is it to analyse the software's source release to: (a) understand its implementation architecture, and (b) understand individual source code files and how they fit into the implementation architecture?

- Source code is structured into modules or packages.
 - Yes
- Source code structure relates clearly to the architecture or design.
 - Yes, but could be improved
- Project files for IDEs are provided.
 - N/A
- Source code repository is a revision control system.
 - Yes
- Structure of the source code repository and how this maps to the software's components is documented.
 - Yes. See the README file and linked documents.
- Source releases are snapshots of the repository.
 - Yes: PyPI releases are created from commits on the repository master branch.
- Source code is commented.
 - Yes
- Source code comments are written in an API document generation mark-up language e.g. JavaDoc or Doxygen.
 - N/A
- Source code is laid out and indented well.
 - Yes
- Source code uses sensible class, package and variable names.
 - Yes, mostly
- There are no old source code files that should be handled by version control e.g. "SomeComponentOld.java".
 - Usually not. Some are kept transiently, and removed after changes are finalized.
- There is no commented out code.
 - Some, but generally only kept transiently.
- There are no TODOs in the code.
 - Lots of TODOs
- Coding standards are required to be observed.
 - No enforcement
- Project-specific coding standards are consistent with community or generic coding standards (e.g. for

C, Java, FORTRAN etc.).

- Yes

Changeability:

How straightforward is it to modify the software to: (a) address issues, (b) modify functionality, and (c) add new functionality?

- Project has defined a contributions policy.
 - No
- Contributions policy is publicly available.
 - No
- Contributors retain copyright/IP of their contributions.
 - Yes (by default; there is no assignment process).
- Users, user-developers and developers who are not project members can contribute.
 - Yes, through gatekeeper
- Project has defined a stability/deprecation policy for components, APIs etc.
 - No
- Stability/deprecation policy is publicly available.
 - N/A
- Releases document deprecated components/APIs in that release.
 - N/A (no components or APIs have been deprecated)
- Releases document removed/changed components/APIs in that release.
 - N/A (no components or APIs have been removed/changed)
- Changes in the source code repository are e-mailed to a mailing list.
 - No
- This e-mailing list can be subscribed to by anyone.
 - N/A

Evolvability:

To what extent will the product be developed in the future: (a) for a future release, and (b) within a roadmap for the product? * Web site describes project roadmap or plans or milestones (either on a web page or within a ticketing system). * No * Web site describes how project is funded/sustained. * No * See also Wf4ever project * We site describes end dates of current funding lines. * No * See also Wf4ever project

Interoperability:

To what extent does the software's interoperability: (a) meet appropriate open standards, and (b) function with required and optional third-party components?

- Uses open standards.
 - Yes (HTTP, RDF, etc.)
- Uses mature, ratified, non-draft open standards.
 - Yes
- Provides tests demonstrating compliance to open standards.
 - No

3.5 Summary of evaluation for checklist service

The checklist evaluation service is targeted at developers who wish to use the checklist service to provide quality indications of Research Objects in other components of the Wf4Ever workflow preservation software,

and more generally. Our experiences indicate that the API is easy to use, and the code base is capable of being understood and maintained by third party developers.

Checklist creation needs further work to make it accessible to end users; we have made some initial steps towards this goal⁵⁵ ⁵⁶. The adaptation of checklists by other members of the project team indicates that existing checklist descriptions are somewhat understandable and editable by developers with some knowledge of RDF, SPARQL and the RO Model. The main issue here is for the checklist designer to have a clear knowledge of the form of RO annotations used by the ROs to be evaluated (or by the community to whom the checklist is addressed).

The current checklist service is capable of a fair range of requirements arising from our own project, and also of those reported in other quality evaluation work. There are some requirements from other work that our implementation does not currently support, but many of these can be addressed by exploiting the extension points in our design to add new capabilities.

Checklist evaluation performance has been adequate for much of our use to date, but we have identified potential problems with ROs containing large annotations, and especially containing large numbers of annotations. Our benchmarking indicates that the problems are in the area of RDF load and parse times, which are somewhat determined by the RDF handling library used. The most problematic aspect in our work has been the size of provenance traces generated from Taverna workflow runs, which contain a lot of information that is not used (so far) in any checklist. We have given some consideration to alternative strategies for dealing with large RO annotations, but progress on this will be a topic for future work.

The project has many of the technical requirements for sustainability and maintainability in place (version control, unit tests, commented source code, etc.). The main missing technical feature is to use a continuous integration platform. But it is clear from the SSI checklist that there are many aspects of developer community building and communication that are not yet in place. These are points that should be addressed if a future project attempts to move this service from a research prototype to a fully fledged software product.

4 Stability service

@@TODO: Esteban this section?

4.1 Component description

@@brief description

See also: * (D4.2) * (other links)

4.2 Usability study

@@links to supporting documentation??

4.2.1 User perspective

@@description of end-user facing usability

@@cover visualization or ensure it is covered in other deliverables. Cross-ref?

⁵⁵<https://github.com/wf4ever/ro-manager/blob/master/src/checklist/README.md>

⁵⁶<https://github.com/wf4ever/ro-manager/blob/master/src/checklist/mkminim.md>

Survey @@TODO: as appropriate, replace “N/A” or TODO by appropriate survey responses

General usability:

- Visibility of system status. Does it give users appropriate feedback within reasonable time?
 - N/A
- Match between system and the real world. Does it speak the user’s language and make information appear in a natural and logical order? Are implementation-specific details hidden from the user?
 - N/A
- User control and freedom. Does it provide clearly marked exits, undo and redo?
 - N/A
- Consistency and standards. Is it consistent within the software, with other similar packages and with platform conventions?
 - N/A
- Error prevention. Does it prevent errors in the first place or help users avoid making them?
 - N/A
- Recognition rather than recall. Does it make objects, actions and options visible and reduce the amount of information a user has to remember?
 - N/A
- Flexibility and efficiency of use. Does it offer short-cuts and support macros for frequently- done action sequences?
 - N/A
- Aesthetic and minimalist design. Does it avoid showing irrelevant or rarely-needed information?
 - N/A
- Help users recognize, diagnose, and recover from errors. Does it make errors clear, comprehensible, precise and suggest solutions if possible?
 - N/A
- Help and documentation. Does it provide concise, accurate, clear, easily-searchable task- oriented doc centred around concrete lists of steps?
 - N/A
- Robustness. Are responses sensible when instructions are not followed; e.g., wrong commands, illegal values, etc.?
 - N/A
- Obviousness. Can tasks be accomplished without any consultation of user documents or other material?
 - N/A

Release packaging:

- Are the binary releases packaged for immediate use in a suitable archive format?
 - @@TODO
- Is it clear how to get the software from the web site? Does it have version numbers?
 - @@TODO
- Is it clear from the web site or user doc what other packages are required?
 - @@TODO
- Is it clear what the licencing and copyright is on the web site?
 - @@TODO
- How to get started. Is there a README, FAQ?
 - @@TODO

User documentation:

- Are thererelevant user documents?

- @@TODO (links)
- Is the user documentation accurate?
 - @@TODO
- Does it partition user, user-developer and developer information or mix it all together?
 - @@TODO
- Is the user doc online?
 - @@TODO
- Are there any supporting tutorials?
 - @@TODO
- If so, do these list the versions they apply to?
 - N/A
- If so, are they task-oriented, structured around helping users achieve their tasks?
 - N/A

Help and support:

- Is there a list of known bugs and issues, or a bug/issue tracker?
 - @@TODO
- Is it clear how to ask for help e.g. where to e-mail or how to enter bugs/issues
 - @@TODO
- Are there e-mail list archives or forums?
 - @@TODO
- If so, is there evidence of use?
 - N/A
- Are they searchable?
 - N/A
- Is there a bug/issue tracker?
 - @@TODO
- If so, there evidence of use?
 - N/A
- If so, does it seem that bugs and issues are resolved or, at least, looked at?
 - N/A
- Is it clear what quality of service a user expect in terms of support e.g. best effort, reasonable effort, reply in 24 hours etc.?
 - @@TODO
- Is it clear how to contribute bugs, issues, corrections (e.g. in tutorials or user doc) or ideas?
 - @@TODO

4.2.2 User-developer perspective

@@TODO: describe user-developer capabilities and interface

Survey

- How easy is it to set up development environment to write code that uses the software or service?
 - @@TODO
- Is it clear what third-party tools and software you need, which versions you need, where to get these and how to set them up?
 - @@TODO
- Are there tutorials available for user-developers?

- @@TODO
- If so, Do these list the versions they apply to? Are they accurate, and understandable?
 - N/A
- Is there example code that can be compiled, customised and used?
 - @@TODO
- How accurate, understandable and complete is the API documentation? Does it provide examples of use?
 - @@TODO
- For services, is there information about quality of service? e.g. number of requests that can be run in a specific time period. How do user-developers find out when services might be down etc.
 - @@TODO
- For services, is there information about copyright and licencing as to how the services can be used? e.g. for non-commercial purposes only, does the project have to be credited etc. Is there information on how any data can be used, who owns it etc.?
 - @@TODO
- Is the copyright and licencing of the software and third-party dependencies clear and documented so you can understand the implications on extensions you write?
 - @@TODO

4.2.3 Developer perspective

@@Briefly describe the nature of the codebase, language used and development practices

@@Link to component web page for developers

Survey

- How easy is it to set up development environment to change the software?
 - @@TODO
- How easy is it to access to up-to-date versions of the source code that reflect changes made since the last release? i.e. access to the source code repository.
 - @@TODO
- How easy is it to understand the structure of the source code repository? Is there information that relates the structure of the source code to the software's architecture?
 - @@TODO
- Is it clear what third-party tools and software you need, which versions you need, where to get these and how to set them up?
 - @@TODO
- How easy is it to compile the code?
 - @@TODO
- How easy is it to build a release bundle or deploy a service?
 - @@TODO
- How easy is it to validate changes you've made? This includes building the software, getting, building and running tests.
 - @@TODO
- Is there design documentation available? How accurate and understandable is it?
 - @@TODO
- Are there tutorials available for developers?
 - @@TODO

- If so, are they accurate, and understandable?
 - @TODO
- How readable is the source code? Well-laid out with good use of white-space and indentation?
 - @TODO
- How accurate or comprehensive is the source code commenting? Does it focus on why the code is as it is?
 - @TODO

4.3 Benchmarking

4.3.1 Capabilities

@TODO - describe evaluation and result

4.3.2 Performance

@TODO - describe evaluation and result

4.4 Sustainability and maintainability

@TODO: brief background information from developer perspective, focusing on access, support and communication

4.4.1 Survey

Identity:

To what extent is the identity of the project/software clear and unique both within its application domain and generally?

- Project/software has its own domain name.
 - @TODO
- Project/software has a logo.
 - @TODO
- Project/software has a distinct name within its application area. A search by Google on the name plus keywords from the application area throws up the project web site in the first page of matches.
 - @TODO
- Project/software has a distinct name regardless of its application area. A search by Google on the name plus keywords from the application area throws up the project web site in the first page of matches.
 - @TODO
- Project/software name does not throw up embarrassing “did you mean...” hits on Google.
 - @TODO
- Project/software name does not violate an existing trade-mark.
 - @TODO
- Project/software name is trade-marked.
 - @TODO

Copyright:

To what extent is it clear who wrote the software and owns its copyright?

- Web site states copyright.
 - @@TODO
- Web site states who developed/develops the software, funders etc.
 - @@TODO
- If there are multiple web sites then these all state exactly the same copyright, licencing and authorship.
 - @@TODO
- Each source code file has a copyright statement.
 - @@TODO
- If supported by the language, each source code file has a copyright statement embedded within a constant.
 - @@TODO

Licencing:

Has an appropriate licence been adopted?

- Web site states licence.
 - @@TODO
- Software (source and binaries) has a licence.
 - @@TODO
- Software has an open source licence.
 - @@TODO
- Software has an Open Software Initiative (OSI) recognised licence (<http://www.opensource.org/>).
 - @@TODO
- Each source code file has a licence header.
 - @@TODO

Governance:

To what extent does the project make its management, or how its software development is managed, transparent?

- Project has defined a governance policy.
 - @@TODO
- Governance policy is publicly available.
 - @@TODO

Community:

To what extent does/will an active user community exist for this product?

- Web site has statement of number of users/developers/members.
 - @@TODO
- Web site has success stories.
 - @@TODO
- Web site has quotes from satisfied users.
 - @@TODO
- Web site has list of important partners or collaborators.
 - @@TODO
- Web site has list of the project's publications.
 - @@TODO

- Web site has list of third-party publications that cite the software.
 - @@TODO
- Web site has list of software that uses/bundles this software.
 - @@TODO
- Users are requested to cite the project if publishing papers based on results derived from the software.
 - @@TODO
- Users are required to cite a boilerplate citation if publishing papers based on results derived from the software.
 - @@TODO
- Users exist who are not members of the project.
 - @@TODO
- Developers exist who are not members of the project.
 - @@TODO

Availability:

To what extent is the software available? (The SSI evaluation uses “accessible” for this quality, but that has been changed here to avoid confusion with other uses of “accessibility”)

- Binary distributions are available (whether for free, payment, registration).
 - @@TODO
- Binary distributions are freely available.
 - @@TODO
- Binary distributions are available without the need for any registration or authorisation of access by the project.
 - @@TODO
- Source distributions are available (whether for free, payment, registration).
 - @@TODO
- Source distributions are freely available.
 - @@TODO
- Source distributions are available without the need for any registration or authorisation of access by the project.
 - @@TODO
- Access to source code repository is available (whether for free, payment, registration).
 - @@TODO
- Anonymous read-only access to source code repository.
 - @@TODO
- Ability to browse source code repository online.
 - @@TODO
- Repository is hosted externally to a single organisation/institution in a sustainable third- party repository (e.g. SourceForge, GoogleCode, LaunchPad, GitHub) which will live beyond the lifetime of any current funding line.
 - @@TODO
- Downloads page shows evidence of regular releases (e.g. six monthly, bi-weekly, etc.).
 - @@TODO

Testability:

How straightforward is it to test the software to verify modifications?

- Project has unit tests.
 - @@TODO
- Project has integration tests.

- @@TODO
- For GUIs, project uses automated GUI test frameworks.
 - @@TODO
- Project has scripts for testing scenarios that have not been automated (e.g. for testing GUIs).
 - @@TODO
- Project recommends tools to check conformance to coding standards.
 - @@TODO
- Project has automated tests to check conformance to coding standards.
 - @@TODO
- Project recommends tools to check test coverage.
 - @@TODO
- Project has automated tests to check test coverage.
 - @@TODO
- A minimum test coverage level that must be met has been defined.
 - @@TODO
- There is an automated test for this minimum test coverage level.
 - @@TODO
- Tests are automatically run nightly.
 - @@TODO
- Continuous integration is supported – tests are automatically run whenever the source code changes.
 - @@TODO
- Test results are visible to all developers/members.
 - @@TODO
- Test results are visible publicly.
 - @@TODO
- Test results are e-mailed to a mailing list.
 - @@TODO
- This e-mailing list can be subscribed to by anyone.
 - @@TODO
- Project specifies how to set up external resources e.g. FTP servers, databases for tests.
 - @@TODO
- Tests create their own files, database tables etc.
 - @@TODO

Portability:

To what extent can the software be used on other platforms?

- Application can be built on and run under Windows.
 - @@TODO
- Application can be built on and run under Windows 7.
 - @@TODO
- Application can be built on and run under Windows XP.
 - @@TODO
- Application can be built on and run under Windows Vista.
 - @@TODO
- Application can be built on and run under UNIX/Linux.
 - @@TODO
- Application can be built on and run under Solaris.
 - @@TODO
- Application can be built on and run under RedHat.

- @@TODO
- Application can be built on and run under Debian.
 - @@TODO
- Application can be built on and run under Fedora.
 - @@TODO
- Application can be built on and run under Ubuntu.
 - @@TODO
- Application can be built on and run under MacOSX.
 - @@TODO
- Browser applications run under Internet Explorer.
 - @@TODO
- Browser applications run under Mozilla Firefox.
 - @@TODO
- Browser applications run under Google Chrome.
 - @@TODO
- Browser applications run under Opera.
 - @@TODO
- Browser applications run under Safari.
 - @@TODO

Supportability:

To what extent will the product be supported currently and in the future?

- Web site has page describing how to get support.
 - @@TODO
- User doc has page describing how to get support.
 - @@TODO
- Software describes how to get support (in a README for command-line tools or a Help=>About window in a GUI).
 - @@TODO
- Above pages/windows/files describe, or link to, a description of “how to ask for help” e.g. cite version number, send transcript, error logs etc.
 - @@TODO
- Project has an e-mail address.
 - @@TODO
- Project e-mail address has project domain name.
 - @@TODO
- E-mails are read by more than one person.
 - @@TODO
- E-mails are archived.
 - @@TODO
- E-mail archives are publicly readable.
 - @@TODO
- E-mail archives are searchable.
 - @@TODO
- Project has a ticketing system.
 - @@TODO
- Ticketing system is publicly readable.
 - @@TODO
- Ticketing system is searchable.

- @@TODO
- Web site has site map or index.
 - @@TODO
- Web site has search facility.
 - @@TODO
- Project resources are hosted externally to a single organisation/institution in a sustainable e-mail archives or ticketing system shows that queries are responded to within a week (not necessarily fixed, but at least looked at and a decision taken as to their priority).
 - @@TODO
- If there is a blog, is it is regularly used.
 - @@TODO
- E-mail lists or forums, if present, have regular posts.
 - @@TODO

Analysability:

How straightforward is it to analyse the software's source release to: (a) To understand its implementation architecture? (b) To understand individual source code files and how they fit into the implementation architecture?

- Source code is structured into modules or packages.
 - @@TODO
- Source code structure relates clearly to the architecture or design.
 - @@TODO
- Project files for IDEs are provided.
 - @@TODO
- Source code repository is a revision control system.
 - @@TODO
- Structure of the source code repository and how this maps to the software's components is documented.
 - @@TODO
- Source releases are snapshots of the repository.
 - @@TODO
- Source code is commented.
 - @@TODO
- Source code comments are written in an API document generation mark-up language e.g. JavaDoc or Doxygen.
 - @@TODO
- Source code is laid out and indented well.
 - @@TODO
- Source code uses sensible class, package and variable names.
 - @@TODO
- There are no old source code files that should be handled by version control e.g. "SomeComponentOld.java".
 - @@TODO
- There is no commented out code.
 - @@TODO
- There are no TODOs in the code.
 - @@TODO
- Coding standards are required to be observed.
 - @@TODO

- Project-specific coding standards are consistent with community or generic coding standards (e.g. for C, Java, FORTRAN etc.).
 - @@TODO

Changeability:

How straightforward is it to modify the software to: (a) Address issues? (b) Modify functionality? (c) Add new functionality?

- Project has defined a contributions policy.
 - @@TODO
- Contributions policy is publicly available.
 - @@TODO
- Contributors retain copyright/IP of their contributions.
 - @@TODO
- Users, user-developers and developers who are not project members can contribute.
 - @@TODO
- Project has defined a stability/deprecation policy for components, APIs etc.
 - @@TODO
- Stability/deprecation policy is publicly available.
 - @@TODO
- Releases document deprecated components/APIs in that release.
 - @@TODO
- Releases document removed/changed components/APIs in that release.
 - @@TODO
- Changes in the source code repository are e-mailed to a mailing list.
 - @@TODO
- This e-mailing list can be subscribed to by anyone.
 - @@TODO

Evolvability:

To what extent will the product be developed in the future: (a) For a future release? (b) Within a roadmap for the product? ï£ijï£ij

- Web site describes project roadmap or plans or milestones (either on a web page or within a ticketing system).
 - @@TODO
- Web site describes how project is funded/sustained.
 - @@TODO
- We site describes end dates of current funding lines.
 - @@TODO

Interoperability:

To what extent does the software's interoperability: (a) Meet appropriate open standards? (b) Function with required third-party components? (c) Function with optional third-party components?

- Uses open standards.
 - @@TODO
- Uses mature, ratified, non-draft open standards.
 - @@TODO
- Provides tests demonstrating compliance to open standards.
 - @@TODO

4.5 Summary of evaluation for stability service

@@TODO: summary of reports and survey indications for component

5 Conclusions

@@TODO: Jose to draft something?

References

- [1] Matthew Gamble and Carole Goble. Quality, trust, and utility of scientific data on the web: Towards a joint model. In *Proceedings of the ACM WebSci'11*, pages 1–8, 2011.
- [2] E. Garcia-Cuesta et al. Design, implementation and deployment of Workflow Integrity and Authenticity Maintenance components - Phase II. Deliverable D4.2v2, Wf4Ever Project, 2013.
- [3] D. Garijo, P. Alper, K. Belhajjame, O. Corcho, Y. Gil, and C. Goble. Common motifs in scientific workflows: An empirical analysis. In *8th IEEE International Conference on eScience 2012*, 8th IEEE International Conference on eScience 2012, Chicago, 2012. IEEE Computer Society Press, USA.
- [4] Belhajjame K Goble CA Mina E Dharuri H Verdes-Montenegro L Garrido J de Roure D Roos M. Hettne K, Wolstencroft K. Best practices for workflow design: how to prevent workflow decay. In *SWAT4LS*, 2012.
- [5] Software Sustainability Institute. Software evaluation: Criteria-based assessment. November 2011.
- [6] Software Sustainability Institute. Software evaluation guide. November 2011.
- [7] Software Sustainability Institute. Software evaluation: Tutorial-based assessment. November 2011.
- [8] Christian Mader, Bernhard Haslhofer, and Antoine Isaac. Finding quality issues in skos vocabularies. *CoRR*, abs/1206.1339, 2012.
- [9] Jun Zhao, Jose Manuel Gomez-Perez, Khalid Belhajjame, and et al. Why workflows break-understanding and combating decay in taverna workflows. In *IEEE eScience*, pages 1–8, 2012.