Providing tool support for unit testing in eXtreme Design

Fiorela Ciroku, Valentina Presutti

Alma Mater Studiorum - Università di Bologna, Bologna, Italy

Abstract

The evaluation of an ontology is a crucial and occasionally overlooked step of the ontology development process. The evaluation can happen in different stages of the development, on different layers, such as lexical, taxonomic, semantic relations, context/application, syntactic, and structural layers, and by different people's roles, depending on the ontology development methodology. A widespread issue regarding the evaluation of ontologies is the lack of tools to support these methodologies. In the eXtreme Design methodology, the ontology evaluation is a central part of the process, which focuses on assessing whether the requirements, either data-driven or story-driven, have been fulfilled by the ontology module. Momentarily, there is only a jar that provides support for the execution of test cases, but there is no support for other aspects of the evaluation process. This paper presents the progress of the development of tool support which provides semi-automated management for unit testing of owl ontologies on GitHub by means of developed-from-scratch actions and based on the Continuous Integration practice. The fundamental features that are currently developed in the tool are: 1) Setup of the testing environment, 2) Crosscheck and parsing of the ontology tester input, 3) Construction and automatic execution of the unit test, and 4) Documentation of the unit test. The evaluation of the tool itself has been prepared and we expect to have a list of new functional and non-functional requirements as well as identification of bugs or refinement of the existing features.

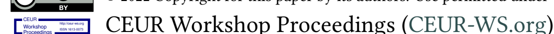
Keywords

Ontology engineering, Ontology testing, Ontology evaluation

1. Introduction

Knowledge graphs are becoming the standard for knowledge management. The construction and the quality of the knowledge graphs strongly rely on the ontology engineering process. The component of the process that assures the quality of the ontology and knowledge graphs is the testing and evaluation. The research regarding this field of study is not extensive and most importantly it covers no support tool-wise and often also methodology-wise for the ontology testers, which differentiates it immensely from software testing practices. The techniques and methodologies used for ontology testing vary based on the stage of the ontology development when they are applied, the layer that they test, and the role that they test.

Specifically, in eXtreme Design (XD), an ontology design methodology that puts in its core

Workshop on Modular Knowledge @ESWC 2022, Hersonissos, Greece, May 29, 2022
fiorela.ciroku2@unibo.it (F. Ciroku); valentina.presutti@unibo.it (V. Presutti)
https://www.unibo.it/sitoweb/fiorela.ciroku2/en (F. Ciroku); https://www.unibo.it/sitoweb/valentina.presutti/en (V. Presutti)
0000-0002-3885-6280 (F. Ciroku); 0000-0002-9380-5160 (V. Presutti)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).
CEUR Workshop Proceedings (CEUR-WS.org)

CEUR-WS.org/Vol-3184/MK_paper1.pdf

the reuse of Ontology Design Patterns (ODPs), the protocol developed for the testing of the ontologies is quite descriptive, but the lack of tool support requires great effort and time from the ontology testers. The protocol covers the testing of the requirements that are gathered in the form of stories from interactions with domain experts and investigations of the datasets that need to be modelled into an ontology. Currently, the work that an ontology tester needs to do to test only one requirement includes: 1) selecting a requirement to test, 2) reformulating the requirement into a SPARQL query, 3) creating a JSON file to represent the expected results of the test, 4) creating a test case owl file and insert all the information, 5) creating test data and adding the reference to the test case, 6) downloading and installing a jar, 7) running the test case in the local terminal or a SPARQL endpoint, and 8) documenting the results. This is a process that is done for each requirement and considering that it is manually done it is prone to syntax and semantic errors that can be dealt with otherwise. Hence, we introduce XDTesting, a tool support for testing ontologies based on the eXtreme Design methodology. The tool is developed as an automation in the GitHub platform and is comprised of numerous features that are in coherence with the testing methodology. The features are identified from interactions with ontology engineers and testers and also from state-of-the-art tools and literature. The rest of the paper is structured as follows. In section 2 we present a summary of featured works related to ontology testing and evaluation. Section 3 provides information about testing based on the eXtreme Design methodology and the tool support that exists for this process. In section 4, we describe how we have collected the initial requirements for the tool and two surveys that we did to understand the state-of-the-art for the tool support. Later, in sections 5 and 6 we express in detail respectively the inception and the development phase of the XD Testing tool automation. The evaluation of the tool is discussed in section 7 and in section 8 we present several action points for the future development of the tool. Lastly, we conclude in section 9.

2. Related work

The foundation of the tool support that we are developing is the XD ontology design and testing methodology. The methodology has been covered in several papers such as [1], [2], [3] where it is described in detail not only in the protocol but also illustrated with examples. Certainly, XD is not the only methodology that supports the testing of the ontologies. In [4], the authors provide a classification of methods and tools for the evaluation of ontologies for industrial practice which gave us a basis of research about the field and a sense of classification of the tools that support the testing. Described in the report are methods such as OntoMetric [5], Natural Language Application metrics, OntoClean [6], EvaLexon [7] and tools such ODEval [8], OntoManager [9]. Another survey of the state-of-art of ontology evaluation is present in [10], where the authors classify the methodologies found into the following categories: those based on comparing the ontology to a "golden standard", those based on using the ontology in an application and evaluating the results, those involving comparisons with a source of data about the domain to be covered by the ontology, those where evaluation is done by humans who try to assess how well the ontology meets a set of predefined criteria, standards, requirements. Similar classification can be found in another survey by Raad in [11]. Meanwhile, in [12], the authors present a model which consists of a meta-ontology, that characterises ontologies as

semiotic objects, and an ontology of ontology validation called oQual, which provides the means to devise the best set of criteria for choosing an ontology over others in the context of a given project. In more recent work, the authors in [13] present a web-based tool called Themis, independent of any ontology development environment, for validating ontologies by means of the application of test expressions which, following lexico-syntactic patterns. While, in [14] Ławrynowicz and Keet introduce the approach of test-driven development (TDD) for ontology authoring. Their tool is implemented as a Protégé plugin so that one can perform a TDD test as a black box test. In another work [15], the development of TDDOnto, which implements a subset of TDD tests, is presented. Mostly all the fore-mentioned works describe methodologies for testing and shortly refer to any tool support that exists to support them. A fact that we need to emphasise is that to our knowledge the literature on the state-of-the-art primarily refers to outdated tools that are no longer maintained or not used in the present. This enforces, even more, our motivation to support the ontology testing process with tools and guarantee a certain level of ontology quality.

3. Testing and its tool support in XD

In XD, the modelling of knowledge is done in the form of modules, which in turn are comprised of building blocks named fragments that are modellings of a set of competency questions. The evaluation of the module starts by testing its ontology fragments by performing competency question verification, inference verification, and error provocation tests. Competency question verification tests allow verifying if the ontology can answer the competency questions that have been selected during the requirement collection phase. Inference verification tests allow verifying that the inference mechanisms are in place, to ensure the correct fulfillment of the inference requirement. Error provocation tests allow verifying how the ontology acts when it is fed random or incorrect data [1].

In addition to these three types of tests, there are two higher-level tests that need to be executed to guarantee the quality of the ontology, named the Integration test and the Regression test. The Integration test allows verifying that the integration of an ontology fragment to a module has been successful. In this case, the ontology engineer shall perform all competency question verification tests, inference verification tests and error provocation tests of the module after the ontology fragment has been added. The Regression test is the rerunning of all the fore-mentioned tests at the level of the ontology network, i.e. executing the tests of all modules that import the module where the ontology fragment was integrated.

The testing of the ontology modules is currently supported only by OWLUnit[1], a java jar that allows running unit tests for ontologies defined according to the OWLUnit Ontology. OWLUnit is also able to run test cases defined by using Ontology Design Pattern's test annotation schema. To execute test cases using OWLUnit, the test engineers must first construct the test case themselves and then use the local terminal or a SPARQL endpoint to execute the test case.

[1] https://github.com/luigi-asprino/owl-unit

4. Requirement collection and surveys

In this section, we describe how we collected the initial set of requirements for the refinement of the existing tool support, OWLUnit, and the request for new features. In addition to the requirement collection, we present two small-scale surveys that we conducted to assess the state-of-the-art of the tool support for testing and the necessity of the testing automation in the GitHub platform.

To collect requirements for the testing automation, we had a brainstorming meeting with researchers from STLab[2], a laboratory dedicated to the representation and processing of knowledge. The participants are researchers that are specialized in ontology engineering and testing based on eXtreme Design and have experience with OWLUnit jar. The aim of the meeting was to discuss the improvement of the current tool support and new features that are necessary to ease the process of testing. The outcome of the meeting has resulted in the identification of the following action points.

1. The tool should enable the testing of an ontology module under development and not published under the official URI of the ontology.
2. The tool should enable the automatic rerun of all test cases when a competency question, SPARQL query, or expected result is updated.
3. The tool should enable the addition of an annotation property or a datatype property to specify the version of the test case.
4. The tool should enable the provision of detailed information in the response messages to understand why the tests fail.[3]
5. Create a test to check the consistency of an ontology. The tool should be able to run a reasoner, and assess the consistency of an ontology that has been provided by the ontology tester.
6. Create a test to check whether a SPARQL query, that is necessary to execute competency questions verification and inferences verification tests, is executable.

After identifying these requirements, we investigated the possibility of using GitHub actions for the automation of the testing process. As standard practice, we surveyed existing actions and apps that are available in the GitHub marketplace. The actions and apps that we took into consideration for analysis fall under the *testing* category. A selection criterion besides the category is also the certification of the creators to guarantee a certain quality of the tools. In total, in the GitHub marketplace, there are 927 actions and apps under the testing category, but merely 37 are verified by Github. The information that we retrieved for each action and app are URL, action name, description, creator, and review. After analysing each of the 37 tools, it resulted that there are no actions or apps that provide support for ontology testing. Investigating beyond these restrictions, we searched for not certified actions or apps in the Marketplace, and still, none support ontology testing.

To argue the choice of why we decided to implement the tool automation in GitHub, we conducted another survey to understand the presence of repositories that contain ontologies or

[2]http://stlab.istc.cnr.it/stlab/

[3]Currently the information that the ontology tester receives in case of failed tests are orderly Java errors or highly general messages that make it difficult to pinpoint the problem in the test case or dataset.

tools to interact with ontologies in GitHub. We searched the keyword *ontology* on GitHub, and as a result, we got 9683 repositories. From these 9683 repositories, we investigated the Top 900 ranking of *Best matched* results with the purpose of categorizing them to assess the percentage of repositories that store ontologies. We classified the repositories under two categories: 1) Repositories that store ontologies and 2) Repositories of tools for handling/editing/visualizing ontologies. The categorization was done manually based on the description of the repository. If the description is missing, the categorization was done based on the name of the repository, e.g. /example-ontology, or the tag of the repository, e.g. ontology. As for the description of the repository, to categorize it under the ontology repository we took into consideration phrases such as ontology of, *example* ontology, ontology for, ontology to describe. While for the categorization under the tool repository, we searched for phrases such as tool, package, visualization, code, manager, editor, etc. Based on the sample under consideration, that is roughly 10% of the total results, it resulted that 65,8% are repositories that store ontologies and 34,2% repositories of tools for handling/editing/visualizing ontologies. Ergo, we decided to develop from scratch an automation, comprised of several actions and workflows, in GitHub to provide tool support for the testing of the numerous ontologies that are stored on this platform.

Considerably, there are many online repositories and libraries such as BioPortal[4], Ontology Design Patterns [5], OLS[6], Obo-Foundry[7], Ontobee[8], etc, whose purpose is to store ontologies of a multitude of domains. Ontologies are also stored locally in private servers, public websites, and collaborative spaces. In contrast to these repositories, GitHub offers features that enable the possibility to have an early alpha tool support that can facilitate the workload of ontology testers instead of building API, web apps, and jars to interact with the fore mentioned repositories. Yet, making the tool useful also for other platforms is an aspect that is already identified and is being planned for the future development of the project.

5. XD Testing tool automation inception

In the present section, we describe the inception phase of the XD Testing tool automation and present an artifact that was produced during this stage. The artifacts that we have produced during the first iteration of the development are a use case diagram and a workflow diagram. It stands to reason that the workflow diagram that we will describe in this section is heavily based on the use case diagram[9].

The artifact, shown in figure 1, is a diagram to present the workflow of the XD Testing tool automation based on XD methodology. As shown in the diagram, there are two actors that interact within the workflow and perform tasks that are in coherence with the use case diagram. The automation is triggered to start whenever an ontology engineer or tester needs an ontology fragment to be tested and to do so provides the name of the ontology fragment in a specific file,

[4]https://bioportal.bioontology.org/

[5]http://ontologydesignpatterns.org/wiki/Main_Page

[6]https://www.ebi.ac.uk/ols/index

[7]https://obofoundry.org/

[8]https://www.ontobee.org/

[9]For more detailed information on the use case diagram, you can browse the website https://fiorelaciroku.github.io/XD-Testing/

for example by writing `RecordingProcess` in the UserInput.txt file. The system listens to this action and creates a structure of directories in GitHub[10] to collect the input for the testing of the fragment `RecordingProcess` and document the test cases that are constructed and executed. Once this action is completed, it creates a Pull Request for the ontology engineer to accept. After the merge to the main branch in GitHub, the ontology engineer can see all the directories that were created by the system and provide the input in the corresponding files. For instance, it can add the information in listing 1 to the CompetencyQuestion.txt file, the SPARQL query in listing 2 in the SPARQLQuery.txt file and lastly the expected results ecxpressed in JSON in listing 3 in ExpectedResult.txt file. Certainly, they need to provide the owl file of the ontology fragment[11]. When the ontology engineer supplies the input, the system checks if the input provided is sufficient for constructing a unit test case. If this is the case, it checks the syntax of the input, more specifically of the SPARQL query and the expected results in JSON. If not, it waits for additional input to be supplied. If the syntax check is passed, the system constructs the unit test and executes it. If the syntax is not correct, then the system waits for a change in the input. The course of the workflow depends on the result of the unit test run. If it is successful, the test run is documented and the process ends. If it is not successful, the system opens a Github issue for the ontology engineer to analyze and waits for a change in the input to restart the testing process.

An important principle that we considered during the inception phase is the continuous integration of the XD Testing tool automation. We need to have a tool that we can build upon and continuously add new features based on several iterations of the development. Furthermore, we aim to develop and release a new version of the tool support after each iteration. Subsequently, the ontology engineer is supported consistently with new features in the process of ontology testing.

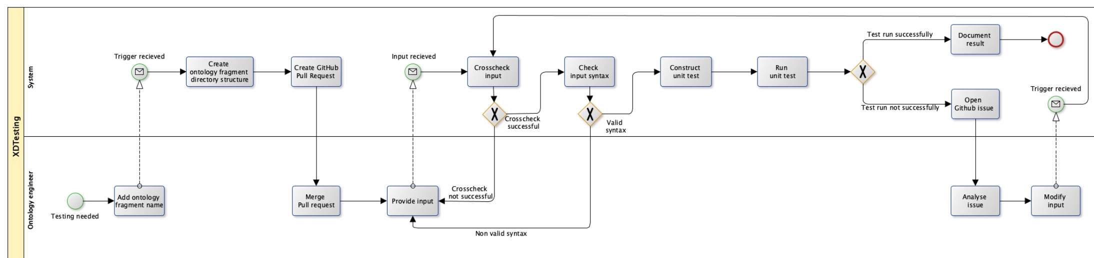


Figure 1: The workflow of the XD Testing tool automation

Listing 1: Competency question

```
CQ01: Which recording is produced by a session?
```

[10]You can find an example of the directories in https://fiorelaciroku.github.io/XD-Testing/

[11]An example of the owl file of the ontology fragment can be found in https://drive.google.com/file/d/1yHWstifYpPlByf1esQ_J9vGEvE4gZx1f/view

Listing 2: SPARQL Query

```
SQCQ01: PREFIX mp: <https://.../RecordingProcess.owl>
SELECT DISTINCT ?recording
WHERE {
?session mp:isSessionOf ?recordingProcess .
?recordingProcess mp:producesRecording ?recording .}
```

Listing 3: Expected results

```
ER01: { "head": {
        "vars": [ "recording" ] } ,
        "results": { "bindings": [ { "recording": {
        "type": "uri" ,
        "value": "https://.../RecordingProcess/SPARQLUnitTest
        /CQDataSet/ComeWithMe" } } ] } }
```

6. XD Testing tool automation development

The development of the tool support is the second phase that we carried out in two iterations so far. In the first iteration of the development, we focused the effort on having a complete working workflow with the most basic features, which are 1) Retrieve input from the user, 2) Create directory structure for the ontology fragment, 3) Crosscheck input, 4) Setup testing environment, 5) Construct unit test, 6) Run of the unit test, 7) Document the unit test, 8) Send discord notification when a unit test case is executed, and 9) Create GitHub issues to report problems. In the second iteration, we have added the features: 1) Check the syntax of the input, 2) Validate the toy dataset, 3) Construct the integration test case, 4) Run the integration test case, and 5) Integrate the ontology fragment to the ontology module.

The *User input retrieval* feature enables the system to read information from files using a bash script. After reading the information, the system uses it in a two-fold manner. More specifically, the user writes in a file the name of the ontology fragment that they want to test and the system retrieves the information and builds the directory structure where the tests are going to be organized and documented. On the other hand, when the user provides the input to construct a test case, the system retrieves the competency question or the general constraint that is to be tested, SPARQL query, and expected result using the corresponding IDs to construct the unit test case.

The objective of the *Ontology fragment directory structure* feature is to create a structure of directories in GitHub for an ontology fragment that is to be tested. The trigger of the automation is the input of the user for the ontology fragment name file inside the directory of an ontology module. The system creates directories for each type of test, namely: Competency question verification test, Inference verification test, Error provocation test, and the last directory Test documentation. In every directory for the tests, there are two sub-folders for storing test cases and toy datasets. Besides the directories, the system creates empty files for storing competency questions, SPARQL queries, expected results, general constraints, and the ontology. Overall,

the action can 1) create a directory structure, 2) add files to directories, 3) show the structure that was created, and 4) make a pull request to the GitHub repository.

The *Crosscheck input* feature verifies that the required input to construct a unit test case is provided from the ontology tester. The tool does this by checking the content of each file, where the input is saved, whether it is empty or not. The required inputs are competency questions, SPARQL queries, expected results, ontology and datasets. The action should validate the input and be able to verify whether all types of input are present in the repository. The alternative option is for the ontology engineer to provide the test case completed. The feature is fully completed and the action can find the directory of the ontology fragment that we are testing, run through the files in the directory and check of each file if it is empty or not. Currently, it displays the result of the validation as a message in the terminal. This result will be used as a condition for the continuation of the test execution.

The *Setup testing environment* feature deals with the installation in a GitHub hosted runner (server) of the technical components that are essential for the execution of the unit test. The components are Java and the OWLUnit jar. Java is downloaded and installed by using a verified GitHub action named *Setup Java JDK*[12]. Meanwhile, for the setup of the OWLUnit jar, we have created a bash script that is capable of retrieving the latest version release of the jar from its GitHub repository and installing it.

The feature *Construct unit test* deals with the filling in of the template for test cases of each type of test. Since all the information that is necessary to construct a test case is present, what this feature does is match the competency question to the corresponding SPARQL query and expected result and insert this information in the template. Another important aspect of the feature is that it makes it possible for the system to generate automatically the prefixes that are used in the test case. Once the construction is complete, the feature creates a Pull request to add the new test case in the correct folder. This request needs to be merged with the main branch of the GitHub repository by the ontology engineer.

The *Run unit test* feature allows the system to execute a test case. The trigger of this automation is the addition of a test case file to any of the sub-folders that store test cases. The feature constructs the path to the test case and by using the Setup testing environment feature it executes the test case with the means of a bash script.

The *Document unit test* feature deals with the documentation of the results of a test case execution. The documentation of the test case is a necessity for the whole procedure of the testing. Even though GitHub provides an output of the action, it is a good practice to have this information stored for documentation purposes. The documentation is done based on a GitHub template that includes information such as the requirement that is being tested, the category of the test, the test case description, the test itself, the input test data, the expected results, the actual results, when a test case was executed, the execution result and the log of the Run unit test feature.

The eXtreme Design methodology foresees that the ontology testers are not the same team as the ontology engineers that develop an ontology module or fragment. When a test case that is being executed by the ontology testers fails or raises issues, the system sends a notification on Discord to the engineering team via the *Send Discord notification* feature. The notification

[12]https://github.com/marketplace/actions/setup-java-jdk

asks for the engineer to analyse the outcome. To build this feature, we have used the *Discord Message Notify*[13] action, the webhook ID, and the token of the Discord server. The notification includes a message and the path of test case documentation. The communication of erroneous test cases can also be done by integrating GitHub with Slack[14] or other platforms that facilitate the creation of a webhook.

The *Create GitHub issue* feature enables the system to create and publish an issue on GitHub. The purpose of the feature is to assign an ontology engineer to analyze an error that was encountered during the test case execution. Considering that syntax problems have already been detected from the system in an earlier step of the framework, the problems that remain in this phase are of a semantic nature. We have developed the feature by using the *Create an issue* action[15] that requires only a secret GitHub token to be used.

The feature *Check syntax* is used to check the syntax of the input, in specific SPARQL queries and expected results, that are provided by the ontology engineer. To check the syntax of SPARQL queries, we have developed a python script and made use of the `rdflib` library. This library makes it possible to import namespaces such as FOAF, RDFS, RDF, OWL, XSD, and most importantly to prepare a query for execution. If the syntax of the query is valid, the script prints "Success!", otherwise it prints the error that was encountered. As for the validation of the expected results, we have developed another python script that imports the `JSON` library and is able to load the string as a JSON file. If the syntax of the expected results is valid as a JSON file, the script prints "Success!" and in the opposite, case it prints the error.

Validate toy dataset feature has been identified during the second iteration of the development for the purpose of catching another type of error that might occur during the execution of a text case. With this feature, we check if the properties that are part of the triples of a toy dataset file are also present in the ontology of the fragment. The check is realized via a python script that firstly imports the `SPARQLWrapper`, declares a wrapper, and then inserts the query that is to be executed. The query is executed and the format of the result of the query is expressed in JSON and is used as a checkpoint in the workflow.

The features related to the integration tests are still under development and they aim to integrate the ontology fragment into the ontology and construct and execute the integration tests. The integration test consists of rerunning competency question verification tests, inference verification tests and error provocation tests of the ontology after the ontology fragment has been added. You can follow the development of these features in GitHub[16].

7. XD Testing tool automation evaluation

The XD Testing tool automation is in the course of its first official evaluation. This evaluation will be realised in the form of a testing session in collaboration with ontology engineers and testers. They will be introduced first to the tool that is documented in the specific guidelines. Later, we will present the exercise, that the participants have to do during the session and instructions on

[13]https://github.com/marketplace/actions/discord-message-notify
[14]https://slack.com/
[15]https://github.com/marketplace/actions/create-an-issue
[16]https://github.com/FiorelaCiroku/XD-Testing

how to do it. For the exercise we have prepared an ontology fragment, a competency question with the respective SPARQL query and expected result, general constraints, and datasets. All this information is displayed in this GitHub page[17]. To document the problems that the participants will encounter during the session we have created two GitHub issue templates. One template is to document reports for bugs and the other for requesting features that they would like to have. We have also opened several discussion boards[18] to light up conversations regarding features, bugs, documentation, and general feedback.

Based on preparatory experiments, we concluded that we will begin the testing when the tool is more user-friendly. This way the participants of the experiments can focus on the functionality rather than the current usability. For this reason, we are implementing an interface that is shortly described in the future work section. We expect the development of the interface to be completed fairly soon and therefore to be ready for the testing of the XDTesting tool.

8. Future work

A continuous driving force for the development of the tool automation is the requirements that we have gathered from discussions with ontology engineers and from the available literature. An immediate action point is to complete the features regarding the integration test and add them to the general testing workflow.

At the same time, we are working to build an interface using Angular JS[19] for receiving the input from the ontology engineers in a more user-friendly way considering that not all end-users of the tool are tech-savvy. This interface requests the user to provide the name of the ontology and the ontology fragment, the competency question, SPARQL query, and expected results and it passes this information to GitHub by producing a JSON file. Part of the interface will also be a dashboard to display the results of the executed test cases within a GitHub repository.

Lastly, another action point that we have identified is to extend the tool automation beyond the GitHub platform. We are considering options such as self-hosted servers, cloud-hosted management, or expansion to other platforms such as GitLab[20]. At this phase of the project, we are discussing the advantages and disadvantages of each option and are yet to come to a conclusion.

9. Conclusion

In this paper, we present and describe in detail XD Testing tool automation, a tool that was built to support the testing phase of the ontology engineering process. The tool itself aims to fulfill an immediate need for support in this field and contribute to the research community by encouraging ontology testing as a process and making it a standard practice. We have described the methodology that we use as a foundation and the choices that we have made regarding the implementation of the tool, the conceptualization, and the development phase based on

[17]https://fiorelaciroku.github.io/XD-Testing/
[18]https://github.com/FiorelaCiroku/XD-Testing/discussions
[19]https://angularjs.org/
[20]https://about.gitlab.com/

agile methodologies and continuous integration practices. We also present a plan for the first evaluation of the XD Testing tool and describe action points that we are currently working on and aim to complete as part of the second iteration of the development.

References

[1] E. Blomqvist, V. Presutti, E. Daga, A. Gangemi, Experimenting with extreme design, in: International Conference on Knowledge Engineering and Knowledge Management, Springer, 2010, pp. 120–134.

[2] V. Presutti, E. Daga, A. Gangemi, E. Blomqvist, extreme design with content ontology design patterns, in: Proc. Workshop on Ontology Patterns, 2009, pp. 83–97.

[3] E. Blomqvist, K. Hammar, V. Presutti, Engineering ontologies with patterns-the extreme design methodology., Ontology Engineering with Ontology Design Patterns (2016) 23–50.

[4] J. Hartmann, P. Spyns, A. Giboin, D. Maynard, R. Cuel, M. Suarez-Figeroa, Y. Sure, Methods for ontology evaluation. (2005).

[5] B. Lantow, Ontometrics: Putting metrics into use for ontology evaluation., in: KEOD, 2016, pp. 186–191.

[6] N. Guarino, C. A. Welty, An overview of ontoclean, Handbook on ontologies (2004) 151–171.

[7] P. Spyns, Validating evalexon: validating a tool for evaluating automatically lexical triples mined from texts (2007).

[8] Ó. Corcho, A. Gómez-Pérez, R. González-Cabero, M. C. Suárez-Figueroa, Odeval: a tool for evaluating rdf (s), daml+ oil, and owl concept taxonomies, in: IFIP International Conference on Artificial Intelligence Applications and Innovations, Springer, 2004, pp. 369–382.

[9] L. Stojanovic, N. Stojanovic, J. Gonzalez, R. Studer, Ontomanager–a system for the usage-based ontology management, in: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", Springer, 2003, pp. 858–875.

[10] J. Brank, M. Grobelnik, D. Mladenic, A survey of ontology evaluation techniques, in: Proceedings of the conference on data mining and data warehouses (SiKDD 2005), Citeseer Ljubljana Slovenia, 2005, pp. 166–170.

[11] J. Raad, C. Cruz, A survey on ontology evaluation methods, in: Proceedings of the International Conference on Knowledge Engineering and Ontology Development, part of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, 2015.

[12] A. Gangemi, C. Catenacci, M. Ciaramita, J. Lehmann, Modelling ontology evaluation and validation, in: European Semantic Web Conference, Springer, 2006, pp. 140–154.

[13] A. Fernández-Izquierdo, R. García-Castro, Themis: a tool for validating ontologies through requirements., in: SEKE, 2019, pp. 573–753.

[14] C. M. Keet, A. Ławrynowicz, Test-driven development of ontologies, in: European Semantic Web Conference, Springer, 2016, pp. 642–657.

[15] A. Lawrynowicz, C. M. Keet, The tddonto tool for test-driven development of dl knowledge bases (2016).

A. Online Resources

- GitHub Marketplace, last accessed 22 April 2022,
- W3C Ontology repositories, last accessed 22 April 2022,
- Crop Ontology Community Website, last accessed 22 April 2022.