Test-Driven Development of Ontologies

C. Maria Keet[1] and Agnieszka Ławrynowicz[2]

[1] Department of Computer Science, University of Cape Town, South Africa
mkeet@cs.uct.ac.za
[2] Institute of Computing Science, Poznan University of Technology, Poland
agnieszka.lawrynowicz@cs.put.poznan.pl

Abstract. Emerging ontology authoring methods to add knowledge to an ontology focus on ameliorating the validation bottleneck. The verification of the newly added axiom is still one of trying and seeing what the reasoner says, because a systematic testbed for ontology authoring is missing. We sought to address this by introducing the approach of test-driven development for ontology authoring. We specify 36 generic tests, as TBox queries and TBox axioms tested through individuals, and structure their inner workings in an 'open box'-way, which cover the OWL 2 DL language features. This is implemented as a Protégé plugin so that one can perform a TDD test as a black box test. We evaluated the two test approaches on their performance. The TBox queries were faster, and that effect is more pronounced the larger the ontology is.

1 Introduction

The process of ontology development has progressed much over the past 20 years, especially by the specification of high-level, information systems-like methodologies [8, 25], and both stand-alone and collaborative tools [9, 10]. But support for effective low-level *ontology authoring*—adding the right axioms and adding the axioms right—has received some attention only more recently. Processes at this 'micro' level of the development may use the reasoner to propose axioms with FORZA [13], use Ontology Design Patterns (ODPs) [6], and repurpose ideas from software engineering practices, notably exploring the notion of unit tests [27], eXtreme Design with ODPs [3], and Competency Question (CQ)-based authoring using SPARQL [23].

However, testing whether a CQ can be answered does not say how to add the knowledge represented in the ontology, FORZA considers simple object properties only, and eXtreme Design limits one to ODPs that do not come out of the blue but have been previously prepared. Put differently, there is no systematic testbed for ontology engineering, other than manual efforts by a knowledge engineer to add or change something and running the reasoner to check its effects. This still puts a high dependency on expert knowledge engineering, which ideally should not be in the realm of an art, but be rather at least a systematic process for good practices.

We aim to address this problem by borrowing another idea from software engineering: *test-driven development* (TDD) [2]. TDD ensures that what is added to the program core (here: ontology) does indeed have the intended effect specified upfront. Moreover, TDD in principle is cognitively a step up from the 'add stuff and lets see what happens'-attitude, therewith deepening the understanding of the ontology authoring process and the logical consequences of an axiom.

There are several scenarios of TDD usage in ontology authoring:

I. CQ-driven TDD Developers (domain experts, knowledge engineers etc) specify CQs. A CQ is translated automatically into one or more axioms. The axiom(s) are the input of the relevant TDD test(s) to be carried out. The developers who specify the CQs could be oblivious to the inner workings of the two-step process of translating the CQ and testing the axiom(s).

II-a. Ontology authoring-driven TDD - the knowledge engineer The knowledge engineer knows which axiom s/he wants to add, types it, which is then fed directly into the TDD system.

II-b. Ontology authoring-driven TDD - the domain expert As there is practically a limited amount of 'types' of axioms to add, one could create templates, alike the notion of the "logical macro" ODP [22], which then map onto *generic*, domain-independent tests (as will be specified in Section 3). For instance, a domain expert could choose the all-some template from a list, i.e., an axiom of the form $C \sqsubseteq \exists R.D$. The domain expert instantiates it with relevant domain entities (e.g., $\mathsf{Professor} \sqsubseteq \exists \mathsf{teaches.Course}$), and the TDD test for the $C \sqsubseteq \exists R.D$ type of axiom is then run automatically. The domain expert need not know the logic, but behind the usability interface, what gets sent to the TDD system is that axiom.

While in each scenario the actual testing can be hidden from the user's view, it is necessary to specify what actually happens during such testing and how it is tested. Here, we assume that either the first step of the CQ process is completed, or the knowledge engineer adds the axiom, or that the template is populated, respectively; i.e., that we are at the stage where the axioms are fed into the TDD test system. To realise the testing, a number of questions have to be answered:

1. Given the TDD procedure in software engineering—check that the desired feature is absent, code it, test again (*test-first* approach)—then what does that mean for ontology testing when transferred to ontology development?
2. TDD requires so-called *mock objects* for 'incomplete' parts of the code; is there a parallel to it in ontology development, or can that be ignored?
3. In what way and where (if at all) can this be integrated as a methodological step in existing ontology engineering methodologies that are typically based on waterfall, iterative, or lifecycle principles?

To work this out for ontologies, we take some inspiration from TDD for conceptual modelling. Tort et al. [26] essentially specify 'unit tests' for each feature/possible addition to a conceptual model, and test such an addition against sample individuals. Translating this to OWL ontologies, such testing is possible by means of ABox individuals, and then instead of using an ad hoc algorithm, one can avail of the automated reasoner. In addition, for ontologies, one can avail of a query language for the TBox, namely, SPARQL-OWL [15], and most of the tests can be specified in that language as well. We define TBox and ABox-driven

TDD tests for the basic axioms one can add to an OWL 2 DL ontology. To examine practical feasibility for the ontology engineer and determine which TDD strategy is the best option, we implemented the TDD tests as a Protégé plugin and evaluated it on performance by comparing TBox and ABox TDD tests for 67 ontologies. The TBox TDD tests outperform the ABox ones except for disjointness and this effect is more pronounced with larger ontologies. Overall, we thus add a new mechanism and tool to the ontology engineer's 'toolbox' to enable systematic development of ontologies in an agile way.

The remainder of the paper is structured as follows. Section 2 describes related works on TDD in software and ontology development. Section 3 summarises the TDD tests and Section 4 evaluates them on performance with the Protégé plugin. We discuss in Section 5 and conclude in Section 6. Data, results, and more detail on the TDD test specifications is available at `https://semantic.cs.put.poznan.pl/wiki/aristoteles/doku.php`.

2 Related Work

To 'transfer' TDD to ontology engineering, we first summarise preliminaries about TDD from software engineering and subsequently discuss related works on tests in ontology engineering.

TDD in software development. TDD was introduced as a software development methodology where one writes new code only if an automated test has failed [2]. TDD permeates the whole development process, which can be summarised as: 1) Write a test for a piece of functionality (that was based on a requirement), 2) Run all tests to check that the new test fails, 3) Write relevant code that passes the test, 4) Run the specific test to verify it passes, 5) Refactor the code, and 6) Run all tests to verify that the changes to the code did not change the external behaviour of the software (regression testing) [24]. The important difference with unit tests, is that TDD is a *test-first* approach rather than *test-last* (design, code, test). TDD results in being more focussed, improves communication, improves understanding of required software behaviour, and reduces design complexity [17]. Quantitatively, TDD produced code passes more externally defined tests—i.e, better software quality—and involves less time spent on debugging, and experiments showed that it is significantly more productive than test-last [11].

TDD has been applied to conceptual data modelling, where each language feature has its own test specification in OCL that involves creating the objects that should, or ought not to, instantiate the UML classes and associations [26]. Tort and Olivé's tool was evaluated with modellers, which made clear, among others, that more time was spent on developing and revising the conceptual model to fix errors than on writing the test cases [26].

Tests in ontology engineering. In ontology engineering, an early explorative work on borrowing the notion of testing from software engineering is described in [27], which explores several adaptation options: testing with the axiom and its negation, formalising CQs, checks by means on integrity constraints, autoepistemic operators, and domain and range assertions. Working with CQs has shown to be the most popular approach, notably [23], who analyse CQs and their patterns for use with SPARQL queries that then would be tested against the ontology. Their focus is on individuals and the formalisation stops at *what* has to be tested, not *how* that can, or should, be done. Earlier work on CQs and queries include the OntologyTest tool for ABox instances, which specifies different types of tests, such as "instantiation tests" (instance checking) and "recovering tests" (query for a class' individuals) and using mock individuals where applicable [7]; other instance-oriented test approaches is RDF/Linked Data [16]. There is also an eXtreme Design NeON plugin with similar functionality and ODP rapid design [3, 21], likewise with RapidOWL [1], which lacks the types of tests, and a more basic variant exists in the EFO Validator[3]. Neither are based on the principle of TDD. The only one that aims to zoom in on unit tests for TBox testing requires the tests to be specified in Clojure and the ontology in Tawny-Owl notation, describes subsumption tests only [28], and the tests are tailored to the actual ontology rather than reusable 'templates' for the tests covering all OWL language features.

Related notions have been proposed in methods for particular types of axioms, such as disjointness [5] and domain and range constraints [13]. Concerning methodologies, none of the 9 methodologies reviewed by [8] are TDD-based, nor is NeON [25]. The Agile-inspired OntoMaven [20] has OntoMvnTest with 'test cases' only for the usual syntax checking, consistency, and entailment [20].

Thus, full TDD ontology engineering has not been proposed yet. While the idea of unit tests—which potentially could become part of TDD tests—has been proposed, there is a dearth of actual specifications as to what exactly is, or should be, going on in such as test. Even when one were to specify basic tests for each language feature, it is unclear whether they can be put together in a modular fashion for the more complex axioms that can be declared with OWL 2. Further, there is no regression testing to check that perhaps an earlier modelled CQ—and thus a passed test—conflicts with a later one.

3 TDD Specification for Ontologies

First the general procedure and preliminaries are introduced, and then the TBox and RBox TDD tests are summarised.

3.1 Preliminaries on Design and Notation of the TDD tests

The generalised TDD test approach is summarised as follows for the default case:

1. input: CQ into axiom, axiom, or template into axiom.
2. given: axiom x of type X to be added to the ontology.
3. check the vocabulary elements of x are in ontology O (itself a TDD test)
4. run TDD test twice:

[3] `http://www.ebi.ac.uk/fgpt/sw/efovalidator/index.html`

(a) the first execution should fail (check $O \nvDash x$ or not present),
(b) update the ontology (add x), and
(c) run the test again which then should pass (check that $O \models x$) and such that there is no new inconsistency or undesirable deduction

5. Run all previous successful tests, which should pass (i.e., regression testing)

There are principally two options for the TDD tests: a test at the TBox-level or always using individuals explicitly asserted in the ABox. We specify tests for both approaches, where possible. For the test specifications, we use the OWL 2 notation for the ontology's vocabulary: $C, D, E, ... \in V_C$, $R, S, ... \in V_{OP}$, and $a, b, ... \in V_I$, and SPARQL-OWL notation [15] where applicable, as it conveniently reuses OWL functional syntax-style notation merged with SPARQL's queried objects (i.e., ?x) for the formulation of the query. For instance, $\alpha \leftarrow$ SubClassOf (?x D) will return all subclasses of class D. Details of SPARQL-OWL and its implementation are described in [15].

Some TBox and all ABox tests require additional classes or individuals for testing purposes only, which resembles the notion of *mock objects* in software engineering [18, 14]. We shall import this notion into the ontology setting, as *mock class* for a temporary OWL class created for the TDD test, *mock individual* for a temporary ABox individual, and *mock axiom* for a temporary axiom. These mock entities are to be removed from the ontology after completion of the test.

Steps 3 and 4a in the sequence listed above may give an impression of epistemic queries. It has to be emphasised that there is a fine distinction between 1) checking when an element is in the vocabulary of the TBox of the ontology (in V_C or V_{OP}) versus autoepistemic queries, and 2) whether something is *logically* true or false versus a *test* evaluating to true or false. In the TDD context, the epistemic-sounding 'not asserted in or inferred from the ontology' is to be understood in the context of a *TDD test*, like whether an ontology has some class C in its vocabulary, not whether it is 'known to exist' in one's open or closed world. Thus, an epistemic query language is not needed for the TBox tests.

3.2 Generic Test Patterns for TBox Axioms

The tests are introduced in pairs, where the primed test names concern the tests with individuals; they are written in SPARQL-OWL notation. They are presented in condensed form due to space limitations. The TDD tests in algorithm-style notation are available in an extended technical report of this paper [12].

Class subsumption, $Test_{cs}$ or $Test'_{cs}$. When the axiom to add is of type $C \sqsubseteq D$, with C and D named classes, then $O \models \neg(C \sqsubseteq D)$ should be true if it were not present. Logically, then in the tableau, $O \cup \neg(\neg(C \sqsubseteq D))$ should be inconsistent, i.e., $O \cup (\neg C \sqcup D)$. Given the current Semantic Web technologies, it is easier to query the ontology for the subclasses of D and to ascertain that C is not in query answer α rather than create and execute tailor-made tableau algorithms:

$\mathbf{Test_{CS}}$ $= \alpha \leftarrow$ SubClassOf(?x D). *If $C \notin \alpha$, then $C \sqsubseteq D$ is neither asserted nor entailed in the ontology; the test fails.* ◀

After adding $C \sqsubseteq D$ to the ontology, the same test is run, which should evaluate to $C \in \alpha$ and therewith $Test_{cs}$ returns 'pass'. The TTD test with individuals checks whether an instance of C is also an instance of D:

Test$'_{\mathbf{cs}}$ = *Create a mock object a and assert $C(a)$. $\alpha \leftarrow$* Type(?x D)*. If $a \notin \alpha$, then $C \sqsubseteq D$ is neither asserted nor entailed in the ontology.* ◀

Class disjointness, $Test_{cd}$ or $Test'_{cd}$. One can assert the complement, $C \sqsubseteq \neg D$, or disjointness, $C \sqcap D \sqsubseteq \bot$. Let us consider the former first (test $Test_{cd_c}$), such that then $\neg(C \sqsubseteq \neg D)$ should be true, or $T(C \sqsubseteq \neg D)$ false (in the sense of 'not be in the ontology'). Testing for the latter only does not suffice, as there are more cases where $O \nvDash C \sqsubseteq D$ holds, but disjointness is not really applicable—being classes in distinct sub-trees in the TBox—or holds when disjointness is asserted already, which is when C and D are sibling classes. For the complement, we simply can query for it in the ontology:

Test$_{\mathbf{cd_c}}$ = $\alpha \leftarrow$ ObjectComplementOf(C ?x)*. If $D \notin \alpha$, then $O \nvDash C \sqsubseteq \neg D$; hence, the test fails.* ◀

For $C \sqcap D \sqsubseteq \bot$, the test is:

Test$_{\mathbf{cd_d}}$ = $\alpha \leftarrow$ DisjointClasses(?x D)*. If $C \notin \alpha$, then $O \nvDash C \sqcap D \sqsubseteq \bot$.* ◀

The ABox option uses a query or classification; availing of the reasoner only:

Test$'_{\mathbf{cd}}$ = *Create individual a, assert $C(a)$ and $D(a)$. ostate $\leftarrow$ Run the reasoner. If ostate is consistent, then either $O \nvDash C \sqsubseteq \neg D$ or $O \nvDash C \sqcap D \sqsubseteq \bot$ directly or through one or both of their superclasses (test fails). Else, the ontology is inconsistent (test passed); thus either $C \sqsubseteq \neg D$ or $C \sqcap D \sqsubseteq \bot$ is already asserted among both their superclasses or among C or D and a superclass of D or C, respectively.* ◀

Further, from a modelling viewpoint, it would make sense to also require C and D to be siblings. The sibling requirement can be added as an extra check in the interface to alert the modeller to it, but not be enforced from a logic viewpoint.

Class equivalence, $Test_{ce}$ and $Test'_{ce}$. When the axiom to add is of the form $C \equiv D$, then $O \models \neg(C \equiv D)$ should be true before the edit, or $O \nvDash C \equiv D$ false. The latter is easier to test—run $Test_{cs}$ twice, once for $C \sqsubseteq D$ and once for $D \sqsubseteq C$—or use one SPARQL-OWL query:

Test$_{\mathbf{ce}}$ = $\alpha \leftarrow$ EquivalentClasses(?x D)*. If $C \notin \alpha$, then $O \nvDash C \equiv D$; the test fails.* ◀

Note that D can be complex here, but C cannot. For class equivalence with individuals, we can extend $Test'_{cs}$:

Test$'_{\mathbf{ce}}$ = *Create a mock object a, assert $C(a)$. Query $\alpha \leftarrow$* Type(?x D)*. If $a \notin \alpha$, then $O \nvDash C \equiv D$ and the test fails; delete $C(a)$ and a. Else, delete $C(a)$, assert $D(a)$. Query $\alpha \leftarrow$* Type(?x C)*. If $a \notin \alpha$, then $O \nvDash C \equiv D$, and the test fails. Delete $D(a)$ and a.* ◀

Simple existential quantification, $Test_{eq}$ or $Test'_{eq}$. The axiom pattern is $C \sqsubseteq \exists R.D$, so $O \nvDash \neg(C \sqsubseteq \exists R.D)$ should be true, or $O \models C \sqsubseteq \exists R.D$ false (or: not asserted) before the ontology edit. One could do a first check that D is not a descendant of R but if it is, then it may be the case that $C' \sqsubseteq \exists R.D$, with C a different class from C'. This still requires one to confirm that C is not a subclass of $\exists R.D$. This can be combined into one query/TDD test:

Test$_{\mathbf{eq}}$ = $\alpha \leftarrow$ SubClassOf(?x ObjectSomeValuesFrom(R D))*. If $C \notin \alpha$, then*

$O \not\models C \sqsubseteq \exists R.D$, *hence the test fails.* ◀
If $C \notin \alpha$, then the axiom is to be added to the ontology, the query run again, and if $C \in \alpha$, then the test cycle is completed.

From a modelling viewpoint, desiring to add a CQ that amounts to $C \sqsubseteq \exists R.\neg D$ may look different, but $\neg D \equiv D'$, so it amounts to testing $C \sqsubseteq \exists R.D'$, i.e., essentially the same pattern. This also can be formulated directly into a SPARQL-OWL query, encapsulated in a TDD test:

Test$_{\mathbf{eq_{nd}}}$ $= \alpha \leftarrow$ SubClassOf(?x ObjectSomeValuesFrom(R ObjectComplementOf(D))). *If* $C \notin \alpha$, *then* $O \not\models C \sqsubseteq \exists R.\neg D$; *hence, the test fails.* ◀
It is slightly different for $C \sqsubseteq \neg\exists R.D$. The query with TDD test is as follows:

Test$_{\mathbf{eq_{nr}}}$ $= \alpha \leftarrow$ SubClassOf(?x ObjectComplementOf(ObjectSomeValuesFrom(R D))). *If* $C \notin \alpha$, *then* $O \not\models C \sqsubseteq \neg\exists R.D$, *and the test fails.* ◀
The TDD test $Test'_{eq}$ with individuals only is as follows:

Test$'_{\mathbf{eq}}$ = *Create mock objects a, assert* $(C \sqcap \neg\exists R.D)(a)$. *ostate* $\leftarrow$ *Run the reasoner. If ostate is consistent, then* $O \not\models C \sqsubseteq \exists R.D$; *test fails. Delete* $(C \sqcap \neg\exists R.D)(a)$, *and* a. ◀
This holds similarly for $C \sqsubseteq \exists R.\neg D$ ($Test'_{eq_{nd}}$). Finally, for $C \sqsubseteq \neg\exists R.D$:

Test$'_{\mathbf{eq_{nr}}}$ = *Create two mock objects, a and b; assert* $C(a)$, $D(b)$, *and* $R(a,b)$. *ostate* $\leftarrow$ *Run the reasoner. If ostate is consistent, then* $O \not\models C \sqsubseteq \neg\exists R.D$, *hence, the test fails. Delete* $C(a)$, $D(b)$, $R(a,b)$, a, *and* b. ◀

Simple universal quantification, $Test_{uq}$ *or* $Test'_{uq}$. The axiom to add is of the pattern $C \sqsubseteq \forall R.D$, so then $O \not\models \neg(C \sqsubseteq \forall R.D)$ should hold, or $O \models C \sqsubseteq \forall R.D$ false (not be present in the ontology), before the ontology edit. This has a similar pattern for the TDD test as the one for existential quantification,

Test$_{\mathbf{uq}}$ $= \alpha \leftarrow$ SubClassOf(?x ObjectAllValuesFrom(R D)). *If* $C \notin \alpha$, *then* $O \not\models C \sqsubseteq \forall R.D$, *hence, the test fails.* ◀
which then can be added and the test ran again. The TDD test for $Test'_{uq}$ is alike $Test'_{eq}$, but then the query is $\alpha \leftarrow$ Type(?x, ObjectAllValuesFrom(R D)).

3.3 Generic Test Patterns for Object Properties

TDD tests for object properties (the RBox) do not lend themselves well for TBox querying, though the automated reasoner can be used for the TDD tests.

Domain axiom, $Test_{da}$ *or* $Test'_{da}$. The TDD needs to check that $\exists R \sqsubseteq C$ that is not yet in O, so $O \models \neg(\exists R \sqsubseteq C)$ should be true, or $O \models \exists R \sqsubseteq C$ false. There are two options with SPARQL-OWL. First, one can query for the domain:

Test$_{\mathbf{da}}$ $= \alpha \leftarrow$ ObjectPropertyDomain(R ?x) *If* $C \notin \alpha$, *then* $O \not\models \exists R \sqsubseteq C$; *test fails.* ◀
Alternatively, one can query for the superclasses of $\exists R$ (it is shorthand for $\exists R.\top$), where the TDD query is: $\alpha \leftarrow$ SubClassOf(SomeValuesFrom(R Thing) ?x). Note that $C \in \alpha$ only will be returned if C is the only domain class of R or when $C \sqcap C'$ (but not if it is $C \sqcup C'$, which is a superclass of C). The ABox test is:

Test$'_{\mathbf{da}}$ = *Check* $R \in V_{OP}$ *and* $C \in V_C$. *Add individuals a and topObj, add* $R(a, topObj)$. *Run the reasoner. If* $a \notin C$, *then* $O \not\models \exists R \sqsubseteq C$ *(also in the strict sense as is or with a conjunction); hence the test fails. Delete a and topObj.* ◀

If the answer is empty, then R does not have any domain specified yet, and if $C \notin \alpha$, then $O \nvDash \exists R \sqsubseteq C$, hence, it can be added and the test run again.

Range axiom, $Test_{ra}$ or $Test'_{ra}$. Thus, $\exists R^{-} \sqsubseteq D$ should not be in the ontology before the TDD test. This is similar to the domain axiom test:

$\mathbf{Test_{ra}} = \alpha \leftarrow$ ObjectPropertyRange(R ?x). *If $D \notin \alpha$, then $O \nvDash \exists R^{-} \sqsubseteq D$; test fails.* ◀
Or one can query $\alpha \leftarrow$ SubClassOf(SomeValuesFrom(ObjectInverseOf(R) Thing) ?x). Then $D \in \alpha$ if $O \models \exists R^{-} \sqsubseteq D$ or $O \models \exists R^{-} \sqsubseteq D \sqcap D'$, and only `owl:Thing` $\in \alpha$ if no range was declared for R. The test with individuals:

$\mathbf{Test'_{ra}} =$ *Check $R \in V_{OP}$ and $D \in V_C$. Add individuals a and $topObj$, add $R(topObj, a)$. If $a \notin D$, then $O \nvDash \exists R^{-} \sqsubseteq D$. Delete $R(topObj, a)$, a, $topObj$.* ◀

Object property subsumption and equivalence, $Test_{ps}$ and $Test_{pe}$, and $Test'_{ps}$ and $Test'_{pe}$. For property subsumption, $R \sqsubseteq S$, we have to test that $O \models \neg(R \sqsubseteq S)$, or that $R \sqsubseteq S$ fails. This is simply:

$\mathbf{Test_{ps}} = \alpha \leftarrow$ SubObjectPropertyOf(?x S) *If $R \notin \alpha$, then $O \nvDash R \sqsubseteq S$; test fails.* ◀
Regarding the ABox variant, for $R \sqsubseteq S$ to hold given the OWL semantics, it means that, given some individuals a and b, that if $R(a, b)$ then $S(a, b)$:

$\mathbf{Test'_{ps}} =$ *Check $R, S \in V_{OP}$. Add individuals a, b, add $R(a, b)$. Run the reasoner. If $S(a, b) \notin \alpha$, then $O \nvDash R \sqsubseteq S$; test fails. Delete $R(a, b)$, a, and b.* ◀
Upon the ontology update, it should infer $S(a, b)$. There is no guarantee that $R \sqsubseteq S$ was added, but $R \equiv S$ instead. This can be observed easily with the following test:

$\mathbf{Test'_{pe}} =$ *Check $R, S \in V_{OP}$. Add mock individuals a, b, c, d, add $R(a, b)$ and $S(c, d)$. Run the reasoner. If $S(a, b) \in \alpha$ and $R(c, d) \notin \alpha$, then $O \models R \sqsubseteq S$ (hence the ontology edit was correct); test fails. Else, i.e. $\{S(a, b), R(c, d)\} \in \alpha$, so $O \models R \equiv S$; test passes. Delete $R(a, b)$ and $S(c, d)$, and a, b, c, d.* ◀
For object property equivalence at the Tbox level, i.e., $R \equiv S$, one could use $Test_{ps}$ twice, or simply use the EquivalentObjectProperties:

$\mathbf{Test_{pe}} = \alpha \leftarrow$ EquivalentObjectProperties(?x S) *If $R \notin \alpha$, then $O \nvDash R \equiv S$; test fails.* ◀

Object property inverses, $Test_{pi}$ and $Test'_{pi}$. There are two options since OWL 2: explicit inverses (e.g., `teaches` with its inverse declared as `taught by`) or 'implicit' inverse (e.g., `teaches` and `teaches`$^{-}$). For the failure-test of TDD, only the former case can be tested. Also here there is a TBox and an ABox approach; their respective tests are:

$\mathbf{Test_{pi}} = \alpha \leftarrow$ InverseObjectProperties(?x S) *If $R \notin \alpha$, then $O \nvDash R \sqsubseteq S^{-}$; test fails.* ◀

$\mathbf{Test'_{pi}} =$ *Check $R, S \in V_{OP}$. Assume S is intended to be the inverse of R (with R and S having different names). Add mock individuals a, b, and add $R(a, b)$. Run the reasoner. If $O \nvDash S(b, a)$, then $O \nvDash R \sqsubseteq S^{-}$; hence, the test fails. Delete a, b.* ◀

Object property chain, $Test_{pc}$ or $Test'_{pc}$. The axiom to be added is one of the permissible chains (except for transitivity; see below), such as $R \circ S \sqsubseteq S$, $S \circ R \sqsubseteq S$, $R \circ S_1 \circ ... \circ S_n \sqsubseteq S$ (with $n > 1$). This is increasingly more cumbersome to test, because many more entities are involved, hence, more opportunity to have incomplete knowledge represented in the ontology and thus more hassle to check all possibilities that lead to not having the desired effect. Aside from searching the owl file for `owl:propertyChainAxiom`, with the relevant properties included in order, the SPARQL-OWL-based TDD test is:

Test$_{\mathbf{pc}}$, for $R \circ S \sqsubseteq S$ = $\alpha \leftarrow$ SubObjectPropertyOf(ObjectPropertyChain(R S) ?x). *If $S \notin \alpha$, then $O \not\models R \circ S \sqsubseteq S$, and the test fails.* ◀

and similarly with the other permutations of property chains. However, either option misses three aspects of chains: 1) a property chain is pointless if the properties involved are never used in the intended way, 2) this cannot ascertain that it does only what was intended, and 3) whether the chain does not go outside OWL 2 due to some of them being not 'simple'. For $O \models R \circ S \sqsubseteq S$ to be interesting for the ontology, also at least one $O \models C \sqsubseteq \exists R.D$ and one $O \models D \sqsubseteq \exists S.E$ should be present. If they all were, then a SPARQL-OWL query $\alpha \leftarrow$ SubClassOf(?x ObjectSomeValuesFrom(S E)) will have $C \in \alpha$. If either of the three axioms are not present, then $C \notin \alpha$. The ABox TDD test is more cumbersome:

Test$'_{\mathbf{pc}}$, for $R \circ S \sqsubseteq S$ = *Check $R, S \in V_{OP}$ and $C, D, E \in V_C$. If $C, D, E \notin V_C$, then add the missing class(es) (C, D, and/or E) as mock classes. Run the test $Test_{eq}$ or $Test'_{eq}$, for both $C \sqsubseteq \exists R.D$ and for $D \sqsubseteq \exists S.E$. If $Test_{eq}$ is false, then add $C \sqsubseteq \exists R.D$, $D \sqsubseteq \exists S.E$, or both, as mock axiom. If $O \models C \sqsubseteq \exists S.D$, then the test is meaningless, for it would not test the property chain. Then add mock class C', mock axiom $C' \sqsubseteq \exists R.D$. Verify with $Test_{eq}$ or $Test'_{eq}$. $\alpha \leftarrow$* SubClassOf(?x ObjectSomeValuesFrom(S E)). *If $C' \notin \alpha$, then $O \not\models R \circ S \sqsubseteq S$; test fails. Else, i.e., $O \not\models C \sqsubseteq \exists S.D$: $\alpha \leftarrow$* SubClassOf(?x ObjectSomeValuesFrom(S E)). *If $C \notin \alpha$, then $O \not\models R \circ S \sqsubseteq S$; test fails. Delete all mock entities.* ◀

Assuming that the test fails, i.e., $C \notin \alpha$ (resp. $C' \notin \alpha$) and thus $O \not\models R \circ S \sqsubseteq S$, then add the chain and run the test again, which then should pass (i.e., $C \in \alpha$). The procedure holds similarly for the other permissible combinations of object properties in a property chain/complex role inclusion.

Object property characteristics, $Test_{p_x}$. TDD tests can be specified for the ABox approach, but only transitivity and local reflexivity have a TBox test.

R is functional, $Test'_{p_f}$, i.e., an object has at most one R-successor:

Test$'_{\mathbf{p}_f}$ = *Check $R \in V_{OP}$ and $a, b, c \in V_I$; if not present, add. Assert mock axioms $R(a, b)$, $R(a, c)$, and $b \neq c$, if not present already. Run reasoner. If O is consistent, then $O \not\models$* Func(R)*, so the test fails. (If O is inconsistent, then the test passes.) Remove mock axioms and individuals, as applicable.* ◀

R is inverse functional, $Test'_{p_{if}}$. This is as above, but then in the other direction, i.e., $R(b, a)$, $R(c, a)$ with b, c declared distinct. Thus:

Test$'_{\mathbf{p}_{if}}$ = *Check $R \in V_{OP}$ and $a, b, c \in V_I$; if not present, add. Assert mock axioms $R(b, a)$, $R(c, a)$, and $b \neq c$, if not present already. Run reasoner. If O is consistent, then $O \not\models$* InvFun(R)*, so the test fails. (If O is inconsistent, then* InvFun(R) *is true.) Remove mock axioms and individuals, as applicable.* ◀

R is transitive, $Test_{p_t}$ or $Test'_{p_t}$. As with object property chains ($Test_{pc}$), transitivity is only 'interesting' if there are at least two related axioms so that one obtains a non-empty deduction; if the relevant axioms are not asserted, they have to be added. The TBox and ABox tests are as follows:

Test$_{\mathbf{p}_t}$ = *Check $R \in V_{OP}$ and $C, D, E, \in V_C$. If $C, D, E, \notin V_C$, then add the missing class(es) (C, D, and/or E as mock classes). If $C \sqsubseteq \exists R.D$ and $D \sqsubseteq \exists R.E$ are not asserted, then add them to O. Query $\alpha \leftarrow$* SubClassOf(?x ObjectSomeValuesFrom(R E)). *If $C \notin \alpha$, then $O \nvDash$* Trans(R), *so the test fails. Remove mock classes and axioms, as applicable.* ◀

Test$'_{\mathbf{p}_t}$ = *Check $R \in V_{OP}$, $a, b, c \in V_I$. If not, introduce mock a, b, c, $R(a, b)$, and $R(b, c)$, if not present already. Run reasoner. If $R(a, c) \notin \alpha$, then $O \nvDash$* Trans(R), *so the test fails. Remove mock entities.* ◀

R is symmetric, $Test'_{p_s}$, Sym(R), so that with $R(a, b)$, it will infer $R(b, a)$. The test-to-fail—assuming $R \in V_{OP}$—is as follows:

Test$'_{\mathbf{p}_s}$ = *Check $R \in V_{OP}$. Introduce a, b as mock objects ($a, b \in V_I$). Assert mock axiom $R(a, b)$. $\alpha \leftarrow$* ObjectPropertyAssertion(R x? a). *If $b \notin \alpha$, then $O \nvDash$* Sym(R), *so the test fails. Remove mock assertions and individuals.* ◀
Alternatively, one can check in the ODE whether $R(b, a)$ is inferred.

R is asymmetric, $Test'_{p_a}$. This is easier to test with its negation, i.e., assert objects symmetric and distinct, then if O is not inconsistent, then $O \nvDash$ Asym(R):

Test$'_{\mathbf{p}_a}$ = *Check $R \in V_{OP}$. Introduce a, b as mock objects and assert mock axioms $R(a, b)$ and $R(b, a)$. Run reasoner. If O is not inconsistent, then $O \nvDash$* Asym(R), *so the test fails. Remove mock axioms and individuals.* ◀

R is reflexive, $Test'_{p_{rg}}$ or $Test'_{p_{rg}}$. The object property can be either globally reflexive (Ref(R)), or locally ($C \sqsubseteq \exists R$.Self). Global reflexivity is uncommon, but if the modeller does want it, then the following test should be executed:

Test$'_{\mathbf{p}_{rg}}$ = *Check $R \in V_{OP}$. Add mock object a. Run the reasoner. If $R(a, a) \notin O$, then $O \nvDash$* Ref(R), *so the test fails. Remove mock object a.* ◀
Adding Ref(R) will have the test evaluate to true. Local reflexivity amounts to checking whether $O \models C \sqsubseteq \exists R$.Self. This is essentially the same as $Test_{eq}$ but then with Self cf. a named D, so there is a TBox and an ABox TDD test:

Test$_{\mathbf{p}_{rl}}$ = $\alpha \leftarrow$ SubClassOf(?x ObjectSomeValuesFrom(R Self)). *If $C \notin \alpha$, then $O \nvDash C \sqsubseteq \exists R.Self$, so the test fails.* ◀

Test$'_{\mathbf{p}_{rl}}$ = *Check $R \in V_{OP}$. Introduce a as mock objects ($a \in V_I$). Assert mock axiom $C(a)$. $\alpha \leftarrow$* Type(?x C), PropertyValue(a R ?x). *If $a \notin \alpha$, then $O \nvDash C \sqsubseteq \exists R.Self$, so the test fails. Remove $C(a)$ and mock object a.* ◀

R is irreflexive, $Test'_{p_{ir}}$. As with asymmetry, the TDD test exploits the converse:

Test$'_{\mathbf{p}_i}$ = *Check $R \in V_{OP}$, and add $a \in V_I$. Add mock axiom $R(a, a)$. Run reasoner. If O is consistent, then $O \nvDash$* Irr(R)*; test fails. (Else, O is inconsistent, and* Irr(R)) *is true. Remove mock axiom and individual, as applicable.* ◀

This concludes the basic tests. While the logic permits that a class on the left-hand side of the inclusion axiom is an unnamed class, we do not consider this here, as due to the tool design of the most widely used ODE, Protégé, the class on the left-hand side of the inclusion is typically a named class.

4 Evaluation with the Protégé Plugin for TDD

In order to support ontology engineers in performing TDD, we have implemented a Protégé plugin, TDDOnto, which provides a view where the user may specify the set of tests to be run. After their execution, the status of the tests is displayed. One also can add a selected axiom to the ontology (and re-run the test).

The aim of the evaluation is to answer *Which TDD approach—queries or mock objects—is better?*, as performance is likely to affect user opinion of TDD. To answer this question, we downloaded the TONES ontologies from OntoHub [`https://ontohub.org/repositories`], of which 67 could be used (those omitted were either in OBO format or had datatypes incompatible with the reasoner). The ontologies were divided into 4 groups, based on the number of axioms: up to 100 (n=20), 100-1000 axioms (n=35), 1000-10,000 axioms (n=10), and over 10,000 (n=2) to measure effect of ontology size. The tests were generated randomly, using the ontology's vocabulary, and each test kind was repeated 3 times to obtain more reliable results as follows. For each axiom kind of the basic form (with C and D as primitive concepts) there is a fixed number of "slots" that can be replaced with URIs. For each test, these slots were randomly filled from the set of URIs existing in the ontology taking into account whether an URI represents a class or a property. The tested axioms with the result of each test are published in the online material. The test machine was a Mac Book Air with 1.3 GHz Intel Core i5 CPU and 4 GB RAM. The OWL reasoner was HermiT 1.3.8, which is the same that is built-in into OWL-BGP to ensure fair comparison.

The first observation during our experiments was that not all the features of OWL 2 are covered by OWL-BGP, in particular the RBox axioms (e.g., `subPropertyOf` and property characteristics). Therefore, we only present the comparative results of the tests that could be run in both settings: ABox tests and TBox tests with use of the SPARQL-OWL query answering technology implemented in the OWL-BGP tool.

The performance results per group of ontologies are presented in Fig. 1. Each box plot has the median m (horizontal line); the first and third quartile (bottom and top line of the box); the lowest value above $m - 1.5 \cdot IQR$ (horizontal line below the box), and the highest value below $m + 1.5 \cdot IQR$ (horizontal line above the box), where IQR (interquartile range) is represented with the height of the box; outliers are points above and below of the short lines. It is evident that TBox (SPARQL-OWL) tests are generally faster than the ABox ones, and these differences are larger in the sets of larger ontologies. A comparison was done also between two alternative technologies for executing a TBox test—based on SPARQL-OWL and based on OWL API with the reasoner—showing even better performance of the TBox based TDD tests versus ABox based ones (results available in the online material). Before running any test on an ontology, we also measured ontology classification time, which is also included in Fig. 1: it is higher on average in comparison to the times of running the test. Performance by TDD test type and the kind of axiom is shown in Fig. 2, showing the better general performance of the TBox approach in more detail, except for disjointness.

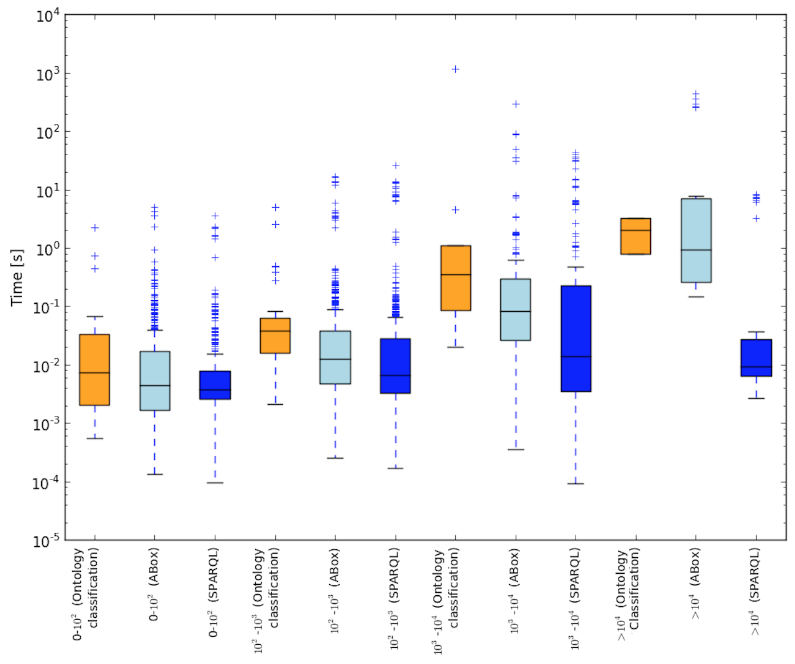

Fig. 1. Performance times by ontology size (four groups, with lower and upper number of the axioms of the ontologies in that group), and classification and test type for each.

5 Discussion

The current alternative to TDD tests is browsing the ontology for the axiom. This is problematic, for then one does not know the implications it is responsible for, it results in cognitive overload that hampers ontology development, and one easily overlooks something. Instead, TDD can manage this in one fell swoop. In addition, the TDD tests also facilitate regression testing.

On specifying and implementing a TDD tool TBox tests can be implemented in different ways; e.g., in some instances, one could use the DL query tab in Protégé; e.g., T_{cs}'s as: `D` and *select* `Sub classes`, without the hassle of unnamed classes (complex class expressions) on the right-hand-side of the inclusion axiom (not supported by BGP [15]). However, it lacks functionality for object property tests (as did all others, it appeared during evaluation); one still can test the sequence 'manually' and check the classification results, though.

The core technological consideration, however, is the technique to obtain the answer of a TDD test: SPARQL SELECT-queries, SPARQL-OWL's BGP (with SPARQL engine and HermiT), or SPARQL-DL with ASK queries and the OWL API. Neither could do all TDD tests in their current version. Regarding performance, the difference between the ABox and TBox tests are explainable—the former always modifies the ontology, so requires an extra classification step—though less so for disjointness or the difference being larger (subsumption, equiv-

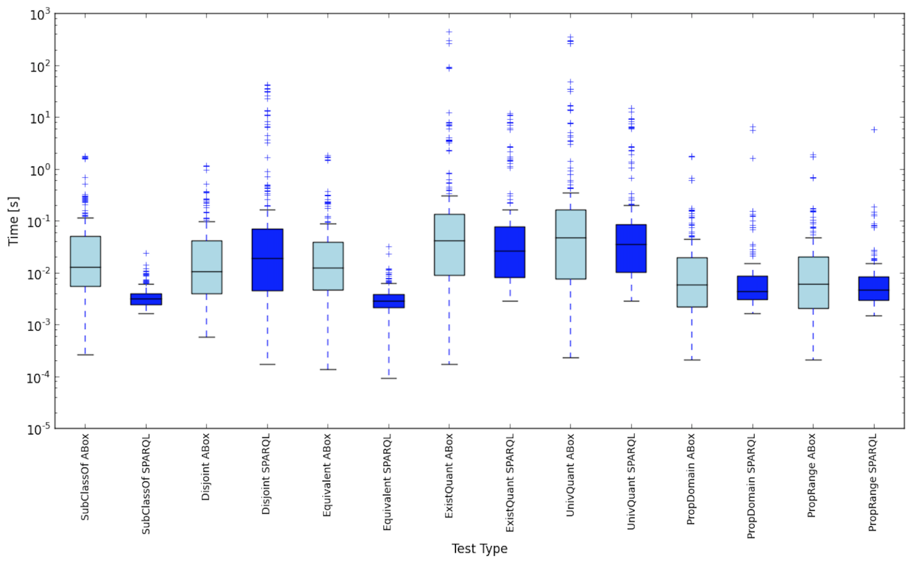

Fig. 2. Test computation times per test type and per the kind of the tested axiom.

alence) or smaller (queries with quantifiers). Overall performance is likely to vary also by reasoner [19], and, as observed, by ontology size. This is a topic of further investigation.

A related issue is the maturity of the tools. Several ontologies had datatype errors, and there were the aforementioned RBox tests limitations. Therefore, we tested only what could be done with current technologies (the scope is TDD evaluation, not extending other tools), and infer tendencies from that so as to have an experimentally motivated basis for deciding which technique likely will have the best chance of success, hence, is the best candidate for extending the corresponding tool. This means using TBox TDD tests, where possible.

A step toward a TDD ontology engineering methodology A methodology is a structured collection of methods and techniques, processes, people having roles possibly in teams, and quality measures and standards across the process (see, e.g., [4]). A foundational step in the direction of a TDD ontology development methodology that indicates where and how it differs from the typical waterfall, iterative, or lifecycle-based methodologies is summarised in Fig. 3, adapting the software development TDD procedure. One can refine these steps, such as managing the deductions following from the ontology update and how to handle an inconsistency or undesirable deduction due to contradictory CQs. Refactoring could include, e.g., removing an explicitly declared axiom from a subclass once it is asserted for its superclass. These details are left for future work. Once implemented, a comparison of methodologies is also to be carried out.

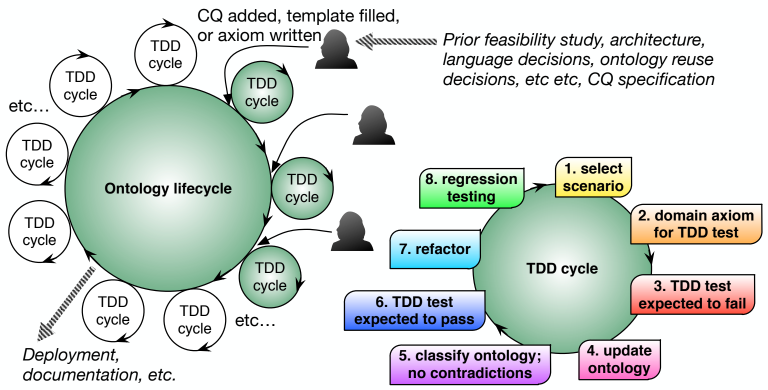
CQ added, template filled,
or axiom written
Prior feasibility study, architecture, language decisions, ontology reuse decisions, etc etc, CQ specification
TDD cycle
etc…
Ontology lifecycle
Deployment, documentation, etc.
1. select scenario
2. domain axiom for TDD test
3. TDD test expected to fail
4. update ontology
5. classify ontology; no contradictions
6. TDD test expected to pass
7. refactor
8. regression testing
TDD cycle

Fig. 3. Sketch of a possible ontology lifecycle that focuses on TDD, and the typical, default, sequence of steps of the TDD procedure summarised in key terms.

6 Conclusions

This paper introduced 36 tests for *Test-Driven Development* of ontologies, specifying what has to be tested, and how. Tests were specified both at the TBox-level with queries and for ABox individuals, using mock entities. The implementation of the main tests demonstrated that the TBox test approach performs better, which is more pronounced with larger ontologies. A high-level 8-step process for TDD ontology engineering was proposed.

Future work pertains to extending tools to also implement the remaining tests, elaborate on the methodology, and conduct use-case evaluations.

Acknowledgments This research has been supported by the National Science Centre, Poland, within the grant number 2014/13/D/ST6/02076.

References

1. Auer, S.: The RapidOWL methodology–towards Agile knowledge engineering. In: Proc. of WETICE'06. pp. 352–357. IEEE Computer Society (June 2006)
2. Beck, K.: Test-Driven Development: by example. Addison-Wesley, Boston, MA (2004)
3. Blomqvist, E., Sepour, A.S., Presutti, V.: Ontology testing – methodology and tool. In: Proc. of EKAW'12. LNAI, vol. 7603, pp. 216–226. Springer (2012)
4. Cockburn, A.: Selecting a project's methodology. IEEE Softw. 17(4), 64–71 (2000)
5. Ferré, S., Rudolph, S.: Advocatus diaboli – exploratory enrichment of ontologies with negative constraints. In: Proc. of EKAW'12. LNAI, vol. 7603, pp. 42–56. Springer (2012), Oct 8-12, Galway, Ireland
6. Gangemi, A., Presutti, V.: Ontology design patterns. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies, pp. 221–243. Springer Verlag (2009)
7. Garca-Ramos, S., Otero, A., Fernández-López, M.: OntologyTest: A tool to evaluate ontologies through tests defined by the user. In: Proc. of IWANN 2009 Workshops, Part II. LNCS, vol. 5518, pp. 91–98. Springer (2009)

8. Garcia, A., O'Neill, K., Garcia, L.J., Lord, P., Stevens, R., Corcho, O., Gibson, F.: Developing ontologies within decentralized settings. In: Chen, H., et al. (eds.) Semantic e-Science. Annals of Information Systems 11, pp. 99–139. Springer (2010)
9. Gennari, J.H., et al.: The evolution of Protégé: an environment for knowledge-based systems development. Int. J. of Hum.-Comp. St. 58(1), 89–123 (2003)
10. Ghidini, C., Kump, B., Lindstaedt, S., Mabhub, N., Pammer, V., Rospocher, M., Serafini, L.: Moki: The enterprise modelling wiki. In: Proc, of ESWC'09 (2009), Heraklion, Greece, 2009 (demo)
11. Janzen, D.S.: Software architecture improvement through test-driven development. In: Companion to ACM SIGPLAN'05. pp. 240–241. ACM Proceedings (2005)
12. Keet, C.M., Ławrynowicz, A.: Test-driven development of ontologies (extended version). Tech. Rep. 1512.06211 (Dec 2015), arxiv.org `http://arxiv.org/abs/1512.06211`
13. Keet, C.M., Khan, M.T., Ghidini, C.: Ontology authoring with FORZA. In: Proc. of CIKM'13. pp. 569–578. ACM proceedings (2013)
14. Kim, T., Park, C., Wu, C.: Mock object models for test driven development. In: Proc. of SERA06. IEEE Computer Society (2006)
15. Kollia, I., Glimm, B., Horrocks, I.: SPARQL Query Answering over OWL Ontologies. In: Proc, of ESWC'11. LNCS, vol. 6643, pp. 382–396. Springer (2011)
16. Kontokostas, D., Westphal, P., Auer, S., Hellmann, S., Lehmann, J., Cornelissen, R., Zaveri, A.: Test-driven evaluation of linked data quality. In: Proc. of WWW'14. pp. 747–758. ACM proceedings (2014)
17. Kumar, S., Bansal, S.: Comparative study of test driven development with traditional techniques. Int. J. Soft Comp. & Eng. 3(1), 352–360 (2013)
18. Mackinnon, T., Freeman, S., Craig, P.: Extreme Programming Examined, chap. Endo-testing: unit testing with mock objects, pp. 287–301. Addison-Wesley, Boston, MA (2001)
19. Parsia, B., Matentzoglu, N., Goncalves, R., Glimm, B., Steigmiller, A.: The OWL Reasoner Evaluation (ORE) 2015 competition report. In: Proc. of SSWS'15. CEUR-WS, vol. 1457 (2015), bethlehem, USA, Oct 11, 2015.
20. Paschke, A., Schaefermeier, R.: Aspect OntoMaven - aspect-oriented ontology development and configuration with OntoMaven. Tech. Rep. 1507.00212v1, Free University of Berlin (July 2015), `http://arxiv.org/abs/1507.00212`
21. Presutti, V., Daga, E., et al.: extreme design with content ontology design patterns. In: Proc. of WS on OP'09. CEUR-WS, vol. 516, pp. 83–97 (2009)
22. Presutti, V., et al.: A library of ontology design patterns: reusable solutions for collaborative design of networked ontologies. NeOn deliverable D2.5.1, NeOn Project, ISTC-CNR (2008)
23. Ren, Y., Parvizi, A., Mellish, C., Pan, J.Z., van Deemter, K., Stevens, R.: Towards competency question-driven ontology authoring. In: Proc. of ESWC'14. LNCS, vol. 8465, p. 752767. Springer (2014)
24. Shrivastava, D.P., Jain, R.: Metrics for test case design in test driven development. Int. J. of Comp. Th. & Eng. 2(6), 952–956 (2010)
25. Suárez-Figueroa, M.C., et al.: NeOn methodology for building contextualized ontology networks. NeOn Deliverable D5.4.1, NeOn Project (2008)
26. Tort, A., Olivé, A., Sancho, M.R.: An approach to test-driven development of conceptual schemas. Data & Knowledge Engineering 70, 1088–1111 (2011)
27. Vrandečić, D., Gangemi, A.: Unit tests for ontologies. In: OTM workshops 2006. LNCS, vol. 4278, pp. 1012–1020. Springer (2006)
28. Warrender, J.D., Lord, P.: How, What and Why to test an ontology. Technical Report 1505.04112, Newcastle University (2015), http://arxiv.org/abs/1505.04112