# Toward Test-Driven Development for Ontologies

C. Maria Keet[1]

Department of Computer Science, University of Cape Town, South Africa
mkeet@cs.uct.ac.za

*OWLED-ORE'16, 20 November, 2016*

[1]joint work with Agnieszka Ławrynowicz, Poznan University of Technology, Poland, and Kieren Davies, UCT

# Outline

# Outline

# Introduction

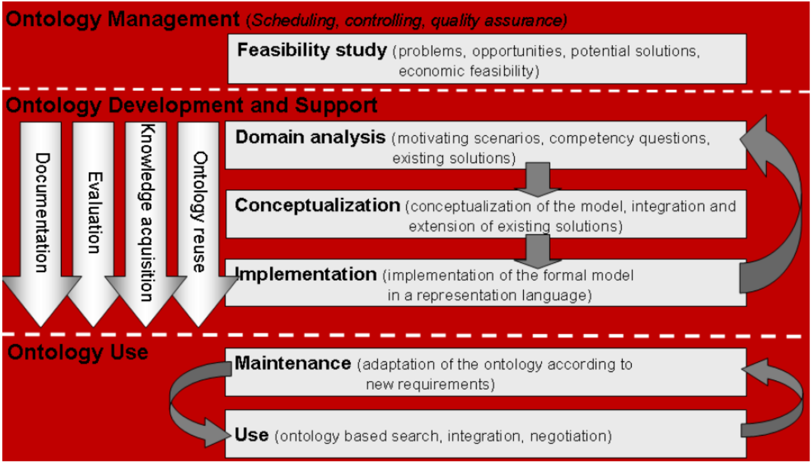
- Ontologies
  - For their own sake
  - For communication
  - Used for many different ontology-driven information systems (database integration and linking, recommender systems, NLP, textbook annotation and search, question generation, Q&A systems, etc.)

# Introduction

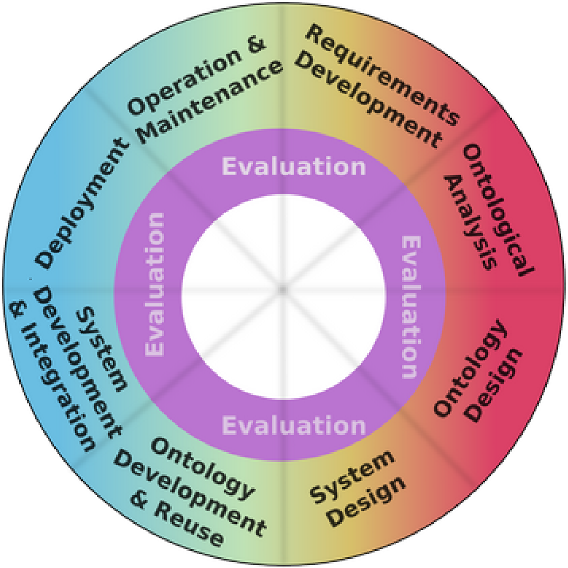
- Ontologies
  - For their own sake
  - For communication
  - Used for many different ontology-driven information systems (database integration and linking, recommender systems, NLP, textbook annotation and search, question generation, Q&A systems, etc.)

⇒ Someone has to build them, *somehow*

# Typical stages of macro-level methodologies

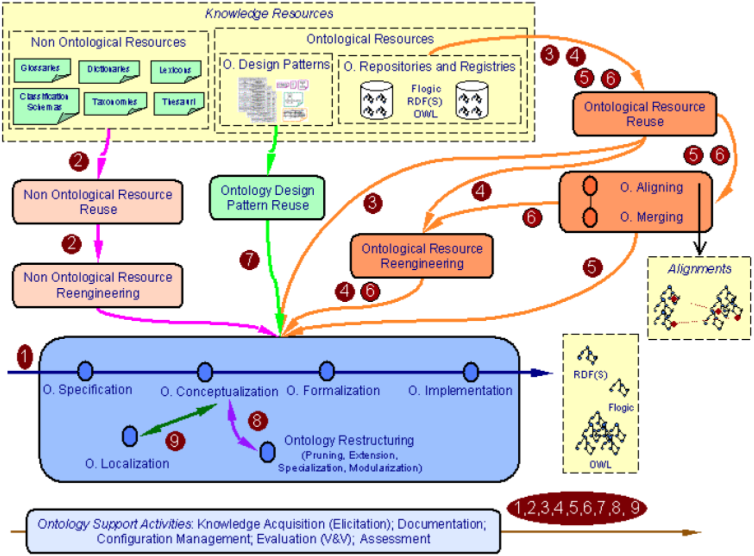



*(Source: Simperl et al., 2010)*

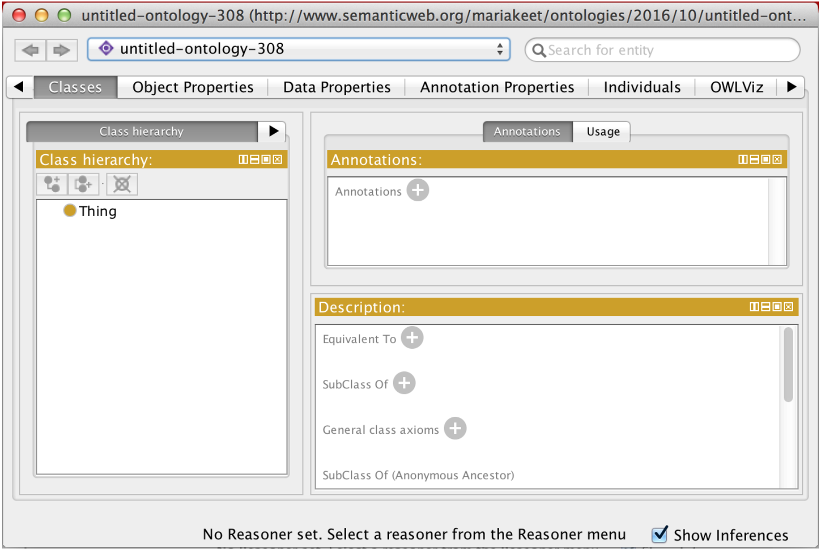



*Ontology Summit 2013's lifecycle model* (`http://ontolog.cim3.net/cgi-bin/wiki.pl?OntologySummit2013_Communique`)

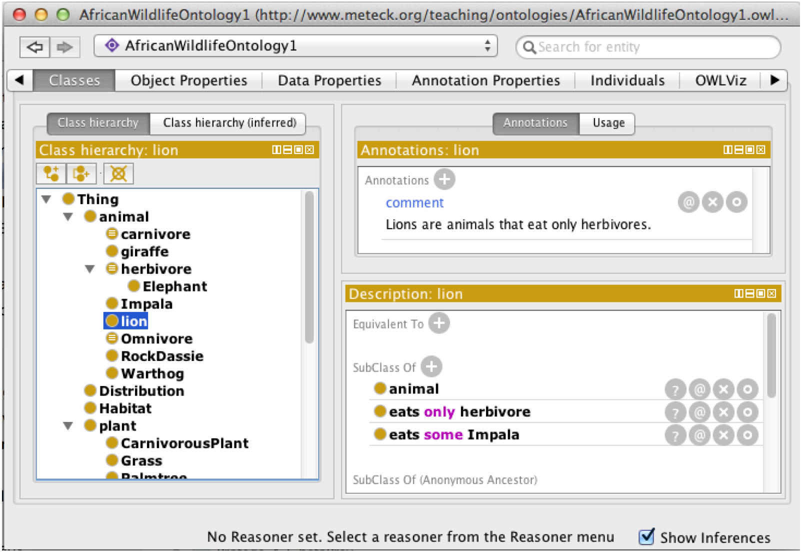
# Scenarios for building Ontology Networks (NEON methodology)



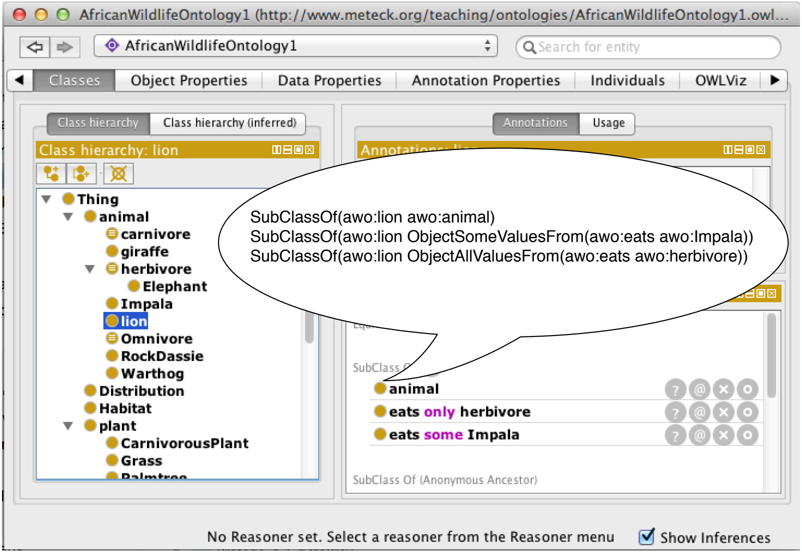
# And then you open an ontology editor...



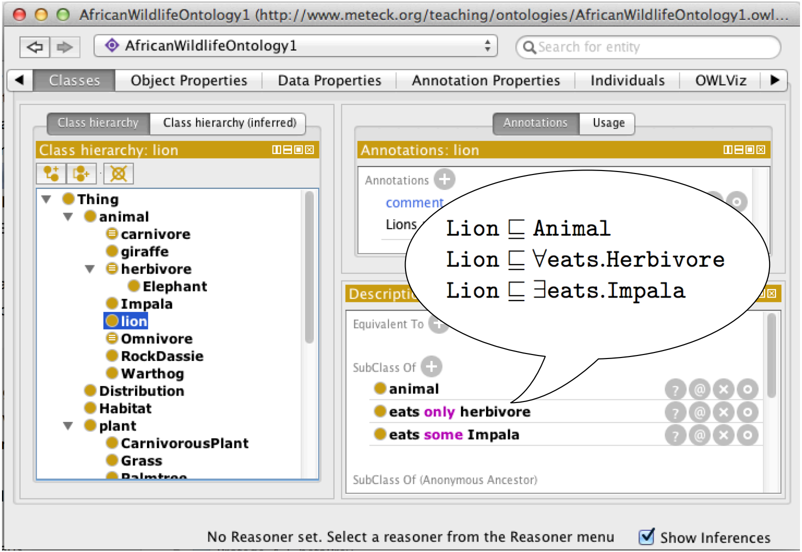
# Or if you have something to start with:



# Behind the facade



# And behind that serialisation



# Ontology development at the ‘micro-level’ level (cf. macro)

- We need to get those axioms into the ontology

# Ontology development at the 'micro-level' level (cf. macro)

- We need to get those axioms into the ontology
- The actual modelling, or *ontology authoring*, using micro-level guidelines and tools
  - Methods, such as reverse engineering and text mining to start, OntoClean and OntoPartS to improve an ontology's quality
  - Parameters that affect ontology development, such as purpose, starting/legacy material, language
  - Tools to model, to reason, to debug, to integrate, to link to data

# Ontology authoring

- Ontology authoring: on adding axioms to the Knowledge base

  Q1 "Does my ontology have axiom X?"
  - where X is, e.g., *all giraffes eat some twigs*
  - i.e., $Giraffe \sqsubseteq \exists eat.Twig$

- Current approaches:
  - For Q1: browsing, searching the *asserted* knowledge

# Ontology authoring

- Ontology authoring: on adding axioms to the Knowledge base
  - Q1 "Does my ontology have axiom X?"
    - where X is, e.g., *all giraffes eat some twigs*
    - i.e., $Giraffe \sqsubseteq \exists eat.Twig$
  - Q2 "Will it still be consistent/class satisfiable if I add X?"
    - add, and try and see what the reasoner says about it
- Current approaches:
  - For Q1: browsing, searching the *asserted* knowledge
  - For Q2: essentially a *test-last* approach

# Ontology authoring

- Ontology authoring: on adding axioms to the Knowledge base

  Q1 "Does my ontology have axiom X?"
  - where X is, e.g., *all giraffes eat some twigs*
  - i.e., $Giraffe \sqsubseteq \exists eat.Twig$

  Q2 "Will it still be consistent/class satisfiable if I add X?"
  - add, and try and see what the reasoner says about it
- Current approaches:
  - For Q1: browsing, searching the *asserted* knowledge
  - For Q2: essentially a *test-last* approach
- Cumbersome and time-consuming with larger ontologies
- Missing: a systematic testbed to do this in a methodical fashion
- It would need to relate to those macro-level processes

# Addressing these issues

$\Rightarrow$ Reuse software engineering's notion of **Test-Driven Development**, based on **test-first**

# (Recap) TDD in software development

- Methodology where one writes new code only if an automated test has failed [Beck(2004)].
- TDD permeates the whole development process
- TDD is a *test-first* approach rather than *test-last* (design, code, test) of unit tests
- More focussed, improves communication, improves understanding of required software behaviour, reduces design complexity [Kumar and Bansal(2013)]
- TDD produced code passes more externally defined tests—i.e, better software quality—and less time spent on debugging [Janzen(2005)]

# Several scenarios of TDD usage in ontology authoring

I. CQ-driven TDD Specify CQ, translate it into one or more axioms, which are the input of the relevant TDD test(s)

# Several scenarios of TDD usage in ontology authoring

I. CQ-driven TDD Specify CQ, translate it into one or more axioms, which are the input of the relevant TDD test(s)

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

# Several scenarios of TDD usage in ontology authoring

I. CQ-driven TDD Specify CQ, translate it into one or more axioms, which are the input of the relevant TDD test(s)

II-a. Ontology authoring-driven TDD - the knowledge engineer who 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 uses a template or "logical macro" ODP [Presutti et al.(2008)], which map onto generic tests; e.g.:

- the all-some template, i.e., an axiom of the form $C \sqsubseteq \exists R.D$
- instantiate with relevant domain entities; e.g., $Professor \sqsubseteq \exists teaches.Course$
- the TDD test for the $C \sqsubseteq \exists R.D$ type of axiom is then run automatically

behind the usability interface, what gets sent to the TDD system is that axiom

# To realise TDD for ontology authoring, one can ask:

Q1: What does TDD mean for 'ontology testing'?

Q2: Do *mock objects* for 'incomplete' parts make sense for ontologies?

Q3: What would be an efficient way to realise the testing?

Q4: In what way and where (if at all) can this be integrated as a methodological step in existing ontology engineering methodologies?

# TDD in conceptual modelling [Tort et al.(2011)]

- Applied to UML class diagrams
- Test specification in OCL
- Each language feature has its own test specification involves creating the objects that should, or ought not to, instantiate the UML classes and associations
- Evaluation: (a.o.) more time was spent on modelling to fix errors than on writing the test cases

# Tests in ontology engineering

- Early explorative work borrowing notion of testing [Vrandečić and Gangemi(2006)]—no framework, testbed
- CQs: patterns [Ren et al.(2014)], formalise into SPARQL queries—what, not how
- Instance-oriented approaches [Garca-Ramos et al.(2009), Kontokostas et al.(2014)], eXtreme Design NeON plugin, ODP rapid design [Blomqvist et al.(2012), Presutti et al.(2009)], RapidOWL [Auer(2006)]
- Tests for particular types of axioms:
  - disjointness [Ferré and Rudolph(2012)]
  - adding part-whole relation based domain and range constraints [Keet et al.(2013)]

# Tests in ontology engineering

- Tawny-Owl's subsumption tests [Warrender and Lord(2015)]. Tests tailored to the actual ontology rather than reusable 'templates' for the tests covering all OWL language features
- SCONE, BDD, focussing on natural language and examples, Cucumber at the back (F. Neuhaus, 2015)
- Methodologies:
  - none of the 9 methodologies reviewed by [Garcia et al.(2010)] are TDD-based
  - The Agile-inspired OntoMaven [Paschke and Schaefermeier(2015)] has OntoMvnTest with 'test cases' only for the usual syntax checking, consistency, and entailment

# Tests in ontology engineering

- Full TDD ontology engineering [Keet and Ławrynowicz(2016), Ławrynowicz and Keet(2016)]
- Idea of unit 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
- No regression testing to check that perhaps an earlier modelled CQ—and thus a passed test—conflicts with a later one

# Outline

# General idea of Test-Driven Development for an ontology

1. Require: domain axiom $x$ of type $X$ is to be added to the ontology; e.g., $x$ may be Professor $\sqsubseteq \exists$teaches.Course, which has pattern $C \sqsubseteq \exists R.D$.
2. Check the vocabulary elements of $x$ are in ontology $O$ (itself a TDD test);
3. Run the TDD test:
   - 3.1 The first execution should fail (check $O \nvDash x$ or not present)
   - 3.2 Update the ontology (add $x$), and
   - 3.3 Run the test again which then should pass (check that $O \models x$) and such that there is no new inconsistency or undesirable deduction
4. Run all previous successful tests, which still have to pass (i.e., regression testing); if not, resolve conflicting knowledge.

# TDD test specification, preliminaries

- 42 test types for $\mathcal{SROIQ}$ [Keet and Ławrynowicz(2016)]
- First iteration:
  - Covering *basic* axioms one can add to the TBox or RBox
  - T-tests: test with terminological knowledge only
    - Use SPARQL-OWL [Kollia et al.(2011)] queries to evaluate the test
    - Use the reasoner directly via OWL API
  - A-tests: test with *mock objects* that must be able to exist

# TDD test specification, preliminaries

- 42 test types for $\mathcal{SROIQ}$ [Keet and Ławrynowicz(2016)]
- First iteration:
  - Covering *basic* axioms one can add to the TBox or RBox
  - T-tests: test with terminological knowledge only
    - Use SPARQL-OWL [Kollia et al.(2011)] queries to evaluate the test
    - Use the reasoner directly via OWL API
  - A-tests: test with *mock objects* that must be able to exist
  - Notation of test in algorithm-style notation

# TDD test specification, preliminaries

- 42 test types for $\mathcal{SROIQ}$ [Keet and Ławrynowicz(2016)]
- First iteration:
  - Covering *basic* axioms one can add to the TBox or RBox
  - T-tests: test with terminological knowledge only
    - Use SPARQL-OWL [Kollia et al.(2011)] queries to evaluate the test
    - Use the reasoner directly via OWL API
  - A-tests: test with *mock objects* that must be able to exist
  - Notation of test in algorithm-style notation
- Second iteration (theory completed):
  - TDD tests for general TBox axioms
  - More feedback (not just 'undefined', 'failed', 'OK')
  - Proofs
  - TDD tests for ABox assertions

# TDD test specification, preliminaries

- 42 test types for $\mathcal{SROIQ}$ [Keet and Ławrynowicz(2016)]
- First iteration:
  - Covering *basic* axioms one can add to the TBox or RBox
  - T-tests: test with terminological knowledge only
    - Use SPARQL-OWL [Kollia et al.(2011)] queries to evaluate the test
    - Use the reasoner directly via OWL API
  - A-tests: test with *mock objects* that must be able to exist
  - Notation of test in algorithm-style notation
- Second iteration (theory completed):
  - TDD tests for general TBox axioms
  - More feedback (not just 'undefined', 'failed', 'OK')
  - Proofs
  - TDD tests for ABox assertions
- Third iteration: dealing with RBox inconsistencies [Keet(2012)], still to implement the algorithm

# Example: a T-test test with SPARQL-OWL

**Require:** Test $T(C \sqsubseteq \exists R.D)$

$\alpha \leftarrow$ SubClassOf(?x ObjectSomeValuesFrom(R D))
**if** $C \notin \alpha$ **then** $\triangleright$ thus, $O \not\models C \sqsubseteq \exists R.D$
    **return** $T(C \sqsubseteq \exists R.D)$ is false
**else**
    **return** $T(C \sqsubseteq \exists R.D)$ is true
**end if**

# Example: A-test with mock objects, using SPARQL-OWL

**Require:** Test $T(C \sqsubseteq \exists R.D)$ ▷ i.e., test $T'_{eq}$

Create a mock object, $a$
Assert $(C \sqcap \neg\exists R.D)(a)$
$ostate \leftarrow$ Run reasoner
**if** $ostate ==$ consistent **then** ▷ thus, then $O \nvDash C \sqsubseteq \exists R.D$
    **return** $\mathrm{T}(C \sqsubseteq \exists R.D)$ is false
**else**
    **return** $\mathrm{T}(C \sqsubseteq \exists R.D)$ is true
**end if**
Delete $(C \sqcap \neg\exists R.D)(a)$ and $a$

Note: using De Morgan in that if the existential quantification were present and had an instance, then $C \sqcap \neg\exists R.D$ should result in an inconsistent ontology, or: in its absence, the ontology is consistent

# Example A-test with mock objects, RBox axiom

**Require:** Test $T(R \sqsubseteq S)$

Check $R, S \in V_{OP}$

Add individuals $a, b$ to the ABox, add $R(a, b)$

Run the reasoner

**if** $O \not\models S(a, b)$ **then** ▷ thus $O \not\models R \sqsubseteq S$

  **return** $T(R \sqsubseteq S)$ is false

**else**

  **return** $T(R \sqsubseteq S)$ is true

**end if**

Delete $R(a, b)$, and individuals $a$ and $b$

# Revisiting the general idea of TDD for an ontology

1. Require: domain axiom $x$ of type $X$ is to be added to the ontology; e.g., $x$ may be Professor $\sqsubseteq$ $\exists$teaches.Course, which has pattern $C \sqsubseteq \exists R.D$.
2. Check the vocabulary elements of $x$ are in ontology $O$ (itself a TDD test);
3. Run the TDD test:
   3.1 The first execution should fail (check $O \not\models x$ or not present)
   3.2 Update the ontology (add $x$), and
   3.3 Run the test again which then should pass (check that $O \models x$) and such that there is no new inconsistency or undesirable deduction
4. Run all previous successful tests, which still have to pass (i.e., regression testing); if not, resolve conflicting knowledge.

# A model for testing–possible test results

- Ontology already inconsistent
- Ontology already incoherent: that is, one or more of its named classes are unsatisfiable.
- Missing entity in axiom: The axiom contains one or more named classes or properties which are not declared in the ontology.
- Axiom causes inconsistency
- Axiom causes incoherence
- Axiom absent: The axiom is not entailed by the ontology, but it could be added without negative consequences.
- Axiom entailed: The axiom is already entailed by the ontology

# Formally

## Definition

Given an ontology $O$ which is consistent and coherent, and an axiom $A$ such that $\Sigma(A) \subseteq \Sigma(O)$, the result of testing $A$ against $O$ is

$$\text{test}_O(A) = \begin{cases} \text{entailed} & \text{if } O \vdash A \\ \text{inconsistent} & \text{if } O \cup A \vdash \bot \\ \text{incoherent} & \text{if } O \cup A \nvdash \bot \\ & \wedge (\exists C \in \Sigma_C(O)) \text{ s.t. } O \cup A \vdash C \sqsubseteq \bot \\ \text{absent} & \text{otherwise} \end{cases}$$

# Generalisation

Note: now $C$ and $D$ can be *any* class expression, not just only a named class

---

**Algorithm 1** Function $\textsc{TestSubclassOf}(C, D)$

---

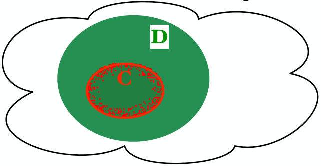
**Input:** $C, D$ class expressions

1: **if** $\textsc{GetInstances}(C \sqcap \neg D) \neq \emptyset$ **then**
2: **return** inconsistent
3: **else if** $\textsc{GetSubclasses}(C \sqcap \neg D) \neq \emptyset$ **then**
4: **return** incoherent
5: **else if** $\textsc{IsSatisfiable}(C \sqcap \neg D) ==$ false **then**
6: **return** entailed
7: **else**
8: **return** absent
9: **end if**

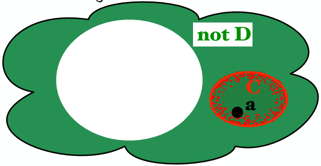
---

# Graphically

We want to test whether this already holds in O.

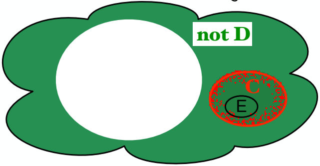



There is an object, **a**, that is a C and not a D...



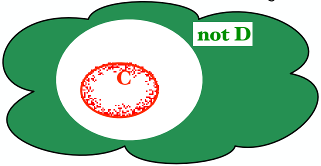

... so O with C is-a D would turn out to be inconsistent.

There is some class E subsumed by C and not a D...




... so C is-a D would cause O to be incoherent.

There cannot be a class E subsumed by C and not a D...




... so C is-a D is entailed already in O.

What remains: C is-a D is **absent**.

# Outline

# Design considerations and issues

- Which technology to use?
- DL Query tab possible
  - to cumbersome, not all tests possible
- SPARQL-OWL's implementation OWL-BGP and its SPARQL SELECT, SPARQL answering engine, and Hermit v1.3.8 [Kollia et al.(2011)]
  - Limited RBox tests (note: does not implement ASK queries)
- SPARQL-DL's implementation with its ASK queries
  - Limited RBox tests
- Use just the OWL API + a DL reasoner

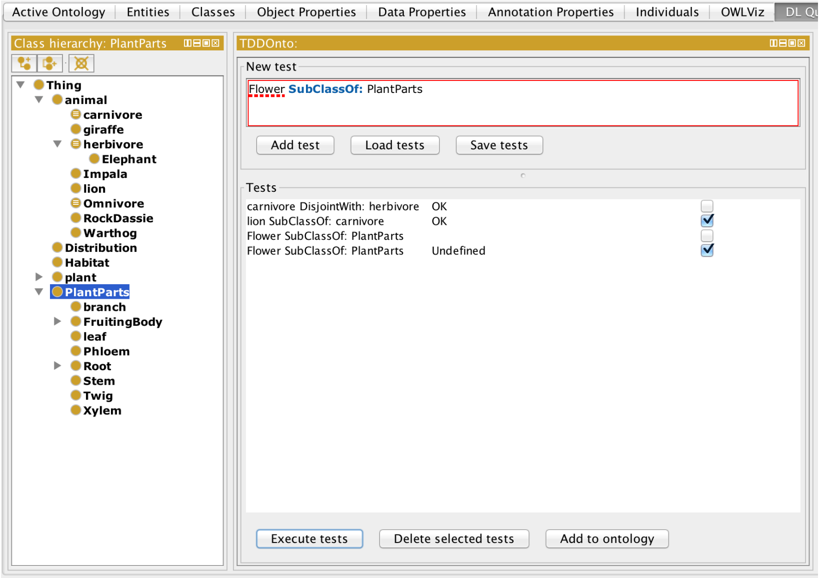
# TDDOnto

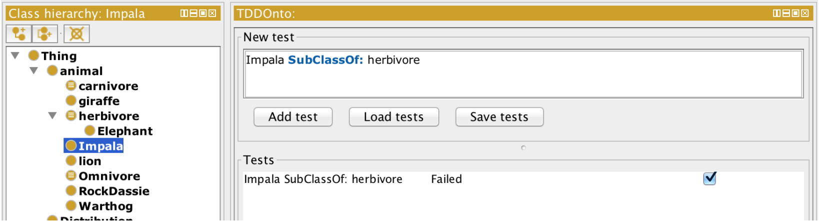
- TDDOnto tool as Protégé plugin
- Manages test specification and execution, ontology update
- ‘wraps’ around the actual execution of the test (SPARQL query, reasoner) for creation/deletion mock entities, the true/false returned

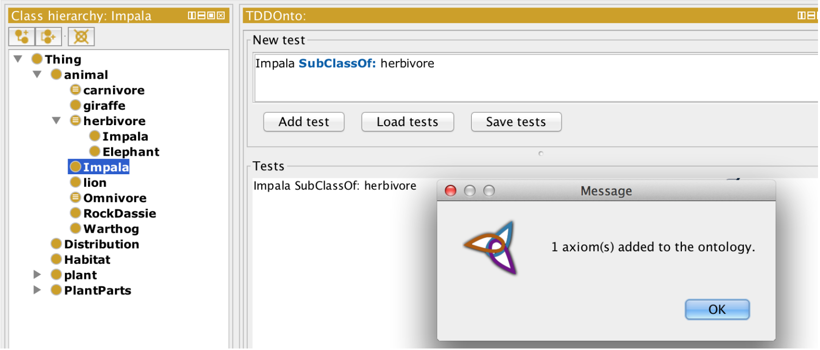
# TDDOnto

- TDDOnto tool as Protégé plugin
- Manages test specification and execution, ontology update
- 'wraps' around the actual execution of the test (SPARQL query, reasoner) for creation/deletion mock entities, the true/false returned
- To make a long story short: the current version of TDDonto uses the reasoner, for it is the fastest of the three options...

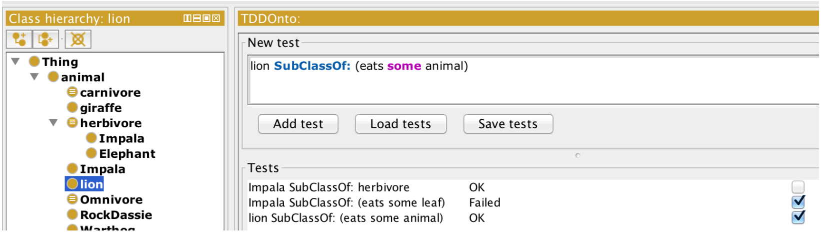
# Screenshots




Class hierarchy: Impala
Thing
animal
carnivore
giraffe
herbivore
Elephant
Impala
lion
Omnivore
RockDassie
Warthog
TDDOnto:
New test
Impala SubClassOf: herbivore
Add test
Load tests
Save tests
Tests
Impala SubClassOf: herbivore
Failed


Class hierarchy: Impala
Thing
animal
carnivore
giraffe
herbivore
Impala
Elephant
Impala
lion
Omnivore
RockDassie
Warthog
Distribution
Habitat
plant
PlantParts
TDDOnto:
New test
Impala SubClassOf: herbivore
Add test
Load tests
Save tests
Tests
Impala SubClassOf: herbivore
Message
1 axiom(s) added to the ontology.
OK

# Screenshots



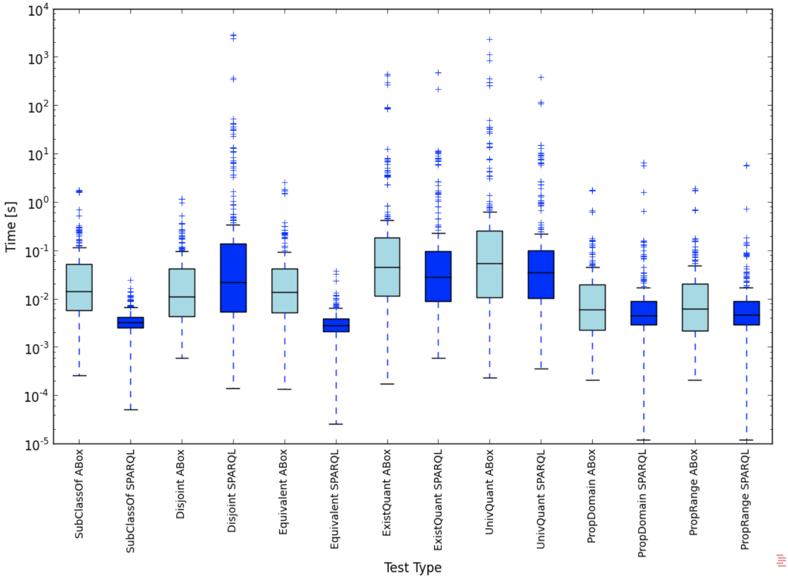
# Evaluation

- Which TDD approach has better performance: T-test with SPARQL queries using OWL-BGP, mock objects with the A-tests, or T-tests with the reasoner using the OWL API?
- Hypotheses:
  - H1: Query-based T-test TDD is faster than A-test mock object-based TDD tests.
  - H2: Classification time of the ontology contributes the most to overall performance (time) of a TDD test.
  - H3: The TDD tests with OWL (1) ontologies are faster than on OWL 2 DL ontologies.

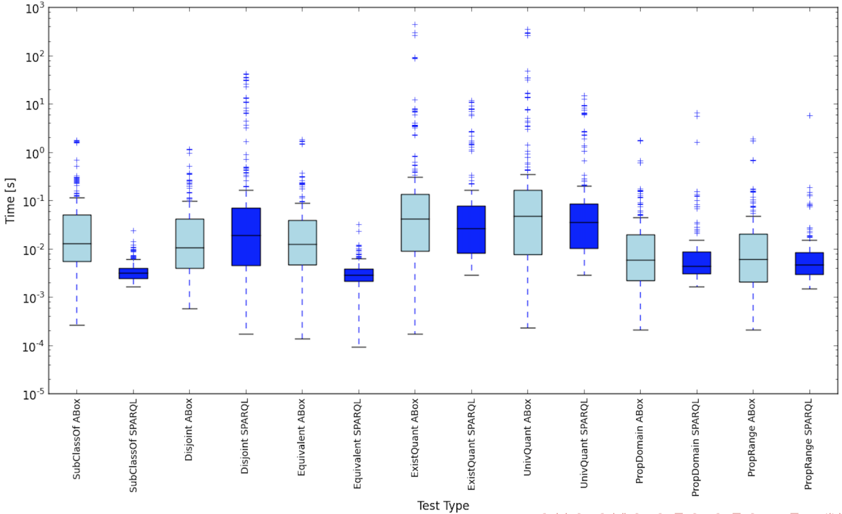
# Evaluation

- Data: OWL ontologies from TONES (via Ontohub), manually collected 20 OWL 2 ontologies. total 82 ontologies
- Group ontologies by size: up to 100 ($n$=20), 100-1000 axioms ($n$=35), 1000-10,000 axioms ($n$=10), over 10,000 ($n$=2)
- OWL-BGP with built-in Hermit v1.3.8, OWL API + reasoner (also Hermit v1.3.8)
- Mac Book Air: 1.3 GHz Intel Core i5 CPU, 4 GB RAM
- Tests: use URIs of the ontology, randomly for the type. Repeated 3 times

# Mock objects (light blue) vs. SPARQL-OWL (dark blue)



# Mock objects vs. SPARQL-OWL, OWL 1 only



# Hypothesis H1

H1: Query-based T-test TDD is faster than A-tests with mock objects.

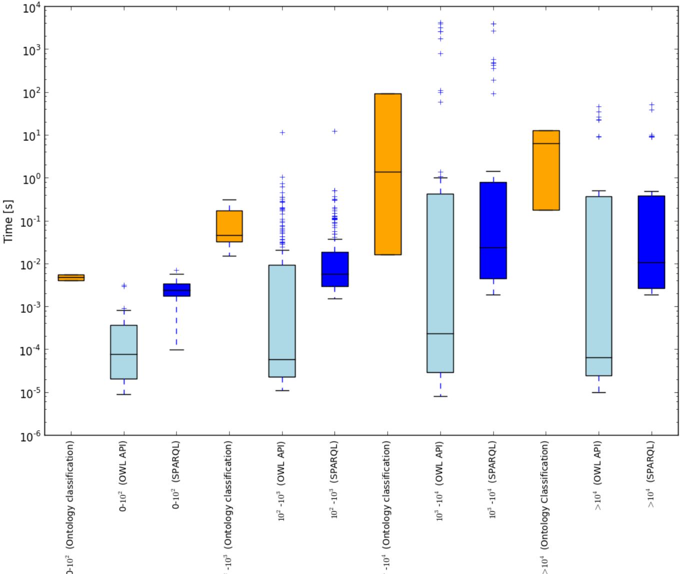
- Avg A-test: 5.191s, sd of 71.491s, and median of 0.014s
- Avg T-test (OWL-BGP): 6.244s, sd 113.605s, and median 0.005s
- t-test with $H1_0$ of identical average scores and the threshold of 5%, with all ontologies:
  - $t$=-0.322 and p=0.748
  - therefore we cannot reject the null hypothesis

# Hypothesis H1

H1: Query-based T-test TDD is faster than A-tests with mock objects.

- Avg A-test: 5.191s, sd of 71.491s, and median of 0.014s
- Avg T-test (OWL-BGP): 6.244s, sd 113.605s, and median 0.005s
- t-test with $H1_0$ of identical average scores and the threshold of 5%, with all ontologies:
  - $t$=-0.322 and p=0.748
  - therefore we cannot reject the null hypothesis
- t-test with $H1_0$, but with OWL 1 ontologies only:
  - $t$=2.959 and p=0.003,
  - therefore we can reject the null hypothesis ⇒ the query-based T-tests are significantly faster than the A-tests with mock objects

# Classification vs TDD T-test, OWL 2 DL, by size

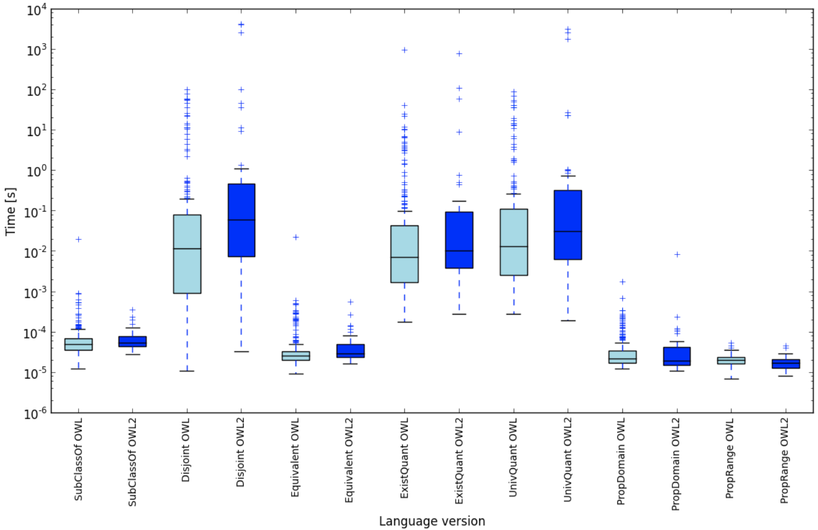


# Hypothesis H2

H2: Classification time of the ontology contributes the most to overall performance (time) of a TDD test.

- A-test: Average classification time 15.990s (sd 128.264s), median 0.040s vs. avg test time 5.191s (sd 71.491s) and median 0.013s
- T-test (OWL-BGP): respectively, avg 15.954s (sd 28.267s) and median 0.040s, vs 6.244s (sd 113.606s) and median 0.005s
- We didn't quite expect that TDD would be faster on average
- Reasons: some outliers, and for repeated querying one does not need to classify each time

# OWL API+Reasoner, OWL vs OWL 2



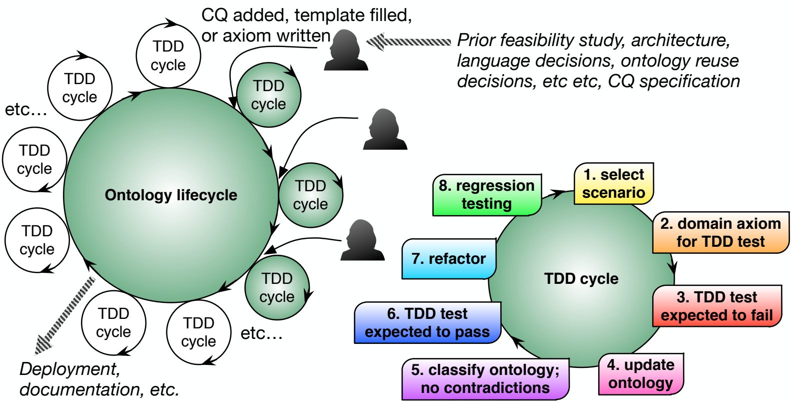
# Hypothesis H3

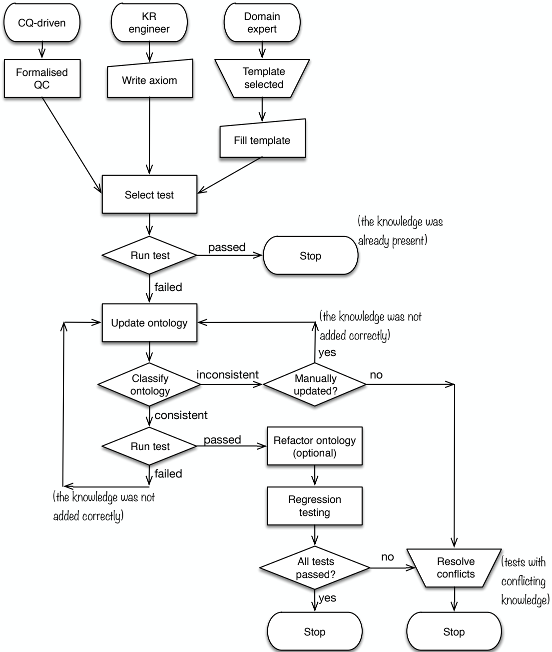
H3: The TDD tests on OWL (1) ontologies are faster than on OWL 2 DL ontologies.

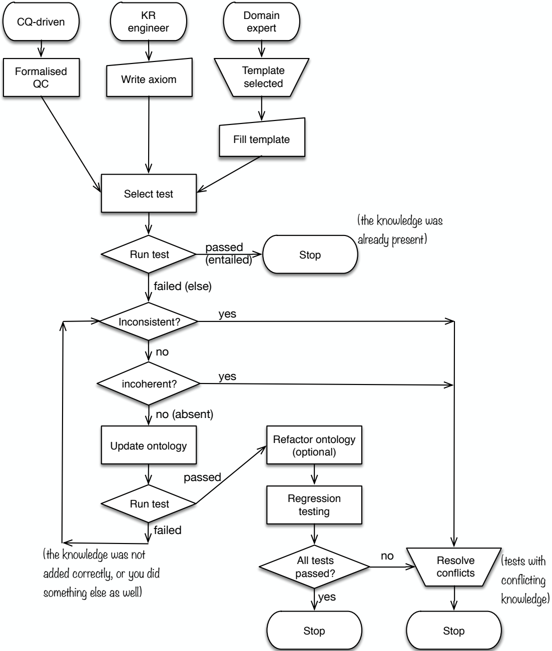
- T-test values are $t$=-7.425 and p=1.309e-13;
- Thus, tests on OWL ontologies were significantly faster
- As expected, based on the theory

# Outline

# Methodology sketch




CQ-driven
KR engineer
Domain expert
Formalised QC
Write axiom
Template selected
Fill template
Select test
(the knowledge was already present)
Run test
passed
Stop
failed
Update ontology
(the knowledge was not added correctly)
yes
Classify ontology
inconsistent
Manually updated?
no
consistent
Run test
passed
Refactor ontology (optional)
failed
(the knowledge was not added correctly)
Regression testing
All tests passed?
no
Resolve conflicts
(tests with conflicting knowledge)
yes
Stop
Stop


CQ-driven
KR engineer
Domain expert
Formalised QC
Write axiom
Template selected
Fill template
Select test
Run test
passed (entailed)
Stop
(the knowledge was already present)
failed (else)
Inconsistent?
yes
no
incoherent?
yes
no (absent)
Update ontology
Refactor ontology (optional)
passed
Run test
failed
Regression testing
(the knowledge was not added correctly, or you did something else as well)
All tests passed?
no
Resolve conflicts
(tests with conflicting knowledge)
yes
Stop
Stop

# Notes

- The picture shows the basic loop only

# Notes

- The picture shows the basic loop only
- What if Step 5 goes wrong (inconsistent/incoherent ontology):
- What if after refactoring (step 7), regression (step 8) fails:

# Notes

- The picture shows the basic loop only
- What if Step 5 goes wrong (inconsistent/incoherent ontology):
  - What is the source of the inconsistency?
  - Was it a previous test, hence, contradicting CQs?
- What if after refactoring (step 7), regression (step 8) fails:

# Notes

- The picture shows the basic loop only
- What if Step 5 goes wrong (inconsistent/incoherent ontology):
  - What is the source of the inconsistency?
  - Was it a previous test, hence, contradicting CQs?
- What if after refactoring (step 7), regression (step 8) fails:
  - Is a previous test obsolete?
  - Error introduced in the refactoring?

# Notes

- The picture shows the basic loop only
- What if Step 5 goes wrong (inconsistent/incoherent ontology):
  - What is the source of the inconsistency?
  - Was it a previous test, hence, contradicting CQs?
- What if after refactoring (step 7), regression (step 8) fails:
  - Is a previous test obsolete?
  - Error introduced in the refactoring?
- What does ‘refactoring’ an ontology mean anyway?
  - (we have some ideas, but too preliminary at this stage)

# Outline

# Conclusions

- First comprehensive specification of TDD for ontology authoring
- Rigorous, formal foundation, with proofs
- Sketch of a revised ontology development methodology
- TDDonto, a Protégé plugin for Test-Driven Development tests
- Performance evaluation:
    - TDD tests outperformed classification reasoning
    - TBox-based test strategy was faster in general than ABox-based (significantly so for OWL 1 ontologies)
    - OWL API+reasoner options for TBox TDD tests had better median performance than SPARQL-OWL (OWL-BGP) TBox TDD tests
    - TDD tests on OWL ontologies are significantly faster overall than on OWL 2 DL ontologies

# References I


S. Auer.
The RapidOWL methodology–towards Agile knowledge engineering.
In *Proc. of WETICE'06*, pages 352–357. IEEE Computer Society, June 2006.
doi: 10.1109/WETICE.2006.67.

Kent Beck.
*Test-Driven Development: by example.*
Addison-Wesley, Boston, MA, 2004.

E Blomqvist, A. S. Sepour, and V. Presutti.
Ontology testing – methodology and tool.
In *Proc. of EKAW'12*, volume 7603 of *LNAI*, pages 216–226. Springer, 2012.

Sebastien Ferré and Sebastian Rudolph.
Advocatus diaboli exploratory enrichment of ontologies with negative constraints.
In *Proc. of EKAW'12*, volume 7603 of *LNAI*, pages 42–56. Springer, 2012.
Oct 8-12, Galway, Ireland.

S. Garca-Ramos, A. Otero, and M Fernández-López.
OntologyTest: A tool to evaluate ontologies through tests defined by the user.
In *Proc. of IWANN 2009 Workshops, Part II*, volume 5518 of *LNCS*, pages 91–98. Springer, 2009.

Alexander Garcia, Kieran O'Neill, Leyla Jael Garcia, Phillip Lord, Robert Stevens, Óscar Corcho, and Frank Gibson.
Developing ontologies within decentralized settings.
In H. Chen et al., editors, *Semantic e-Science. Annals of Information Systems 11*, pages 99–139. Springer, 2010.

# References II


David S. Janzen.
Software architecture improvement through test-driven development.
In *Companion to ACM SIGPLAN'05*, pages 240–241. ACM Proceedings, 2005.

C. M. Keet and A. Ławrynowicz.
Test-driven development of ontologies.
In *13th Extended Semantic Web Conference (ESWC'16)*, LNCS. Springer, 2016.
29 May - 2 June, 2016, Crete, Greece.

C. Maria Keet.
Detecting and revising flaws in OWL object property expressions.
In A. ten Teije et al., editors, *18th International Conference on Knowledge Engineering and Knowledge Management (EKAW'12)*, volume 7603 of *LNAI*, pages 252–266. Springer, 2012.
URL `http://www.meteck.org/files/EKAW12subProsChains.pdf`.
Oct 8-12, Galway, Ireland.

C. Maria Keet, M. Tahir Khan, and Chiara Ghidini.
Ontology authoring with FORZA.
In *Proc. of CIKM'13*, pages 569–578. ACM proceedings, 2013.

Ilianna Kollia, Birte Glimm, and Ian Horrocks.
SPARQL Query Answering over OWL Ontologies.
In *Proc, of ESWC'11*, volume 6643 of *LNCS*, pages 382–396. Springer, 2011.
29 May-2 June, 2011, Heraklion, Crete, Greece.

# References III


D. Kontokostas, P. Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri.
Test-driven evaluation of linked data quality.
In *Proc. of WWW'14*, pages 747–758. ACM proceedings, 2014.

Shaweta Kumar and Sanjeev Bansal.
Comparative study of test driven development with traditional techniques.
*Int. J. Soft Comp. & Eng.*, 3(1):352–360, 2013.

A. Ławrynowicz and C. M. Keet.
The tddonto tool for test-driven development of dl knowledge bases.
In R. Peñaloza and M. Lenzerini, editors, *29th International Workshop on Description Logics (DL'16)*, volume 1577 of *CEUR-WS*, 2016.
22-25 April 2016, Cape Town, South Africa.

Adrian Paschke and Ralph Schaefermeier.
Aspect OntoMaven - aspect-oriented ontology development and configuration with OntoMaven.
Technical Report 1507.00212v1, Free University of Berlin, July 2015.
URL `http://arxiv.org/abs/1507.00212`.

V. Presutti, E Daga, et al.
extreme design with content ontology design patterns.
In *Proc. of WS on OP'09*, volume 516 of *CEUR-WS*, pages 83–97, 2009.

V. Presutti 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.

# References IV


Yuan Ren, Artemis Parvizi, Chris Mellish, Jeff Z. Pan, Kees van Deemter, and Robert Stevens.
Towards competency question-driven ontology authoring.
In *Proc. of ESWC'14*, volume 8465 of *LNCS*, page 752767. Springer, 2014.

Albert Tort, Antoni Olivé, and Maria-Ribera Sancho.
An approach to test-driven development of conceptual schemas.
*Data & Knowledge Engineering*, 70:1088–1111, 2011.

Danny Vrandečić and Aldo Gangemi.
Unit tests for ontologies.
In *OTM workshops 2006*, volume 4278 of *LNCS*, pages 1012–1020. Springer, 2006.

J. D. Warrender and P. Lord.
How, What and Why to test an ontology.
Technical Report 1505.04112, Newcastle University, 2015.
http://arxiv.org/abs/1505.04112.

# Thank you!

More details in DL16 and ESWC16 papers.
TDDonto can be downloaded from
`https://semantic.cs.put.poznan.pl/wiki/aristoteles/doku.php`

# Questions?