# COMPUTING
# SCIENCE

## How, What and Why to test an ontology

Jennifer D. Warrender and Phillip Lord

**TECHNICAL REPORT SERIES**

How, What and Why to test an ontology

J. D. Warrender and P. Lord

**Abstract**

Ontology development relates to software development in that they both involve the production of formal computational knowledge. It is possible, therefore, that some of the techniques used in software engineering could also be used for ontologies; for example, in software engineering testing is a well-established process, and part of many different methodologies.

The application of testing to ontologies, therefore, seems attractive. The Karyotype Ontology is developed using the novel Tawny-OWL library. This provides a fully programmatic environment for ontology development, which includes a complete test harness.

In this paper, we describe how we have used this harness to build an extensive series of tests as well as used a commodity continuous integration system to link testing deeply into our development process; this environment, is applicable to any OWL ontology whether written using Tawny-OWL or not. Moreover, we present a novel analysis of our tests, introducing a new classification of what our different tests are. For each class of test, we describe why we use these tests, also by comparison to software tests. We believe that this systematic comparison between ontology and software development will help us move to a more agile form of ontology development.

# Bibliographical details

WARRENDER, J. D; LORD, P;
How, What and Why to test an ontology
[By] J. D. Warrender and P. Lord
Newcastle upon Tyne: Newcastle University: Computing Science, 2015.

(Newcastle University, Computing Science, Technical Report Series, No. CS-TR-1466)

**Added entries**

NEWCASTLE UNIVERSITY
Computing Science. Technical Report Series.  CS-TR-1466

**Abstract**

Ontology development relates to software development in that they both involve the production of formal computational knowledge. It is possible, therefore, that some of the techniques used in software engineering could also be used for ontologies; for example, in software engineering testing is a well-established process, and part of many different methodologies.

The application of testing to ontologies, therefore, seems attractive. The Karyotype Ontology is developed using the novel Tawny-OWL library. This provides a fully programmatic environment for ontology development, which includes a complete test harness.

In this paper, we describe how we have used this harness to build an extensive series of tests as well as used a commodity continuous integration system to link testing deeply into our development process; this environment, is applicable to any OWL ontology whether written using Tawny-OWL or not. Moreover, we present a novel analysis of our tests, introducing a new classification of what our different tests are. For each class of test, we describe why we use these tests, also by comparison to software tests. We believe that this systematic comparison between ontology and software development will help us move to a more agile form of ontology development.

**About the authors**

Jennifer P. Warrender is a PHD student in the School of Computing Science at Newcastle University. Her research is focused on ways in which to consistently represent knowledge in ontologies.

Phillip Lord is a lecturer at the School of Computing Science, Newcastle University. His research covers a variety of different areas; mostly it focuses on the use of ontological technology in biology, or more generally mechanisms for presenting and publishing scientific information.

**Suggested keywords**

KARYOTYPE ONTOLOGY
TAWNY-OWL
ONTOLOGY DEVELOPMENT
TAWNY-KARYOTYPE
CONTINUOUS INTEGRATION
INTEGRATION TESTING
BIO-ONTOLOGY

# How, What and Why to test an ontology

Jennifer D. Warrender and Phillip Lord*

School of Computing Science, Newcastle University, Newcastle-upon-Tyne, UK

## ABSTRACT

Ontology development relates to software development in that they both involve the production of formal computational knowledge. It is possible, therefore, that some of the techniques used in software engineering could also be used for ontologies; for example, in software engineering *testing* is a well-established process, and part of many different methodologies.

The application of testing to ontologies, therefore, seems attractive. The Karyotype Ontology is developed using the novel Tawny-OWL library. This provides a fully programmatic environment for ontology development, which includes a complete test harness.

In this paper, we describe *how* we have used this harness to build an extensive series of tests as well as used a commodity continuous integration system to link testing deeply into our development process; this environment, is applicable to any OWL ontology whether written using Tawny-OWL or not. Moreover, we present a novel analysis of our tests, introducing a new classification of *what* our different tests are. For each class of test, we describe *why* we use these tests, also by comparison to software tests. We believe that this systematic comparison between ontology and software development will help us move to a more agile form of ontology development.

## 1 INTRODUCTION

Karyotypes have a long history in biology, being used to assess chromosome rearrangement in many different organisms. In humans, this knowledge is used diagnostically for many genetic abnormalities. The use of cytogenetic analysis is cheap, non-invasive and simple, so remains useful. The representation of karyotypes though, is not simple. The specification in humans is a hundred-page book, with no computational definition (Shaffer *et al.*, 2012). The representation is a string with no formal grammar which is difficult to manage computationally.

To address this problem, we have developed the Karyotype Ontology, which provides a fully computational representation in the form of an OWL ontology (Warrender and Lord, 2013b).

Ontology development bears many similarities to software development; both involve taking complex knowledge and producing a computational amenable representation of that knowledge. For the Karyotype Ontology, we have extended this similarity further. It has been developed using Tawny-OWL (Lord, 2013), a fully programmatic ontology development environment.

Tawny-OWL is a library, implemented in Clojure which is an implementation of the Lisp programming language, running on the Java Virtual Machine. It uses the OWL API (Horridge and Bechhofer, 2011), which is the same library underlying Protégé 4 and upward. It allows constructions of ontologies programmatically, so rather than adding classes and properties individually, a large number of entities can be generated according to patterns defined in Clojure (Warrender and Lord, 2013a). The Karyotype Ontology defines a number of these patterns, which are used to generate a large number of classes – in one case, a single pattern is used to generate 1248 classes. In essence, Tawny-OWL allows us to recast ontology development as a form of software development, through the use of functional abstraction.

Tawny-OWL also allows us to use other parts of the software engineering process; more specifically testing in order to apply quality control. Historically, ontology testing has been achieved through the use of DL queries, SPARQL queries and reasoners to ensure the internal consistency and satisfiability of an ontology. These have been encapsulated in bespoke tools such as the efovalidator[1] that can only be used for the validation and unit testing of EFO (Malone *et al.*, 2010). More recently, ontology testing has evolved by incorporating the use of continuous integration systems, as it enables tests to be run frequently and in a clean environment (Mungall *et al.*, 2012). Here, the authors support integration testing while providing releases of OBO ontologies that are internally consist as well as consistent with external ontologies and information sources. This tool[2] was initially used to help with the development and maintenance of GO (Ashburner *et al.*, 2000) and Human Phenotype Ontology (Köhler *et al.*, 2014), but is not specific to those domains.

While testing and continuous integration are not novel in the ontological community, Tawny-OWL has the advantage of not requiring any specialist installation. Clojure comes with a full test harness, a build tool for running the tests and is supported by various continuous integration testing services. In addition, we can use Tawny-OWL to interact with external ontologies such as GO and OBI. As tests are simple to use within Tawny-OWL, this has meant that we have produced a very large test library for the Karyotype Ontology (currently containing 3088 tests).

In this paper, we describe how we have developed this test suite, including our use of a spreadsheet to define tests rapidly. We analyse the different kinds of test and present a novel test classification, describing the purpose of each form of test. Taken together, this work represents a systematic attempt to re-purpose software engineering testing for use within ontology development.

## 2 THE KARYOTYPE ONTOLOGY

A karyotype describes the number of chromosomes and any alterations from the normal. These are visible under the light microscope, and when stained have a characteristic banding pattern

*To whom correspondence should be addressed: phillip.lord@newcastle.ac.uk

---

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

[2] https://github.com/owlcollab/owltools/tree/master/OWLTools-Oort

which can be used to distinguish between different chromosomes and the positions on these chromosomes.

Alterations are described by their type, such as inversions, deletions or duplications and by their location, specified by a chromosome number and band number. So, `46,XY,t(1;3)(p22;q13.1)` describes a male with a translocation from chromosome `1p22` to chromosome `3q13.1`. To describe a karyotype adequately, a unique class needs to be defined for each band, of which there are 1224.

The Karyotype Ontology is developed by specifying the bands in a literal Clojure data structure, and then using this to generate the appropriate classes. For example, the following data structure,:

```
[
 "p10"
 ["p11" "p11.1" "p11.2"]
]
```

describes part of Chromosome 1 which has bands `p10`, and `p11` in turn has two sub-bands `p11.1` and `p11.2`. The representation was chosen for ease of legibility/typing. We then use Tawny-OWL to programmatically expand this data structure into 4 classes, coerced into a tree, and a set of relationships using code specific to the Karyotype Ontology.

## 3 THE KARYOTYPE TEST HARNESS

Of course, programmers have always tested their code, but *test-driven* development methodologies emphasise the importance of writing large numbers of test. A key feature has been the development of a *tests harness*. This provides a method for defining tests separate from the main body of code and a mechanism for running all of the tests regularly in batch. This enables a more agile form of development, since tests can be run after any change, detecting if any unexpected changes have occurred.

Clojure provides a test harness which can be used directly with the Karyotype Ontology. For example, the following statements define two tests[3] which will succeed if the ontology `human` is both consistent and coherent according to the reasoner. That is we are asserting that (`r/consistent? human`) returns a value which `is` true[4].

```
(deftest Basic
 (is (r/consistent? human))
 (is (r/coherent? human)))
```

These tests can be run either individually or in batch using a single command. In total, Tawny-OWL itself contains over 3000 individual assertions. Next, we discuss the kinds of tests that we are running.

## 4 THE ONTOLOGY OF ONTOLOGY TESTS

In this section, we use the following terminology to distinguish between:

- **tawny-karyotype**[5]: the programmatic code written in Clojure, which uses the Tawny-OWL library.

- **the Karyotype Ontology:** the ontology in OWL, either as a set of in-memory Java objects, or as a serialisation as an OWL-XML file, which is generated by tawny-karyotype.

The first kind of test in tawny-karyotype we describe as **software-bound** tests and consists of traditional unit tests. These are tests where neither the test nor the code that it tests makes a direct reference to any ontology object. For example, during the construction of the Karyotype Ontology, it is useful to be able to determine whether a string, used as a label for a band, is either on the long (`p`) or short (`q`) arm of a chromosome. For this purpose, we have defined a *predicate* function as follows:

```
(defn str-pband? [band]
 (re-find #"p" band)
```

Here `defn` introduces a function with name `str-pband?` and formal parameter `band`. This returns true if we `re-find` the regular expression `#"p"` in `band`. This function is tested against a number of different band labels. The following examples test that the function returns both true and false correctly.

```
(is (h/str-pband? "HumanChromosome1Bandp10"))
(is (not (h/str-pband? "HumanChromosome1Bandq10")))
```

There are 53 of this kind of test. In this case, representative examples have been generated by hand, and the tests have been directly written in Clojure.

The second kind of test we call an **ontology-bound** test, as it refers to one or more ontology classes or properties. Most of these use predicates provided by Tawny-OWL or tawny-karyotype. For this reason, ontology-bound tests are also software-bound. For example, the following predicate function is defined as part of tawny-karyotype; this function depends on the `superclass?` function (defined in Tawny-OWL) and checks to see if `x` is a subclass of `HumanChromosomeBand`. In this example, `HumanChromosomeBand` is a term of the Karyotype Ontology, as would be the value passed into `x`.

```
(defn band? [x]
 (or
  (= x HumanChromosomeBand)
  (superclass? human x HumanChromosomeBand)))
```

This predicate function can then be used to test that the Karyotype Ontology correctly asserts that the class representing `1p10` is in fact a chromosome band.

```
(is (h/band? h/HumanChromosome1Bandp10))
```

There are 759 of this kind of test. As with software-bound tests, these tests have been written by hand.

The third kind of test, we call a **reasoner-bound** test as it uses computational reasoning to determine whether the test passes or not. All reasoner-bound tests are also ontology-bound. These tests determine whether the asserted conditions are fulfilled or not[6]. As an example, we might make this assertion, which says that `46,XY` should be a male karyotype.

```
(is (r/isuperclass? i/k46_XY n/MaleKaryotype))
```

---

There are 2273 of this kind of test. The majority of these tests are not directly asserted in Clojure source; we describe how these are generated in Section 5.

Finally, there is one final type of test which we call **probe-bound**. This form of test first changes the ontology in some way, tests assertions using this changed ontology, and lastly reverts these changes. Probe-bound tests are also reasoner-bound. In the following example, we assert a subclass of both HumanAutosome and HumanSexChromosome, then define a test assertion that states the ontology should now be incoherent. Tawny-OWL provides specific support for this form of test (with-probe-entities), as it is critical that any entities created during the tests are removed again to ensure independence.

```
(is
 (not
  (with−probe−entities
   [_ (owl−class "_"
        :super  HumanAutosome
                HumanSexChromosome)]
    (r/coherent?))))
```

We describe this form of test for completeness, as there are currently only three of these tests in the Karyotype Ontology.

## 5  SPECIFYING REASONER-BOUND TESTS WITH FACETS

While Tawny-OWL and Clojure provide a reasonably convenient syntax for specifying most of our tests, it is not ideal for all of them. A large number of tests for the Karyotype Ontology test the behaviour of a set of classes which are, effectively, competency questions for our ontology (Ren *et al.*, 2014). The International System for human Cytogenetic Nomenclature (ISCN) contains a large number of examples often describing well known conditions or syndromes. These have been encoded as an ontology as part of tawny-karyotype. The informal nature of the ISCN as a specification means that these examples are the best mechanism to ensure that the Karyotype Ontology fulfils the ISCN specification.

```
(defclass k45_X
 :super  ISCNExampleKaryotype
  (owl−some b/derivedFrom
   (owl−and
    (owl−some b/derivedFrom b/k46_XN)
     (e/deletion 1 h/HumanSexChromosome))))
```

Having defined these example classes it is, of course, useful to test that they perform as expected when reasoning. We have achieved this by defining a set of defined classes, which should result in a complex polyhierarchy after reasoning. We use these classes as facets in a spreadsheet. Currently, we define 18 facets, with a true/false/unknown value. For example, 45,X is defined as **NOT** male, female or haploid, but **IS** diploid, as shown in Table 1.

This spreadsheet is read at test time[7] using the Incanter library[8]. For example, two of the facets for 45,X are interpreted as these assertions:

```
(is (not (r/isuperclass? i/k45_X n/MaleKaryotype)))
(is (r/isuperclass? i/k45_X n/DiploidKaryotype))
```

---

[7]  Actually, it is translated to a Clojure and is automatically updated when necessary, which is a usability and performance enhancement.

[8]  https://github.com/incanter/incanter

**Table 1.** Table showing an excerpt of the ISCN examples facet spreadsheet. For each facet we define the value as either: true (1), false (-1), or unknown (0).

| Karyotype | Female | Male | Haploid | Diploid |
|-----------|--------|------|---------|---------|
| 45,X      | -1     | -1   | -1      | 1       |
| 45,XX,−22 | 1      | -1   | -1      | 1       |
| 45,X,−X   | 1      | -1   | -1      | 1       |
| 45,X,−Y   | -1     | 1    | -1      | 1       |

The use of a spreadsheet in this way provides a clean and consistent user interface for specifying facet values. As Tawny-OWL is fully programmatic, it is straight-forward to store this spreadsheet as part of the source code of tawny-karyotype which simplifies future updates, and has allowed us to specify a large number of tests for the Karyotype Ontology. In addition, this simplifies continuous integration, which we describe next.

## 6  CONTINUOUS INTEGRATION

Continuous integration (CI) is a software development process where code is tested against its dependencies (and code that depends on it) regularly; in most cases, developers now test their code after every commit to their version control system. CI provides two key features in addition to "normal" testing. Firstly, it is responsive to changes in any dependencies, allowing problems to be detected very early. Secondly, it is normally performed in a "clean" environment, supporting reproducibility.

As tawny-karyotype is using a standard test environment, it is very easy to set up CI. In our case, we are using TravisCI[9]. By design the Karyotype Ontology has no dependencies; the CI in this case, tests against changes in the software dependencies (Tawny-OWL, the OWL API, HermiT (Shearer *et al.*, 2008), and Clojure).

## 7  DISCUSSION

In this paper, we have described our approach to testing the Karyotype Ontology. The four different kinds of tests that we describe all have different purposes. The first of these, software-bound is strictly not a form of ontology testing at all, but unit testing for the software involved in ontology development. It is, however, an essential part of our test suite, as it helps to isolate errors which occur purely as a result of our ontology development software. Ontology-bound tests directly test our ontology, and ensure it describes the world correctly – in essence, they are the ontological equivalent of unit tests. The final two forms of tests are equivalent to functional tests, ensuring the ontology reasons as we expect. Our taxonomy and test usage differs from previous work by Garca-Ramos *et al.* (2009), as we test only T-Box (class) reasoning while they test the A-Box. In addition, we introduce tests for parts of the infrastructure outside of the base ontology.

The use of Tawny-OWL has also allowed us to specify tests as facets in a spreadsheet. Defining a test assertion by filling a cell, means we can test the karyotype ontology extensively (see Table 2). Although, in our case, we have built the tests using Tawny-OWL with an ontology written in Tawny-OWL, it is important to note that the test environment is de-coupled from the ontology development.

---

[9]  https://travis-ci.org/

Tawny-OWL can use ontologies written in OWL (by Protégé, for instance) and then test them.

**Table 2.** Table showing the number of assertions for each test type.

| Test Class | Software | Ontology | Reasoner | Probe |
|---|---|---|---|---|
| Base | 0 | 0 | 2 | 0 |
| Events | 3 | 600 | 2 | 0 |
| Features | 0 | 0 | 2 | 0 |
| Human | 50 | 58 | 2 | 0 |
| ISCN Examples | 0 | 0 | 2156 | 0 |
| Karyotype | 0 | 1 | 2 | 0 |
| Named | 0 | 0 | 83 | 0 |
| Parse | 0 | 28 | 2 | 0 |
| Random | 0 | 41 | 15 | 3 |
| Resolutions | 0 | 32 | 7 | 0 |
| Total | 53 | 759 | 2273 | 3 |

We have also briefly described our use of TravisCI, which performs *integration testing*. The Karyotype Ontology itself has no ontology dependencies, but we have generated an example ontology which is a dependency of the Karyotype Ontology and helps to form a test suite for it. We believe, that the Karyotype Ontology is rather unusual in having no ontological dependencies. Integration testing is likely to bear even more fruit for ontologies with a large or complex dependency graph.

Continuing the metaphor to software engineering, there are currently several forms of testing that we do not perform on the Karyotype Ontology. One common problem with ontology development is understanding reasoner performance, especially the overall reasoning time. The use of Tawny-OWL does allow performance testing; for instance, we have extensively compared several different axiomatisations for parts of the Karyotype Ontology[10]. This form of testing is a *non-functional* test. We do not currently check overall reasoning performance as part of our automated test suite, but this is possible and is likely to be included in tawny-karyotype in the future.

We would also like to test aspects of the ontology other than the class hierarchy, including extra-logical aspects such as labels or definitions. Historically, this form of testing is quite difficult in Tawny-OWL because we lacked a good mechanism for querying an ontology syntactically; however, an initial implementation for such a mechanism (called, prosaically, `tawny.query`) is now in place.

There are a number of tools available for software testing which an equivalent is not currently available for ontology development within Tawny-OWL, but which would be extremely useful. We currently, for instance, cannot assess the state of *coverage* for the Karyotype Ontology as we have neither a tool nor a clear understanding of how it should assessed for ontologies.

Despite these limitations, the use of Tawny-OWL has allowed us to recast testing of the Karyotype Ontology as a form of software testing. We have reused many standard tools to enable this process, and they perform well. In addition, we have made use of programmatic nature of Tawny-OWL to allow specification of tests using spreadsheets as source code, using the extensibility of Tawny-OWL, something we have also found useful for ontology

---

[10] https://github.com/jaydchan/tawny-karyotype-scaling

development. As Tawny-OWL is built on the OWL API, it can offer these capabilities to any ontology, whatever the development environment. So while the work reported on here is specific to the Karyotype Ontology, we believe that the classification of ontology tests and the tooling is generic, and we look forward the application of these forms of tests to many other ontologies.

## ACKNOWLEDGEMENTS

## REFERENCES

Ashburner, M. *et al.* (2000). Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nature Genetics*, **25**(1), 25–29.

Garca-Ramos, S., Otero, A., and Fernndez-Lpez, M. (2009). Ontologytest: A tool to evaluate ontologies through tests defined by the user. In S. Omatu *et al.*, editors, *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, volume 5518 of *Lecture Notes in Computer Science*, pages 91–98. Springer Berlin Heidelberg.

Horridge, M. and Bechhofer, S. (2011). The OWL API: A Java API for OWL ontologies. *Semant. web*, **2**(1), 11–21.

Köhler, S. *et al.* (2014). The human phenotype ontology project: linking molecular biology and disease through phenotype data. *Nucleic Acids Research*, **42**(D1), D966–D974.

Lord, P. (2013). The Semantic Web takes Wing: Programming Ontologies with Tawny-OWL. http://arxiv.org/abs/1303.0213.

Malone, J., Holloway, E., Adamusiak, T., Kapushesky, M., Zheng, J., Kolesnikov, N., Zhukova, A., Brazma, A., and Parkinson, H. (2010). Modeling Sample Variables with an Experimental Factor Ontology. *Bioinformatics (Oxford, England)*.

Mungall, C., Dietze, H., Carbon, S., Ireland, A., Bauer, S., and Lewis, S. (2012). Continuous integration of open biological ontology libraries. http://bio-ontologies.knowledgeblog.org/405.

Ren, Y., Parvizi, A., Mellish, C., Pan, J., van Deemter, K., and Stevens, R. (2014). Towards competency question-driven ontology authoring. In V. Presutti *et al.*, editors, *The Semantic Web: Trends and Challenges*, volume 8465 of *Lecture Notes in Computer Science*, pages 752–767. Springer International Publishing.

Shaffer, L. G., McGowan-Jordan, J., and Schmid, M., editors (2012). *ISCN 2013: An International System for Human Cytogenetic Nomenclature (2013)*. Karger.

Shearer, R., Motik, B., and Horrocks, I. (2008). HermiT: A Highly-Efficient OWL Reasoner. In A. Ruttenberg, U. Sattler, and C. Dolbear, editors, *Proc. of the 5th Int. Workshop on OWL: Experiences and Directions (OWLED 2008 EU)*, Karlsruhe, Germany.

Warrender, J. D. and Lord, P. (2013a). A pattern-driven approach to biomedical ontology engineering. *SWAT4LS 2013*.

Warrender, J. D. and Lord, P. (2013b). The Karyotype Ontology: a computational representation for human cytogenetic patterns. *Bio-Ontologies 2013*.