## FIBO Adaptation of OQuaRE and OOPS!

**Reference:** Ontology Summit 2013 Hackathon HC-03

# Introduction

This table is the product of the Ontology Summit 2013 Hackathon carried out by the author along with a number of EDM Council FIBO regulars and Ontology Summit participants.

The table is based in one ontology evaluation method called OQuaRE, and draws extensively on material from another ontology evaluation method, called OOPS!

A further ontology evaluation method, GOEF, was also presented at the 2013 Summit but did not participate in the FIBO-led hackathon. Features of that methodology should also be considered and added to the FIBO evaluation framework as appropriate, in particular the GOEF approach to use cases.

## Table Development

The table was worked on throughout the Hackathon (a period of 3 days). For each line entry in the table, representing one aspect of the ontology, we identified two things:

1. Evaluation tools and techniques, including automated programs, from both OOPS! And OQuaRE;
2. Whether these measures should be applied to the FIBO conceptual ontology, to operational ontologies derived from this, or both (shown as “C” and “O“ in the final column)

Additional notes were made by the author in the existing narrative columns; these are tidied up in the current document.

# Ontology Evaluation Methods Used

## The Ontology Quality Evaluation Framework (OQuaRE)

### Overview

OQuaRE is an ontology evaluation framework developed by Jesualdo Tomás Fernandez-Breis and Astrid Duque-Ramos at the Departamento de Informática y Sistemas, Universidad de Murcia, Spain.

OQuaRE takes the principles set out in a conventional technology application assessment framework called SQuaRE, and adapts these to the evaluation of ontologies.

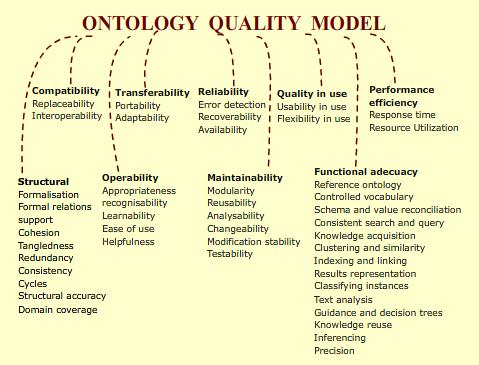
In view of the fact that ontologies are a structural rather than a behavioral artifact, the OQuaRE framework adds a further set of criteria under the heading “Structural” which are not in SQuaRE.

### Explanation from the Originators (academic)

OQUARE[[1]](#id.prvle5lwqegv) is a framework for evaluating the quality of ontologies based on the standard ISO/IEC 25000:2005 for Software product Quality Requirements and Evaluation known as SQuaRE [[2]](#id.yptqb6k48ww5). recent efforts have been put on investigating how such software quality standards can be adapted to measure ontology quality, an approach based on the ISO 9126 was presented In Fernandez-Breis, Egaña, and Stevens (2009)[[3]](#id.efay5l2vdf01).

OQuaRE aims to define all the elements required for ontology evaluation: evaluation support, evaluation process and metrics. The current version of our framework includes, so far, the quality model and the quality metrics, because they are the basic modules for the technical evaluation of the quality of the ontology, and which are explained next.

This is summarized in Figure 1 below:



**Figure 1: OQuaRE Framework**

### The OQuaRE Quality Model

This model reuses and adapts the following SQuaRE characteristics to ontologies: functional adequacy, reliability, operability, maintainability, compatibility, transferability, performance efficiency and quality in use. Moreover, according to requirements, principles and characteristics of ontologies and to the state of the art of ontology evaluation [[3]](#id.tv0dbjb3vcaj), structural features of ontologies are important to evaluate their quality, but they are not considered in the standard. Consequently, we added such characteristics to our framework. In order to determine the quality sub-characteristics, we combined the ones suggested in SQuaRE with the ones suggested by state-of-the-art methods from the ontology evaluation community. The functional adequacy sub-characteristics are the intended uses for ontologies identified in [[4]](#id.khps6u8z19cq),

## FIBO Quality Metrics and Tests

The tests which fall under OQuaRE are identified in the table by their mnemonics, e.g. “ANOnto” which is a measure of annotation richness.

# OOPS! Overview

The OOPS! Ontology evaluation framework provides a set of evaluation metrics, some of them automated.

## OOPS! (OntOlogy Pitfall Scanner!)

OOPS! is an ontology evaluation tool developed by the Ontology Engineering Group (OEG) at the Polytechnic University of Madrid, Spain (UPM, ES) that helps a developer detect some of the most common pitfalls appearing when developing ontologies.

Home page: <http://www.oeg-upm.net/oops>

OOPS! Is available on line as a web site where anyone can submit an XML fragment containing reference to the ontology to be test.

Such requests are submitted in the following format:

<?xml version="1.0" encoding="UTF-8"?>

<OOPSRequest>

<OntologyUrl>http://www.cc.uah.es/ie/ont/learning-resources.owl</OntologyUrl>

<OntologyContent></OntologyContent>

<Pitfalls>10</Pitfalls>

<OutputFormat></OutputFormat>

</OOPSRequest>

There is also a RESTFul Web Service at: <http://oops-ws.oeg-upm.net/>

Ontologies may also be submitted directly or via their URL at the pitfalls description page <http://oeg-lia3.dia.fi.upm.es/oops/catalogue.jsp> (as noted below).

### OOPS! Usage in FIBO Table

The OOPS! measures are evaluated in the FIBO table by their numbers, e.g. P22. The descriptions of these is found in:

<http://oeg-lia3.dia.fi.upm.es/oops/catalogue.jsp>

The full catalog of numbered “pitfalls” is given as Annex B.

### Testing

Note that the above page is also a link where ontologies may be entered directly or the URL of an ontology may be given.

Note however that links between ontologies are not supported. For the Hackathon we used WebProtégé to create a single ontology for all the material under test and submitted this to that link.

# FIBO OQuaRE Table

**STRUCTURAL**

**T**his category is the only one in this framework that is not specified as such in SQuaRE, but it is important when evaluating ontologies, since it accounts for ontology quality factors such as consistency, formalization, redundancy or tangledness.

| **Sub-characteristic** | **Definition** | **Description** | **Metrics** | **Conc/Op** |
| --- | --- | --- | --- | --- |
| Formalisation | Capability of the ontology to support reasoning in principle.  Use of upper level partitions with disjoints to ensure semantics of classes are respected. | An ontology should be expressed in a common formal language e.g. for bio-ontologies in agreement with the ontology principle FP 002 format [[5]](#id.h3366vbt6174): OBO Format, OWL or OWL2 concrete syntax, RDF/XML, OWL2-XML, OWL2-Manchester Syntax, Common Logic concrete syntax, CLIF, Conceptual Graphs, etc. | P23. Using incorrectly ontology elements | C  O |
| Formalisation | Capability of the ontology to support reasoning in an application, in practice.  Making sure the ontology meets computational criteria. | Application of DL-safe rules (if in a DL language OWL or OWL2) and other mathematical measures to ensure the suitability of the ontology for a reasoning based application..  Other tests in real life e.g. run tests on sample data, can establish computability without necessarily being able to prove computability.  Level of expressivity v formalization is a balance. | (measures exist)  <http://oa.upm.es/5453/1/OntologyTest.pdf> | O |
| Formal relations support | Capability of the ontology to represent relations supported for formal theories different to the formal support for taxonomy | It accounts for the types of formal relations supported by the ontology, different to “is a” relations  This covers how much information is conveyed by the ontology. | [RROnto](#id.y7agimkrm72m) | C  O |
| Structural accuracy | Degree of the correctness of the terms used in the ontology  Requires human review. Domain experts.  Be able to indicate whether this is an issue | Correctness of taxonomic links, use of upper level with disjoint categories, consideration of "all some rule" in case of existential restrictions, value restrictions only if disjoint partitions available, domain and range restrictions of object properties, sufficient metadata and annotation properties, free text definitions where necessary | Experts, P5, P14, P15, P16, P17.P19, P23. P25 | C |
| Consistency | Degree of the consistency of the ontology | Consistent naming conventions, logical consistency, structural consistency, consistent distinction class - instance  e.g. Capitals, plurals, Standards, etc. | P22 | C  O |
| Tangledness | This measures the distribution of multiple parent categories, so that it is related to the existence of multiple inheritance. | Note this is a good thing in a conceptual ontology and a bad thing in an application ontology | [TMOnto](#id.471626o02uoe) | C  O |
| Cycles | The existence of cycles through a particular semantic relation is usually a sign of bad design, since they may lead to inconsistencies. |  | P6 | C  O |
| Cohesion | An ontology has a high cohesion if the classes are strongly related. | Density of relationships among classes.  Disconnected parts of the ontology (islands) | [LCOMOnto](#id.4ljburjm71fu)  P4  Coh-QA | C |
| Domain coverage | The degree to which The ontology covers the specified domain | This is evaluated by experts.  Also cross reference to domain technical standards for message and data, to assess coverage (ontology should have semantics of all the terms in those tech standards)  Intersection of terms in the ontology and the domain technical standards.  Annotations in the ontology identifying mapping to domain technical standards.  Term origin: provenance of the term  Mapping (future): real time mapping of the whole of the terms in target industry standards (regression test). |  | C |
| Redundancy | Existence of properties that can be inferred from those already in the model;  Existence of properties that add no meaning | Some redundancy types are: Inferred information more than once from the relations, classes and instances found in ontology.  The same/different formal definition of classes, properties or instances referring to different/same classes, properties or instances [[6]](#id.ygs1455tvls0)  For the former, these may be beneficial in conceptual ontologies, where they can add labels, localized definitions, and mapping to data / message tech models; in operational ontologies they are not needed.  For the latter, these should not be in either style of ontology and should be detected and removed. | P2, P32 | O |

## Functional adequacy [[5]](#id.khps6u8z19cq)

R = Reuse of other ontologies (for FIBO to refer to)

**The capability of the ontologies to provide concrete functions.**

| **Sub-characteristic** | **Definition** | **Description** | **Metrics** | **Conc/Op** |
| --- | --- | --- | --- | --- |
| Reference ontology | Degree in which the ontology can be used as a reference resource for the particular domain the ontology is built for. | Machines can exploit better the reference knowledge offered by an ontology which has a more explicit structure.  Relates to the other ‘C’ metrics especially domain coverage, consistency, redundancy etc.  Structural characteristics all apply. |  | C  R |
| Controlled vocabulary | Capability of the ontology to avoid heterogeneity of the terms.  Segregate ontology issues (synonyms, no heteronyms) from terminological standard ‘layer’ in future SBVR support, which deals with vocabulary matters. | The labels of the ontological entities are used to avoid heterogeneity, which would complicate data processing and analysis. Ontologies provide terminology management, unifying the terms used for referring to the included knowledge entities which should have an unambiguous and non-redundant definition. Ontological concepts are described with different terminologies, different meanings are assigned to the same word in different contexts and different taxonomies are examples of synonymy, polysemy and structural heterogeneity [[7]](#id.k76mj55vfagb). | [ANOnto](#id.johfaapej3iq)  P1, P2 P9 | C |
| Schema and value reconciliation. | Degree in which ontology provide a common data model that can be applied to reconciliation and integration.  conceptual ontology: this is a primary use case  Operational: federated semantic querying applications | Integration is building a new ontology reusing other available ontologies. = reusability (in Maintainability). Mappings; formalization.  Different ontologies with subjective features and particular perspective on the world, cannot be compared without reconciliation and integration, which are necessary to interchange, migration and standardization of information and knowledge of such ontologies.  See e.g. Partridge, Smith on use of upper ontology partitions – Ontology Summit 2013 Track C sessions | [RROnto](#id.y7agimkrm72m), [AROnto](#id.aj78lucgdzut), Formal degree, consistency degree | C  O\* |
| Consistent search and query | The degree which the formal model and structure of the ontology provide a semantic context to evaluate which are the data wanted by the users, allowing better querying and searching methods | Structure of ontology, to make more consistent search queries.  Annotation richness - does not contribute to semantic search. May be relevant in other forms of search (keywords)? | [ANOnto](#id.johfaapej3iq),[RROnto](#id.y7agimkrm72m), [AROnto](#id.aj78lucgdzut),[INROnto](#id.jr26w8s1973o), Formal degree  P8 P11 P12 P4 P30 P31 P32 P13  (P2) | O |
| Knowledge acquisition – representation. | Capability of the Ontology to represent the knowledge acquired. (ability to support a knowledge base of individuals) | Knowledge acquisition is the gathering, storage, and encoding of existing information.  Individuals: OntoQA  OOPS! - ability to support KB adequately  also relationship richness etc. e.g NOMOnto | [ANOnto](#id.johfaapej3iq), [RROnto](#id.y7agimkrm72m),[NOMOnto](#id.1a31segokri8)  P9, P14 - 18, P23  Test: sample data KB | C |
| Clustering | Degree in which the annotations of data with respect to ontology terms can be used for clustering such data against the aspects of the ontology. | Clustering can be defined as the process of organizing objects into groups whose members are more similar to each other than to individuals in other groups.  SME Review / Externality Review: detection of classes which have similar properties to other classes elsewhere.  Also ontology modularity (maintainability) and partitioning. Plugins for this in NeOn ontology editor (Eclipse based editor).  <http://neon-toolkit.org/wiki/Ontology_Module_Partition> for the plugin - classes per cluster, use plugin as a check.  There is a paper that analyzes the labels of every class; how used in subclasses. ISO 11179 | [ANOnto](#id.johfaapej3iq),P2  Import to NeOn ecosystem and run plugins | C |
| Similarity | a. Capability of the component of the ontology to be compared for similarity  b. Similarity of one ontology to another | a. There exist different similarity measures: Taxonomy similarity, Relation similarity, Attribute similarity, semantic similarity. [[8]](#id.lccjikx3rckv)  Good thing v bad thing e.g. non identification of similar classes (P12, P30) v ability to compare (Good)  Availability of SKOS annotations defining the broader, narrower meanings between elements.  SKOS  <http://skos.um.es/unescothes/>  <http://databases.unesco.org/thesaurus/>  b. Run OntoQA on two ontologies to compare output for similarity between ontologies (numeric output); run OOPS! to find similarities expressed as similar pitfalls? Would need to inspect after. Some Ps would give clues e.g. inverses etc.  Semantic similarity: OOPS! tests on synonymy etc. may be relevant.  Use ontology matching tool to measure semantic similarity.  Mappings e.g. Equivalent Class usage. If mappings are complex (tangled) this would show that ontologies on the same subject may be modeling it differently. If mappings more direct, then they are similar. (see refs) | [RROnto](#id.y7agimkrm72m), [AROnto](#id.aj78lucgdzut),  P12 P30  SKOS usage  Visual inspection  b. See refs - Euzenat; Gracia  Jorge Gracia (Tool: CIDER)  http://sid.cps.unizar.es/SEMANTICWEB/ALIGNMENT/ | C  C |
| Indexing and linking | Degree in which the classes defined in the ontology can act as indexes for quick information retrieval | See search and query. This covers the search aspect of this. | [RROnto](#id.y7agimkrm72m), [AROnto](#id.aj78lucgdzut),[INROnto](#id.jr26w8s1973o)  P21 P30 P31 P27 | C  O |
| Results representation | Capability of the ontology to analyze complex results such as microarrays experiments. | Application examples include SPARQL queries across complex networks of instruments and business entities, returning query results in tabular form.  Are there quality measures for the ontology to support this? e.g. completeness and consistency of properties? Mix of datatype property v object property. | [CROnto](#id.bvdfrstk50wm), [AROnto](#id.aj78lucgdzut),  INROnto, NACOnto NOCOnto (No of Ancestor classes,child classes) | O |
| Classifying instances | Degree in which ontology Instances can be recognized as member of a certain class | Need to be able to populate a KB with individuals that are instances of the correct class in FIBO. Therefore annotation; Also misuse of classes, properties etc. (Pitfalls);  Misuse of class / instance relations. Also punning. Domain and range measure also apply. | OntoQA  ANOnto  P20; P13 P14, 15, 16  P17 P11 P19  Tests on example instance data | C  O |
| Text analysis | Capability of the structure of the ontology to helps detecting associations between words or concepts and classifying word types. | Vocabulary issues - SBVR and / or SKOS layers on top of FIBO, not FIBO itself. | Formal degree  [ANOnto](#id.johfaapej3iq)  P1, P2 P9,  P30 P31 P27 | C |
| Guidance | Capability of the ontology to guide the specification of domain theories. | Ontology by capturing knowledge about a domain and encapsulating constraints about class membership provides guidance in the specification of domain theories and supports decision making processes.  Necessary and sufficient properties; use of OWL restrictions is important here. | [AROnto](#id.aj78lucgdzut),[INROnto](#id.jr26w8s1973o)  P14 P15 P16 (by inspection) | C  O |
| Decision trees | Capability of the ontology to be used building Decision trees. | Decision trees are used to represent the logical structures of classification rules for domain specific empirical data. See also Classifying Instances above.  Classification and Rules:  This relates partly to the ability to add business rules to operational ontology (e.g. SWRL, RIF, R) on top of what’s in the Conceptual Ontology content. This is for classifications that are based on relationships among other classes.  Quality requirements to support this: For SWRL, these have to be in OWL; have to have adequate information to define the rules i.e. the predicates for the rules are well defined ontological classes in the ontology | [AROnto](#id.aj78lucgdzut),[INROnto](#id.jr26w8s1973o), [TMOnto](#id.471626o02uoe)  RROnto CROnto NOMOnto  P16; all except annotation related | O |
| Knowledge reuse | The degree to which The ontology knowledge can be used to build other ontologies (knowledge in the ontology) not knowledge in a KB (individuals). | This entry: structural considerations in the ability to reuse knowledge.  See Maintainability/Reusability for other reuse aspects. | [ANOnto](#id.johfaapej3iq),[AROnto](#id.aj78lucgdzut),[INROnto](#id.jr26w8s1973o), Formal degree, Consistency degree, [NOMOnto](#id.1a31segokri8), [LCOMOnto](#id.4ljburjm71fu)  P-all | C |
| Inference | The degree to which The formal model of the ontology can be used by reasoners to make implicit knowledge explicit. | Inference expands the knowledge base with additional information using the existing data, metadata, and rules.  Measures of the expressivity will cover this | formal degree,[RROnto](#id.y7agimkrm72m), [CROnto](#id.bvdfrstk50wm) AROnto  P4,P5, P6, P11- P18, P27-P31, P33 | O |
| Precision | The degree to which The ontology provides the right or specified results with the needed degree of accuracy | Two aspects of precision: how fine / deep and how accurate.  a. measures relating to how correct e.g. misuse of things  b. Depth of class and relationship hierarchies;  The business coverage as described above in ‘Domain Coverage’ section. | DITOnto, INROnto,  P23 P28 P29 P5  P27 | C  O |

## Maintainability

**The capability of ontologies to be modified for changes in environments, in requirements or in functional specifications.**

| **Sub-characteristic** | **Definition** | **Description** | **Metrics** | **BCO/OO** |
| --- | --- | --- | --- | --- |
| Modularity | The degree to which the ontology is composed of discrete components such that a change to one component has minimal impact on other components. | Balance of size of modules, number / complexity of OWL imports, and segregation of concerns (by partition, by classification facet, by business application / use case / 3rdness “context” class relations?...) | [WMCOnto](#id.v4aib5qwgu1w), [CBOnto](#id.p5xl7ygbalru) | C |
| Reusability | The degree to which an asset (part of) the ontology can be used in more than one ontology, or in building other assets | This can be measured by percent of classes that could be reused; degree of abstraction in the class hierarchy.  2 types of reuse: semantic and structural adequacy.  **Semantic** = abstraction = class hierarchy depth; Covered by DITOnto.  Need to state what figures correspond to levels of reusability.  Partitions / mappings / overlaps to other ontologies and upper ontologies use.  Also indicates when an ontology is well documented enough that you know it cannot be used for your own application (still a +ve thing).  Positive and negative scores.  **Structural:** NOCOnto,  Important for foundational ontologies e.g. FIBO Foundations.  Important in modular ontology standards structure generally - aids extensibility of the business domain semantics.  Availability for others to use - must be published  Reusability internally - need not be.  Examples of use  See also learnability and related items. Maintainability also (reusability).  Versioning etc. also important (maintainability) | [WMCOnto](#id.v4aib5qwgu1w), [CBOnto](#id.p5xl7ygbalru) [DITOnto](#id.o0wnsxuh5hc7),[NOCOnto](#id.7m0hberma3ab), [RFCOnto](#id.m16l2att3k84),[NOMOnto](#id.1a31segokri8)  Inspection  Availability  Examples | C |
| Analyzability | The degree to which The ontology can be diagnosed for deficiencies or causes of failures (inconsistences), or for the parts to be modified to be identified | Check whether the ontology is formalized in an ontology language, and that there is a reasoner that can be applied.  Consistency checking in a reasoner is available.  Individual pitfalls also apply (inconsistencies). | [LCOMOnto](#id.4ljburjm71fu), [WMCOnto](#id.v4aib5qwgu1w), [CBOnto](#id.p5xl7ygbalru) [DITOnto](#id.o0wnsxuh5hc7), [RFCOnto](#id.m16l2att3k84),[NOMOnto](#id.1a31segokri8),  P19 | C  O |
| Changeability | The degree to which The Ontology enables a specified modification to be implemented. The ease with which an ontology can be modified | Some kinds of changes in the ontology are: Add or remove classes, axioms, logical axioms, annotations, explicitly stated axioms or annotations and inferred axioms that are entailed by ontology.  A measure of this is the extent to which concepts are abstracted from the specific to the most atomic (archetypical) with suitable levels in between. Not needed in an application ontology but vital in a conceptual ontology. | [WMCOnto](#id.v4aib5qwgu1w), [CBOnto](#id.p5xl7ygbalru) [DITOnto](#id.o0wnsxuh5hc7),[NOCOnto](#id.7m0hberma3ab), [RFCOnto](#id.m16l2att3k84),[NOMOnto](#id.1a31segokri8)  [LCOMOnto](#id.4ljburjm71fu), [NOMOnto](#id.1a31segokri8), | C  O |
| Modification stability | The degree to which The ontology can avoid unexpected effects from modifications of the knowledge (terms, classes, properties, etc.). | Ontology changes could modify ontology specification or conceptualization and have negative effect over ontology.  If classes are suitably abstracted then all changes should be additive; if not they will not be. | [WMCOnto](#id.v4aib5qwgu1w), [CBOnto](#id.p5xl7ygbalru), [RFCOnto](#id.m16l2att3k84), [COMOnto](#id.4ljburjm71fu), [NOCOnto](#id.7m0hberma3ab),  (no P) | C  O |
| Testability | The degree to which the ontology modified can be validated. | As well as the metrics given here, there is the possibility to create a set of standard SPARQL queries and test individuals for regression testing.  Also use of ACE with a human in the loop, to validate implications of changes. | [WMCOnto](#id.v4aib5qwgu1w), [CBOnto](#id.p5xl7ygbalru), [RFCOnto](#id.m16l2att3k84), [DITOnto](#id.o0wnsxuh5hc7), [NOMOnto](#id.1a31segokri8), [LCOMOnto](#id.4ljburjm71fu) | C  O |

## Compatibility

**he ability of two or more software components to exchange information and/or to perform their required functions while sharing the same hardware or software environment**

| **Sub-characteristic** | **Definition** | **Description** | **Metrics** | **BCO/OO** |
| --- | --- | --- | --- | --- |
| Replaceability | The degree to which The ontology can be used in place of another specified Ontology for the same purpose in the same environment. | The ability for users of operational ontologies to ignore some ontology that is referenced in the BCO and replace it with another or their own, e.g. replace geopolitical concepts from FIBO with their own geopolitical ontology.  This equates to a measure of how well ontologies are isolated / how boundaries are drawn.  Formalization, Modularization. | [WMCOnto](#id.v4aib5qwgu1w), [CBOnto](#id.p5xl7ygbalru) [DITOnto](#id.o0wnsxuh5hc7),[NOCOnto](#id.7m0hberma3ab), [NOMOnto](#id.1a31segokri8) | C |
| Interoperability | The degree to which the ontology can be cooperatively operable combining its knowledge with one or more other ontologies. | Ontology matching consists of matching a concept from one ontology to another.  Annotation / SKOS and other metadata (at element and ontology level).  See point made about abstraction levels in the preceding table (on changeability etc.). This also applies here. | Annotation depth  [WMCOnto](#id.v4aib5qwgu1w), [CBOnto](#id.p5xl7ygbalru) | C  O |

## Transferability

**The degree to which the software product can be transferred from one environment to another**

| **Sub-characteristic** | **Definition** | **Description** | **Metrics** | **BCO/OO** |
| --- | --- | --- | --- | --- |
| Portability | The degree in which an Ontology or one part of the ontology can be transferred from one hardware or software environment to another | The degree to which the ontology can be translated between different formal languages, or how easily the code can be moved to another language.  What constructs of the language are used, which do not exist in other ontologies languages?  Also what design conventions are in use which can or cannot be expressed in another environment.  Existence of software to change from one language to another. | Inspection; language | C |
| Adaptability | The degree to which The ontology can be adapted for different specified environments (languages, expressivity levels) without applying actions or means other than those provided for this purpose for the Ontology considered. | Difference is 1 is whether the ontology can be transferred and 2 is how it can be adapted if it cannot be ported directly.  See also the considerations above. | [WMCOnto](#id.v4aib5qwgu1w), [CBOnto](#id.p5xl7ygbalru) [DITOnto](#id.o0wnsxuh5hc7), [RFCOnto](#id.m16l2att3k84), | C |

## Operability

**Effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.**

| **Sub-characteristic** | **Definition** | **Description** | **Metrics** | **BCO/OO** |
| --- | --- | --- | --- | --- |
| Appropriateness recognizability | The degree to which the Ontology enables users to recognize whether it is appropriate for their needs. | The ability to recognize the appropriateness of the functions from initial impressions of the ontology and/or any associated documentation such as Manuals, guides, comments.  Relates to documentation, annotation. Also domain and range | Annotation  P8, P20, P11 | C |
| Informativeness | Capability of the ontology to be informative. Defines how well the ontology content is communicated (particularly to modelers) so that future changes are made with an understanding of what is there now. | as above; see also learnability.  (this row was added during our weekend session but in fact it replicates what’s in Learnability and others).  May remove this row. | [ANOnto](#id.johfaapej3iq)  AROnto | C |
| Learnability | The degree to which the ontology enables users to learn its application.  Split this out to cover aspects of how well the information in the ontology may be understood. | Effectiveness of the user documentation and/or help system.  Also documentation external to the ontology; annotation / comments, domain and range as above  Use of labels (simple ontology); use of SKOS annotations for definitions, notes.  Availability of business facing presentation (boxes-and-lines diagram; spreadsheet; CNL); whether the constructs in the ontology are amenable to these presentations (or have equivalent relations which are); e.g. OWL Restrictions are not learnable to all but a professional modeler | [LCOMOnto](#id.4ljburjm71fu), [WMCOnto](#id.v4aib5qwgu1w), [CBOnto](#id.p5xl7ygbalru) [RFCOnto](#id.m16l2att3k84),[NOMOnto](#id.1a31segokri8), [NOCOnto](#id.7m0hberma3ab), AROnto  ANOnto  P8. Missing annotations  P11, P20  Examples  Documentation  Comments | C |
| Ease of use | The degree to which the ontology makes it easy for users to operate and control it. | Ease of formulation of queries?  Size of ontology; tangledness | DITOnto | C  O |
| Helpfulness | The degree to which the Ontology provides help when users need assistance.  FIBO: In future, for different business visualizations, this row would cover measures of the ontology related to those specific viz techniques. | The ontology provides clear error messages, manuals and guides to help the users, including help itself being comprehensive, effective and easy to find.  See measures under Learnability. This entry focuses on internal documentation / comments / annotations. | See above | C |

## Reliability

**Capability of an ontology to maintain its level of performance under stated conditions for a given period of time**

| **Sub-characteristic** | **Definition** | **Description** | **Metrics** | **BCO/OO** |
| --- | --- | --- | --- | --- |
| Error detection | The degree to which The Ontology enables users to detect faults. | Some of the faults are: Inconsistency, incompleteness and redundancy.  Formal language used, and ability to analyze this in a reasoner.  Can use SPARQL queries to describe patterns that are not allowed.  OntologyTesting  OntoQA  OntoClean? | Language;  abillity to run OOPS! and OQuaRE in that syntax.  SPARQL | O |
| Recoverability | The degree to which the Ontology can re-establish a specified level of performance and recover the data directly affected in the case of a failure. | An application not an ontology metric. | [LCOMOnto](#id.4ljburjm71fu),[WMCOnto](#id.v4aib5qwgu1w), [NOMOnto](#id.1a31segokri8), [DITOnto](#id.o0wnsxuh5hc7), [R](#id.m16l2att3k84) | N/A |
| Availability | The degree to which an ontology or part of it is operational and available when required for use with different applications | Related to adaptability. See measures for that entry.  Also availability on line - whether URIs can be dereferenced.  Several levels of availability:  - described in a paper so you can figure out ontology  - available only from the editor  - available on line  - freely available on line under license  Also the language used - determines how it can be made available e.g. OWL more available (existence of tools);  Available in more than one language (e.g. OWL + CLIF + OBO  Available in more than one file format (e.g. ODM, RDF/OWL XML, | See Adaptability | C |

## Performance Efficiency

**Relationship between the level of performance of the software and the amount of resources used, under stated conditions, taking into account elements such as the time response, or memory consumption.**

| **Sub-characteristic** | **Definition** | **Description** | **Metrics** | **BCO/OO** |
| --- | --- | --- | --- | --- |
| Response time | The degree to which The Ontology provides appropriate response and processing times from and throughput rates when performing its function (Queries and reasoning), under stated conditions. | Mathematical measures of ontology reasoning time frames / DL-safe rules, finite v non finite computational time etc.  Applies to operational ontologies only (application constraint)  Also structure - this may impact reasoning speeds/ efficiency etc. or of queries.  Other aspects that determine ease or difficulty of queries (SPARQL).  The expressivity of the ontology e.g.OWL-Full = more expressive, less responsive; OWL DL more efficient, less expressive.    Cycles related pitfalls?  All reasoning based measures apply.  Size also a consideration. | OWL-full Pitfall | O |
| Resource Utilization | The degree to which the application uses appropriate amounts and types of resources when The Ontology performs its function under stated conditions. | Same as Response Time but relates to Resource rather than response time - these two issues are combined in the measures described above. | as above | O |

## Quality in use

**Quality in a particular context of use. Quality in use is the degree to which a product used by specific users meets their needs to achieve specific goals**

| **Sub-characteristic** | **Definition** | **Description** | **Metrics** | **BCO/OO** |
| --- | --- | --- | --- | --- |
| Usability in use | **Effectiveness in use:** the degree to which specified users can achieve specified goals with accuracy and completeness in a specified context of use.  **Efficiency in use:** The degree to which specified users expend appropriate amounts of resources in relation to the effectiveness achieved in a specified context of use.  **Satisfaction in use:** The degree to which users are satisfied in a specified context of use.  Satisfaction is further subdivided into sub-sub-characteristics: Likability (cognitive satisfaction), Pleasure (emotional satisfaction), Comfort (physical satisfaction), Trust. |  |  | App |
| Flexibility in use | **Context conformity in use:** The degree to which usability in use meets requirements in all the intended contexts of use.  **Context extendibility in use:** The degree of usability in use in contexts beyond those initially intended | Business domain views and support thereof? |  | App |

## Annex A: OQuaRE Quality metrics

OQuaRE permits the definition of the quality model in terms of quality characteristics. In this way, this standard suggests a series of quality characterstics that should be used for measuring quality. Each quality characteristic has a set of quality subcharacteristics associated and each subcharacteristic has a set of primitives measures associated. For the the definition of the metrics (primitives), the following notation has been adopted:

C1; C2; …; Cn: Classes of the ontology. RC1; RC2; …; RCk: Relationships of the class Ci. PC1; PC2; …; PCz: Properties of the class Ci. IC1; IC2; …; ICm: Individuals of the class Ci. SupC1; SupC2; …; SupCm: Direct superclasses of a given class C. Thing: Root class of the ontology.

Some of the metrics like Coupling Between Objects (CBO), Depth of Inheritance Tree (DIT), Number Of Children (NOC), Response For a Class (RFC), Weighted Method Count (WMC), (Chidamber and Kemerer, 1994)[[9]](#id.5dut6sruee33) and Number Of local Methods (NOM) by (Li and Henry, 1993)[[10]](#id.elwguywwghai) were selected from Software Engineering and, in particular, Object-oriented Programming (OOP) and adapted to ontologies. Despite ontologies and object oriented design having different properties, there are a series of shared notions as the existence of classes, individuals and properties that can be exploited to adapt OOP metrics to ontologies. And reused other metrics developed by the ontology engineering community, especially for the structural properties from, for instance, Yao, Orme, and Etzkorn (2005)[[11]](#id.2dc2iyfrs1fp) or Tartir and Arpinar (2007)[[12]](#id.n1eiufwp5mbq) and Gangemi, Catenacci, Ciaramita, and Lehmann [[13]](#id.n1eiufwp5mbq)

## LCOMOnto - Lack of Cohesion in Methods

Semantic and conceptual relatedness of classes. It can be used to measure the separation of responsibilities and independence of components of ontologies LCOMOnto=∑path(|C(leaf)i|)/m , where path|C(leaf)i| is the length of the path from the leaf class i to Thing, and m is the total number of paths in the ontology

## WMCOnto - Weighted Method Count

Mean number of properties and relationships per class WMCOnto(∑|PCi|+∑|RCi|) ∕∑|Ci| , where Ci is the i-th class in the ontology

## DITOnto - Depth of subsumption hierarchy

Length of the largest path from Thing to a leaf class DITOnto=Max (∑D|Ci|), where Ci are the classes and D|Ci| is the length of the path from the i-th leaf class of the ontology to Thing

## NACOnto - Number of Ancestor Classes

Mean number of ancestor classes per leaf class. It is the number of direct superclasses per leaf class NACOnto=∑|SupC(Leaf)i|/∑|C(leaf)i)|

## NOCOnto - Number of Children

Mean number of direct subclasses. It is the number of relationships divided by the number of classes minus the relationships of Thing NOCOnto=∑| RCi| ∕(∑|Ci| -| RThing|)

## CBOOnto - Coupling between Objects

Number of related classes. It is the average number of the direct parents per class minus the relationships of Thing CBOOnto=∑|SupCi|/(∑|Ci| -| RThing|)

## RFCOnto - Response for a class

Number of properties that can be directly accessed from the class RFCOnto=(∑|PCi|+∑|SupCi|/(∑|Ci|

## NOMOnto - Number of properties

Number of properties per class NOMOnto=∑| PCi|∕∑|Ci|

## RROnto - Properties Richness

Number of properties defined in the ontology divided by the number of relationships and properties RROnto=∑| PCi| ∕∑(|RCi| + ∑|Ci|)

## AROnto - Attribute Richness

Mean number of attributes per class AROnto=∑|AttCi|∕∑|Ci|

## INROnto - Relationships per class

Mean number of relationships per class INROnto=∑| RCi| /∑|Ci|

## CROnto - Class Richness

Mean number of instances per class CROnto=∑| ICi| / ∑|Ci |; where ICi, is the set of individuals of the Ci

## ANOnto - Annotation Richness

Mean number of annotations per class ANOnto=∑| ACi| /∑|Ci|; where Ci is the i-th class in the ontology

## TMOnto - Tangledness

Mean number of parents per class, of properties and relationships per class TMOnto=∑| RCi| /∑|Ci|-∑|C(DP)i|; where Ci is the i-th class in the ontology and C(DP)i is thei-th class in the ontology with more than one direct parent.

# Annex B: OOPS! Pitfalls Catalog

## OOPS! Author Introduction

Here you can find a catalogue of pitfalls that usually appear when developing ontologies. Some of them are very common and have been identified by several works about ontology evaluation (see [References](http://oeg-lia3.dia.fi.upm.es/oops/catalogue.jsp#references)).

We would like to help you to find as many pitfalls as possible in your ontology developments. However, some of them depend on the domain being modelled or the requirements specified for each particular ontology. Up to now, OOPS! can identify semi-automatically those pitfalls in the catalogue with the title in **bold**. We encourage you to keep an eye of those pitfalls that OOPS! is not able to detect yet. It is a good idea to revise the ontology manually looking for them.

## Pitfalls

* P01. Creating polysemous elements: an ontology element whose name has different meanings is included in the ontology to represent more than one conceptual idea. For example, the class “Theatre” is used to represent both the artistic discipline and the place in which a play is performed.
* **P02. Creating synonyms as classes:** several classes whose identifiers are synonyms are created and defined as equivalent. As an example we could define “Car”, “Motorcar” and “Automobile” as equivalent classes. Another example is to define the classes “Waterfall” and “Cascade” as equivalents. This pitfall is related to the guidelines presented in [2] which explain that synonyms for the same concept do not represent different classes.
* **P03. Creating the relationship “is” instead of using ''rdfs:subClassOf'', ''rdf:type'' or ''owl:sameAs'':** the “is” relationship is created in the ontology instead of using OWL primitives for representing the subclass relationship (“subclassOf”), the membership to a class (“instanceOf”), or the equality between instances (“sameAs”). An example of this type of pitfall is to define the class “Actor” in the following way ‘Actor ≡ Person ∩ ∃interprets.Actuation ∩ ∃is.Man’. This pitfall is related to the guidelines for understanding the “is-a” relation provided in [2].
* **P04. Creating unconnected ontology elements:** ontology elements (classes, relationships or attributes) are created with no relation to the rest of the ontology. An example of this type of pitfall is to create the relationship “memberOfTeam” and to miss the class representing teams; thus, the relationship created is isolated in the ontology.
* **P05. Defining wrong inverse relationships:** two relationships are defined as inverse relations when they are not necessarily inverse. For example, something is sold or something is bought; in this case, the relationships “isSoldIn” and “isBoughtIn” are not inverse.
* **P06. Including cycles in the hierarchy [1, 2]:** a cycle between two classes in the hierarchy is included in the ontology, although it is not intended to have such classes as equivalent. That is, some class A has a subclass B and at the same time B is a superclass of A. An example of this type of pitfall is represented by the class “Professor” as subclass of “Person”, and the class “Person” as subclass of “Professor”.
* **P07. Merging different concepts in the same class:** a class is created whose identifier is referring to two or more different concepts. An example of this type of pitfall is to create the class “StyleAndPeriod”, or “ProductOrService”.
* **P08. Missing annotations:** ontology terms lack annotations properties. This kind of properties improves the ontology understanding and usability from a user point of view.
* P09. Missing basic information: needed information is not included in the ontology. Sometimes this pitfall is related with the requirements in the ORSD that are not covered by the ontology. Other times it is related with knowledge that could be added to the ontology in order to make it more complete. An example of this type of pitfall is to create the relationship “startsIn” to represent that the routes have a starting point in a particular location; and to miss the relationship “endsIn” to show that a route has an end point. Another example is to create the relationship “follows” when modelling order relations; and do not create its inverse relationship “precedes”.
* **P10. Missing disjointness [1, 2, 3]:** the ontology lacks disjoint axioms between classes or between properties that should be defined as disjoint. For example, we can create the classes “Odd” and “Even” (or the classes “Prime” and “Composite”) without being disjoint; such representation is not correct based on the definition of these types of numbers.
* **P11. Missing domain or range in properties:**relationships and/or attributes without domain or range (or none of them) are included in the ontology. There are situations in which the relation is very general and the range should be the most general concept “Thing”. However, in other cases, the relations are more specific and it could be a good practice to specify its domain and/or range. An example of this type of pitfall is to create the relationship “hasWritten” in an ontology about art in which the relationship domain should be “Writer” and the relationship range should be “LiteraryWork”. This pitfall is related to the common error when defining ranges and domains described in [3].
* **P12. Missing equivalent properties:** when an ontology is imported into another, classes that are duplicated in both ontologies are normally defined as equivalent classes. However, the ontology developer misses the definition of equivalent properties in those cases of duplicated relationships and attributes. For example, the classes “CITY” and “City” in two different ontologies are defined as equivalent classes; however, relationships “hasMember” and “has-Member” in two different ontologies are not defined as equivalent relations.
* **P13. Missing inverse relationships:** this pitfall appears when a relationship (except for the symmetric ones) has not an inverse relationship defined within the ontology. For example, the case in which the ontology developer omits the inverse definition between the relations “hasLanguageCode” and “isCodeOf”, or between “hasReferee” and “isRefereeOf”.
* P14. Misusing ''owl:allValuesFrom'' [3]: this pitfall can appear in two different ways. In the first, the anomaly is to use the universal restriction (“allValuesFrom”) as the default qualifier instead of using the existential restriction (“someValuesFrom”). This means that the developer thinks that “allValuesFrom” implies “someValuesFrom”. In the second, the mistake is to include “allValuesFrom” to close off the possibility of further additions for a given property. An example of this type of pitfall is to define the class “Book” in the following way ‘Book ≡ ∃producedBy.Writer ∩ ∀uses.Paper’ and closing the possibility of adding “Ink” as an element used in the writing.
* P15. Misusing “not some” and “some not” [3]: to mistake the representation of “some not” for “not some”, or the other way round. An example of this type of pitfall is to define a vegetarian pizza as any pizza which both has some topping which is not meat and also has some topping which is not fish. This example is explained in more detail in [3].
* P16. Misusing primitive and defined classes [3]: to fail to make the definition ‘complete’ rather than ‘partial’ (or ‘necessary and sufficient’ rather than just ‘necessary). It is critical to understand that, in general, nothing will be inferred to be subsumed under a primitive class by the classifier. This pitfall implies that the developer does not understand the open world assumption. A more detailed explanation and examples can be found in [3].
* P17. Specializing a hierarchy exceedingly: the hierarchy in the ontology is specialized in such a way that the final leaves cannot have instances, because they are actually instances and should have been created in this way instead of being created as classes. Authors in [2] provide guidelines for distinguishing between a class and an instance when modelling hierarchies. An example of this type of pitfall is to create the class “RatingOfRestaurants” and the classes “1fork”, “2forks”, and so on, as subclasses instead of as instances. Another example is to create the classes “Madrid”, “Barcelona”, “Sevilla”, and so on as subclasses of “Place”. This pitfall could be also named “Individuals are not Classes”.
* P18. Specifying the domain or range exceedingly [2, 3]: not to find a domain or a range that is general enough. An example of this type of pitfall is to restrict the domain of the relationship “isOfficialLanguage” to the class “City”, instead of allowing also the class “Country” to have official language or a more general concept such as “GeopoliticalObject”.
* **P19. Swapping intersection and union:** the ranges and/or domains of the properties (relationships and attributes) are defined by intersecting several classes in cases in which the ranges and/or domains should be the union of such classes. An example of this type of pitfall is to create the relationship “takesPlaceIn” with domain “OlympicGames” and with range the intersection of the classes “City” and “Nation”. Another example can be to create the attribute “Name” for the classes “City” and “Drink” and to define its domain as the intersection of both classes. This pitfall is related to the common error that appears when defining ranges and domains described in [3] and also related to the guidelines for defining these elements provided in [2].
* **P20. Misusing ontology annotations:** The contents of some annotation properties are swapped or misused. An example of this type of pitfall is to include in the Label annotation of the class “Crossroads” the following sentence ’the place of intersection of two or more roads’; and to include in the Comment annotation the word ‘Crossroads’.
* **P21. Using a miscellaneous class:** to create in a hierarchy a class that contains the instances that do not belong to the sibling classes instead of classifying such instances as instances of the class in the upper level of the hierarchy. This class is normally named “Other” or “Miscellaneous”. An example of this type of pitfall is to create the class “HydrographicalResource”, and the subclasses “Stream”, “Waterfall”, etc., and also the subclass “OtherRiverElement”.
* **P22. Using different naming criteria in the ontology:** Ontology elements are not named using the same convention within the whole ontology. It is considered a good practice that the rules and style of lexical encoding for naming the different ontology elements is homogeneous within the ontology. One possibility for rules is that concept names start with capital letters and property names start with non-capital letters. In the case of style, there are different options such as camel case, hyphen style, underscore style, and the combinations. Some notions about naming conventions are provided in [2]. For example, this pitfall appears when a class is named by starting with upper case, e.g. “Ingredient”, and its subclasses by starting with lower case, e.g. “flour”, “milk”, etc.
* P23. Using incorrectly ontology elements: an ontology element (class, relationship or attribute) is used to model a part of the ontology that should be modelled with a different element. A particular case of this pitfall regarding to the misuse of classes and property values is addressed in [2]. An example of this type of pitfall is to create the relationship “isEcological” between an instance of “Car” and the instance “Yes” or “No”, instead of creating the attribute “isEcological” whose range is Boolean.
* **P24. Using recursive definition:** an ontology element is used in its own definition. For example, it is used to create the relationship “hasFork” and to establish as its range the following ’the set of restaurants that have at least one value for the relationship “hasFork”.
* **P25. Defining a relationship inverse to itself:** a relationship is defined as inverse of itself. In this case, this property could have been defined as “owl:SymmetricProperty” instead. An example of this type of pitfall is to create the relationship “hasBorderWith” and to state that “hasBorderWith” is its inverse relationship.
* **P26. Defining inverse relationships for a symmetric one:** a relationship is defined as “owl:SymmetricProperty” and there is also a relationship (it could be itself or another relationship) defined as its inverse. For example, the symmetric relationship “farFrom” has an inverse relationships defined, e.g. itself, “farFrom”.
* **P27. Defining wrong equivalent relationships:** two relationships are defined as equivalent relations when they are not necessarily. For example, we can mix up common relationships that could hold between several types of entities, as "hasPart" defined between human body parts and the same relationship relating research plans as part of research projects.
* **P28. Defining wrong symmetric relationships:** a relationship is defined as symmetric when it is not necessarily. The domain defined for a symmetric relationship is different from its range. This could happen because the relationship might not be symmetric, for example defining the relation "pastProject" between the concepts "Agent" and "Project". This situation can also appear due to the domain and range are too specific, for example, if we define the symmetric relationship "hasSpouse" between the concepts "Man" and "Woman" instead of using the concept "Person" both as domain and range of such a relationship.
* **P29. Defining wrong transitive relationships:** a relationship is defined as transitive when it is not necessarily. The domain defined for a transitive relationship is different from its range. An example of this type of error is to create the relationship "participatesIn", which domain is the union of the concepts "Team" and "Individual" and which range is the concept "Event", defining the relationship as transitive.
* **P30. Missing equivalent classes:** when an ontology is imported into another, classes with the same conceptual meaning that are duplicated in both ontologies should be defined as equivalent classes, in order to benefit the interoperability among both ontologies. However, the ontology developer misses the definition of equivalent classes in those cases of duplicated concepts. An example of this pitfall can be not to have the equivalent knowledge explicitly defined between 'Trainer' (class in the imported ontology) and 'Coach' (class in the ontology about sports being developed).
* **P31. Defining wrong equivalent classes:** two classes are defined as equivalent when they are not necessarily. For example, defining “Car” as equivalent to “Vehicle”.
* **P32. Several classes with the same label:** two or more classes have the same content in the rdfs:Label annotation. In some cases they could be defined as equivalent classes (e.g. if they are defined in different namespaces) or they could be replaced by a single class with one or more labels (e.g. if they are defined in the same namespace).
* **P33. Creating a property chain with just one property:** A property chain including only one property in the antecedent part is created. In this case it could be more appropriate to create the property in the consequent equivalent to the one in the antecedent of the chain. For example, if the following property chain is created: isInChargeOf -> supervises.
* **P34. Untyped class [4]:** a resource is used as a class, e.g. appearing as the object of an rdf:type, rdfs:domain, or rdfs:range statement, or as the subject or object of an rdfs:subClassOf statement, without having been declared as a Class.
* **P35. Untyped property [4]:** a resource is used as a property, e.g. appearing as the subject or object of an rdfs:subPropertyOf statement, without having been declared as a rdf:Property or some subclass of it.
* *(Linked Data Feature)***P36. URI contains file extension:** Guidelines in [5] suggest avoiding file extension in persistent URIs, particularly those related to the technology used, as for example “.php” or “.py”. In our case we have adapted it to the ontology web languages used to formalized ontologies and their serializations. In this regard, we consider as pitfall including file extensions as “.owl”, “.rdf”, “.ttl”, “.n3” and “.rdfxml” in an ontology URI. An example of this pitfall is defining an ontology uri as “http://www.biopax.org/release/biopax-level3.owl" containg the extension “.owl” related to the technology used.
* *(Linked Data Feature)***P37. Ontology not available:** This bad practice is about not meeting LOD1 from Linked Data star system that stars “On the web” and LDV1 that says “Publish your vocabulary on the Web at a stable URI”. An example of this pitfall could be the following case: “Ontology Security (ontosec)” which URI is http://www.semanticweb.org/ontologies/2008/11/OntologySecurity.owl and it is not available online as RDF nor as HTML (at the moment of carrying out this work).
* **P38. No OWL ontology declaration:** The owl:Ontology tag aims at gathering metadata about a given ontology as version information, creation date, etc. It is also used to declare the inclusion of other ontologies. Not declaring this tag is consider as a bad practice for owl ontologies as it is a symptom of not providing useful metadata as proposed in “LDV2”. Example: “Creative Commons Rights Expression Language (cc)” ontology with URI http://creativecommons.org/ns does not have any owl:Ontology declaration in its RDF file even though there are other OWL elements used as, for example, owl:equivalentProperty.
* *(Linked Data Feature)***P39. Ambiguous namespace:** In the case of not having defined the ontology URI nor the xml:base namespace the ontology namespaces is matched to the file location. This situation is not desirable as the location of a file might change while the ontology should remain stable as proposed in “LDV1”. Example: “Basic Access Control ontology (acl)” with URI http://www.w3.org/ns/auth/acl has no owl:Ontology tag nor xml:base definition.
* *(Linked Data Feature)***P40. Namespace hijacking:** this bad practice refers to the situation when an ontology is reusing or referring to terms from other namespaces that are not defined in such namespace. This is an undesirable situation as no information could be retrieve when looking up those undefined terms, in addition, there would be no meaning or semantic behind them. In addition this practice is against Linked Data publishing guidelines provided in [6] “Only define new terms in a namespace that you control.” Example: the “WSMO-Lite Ontology (wl)” which URI is http://www.wsmo.org/ns/wsmo-lite#, uses http://www.w3.org/2000/01/rdf-schema#Property" that is not defined in the rdf namespace (http://www.w3.org/2000/01/rdf-schema#) instead of using http://www.w3.org/1999/02/22-rdf-syntax-ns#Property, that is ac-tually defined in the rdfs namespace (http://www.w3.org/1999/02/22-rdf-syntax-ns#).

**References**

1. ↑ A. Duque-Ramos, J.T. Fernández-Breis, R. Stevens, N. Aussenac-Gilles, Oquare: a square-based approach for evaluating the quality of ontologies Journal of Research and Practice in Information Technology 43(2011) 159-73.
2. ↑ ISO25000 2005. ISO/IEC 25000 2005, Software engineering - Software product quality requirements and evaluation (SQuaRE) - guide to square (ISO/IEC 25000). Geneva, Switzerland: International Organization for Standardization.
3. Fernández-Breis, J., Egaña, M. and Stevens, R. (2009): A quality evaluation framework for bio-ontologies. In: PRECEDINGS, N., ed. ICBO International Conference on Biomedical Ontology, University at Buffalo, NY. 136–139
4. ↑ GANGEMI, A., CATENACCI, C., CIARAMITA, M. and LEHMANN, J. (2006): Modelling ontology evaluation and validation. Semantic Web: Research and Applications, Proceedings, 4011: 140 –154.
5. ↑ STEVENS, R. and LORD, P. (2009): Application of ontologies in bioinformatics. In: BERNUS, P., BŁAŻEWICS, J., SCHMIDT, G., SHAW, M., STAAB, S. and STUDER, R. (eds.) Handbook on Ontologies. Springer Berlin Heidelberg.
6. ↑ <http://www.obofoundry.org/wiki/index.php/FP_002_format>
7. ↑ FAHAD M., QADIR, M. (2008). A Framework for Ontology Evaluation. In Proceedings International Conference on Conceptual Structures (ICCS'08), Toulouse, France, July, page 711.
8. ↑ KIU, C.-C., & LEE, C.-S. (2006). Ontology Mapping and Merging through OntoDNA for Learning Object Reusability. Educational Technology & Society, 9 (3), 27-42
9. ↑ MAEDCHE A., ZACHARIAS V. (2002). Clustering Ontology-based Metadata in the Semantic Web. In Proceedings 6th European Conference on Principles of Data Mining and Knowledge Discovery, Springer-Verlag, London, Uk, 348-360
10. ↑ CHIDAMBER, S.R. and KEMERER, C.F. (1994): metric suite for object oriented design. IEEE Transactions on SoftwareEngineering, 467–493.
11. ↑ LI, W. and HENRY, S. (1993): Object-oriented metrics that predict maintainability. Journal of Systems and Software, 23:111–122.
12. ↑ YAO, H., ORME, A. and ETZKORN, L. (2005): Cohesion metrics for ontology design and application. Journal of ComputerScience, 1.
13. ↑ TARTIR, S. and ARPINAR, I.B. (2007): Ontology evaluation and ranking using OntoQA. ICSC 2007: International Conference on Semantic Computing, Proceedings, 185–192.
14. ↑ GANGEMI, A., CATENACCI, C., CIARAMITA, M. and LEHMANN, J. (2006): Modelling ontology evaluation and validation. Semantic Web: Research and Applications, Proceedings, 4011: 140 –154.
15. "A Pitfall Catalogue and OOPS!: An Approach to Ontology Validation", presentation by Ms. Maria Poveda Villalon (Universidad Politécnica de Madrid), Dr. Mari Carmen Suarez Figueroa (Universidad Politécnica de Madrid) and Dr. Asuncion Gomez Perez (Universidad Politécnica de Madrid) - given at the OntologySummit2013 on 2013.01.31