Knowledge and Information Systems (2021) 63:867–903
https://doi.org/10.1007/s10115-021-01545-9

REGULAR PAPER

Towards metrics-driven ontology engineering

Alba Fernández-Izquierdo[1] · María Poveda-Villalón[1] · Asunción Gómez-Pérez[1] · Raúl García-Castro[1]

Received: 21 February 2019 / Revised: 4 December 2020 / Accepted: 30 December 2020 / Published online: 23 February 2021

Abstract

The software engineering field is continuously making an effort to improve the effectiveness of the software development process. This improvement is performed by developing quantitative measures that can be used to enhance the quality of software products and to more accurately describe, better understand and manage the software development life cycle. Even if the ontology engineering field is constantly adopting practices from software engineering, it has not yet reached a state in which metrics are an integral part of ontology engineering processes and support making evidence-based decisions over the process and its outputs. Up to now, ontology metrics are mainly focused on the ontology implementation and do not take into account the development process or other artefacts that can help assessing the quality of the ontology, e.g. its requirements. This work envisions the need for a metrics-driven ontology engineering process and, as a first step, presents a set of metrics for ontology engineering which are obtained from artefacts generated during the ontology development process and from the process itself. The approach is validated by measuring the ontology engineering process carried out in a research project and by showing how the proposed metrics can be used to improve the efficiency of the process by making predictions, such as the effort needed to implement an ontology, or assessments, such as the coverage of the ontology according to its requirements.

Keywords Metrics · Ontology engineering · Requirements · Ontology development

✉ Alba Fernández-Izquierdo
albafernandez@fi.upm.es

María Poveda-Villalón
mpoveda@fi.upm.es

Asunción Gómez-Pérez
asun@fi.upm.es

Raúl García-Castro
rgarcia@fi.upm.es

1 Ontology Engineering Group, Escuela Técnica Superior de Ingenieros Informáticos, Universidad Politécnica de Madrid, Madrid, Spain

1 Introduction

Software metrics play an important role in the software engineering field, supporting both development and managerial decision-making during the software life cycle. These software metrics are not only related to the source code itself, but also to other artefacts that are part of the software main product (e.g. requirements, documentation and tests) and to the activities needed to obtain such artefacts. This diversity of metrics enables software engineers to have enough information to make different types of predictions, assessments and trade-offs, such as effort and time predictions or software quality analysis [15].

Similarly, in the ontology engineering field different metrics exist which try to assess the quality of ontologies by measuring reliability, reusability or cohesion, among other aspects. However, the metrics proposed until now are mostly focused on the ontology implementation, and they do not take into account other artefacts produced during the ontology development process or even the development process itself. Moreover, they only consider the structure of the ontology [55].

There are some works that are sceptical about the use of metrics [34], which claim that people expect too much from them as a management tool and that the importance associated to a metric leads to the risk of chasing the metric rather than focusing on producing good products. However, several works and projects (e.g. [15,45,56]) defend that appropriate metrics can help to identify potential problematic areas that may lead to issues or errors in fields such as software engineering. In any case, to ascertain if a project or product is healthy, i.e. if it is maintainable, readable, stable and simple, some measures of its health are needed. Therefore, having specific metrics to control the ontology development process and the artefacts generated from it can lead to evidence-based decision-making and to an improvement in the efficiency of the development process.

In fact, in real-world projects it is necessary to be aware of the status of the development process in order to be able to plan and manage it in an efficient way. In the ontology engineering field, metrics should provide information about the resources and the development time needed to implement the ontology, which is important for the manager of the project to be able to control the costs of the development and to estimate the effort needed for future development tasks. This required effort can also be influenced by the complexity of the ontology requirements that need to be implemented and, because of that, it is also useful to provide measures that indicate how complex a requirement is. As an example, a simple ontological requirement that states "What is a sensor?" usually leads to a solely new class in the ontology, but a complex one such as "Users can have different roles in the organisation" includes several properties. Thus, it is probable that this latter one will require more effort to be conceptualised and implemented. Metrics should also provide information about some aspects of ontology coverage, such as the number of requirements implemented by the ontology, which are important for the ontology developers to be able to know whether the ontology is complete.

Taking into account that ontologies are being increasingly adopted in information systems, it can be considered that ontology development processes may benefit from the application of common software engineering practices in order to be aligned with software development and to ease the integration between software and ontologies. Therefore, inspired by software engineering practices and metrics, where metrics are widely integrated in the software development process, this paper defines ontology engineering metrics that go beyond the ontology implementation and take into account the ontology development process and other artefacts generated from it, such as the requirements specification document or the test suite generated from it.

This work aims to address the following research question: "*Does the use of metrics allow extracting information that supports taking decisions during ontology development?*". To address this research question, a set of hypotheses has been defined to analyse the effect of the different artefacts involved in the development process. This set of hypotheses does not cover the whole question, but it allows to identify the information that could be extracted from metrics in several situations:

H1. The number of requirements influences the number of defined tests.
H2. The number of requirements influences the development time of the ontology.
H3. The number of requirements influences the size of the ontology.
H5. The complexity of requirements influences the number of defined tests.
H6. The complexity of requirements influences the development time of the ontology.
H7. The complexity of requirements influences the size of the ontology.
H8. The volatility of requirements influences the volatility of the ontology.
H9. The number of tests influences the number of tested terms.
H10. The number of tested terms influences the coverage of the ontology.

To validate these hypotheses, the metrics have been applied in a real use case allowing the actors involved in the development process to know the status of the process, the ontology and how both the process and the ontology evolve over time.

This paper is organised as follows. Section 2 presents the state of the art in software and ontology engineering metrics. Section 3 presents the methodology carried out in order to propose the set of metrics, while such metrics are described in Sect. 4. Finally, Sect. 5 shows the validation of the hypotheses in a real-world scenario, Sect. 6 presents the discussion related to the metrics and the obtained results, and Sect. 7 presents the conclusions obtained and gives an outlook into future work.

2 State of the art

This section presents a summary of metrics for software and ontology engineering. The criteria followed for including metrics in this section were based on the availability of the metrics and the specific formulas to calculate them, as well as on the avoidance of duplicated metrics.

Software metrics, widely used to assess the quality of software engineering products and the efficiency of the development process, are used in this work as inspiration to improve the ontology metrics defined in the literature so far.

2.1 Software engineering metrics

In software engineering, metrics have been of interest for years both from the research perspective and in the industrial practice, providing several benefits to software development, such as defect prediction [33] or quality assessment [24]. However, the most significant benefit of these metrics is the ability to provide information to support managerial decision making during software development and testing. In fact, the long-term goal of software measurement is to use measures to make judgements about software quality, which may influence managerial decision making. For example, managers can use process measurements to decide if process changes should be made and product metrics to help estimate the effort required to make software changes [47].

The first software metric dates back to the mid-1960s when the *Lines of Code* metric (*LOC* or *KLOC*, for thousands of lines of code) was used as the basis for measuring programming productivity and effort [15]. In the late 1960s, *LOC* was also used as the basis for measuring program quality; it was used to calculate derived metrics such as the number of defects per *KLOC*. The critical assumption of this metrics is that it considers the size of the program as the key aspect to measure quality and effort.

From the 1970s, new measurement techniques were developed, starting with measures for software complexity, such as the *Cyclomatic complexity* [32], and measures of functional size, such as the *Functional points* metric [1].

Nowadays, software metrics are divided into two categories, namely **product metrics**, which are used to measure internal attributes of a software system, and **process metrics** which are used to obtain measurements about the software process [47]. Product metrics can be related to the different artefacts produced in the software development process, such as the source code, the requirements specification or the user documentation. Process metrics are usually related to the software life cycle, although there are also process metrics which are associated to different artefacts, such as the requirements specification.

Regarding **source code product metrics**, the work presented by Fenton [14] includes the *COCOMO 2.0 model* [4] for predicting project cost, effort and calendar time.*COCOMO 2.0* provides different effort estimating models based on the stage of the development of the project. As inputs, *COCOMO 2.0* requires the level of information available to the user at that time. Thus, during the earliest conceptual stages of a project, the models uses object points, e.g. screens, reports and modules of the language, to compute effort. During the early design stages, it uses as input the unadjusted function points, which represent the functional size of the software implementation based on the logical view of an application. Finally, once an architecture has been selected, *LOC* is the input to the COCOMO model.

Dealing with the same topic, Kan's work [24] includes the *Changed source instructions metric (CSI)*, which counts the new and changed source instructions in a software implementation. Another product metric presented in this collection is the *Problems per user month (PUM)*, which represents the total number of problems that customers reported for a time period in addition to the total number of license-months of the software during the period. In addition to this, Sommerville [47] describes the *Fan-in*, *Fan-out* and *Depth of conditional nesting* metrics. *Fan-in* is a measure of the number of functions or methods that call another function or method and *Fan-out* is the number of functions or methods that are called by a given function or method. *Depth of conditional nesting* measures the depth of nesting of "if" statements in a program.

Concerning **requirements specification product metrics**, Fenton and Bieman [14] points out that the length of the requirements specification might be related to the complexity of the code and, therefore, it is useful to measure it. An example of this type of metrics is the *Specification weight*, formerly the *Bang* metrics [12], that involves two metrics: *Function bang* metric, which is based on the number of functional primitives in any data flow diagram, and *Data bang* metric, which measures the number of entities in the entity relationship model. Costello and Liu [6] present a collection of metrics which includes *Requirements fault density* and *Requirements completeness*. *Requirements fault density* indicates the number of requirement faults that are initially detected during test execution or during the analysis of the tests after being executed and *Requirements completeness* indicates the completeness of all sections of requirement specifications and the degree of decomposition of allocated higher level requirements.

Davis et al. [7] define a set of 18 product metrics to measure the software requirements specification (SRS) quality, which are associated to attributes of the requirements specification document. The authors provide formulas for the following metrics: (1) *Unambiguity*, which measures the percentage of requirements that have been interpreted in an unique manner by all of its reviewers; (2) *Completeness*, which measures different aspects related to the completeness of a SRS; (3) *SRS correctness*, which calculates the percentage of the requirements specification that has been validated; (4) *Understandability*, which measures the percentage of requirements for which all reviewers thought they understood; (5) *SRS verifiability*, which considers the cost and time necessary to verify a requirement; (6) *SRS internal consistency*, which calculates the percentage of requirements that are not in conflict; (7) *SRS external consistency*, which measures the percentage of requirements that are consistent with all other documents; (8) *SRS conciseness*, which counts the number of pages in the requirement specification document; (9) *SRS design independence*, which measures the percentage of possible solution systems that are eliminated by adding the overly constraining requirements; (10) *SRS annotated with relative importance*, which calculates the percentage of requirements that are annotated with their importance; (11) *Non-redundant SRS*, which calculates the percentage of requirements that are stated more than once; and (12) *Traced*, which measures the percentage of requirements that are traced.

Lastly, Iqbal et al. [23] introduce a set of 9 requirements specification product and process metrics, among which are the *Uniqueness*, the *Modifiable*, the *Misinterpreted requirements*, the *Requirement testing* and the *SRS quality* metrics.

Fenton and Bieman [14] also describe several **test case product metrics**, such as *Branch coverage*, which finds the set of paths such that every edge lies on at least one path, and *Test effectiveness ratio*, which it is obtained by dividing the number of tests exercised at least once and the total number of tests.

With respect to **user documentation product metrics**, Sommerville also mentions *Fog index* [21], which indicates the average length of words and sentences in documents and is used in software engineering to measure the readability of the user documentation. The higher the value of a document's Fog index, the more difficult the document is to understand.

To conclude, in relation to **process metrics** Fenton and Bieman [14] describes *Productivity measure*, where the size of the output is compared with the amount of effort. Kan [24] describes a set of process quality metrics which tracks defect arrival during formal machine testing. These metrics include *Defect density during machine testing* metric, which measures the number of defects per LOC or function point, and *Defect arrival pattern during machine testing*, which calculates time between failures. Sommerville distinguishes three types of metrics: (a) the time taken for a particular process to be completed, e.g. the total time devoted to the software development process, (b) the resources required for a particular process, e.g. total effort in person-days, and (c) the number of occurrences of a particular event, e.g. the number of defects discovered during code inspection. Thus, these metrics could be related to any artefact generated during the development process. Moreover, Rahman and Devanbu [42] gathered a set of file-based process metrics which can be applied to the code, the requirements, the tests or the documentation. These metrics include: (1) *COMM*, which measures the number of commits made to a file; (2) *ADEV*, which is the number of developers who changed the file; (3) *DDEV*, which is the cumulative number of distinct developers contributed to this file up to this release; (4) *ADD* and *DEL*, which are the normalised (by the total number of added and deleted lines) added and deleted lines in the file; (5) *OWN*, which measures the percentage of the lines authored by the highest contributor of a file; (6) *MINOR*, which measures the number of contributors who authored less than 5% of the code in that file; (7) *OEXP*, which measures the experience of the highest contributor of that file using the percent

Table 1 Summary of software engineering metrics

Artefacts generated during the development process			
Source code	Requirements specification	Tests	User documentation
Metrics category			
Product metrics			
LOC, Cyclomatic complexity, Function points, COCOMO, CSI, PUM, Depth conditional nesting, Fan-in/Fan-out	*Specification weight, Fault density, Unambiguity, Completeness, Correctness, Understandability, Verifiability, Internal and external consistency, Conciseness, Design independance, Annotation, Non-Redundant Requirements Specification, Uniqueness, Modifiable, Misinterpreted requirements, testing*	*COCOMO, Branch Coverage, Effectiveness Ratio*	*COCOMO, Fog index*
Process metrics			
Life cycle (metrics not related to any artefact)			
Productivity, Defect density during machine testing,			
Defect arrival pattern during machine testing, Time devoted to the process,			
Number of occurrences of a particular event, Resources required for a particular process, Changed requirements			
COMM, ADEV, DDEV, ADD, DEL, OWN, MINOR, OEXP, EXP			

of lines he authored in the project at a given point in time; and (8) *EXP*, which measures the geometric mean of the experiences of all the developers. There are also process metrics which are related to specific artefacts, such as the *Changed requirements* metric defined in the work presented by Iqbal et al. [23] which is related to the requirements specification. This latter metric was also introduced by Costello and Liu [6] as *Requirements volatility*.

Table 1 summarises the software engineering metrics described in this section according to the category and to the associated artefact from which each metric is extracted. From this table, it can be concluded that there are several metrics for each of the category and artefact generated from the software development process.

2.2 Ontology engineering metrics

In the ontology engineering field, metrics have also been an important research topic due to the fact that they can help assessing and qualifying ontologies or detecting ontology problems. Different proposals for ontology metrics have been proposed in the past years, measuring several aspects of the ontology implementation. This section provides an overview of the most relevant of those ontology metrics.

The work presented by Duque-Ramos et al. [13] already creates a classification of ontology metrics; however, their proposed classification is based on the use that is made of the metrics, e.g. functional adequacy, compatibility or maintenance, among others. Instead, the metrics presented in this work are classified according to the product from which the metrics are extracted. Due to the fact that all the metrics in the literature are extracted from the ontology implementation; the classification presented in the following paragraphs is based on the different ontology implementation elements needed to calculate them, e.g. ontology classes, ontology hierarchies, ontology properties and ontology axioms.

Concerning metrics related to **ontology classes**, Yao et al. [57] propose a set of metrics which try to measure the relatedness of ontologies. The metrics they propose include *Number of root classes (NoR)*, which counts the number of classes that have no superclass and *Number of leaf classes (NoL)*, which counts the number of classes that have no subclass. Yang et al. [58] also propose a set of metrics related to ontology classes to evaluate the ontology regarding its complexity; this set of metrics includes *Total number of concepts or classes (TNOC)*. Another set of ontology class metrics is the one presented by Kang et al. [25], whose metrics are used to predict reasoning performance. These metrics include *Number of children (NOC)*, and *Expression richness (RCH)*, which measures the ratio between the number of anonymous class expressions and the total number of class expressions. Orme et al. [36] propose three different metrics to measure the level of coupling in ontologies using ontology classes, namely *Number of external classes (NEC)* and *Number of references to external classes (REC)*. Gangemi et al. [19] define a set of metrics related to the ontology structure which includes *Absolute leaf cardinality*, which measures the number of leaf nodes in an ontology and is similar to *Number of leaf classes (NoL)* already presented by Yao et al. [57]. There are also tools, such as the ontology editors Protégé[1] and WebProtégé[2] which calculate ontology metrics. These editors calculate, among other metrics, *Number of classes* and *Max siblings*, which displays the maximum number of siblings for any class. The tool OntoMetrics [29] also provides, besides the metrics already defined in Gangemi et al. [19], a new metric related to ontology classes named *Equivalence ratio*.

Regarding metrics related to the **ontology hierarchy**, Yao et al. [57] propose the *Average depth of inheritance of all leaf nodes (ADIT-LN)*, which is defined as the sum of depths of all paths divided by the total number of paths. Gangemi et al. [19] also propose a set of metrics related to the hierarchy to measure the ontology structure. Some of the ontology structure metrics proposed by these authors are *Tangledness*, which is related to the multi-hierarchical nodes of a graph; *Absolute depth*, which is related to the cardinality of paths in a graph; or *Absolute breadth*, which is related to the levels in a graph. Zhe et al. [58] propose *Total number of paths (TNOP)* and *Average paths per concept*, which is calculated by dividing *TNOP* by *TNOC*. Finally, Kang et al. [25] propose *Depth of inheritance (DIT)* and *Cyclomatic complexity (CYC)*, which measures the number of linearly independent paths in the ontology graph. Protégé also calculates metrics related to ontology hierarchy such as *Max depth*, which displays the maximum depth of the ontology hierarchy.

With relation to metrics extracted from the **ontology properties**, Zhe et al. [58] propose *Total number of relations (TNOR)* and *Average relations per concept*, which is calculated by dividing TNOR by TNOC. Tartir et al. [51] provide a set of metrics which evaluates ontology design and its potential for rich knowledge representation from the ontology properties, such as *Relationship Richness* and *Attribute Richness*. The first metric reflects the diversity and placement of relations in the ontology, while the second one measures the number of attributes for each class. OntoMetrics [29] also calculates metrics related to ontology properties, such as *Class/Relation ratio* or *Inverse relations ratio*, among others. Finally, DOGMA-MESS [9] proposes a score based on ontology alignment in order to rank ontologies. Such score is

1 https://protege.stanford.edu/.

2 https://webprotege.stanford.edu/.

calculated by checking if the relevance relations defined by the community are included in the definition of the ontology to be measured.

Regarding metrics extracted from **ontology individuals**, Tartir et al. [51] provide a set of metrics which evaluates the placement of instance data within the ontology and the effective usage of the ontology to represent the knowledge modelled in the ontology. This set includes, among others, *Class Richness*, which is related to how instances are distributed across classes and *Average Population*, which is defined as the number of instances divided by the number of classes. OntoMetrics [29] provides new metrics related to individuals, such as *Class Importance*, which is the percentage of instances that belong to classes at the inheritance subtree rooted at the current class with respect to the total number of instances, or *Class Fullness*, which measures the expected number of instances of each class.

With respect to metrics extracted from **ontology axioms**, Ma et al. [30] propose a set of metrics to measure the cohesion of ontologies. They propose *Number of ontology partitions (NOP)*, which measures the number of semantic partitions of a knowledge base,[3] this is, the number of parts in an ontology that are semantically unrelated to each other, *Number of minimally inconsistent subsets (NMIS)*, which measures the minimally inconsistent subsets in a knowledge base, and *Average value of axiom inconsistencies (AVAI)*.

There are also metrics extracted from the **ontology metadata**. Orme et al. [36] propose *Number of referenced includes (RI)*, which is the number of includes in the ontology implementation, and Tartir et al. [51] propose *Readability*, which indicates the existence of human-readable descriptions in the ontology such as comments or labels. Additionally, the tool OntoCheck [44] displays the percentage and absolute number of nodes having exactly, at least or at most a certain number of selectable metadata elements, parents and children, direct superclasses, subclasses and class usages.

Table 2 summarises the metrics described in this section according to the artefact from which they are extracted. The table clearly shows that, even if there is a large amount of work in ontology engineering related to metrics, the metrics found in the literature only analyse the ontology implementation when making measurements, leaving aside the others artefacts obtained during the ontology development process, such as the documentation or the requirements specification. A detailed analysis on the metrics also shows that, even if the authors relate the metrics to different aspects, such as coupling or cohesion, they are mainly focused on the ontology structure, specially on the ontology classes.

It should be mentioned that there are ontology development methodologies, such as DILIGENT [38], SAMOD [37], GOSPL [11] or HCOME [28], which deal with the assessment of ontologies. Furthermore, there are more ontology evaluation approaches and tools, such as ODEClean [16] and OOPS! [39]. However, this state of the art only includes those metrics that are associated to a specific formula for their calculation. Thus, potential metrics that could be extracted from the information stored in the artefacts considered by these methodologies and tools, although such methodologies and tools formally specify how to obtain the information, are out of scope of this paper.

The work presented in this paper is inspired by the work in software metrics and aims to go a step forward regarding ontology metrics by defining metrics not only related to the ontology implementation itself, but also to other artefacts generated during the ontology development process. Furthermore, having multiple ontology development artefacts will enable defining new metrics that integrate data extracted from two or more artefacts.

[3] The authors of the analysed paper refer to the knowledge base as the set of TBox and Abox.

Table 2 Summary of ontology engineering metric

Artefacts generated during the development process			
Ontology implementation	Requirements specification	Tests	User documentation
Metrics category			
Product metrics			
Class metrics (e.g. *TNOC, NoR, NoL*)			
Hierarchy metrics (e.g. *ADIT-LN, Absolute depth*)			
Property metrics (e.g. *TNOR, Relationship richness, DOGMA-MESS ranking score*)	–	–	–
Individual metrics (e.g. *Class Richness, Class Fullness*)			
Axiom properties metrics (e.g. *NOP, AVAI*)			
Metadata metrics (e.g. *RI, Readability, OntoCheck metrics*)			
Process metrics			
Life cycle (metrics not related to any artefact)			
–			

3 Methodology

Along this section, the workflow followed to propose the set of ontology metrics and its classification is described. As shown in Fig. 1, the steps followed during the process were:

1. *Roles identification*: The different roles involved in the ontology development process, their needs and their risks regarding ontology metrics were identified. This step is further described in Sect. 3.1.
2. *Artefacts identification*: The artefacts generated during the ontology development process that could be potentially useful in order to support the above-mentioned needs and risks were identified. As it will be later detailed in Sect. 3.2, these artefacts refer to the ontology implementation, the ontology requirements specification document and the ontology requirements.
3. *Existing ontology metric analysis*: In order to carry out this step, the current state of the art in ontology metrics was analysed, leading to the identification of already existing ontology metrics that could be useful to support part of the identified needs. This review of the state-of-the-art metrics for ontologies was detailed in Sect. 2.2.
4. *Ontology metrics proposal*: The set of new metrics proposed was developed taking into account: a) the existing ontology metrics in the literature; b) the needs of the different actors involved in the ontology development process; and c) the artefacts generated during the ontology development process. These metrics are also classified according to whether they are product or process metrics. In the case of the product metrics, they are also classified according to whether they are base or calculated metrics. The proposed metrics are described in Sect. 4.

3.1 Roles involved in the ontology development process

In order to define the set of ontology engineering metrics, first the roles that should be involved in any ontology development process were identified, along with what they want to monitor

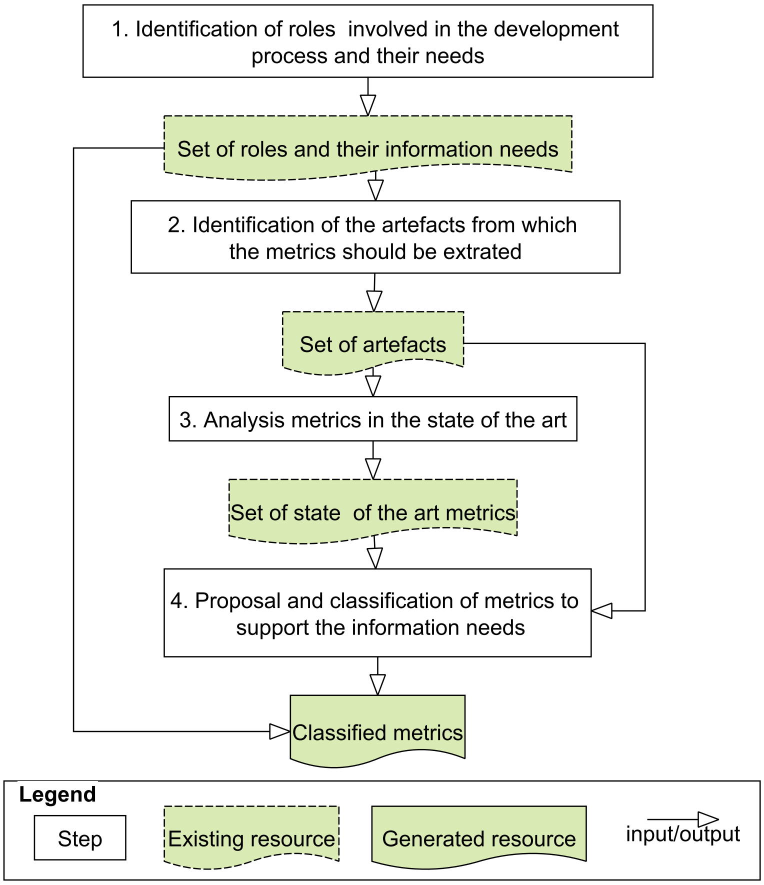

1. Identification of roles involved in the development process and their needs
Set of roles and their information needs
2. Identification of the artefacts from which the metrics should be extrated
Set of artefacts
3. Analysis metrics in the state of the art
Set of state of the art metrics
4. Proposal and classification of metrics to support the information needs
Classified metrics
Legend
Step
Existing resource
Generated resource
input/output

Fig. 1 Activity workflow followed to generate the proposed set of metrics

during this process. Bearing in mind the similarities between the development of ontologies and the development of software, and the fact that the ontology engineering field is facing similar challenges to those seen in software engineering, this role identification was based on the software engineering state of the art [40]. Accordingly, the role identification proposed in the software engineering field was applied to ontology engineering practices. Moreover, different ontology engineering methodologies, such as NeOn [49], SAMOD [37] and UPON [10] were also considered, as well as the goals of the roles involved in their development processes. As a result, the following roles were identified:

- *Team leaders*. People in charge of monitoring and managing the development process and of diagnosing technical and organisational issues. These actors have the following needs:
  - To organise the development tasks.
  - To detect issues that can hinder the development process.
- *Ontology developers*. People responsible of creating the ontologies and all the needed resources to publish and document them. These actors have the following needs:

  - To be aware of whether the ontology they are generating is correct.
  - To be aware of whether the ontology they are generating is complete.

- *Stakeholders*. Customers, users and domain experts who specify the requirements for the ontology to be created. These actors have the following needs:
  - To be aware of whether the ontology they are generating is correct.
  - To be aware of whether the ontology they are generating is complete.
  - To be aware of whether all the requirements they propose are satisfied by the ontology.
  - If they are customers, they need to be aware about the status of the ontology development process.

Also inspired by software engineering, this work takes into account that there are risks that can be monitored using metrics during the development process, such as those mentioned by Pressman [40], namely: (1) **Performance risk**, i.e. the degree of uncertainty that the product will meet its requirements and be fit for its intended use; or (2) **Schedule risk**, i.e. the degree of uncertainty that the project schedule will be maintained and that the product will be delivered on time. The former risk should be monitored by the developers, while the latter one should be monitored by the team leaders.

3.2 Artefacts in the ontology engineering process

The purpose of this work is to show the benefits for ontological engineering of having metrics that go beyond the ontology implementation and extract information from additional ontology engineering artefacts. In this sense, three artefacts are needed to calculate the ontology metrics proposed in this paper, namely:

- The **ontology requirements specification document (ORSD)**, which is the product resultant from the ontology requirements specification activity [49]. In this document, the purpose, requirements and users of the ontology are identified.
- The **ontology implementation**, which is the product resultant from the ontology implementation activity.
- The **ontology requirements test suite (ORTS)**, which is the product generated during the ontology testing process.

These artefacts can be generated iteratively due to the fact that, following ontology development methodologies (e.g. eXtreme design [41], SAMOD [37], NeOn [49], GOSPL [11] and UPON [10]), there can be several versions of an ontology implementation.

Several tools for ontology development can be used to manage these artefacts, including their generation and maintenance. These tools include not only ontology editors, such as Protégé, but also platforms for storing and versioning resources, such as GitHub,[4] and for editing and sharing the requirements specification document, such as online spreadsheets. In the following sections each of these artefacts is further described.

3.2.1 Ontology requirements specification document

The ontology requirements specification document (ORSD) [48] is built to define the purpose of the ontology, the intended users and the requirements the ontology should be able to fulfil [49]. It is usually generated before the implementation of the ontology during the ontology

[4] https://github.com/.

requirements specification activity. This document should include information related to the requirement identifier and the associated competency questions [52]. A competency question is a natural language sentence that expresses a question expected to be answered by the system using the ontology, therefore either the ontology should answer such a question or provide the model supporting the system to answer it. In any case, using competency questions is a well-known and adopted technique for specifying ontological requirements. The ORSD could also include, in order to obtain more metrics, the status of the requirements (e.g. rejected, deprecated or accepted). In case that the ontology development process follows an iterative approach, it could also include the development iteration or sprint in which the requirement is planned to be implemented.

3.2.2 Ontology implementation

The main artefact from which ontology metrics can be extracted is the ontology code itself. The ontology implementation contains the axioms and metadata of the ontology and should be encoded through a formal language such as RDF Schema [5] or OWL [22].

3.2.3 Ontology requirements test suite

An ontology requirements test suite (ORTS) is a collection of test cases that are intended to be used to test a product [31]. The ontology requirements test suite introduced in this paper represents a resource which stores a set of test cases to test the ontology. In case this testing activity is focused on ontology verification, the test suite will verify the ontology against the requirements identified in the ontology requirements specification document [50]. In this verification scenario, the test cases should include the requirements translated into a formal language (e.g. SPARQL[5]) and the identifier of the associated requirement in the ORSD. The implementation of these test cases into a formal language which includes an identifier to associate each test case with an ontology requirement will allow identifying classes and properties defined in the test suite as well as providing traceability between the ontology requirements, the tests cases and the ontology implementation. Additionally, each test case should include its competency question formalised [43] and the expected result of the competency question.

There are several methodologies which deal with this testing activity, such as the work presented by Blomqvist et al. [3] or OntologyTest [20]. Having the requirements formalised allows for a system to automatically check if the ontology implementation meets them [53].

This artefact can be generated before the implementation of the ontology in order to follow a test-driven development approach [26], or after the implementation of the ontology in order to follow a cascade development approach [17].

4 Proposed ontology metrics

Inspired by the software engineering classification of metrics [47], the ontology metrics presented in this paper are also divided into two categories, i.e. product metrics and process metrics. In the ontology engineering context, **product metrics** are used to measure different internal attributes of the resultant ontologies, such as size or complexity of the ontology

[5] https://www.w3.org/TR/rdf-sparql-query/.

implementation. Besides, **process metrics** are used to measure different attributes of the ontology development process, such as time, changes or resources, and enable ontology engineers to assess project status, track potential risks, adjust work-flow, and predict personnel effort.

This section proposes a set of product and process metrics related to each of the artefacts described in Sect. 3.2, i.e. the ORSD, the ontology implementation, and the ORTS. The product metrics are related to a concrete version of these artefacts, while the process metrics are related to two or more versions. In the case that more than one of the mentioned artefacts are available, it is possible to combine the data extracted from them by generating multi-artefact metrics, which are also defined in this section.

To define these metrics, several principles were also followed. First, the work presented here reuses existing metric definitions. In such cases, the reference to the existing bibliographic resource is shown along with the metric. In addition, the work presented here provides metrics related to artefacts that are not analysed in the state of the art so far. Finally, the data extracted from these artefacts was combined to extract measurements with the aim of covering several aspects in the ontology development process, e.g. the analysis of the requirements coverage or the effort needed to develop an ontology. Thus, the objective of this section is to describe metrics based on the needs of the actors identified in Sect. 3.1 and on the integration of data from several of the artefacts presented in Sect. 3.2 rather than to provide an exhaustive analysis about all the feasible ontology metrics that can be extracted from them.

The next subsections describe the proposed metrics. The metrics are organised into tables which include four fields, namely: (a) the artefact or artefacts from which each metric is extracted, (b) the code to identify the metric, (c) the name and definition of each metric, and (d) the formula needed to calculate it. In order to ease readability the product metrics are grouped by **base metrics** (Table 3), which represent raw data, and **calculated metrics** (Table 4), which are derived from base metrics.

4.1 Base product metrics

Table 3 describes the proposed set of base product metrics related to the ORSD, the ORTS and the ontology implementation. It shows by means of the symbol "X" which artefact or artefacts are needed to calculate each metric. These metrics are going to be used to obtain calculated metrics, which are described in the next subsection. These base metrics can be extracted from one or more of the identified artefacts.

4.2 Calculated product metrics

Table 4 describes the proposed set of calculated product metrics related to the three artefacts, i.e. the ORSD, the ORTS and the ontology implementation. This table shows by means of the symbol "X" which artefacts are needed to calculate each metric. These metrics are calculated using several base metrics, which are described in the previous subsection, and are extracted from more than one of the identified artefacts. As an example of calculated metric, the metric *CovReqPtc* is obtained by using the base metrics *NCovReq* and *NReq*, and is extracted from the ORSD and the ontology implementation artefacts.

Table 3 Base product metrics and artefacts from which they are extracted

Artefacts			Code	Name and definition	How to calculate
ORSD	ORTS	Impl.			
X			NReq	*Number of Requirements.* Number of Requirements defined in the ORSD. To increase its granularity, this metric can be subdivided into *Number of added requirements (NAddedReq)*, *Number of rejected requirements (NRejectedReq)* and *Number of accepted requirements (NAcceptedReq)*	Obtained by counting the number of requirements in a given version of the ORSD
X			NPendingReq	*Number of Pending Requirements.* Number of pending to implement requirements after a development iteration	Obtained by counting the number of pending requirements in a given version of the ORSD
		X	NT	*Number of vocabulary Terms.* Number of terms that are included in a particular implementation of an ontology. To increase its granularity, this metric can be subdivided into *Number of classes* [58] and *Number of properties* [58]	Obtained by counting terms (i.e. classes and properties) in an ontology implementation version
	X		NTests	*Number of Test cases.* Number of tests cases defined based on the ORSD	Obtained by counting tests in a given version of the ontology requirements test suite
	X	X	NPassedTests	*Number of Passed Tests.* Number of test cases in the test suite which are passed by the ontology implementation	Obtained by counting the number of tests that are passed by a given version of the ontology implementation
	X	X	TestFaultDensity	*Tests Fault Density.* Number of test cases in the test suite that are not passed by the ontology implementation. Inspired by Costello and Liu [6]	Obtained by counting the number of tests that are not passed by a given version of the ontology implementation
	X	X	NTestedT	*Number of Tested vocabulary Terms.* To increase its granularity, this metric can be subdivided into *Number of tested classes* and *Number of tested properties*	Obtained by counting unduplicated terms (i.e. classes and roperties) in a given version of the ontology test suite
X		X	ReqCompleteness	*Requirements Completeness.* Number of requirements covered by the ontology. Inspired by Costello and Liu [6]	Obtained by counting the number of requirements that are covered by a given version of the ontology implementation
X		X	xReqFaultDensity	*Requirements Fault Density.* Number of requirements faults. Inspired by Costello and Liu [6]	Obtained by counting the number of requirements faults in an ORSD

Table 4 Calculated product metrics and artefacts from which they are extracted

Artefacts			Code	Name and definition	How to calculate
ORSD	ORTS	Impl.			
X		X	CovReqPtc	*Covered Requirements Percentage*. Percentage of requirements covered by the ontology	$CovReqPtc(R_i, O_v) = \frac{NCovReq(R_i, O_v)}{NReq(Req_i)} \times 100$ where O_v represents the version of the ontology implementation to be analysed and Req_i the version of the ORSD
X		X	ReqFaultPtc	*Requirements Fault Percentage*. Percentage of requirements faults	$ReqFaultPtc(R_i, O_v) = \frac{ReqFaultDensity(R_i, O_v)}{NReq(Req_i)} \times 100$ where O_v represents the version of the ontology implementation to be analysed and Req_i the version of the ORSD
X		X	ReqACo	*Requirements Axiom Complexity*. Ratio between the number of ontology axioms, i.e. OWL, RDF and RDFS[a] constructs related to each requirement and the total number of ontology axioms	$ReqACo(Req_j, O_v) = \frac{\sum_{k=1}^{n} hasAxiom(Req_j, A_k(O_v))}{A(O_v)} \times 100$ where $hasAxiom(Req_j, A_k(O_v)) = \begin{cases} 1 & \text{if } Req_j \text{ includes } A_k(O_v) \\ 0 & \text{if } Req_j \text{ not includes } A_k(O_v) \end{cases}$ n is the total number of ontology axioms[b] Req_j represents the requirements in a version of the ORSD and $A(O_v)$ represents the number of considered axioms

Table 4 continued

Artefacts			Code	Name and definition	How to calculate
ORSD	ORTS	Impl.			
X		X	Consistency	*Consistency*. Percentage of requirements in the ORSD that are in conflict. Inspired by Davis et al. [7]	$Consistency(R_i) = \frac{InconsistentReqs(R_i)}{NReq(Req_i)} \times 100$ where O_v represents the version of the ontology implementation to be analysed and Req_i the version of the ORSD
X	X		PendFReqPtc	*Pending Formalized Requirements Percentage*. Percentage of pending requirements which are already included in the test suite	$PendFReqPtc(PendReq_i, S_j) = \frac{\sum_{k=1}^{n} included(PendReq_{i_k}, S_j)}{NPendingReq} \times 100$ where $included(PendReq_{i_k}, S_j) = \begin{cases} 1 & \text{if } S_j \text{ includes } PendReq_{i_k} \\ 0 & \text{if } S_j \text{ does not include } PendReq_{i_k} \end{cases}$ n is the total number of pending requirements in the ORSD version with pending requirements called $PendReq_i$ and S_j represents the test suite version
	X	X	PassedTestPtc	*Passed Test Percentage.* Percentage of tests passed by the ontology implementation	$PassedTestPtc(S_j, O_v) = \frac{NPassedTest(S_j, O_v)}{NTests(S_j)} \times 100$ where S_j represents the a version of the test suite which stores the formalized requirements and O_v the version of the ontology implementation

Table 4 continued

Artefacts			Code	Name and definition	How to calculate
ORSD	ORTS	Impl.			
	X	X	TestFaultPtc	*Test Fault Percentage.* Percentage of tests not passed by the ontology implementation	$TestFaultPtc(S_j, O_v) = \frac{NPassedTest(S_j, O_v)}{NTests(S_j)} \times 100$ where S_j represents the version of the test suite which stores the formalised requirements and O_v the version of the ontology implementation
	X	X	TestEffRatio	*Test Effectiveness Ratio.* Percentage of the ontology implementation terms, including classes and properties, which are defined in the test suite. Inspired by Fenton and Bieman [14]	$TestEffRatio(S_j, O_v) = \frac{NTestedT(S_j)}{NT(O_v)} \times 100$ where S_j represents the test suite which stores the formalised requirements associated to the ontology version O_v
X	X	X	TestedReqPtc	*Tested Requirements Percentage*. Percentage of requirements identified in the ORSD which are formalized in the test suite and passed by an ontology implementation version	$TestedReqPtc(O_v, S_j) = \frac{CovReqPtc(O_v) \times PassedTestPtc(O_v, S_j)}{100}$, where S_j represents the test suite and O_v the ontology to be analysed

[a]The axiom considered from RDF in this metric is: type. The axioms considered from RDFS are: subclassOf, subPropertyOf, domain and range. The axioms considered from OWL are: cardinality, complementOf, disjointWith, equivalentClass, equivalentProperty, FunctionalProperty, hasValue, intersectionOf, InverseFunctionalProperty, inverseOf, maxCardinality, minCardinality, oneOf, someValuesFrom, SymmetricProperty, TransitiveProperty and unionOf. However, this can change depending on the needs of the user

Table 5 Process metrics

Code	Name and definition	How to calculate
DevTime	*Version Development Time*. Time taken to implement a given version of the ontology. This time includes the time taken to carry out all the activities in the development process, e.g. requirements elicitation and evaluation	Obtained by measuring the time of the different activities while developing a version of the ontology. This time can be calculated in minutes, hours or days, depending on the available granularity
ReqDevTime	*Requirement Development Time*. Implementation time taken of each individual requirement in the ontology	
TestDevTime	*Test case Development Time*. Development time taken for each test case in the test suite	
VDevEffort	*Version Development Effort*. Effort taken to implement a given version of the ontology. This effort includes the time taken to carry out all the activities in the development process, e.g. requirements elicitation and evaluation	Obtained by measuring the effort required for the different activities while developing a version of the ontology. This effort can be calculated in Persons-Month
ReqDevEffort	*Requirement Development Effort*. Effort taken to generate the specification of the requirements	
TestDevEffort	*Test case Development Effort*. Effort taken to generate each test case in the test suite	
ReqVolatility	*Requirements volatility*. Number of requirements that changed during the development process	Obtained by counting the number of requirements that have undergone some change during the development process

4.3 Process metrics

In addition to the metrics described in the previous subsections, this paper also describes a set of process metrics which are not extracted from any artefact. These process metrics are related to (1) the time taken for a particular process to be completed; (2) the resources required for a particular process; and (3) the number of occurrences of a particular event.

The proposed process metrics, which are summarised in Table 5, are extracted from the software infrastructure which supports the generation and versioning of the mentioned artefacts. Manual calculation of these metrics is unfeasible, since it would require ontology engineers to keep track of the time spent in each activity.

5 Metrics validation

To provide an assessment of the feasibility of the metrics and to validate the set of hypotheses stated in Sect. 1, an empirical analysis has been carried out in a concrete use case. To perform such assessment, data needs to be collected from different ontologies. There are projects, such as Wikidata [54] or the OBO Foundry [46], which provide their users with a set of metrics that are related, mostly, with counting different terms in their models. However, due to the fact that the only the ontology implementations are openly available and, to the authors' knowledge, no other artefacts are being generated such as ontology requirements or test suites, the validation of the metrics proposed in this paper has been carried out using five ontologies of different sizes which have been developed in the VICINITY European project.[6]

The five ontologies to be analyzed in the project, i.e. the VICINITY Core[7] (Core), the Web of Things[8] (WoT), the WoT mappings (Mappings),[9] the VICINITY Adapters (Adapters),[10] and the Datatypes (Datatypes) ontologies,[11] belong to the VICINITY ontology network and aim to provide interoperability in the IoT domain. The Core ontology represents the information needed to exchange IoT descriptor data between peers through the VICINITY platform; this ontology has been created by following a cross-domain approach and implements requirements from different domain experts. The WoT ontology aims to model the Web of Things domain according to the W3C WoT Working Group[12] descriptions. The Mappings ontology represents the mechanism for accessing the values provided by web things in the VICINITY platform. The Adapters ontology aims to model all the different types of devices and properties that can be defined in the VICINITY platform. It should be mentioned that the Adapters ontologies was created by extracting a module from the core ontology and continuing its development independently as a new ontology from sprint 6. That is, in the first versions of the ontology network, the information about devices and their properties were part of the core module. Finally, the Datatypes ontology aims to model the required and provided datatypes that are used in the interaction patterns of the platform.

To build these ontologies, an agile and iterative development process was followed. For the Core and Adapters ontologies, the team involved in the development process included 15 domain experts, 2 ontology developers and 1 team leader. In the case of the WoT, Mappings and Datatypes ontologies, the team involved 5 domain experts, 2 ontology developers and 1 team leader. The domain experts are the participants in charge of providing the ontology requirements, while the ontology developers design and implement the ontology, and the team leader is in charge of the development process management. One person had the role of team leader but also ontology developer dedicated to the project at 20% of effort. The other ontology developer was dedicated at 50% of effort. Finally, the domain experts dedicated a high percentage of their time at the moment of providing requirements and afterwards acting as consultancy, with minimum participation in the ontology development tasks, when needed. They had high dedication to the project but oriented to software development tasks.

The ontology requirements are prioritised, in order to plan and schedule the development of the ontologies in sprints. The ontology developers are also in charge of creating the test suite from the requirements. For each of the ontologies, the three artefacts mentioned in Sect. 3.2 were generated and are available in the VICINITY ontology portal.[13]

In this particular case, the ORSD was stored in online spreadsheets to facilitate sharing, edition and version control. This ORSD stores the requirements that were identified by domain experts and written in the form of competency questions and statements. The ontology requirements test suite generated to test this use case was aimed at ontology verification against the ORSD; therefore, the identified requirements were formalised into test cases. To

[6] http://vicinity2020.eu/vicinity/.

[7] http://iot.linkeddata.es/def/core/.

[8] http://iot.linkeddata.es/def/wot/.

[9] http://iot.linkeddata.es/def/wot-mappings/.

[10] http://iot.linkeddata.es/def/adapters/.

[11] http://iot.linkeddata.es/def/datatypes/.

[12] https://www.w3.org/WoT/WG/.

[13] http://vicinity.iot.linkeddata.es.

Table 6 Analysed ontologies and their sprints

Ontology	Sprints	Team
Core ontology	6 sprints	15 domain experts, 2 ontology developers and 1 team leader
WoT ontology	4 sprints	5 domain experts, 2 ontology developers and 1 team leader
Mappings ontology	2 sprints	5 domain experts, 2 ontology developers and 1 team leader
Adapters ontology	1 sprint	15 domain experts, 2 ontology developers and 1 team leader
Datatypes ontology	1 sprint	5 domain experts, 2 ontology developers and 1 team leader

validate the requirements on the ontologies, these tests cases are executed on a given version of the ontology implementation. Each ontology implementation was stored in a GitHub repository.[14]

Figure 6 summarises the details of each ontology. More information about these ontologies is available in the VICINITY ontology network portal, including the GitHub repository where each ontology is stored, their ontology requirements and their associated test suite. During each sprint, a set of requirements was planned to be implemented in each ontology.

The following subsections describe the results obtained after gathering the metrics described in Sect. 4 on the five ontologies, with the aim of validating the hypotheses exposed in Sect. 1. Such hypotheses analyse the relation between two metrics, e.g. number of tests and number of requirements. Thus, to validate the hypotheses, the correlation coefficient [27] between the metrics is calculated to determine the strength of the association between them. The formula used to calculate such coefficient is the following:

$$\rho(X, Y) = \frac{\mathbf{Cov}(X, Y)}{\sqrt{\mathbf{Var}(X)\mathbf{Var}(Y)}}$$

The correlation coefficient has a value between − 1 and 1, where 1 is total positive linear correlation, 0 is no linear correlation, and − 1 is total negative linear correlation. The relationship between two variables is generally considered strong when their correlation coefficient value is larger than 0.6 (positive correlation) or lower than − 0.6 (negative correlation).

The metrics are calculated automatically from the three artefacts described in the previous section and from the software tools which support their generation and storage. Based on the previous identification of actors in the development process, Fig. 2 shows potential interests of each actor and the proposed metrics. As an example, ontology developers can be interested in only the product metrics, while team leaders can be interested in both product and process metrics. This figure is not exhaustive, and more relations between actors and metrics could be added.

The following sections are organised according to the interests of the roles involved in the development process and the hypotheses to be validated. To conclude, the limitations found during the analysis are described.

[14] The link to all the Github repositories are indicated in the VICINITY ontology portal: http://vicinity.iot.linkeddata.es/ which due to its version control allows the ontology engineers to be aware of the evolution of the artefacts during the development iterations. Test suites are also stored in the GitHub repository of its associated ontology.

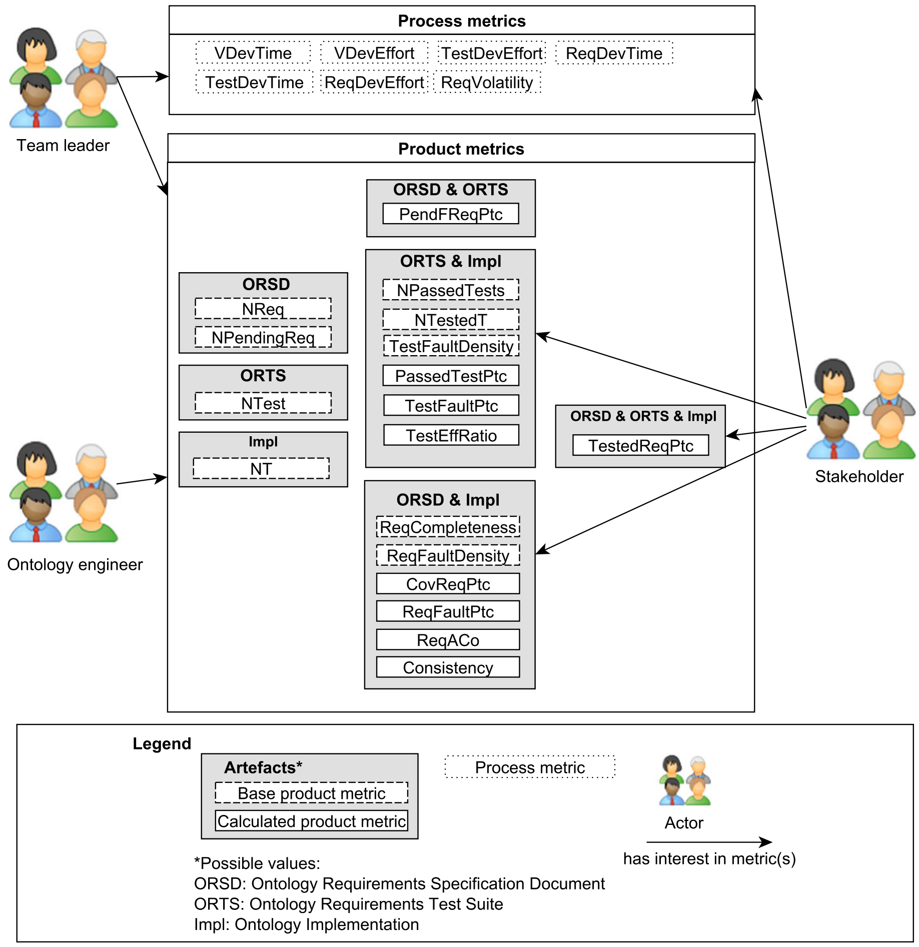


Fig. 2 Ontology engineering metrics and interested actors

5.1 Team leader and Ontology developers: analysis of number of requirements

Ontological requirements can provide team leaders and developers with insights about the development process, since they represent what should be added to an ontology. The number of requirements (*NReq*) metric was used to determine the number of requirements defined for the five ontologies in each sprint, which are summarised in Table 7.

To analyse the possible effects in the development process that could be extracted from the number of requirements, this metric was compared with the number of defined tests, the development time, the size and the expressivity of the five ontologies.

First, the number of tests defined for each ontology was determined and summarised in Table 7. In this project, the tests were generated to verify the requirements and, thus, for the majority of ontologies there is the same number of requirements and tests. Moreover, in those cases where the number of requirements does not coincide with the number of tests, the latter is higher; although there was defined at least one test per requirement, some requirements needed more than one test to be verified. Furthermore, the correlation coefficient between

Table 7 Overview of the sprints

	Sprint 1		Sprint 2		Sprint 3		Sprint 4		Sprint 5		Sprint 6	
	#Req	#Tests	#Req	#Tests	#Req	#Tests	#Req	#Tests	#Req	#Tests	#Req	#Tests
Core	116	116	153	153	156	156	154	154	173	173	67	67
WoT	33	33	32	32	24	24	14	17	–	–	–	–
Mappings	13	13	15	15	–	–	–	–	–	–	–	–
Adapters	–	–	–	–	–	–	–	–	–	–	171	171
Datatypes	–	–	–	–	–	–	–	–	–	–	11	14

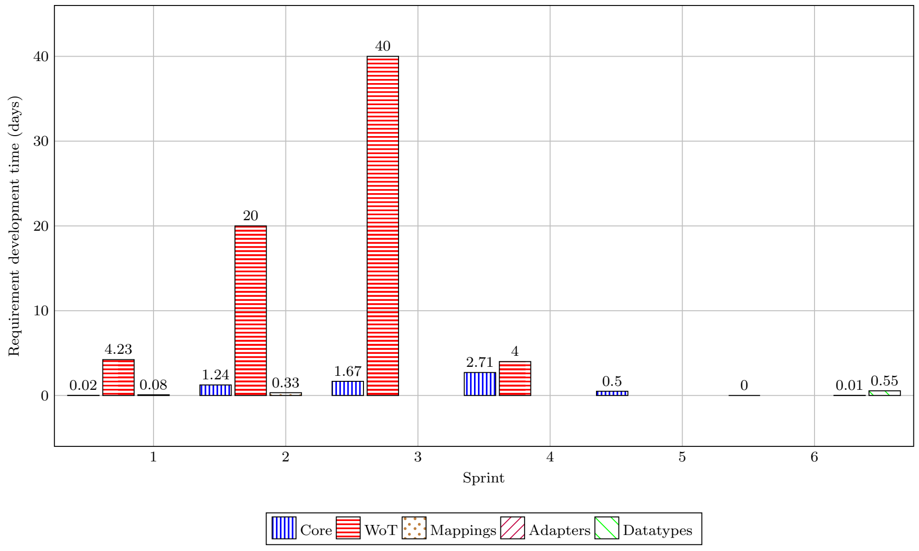


Fig. 3 Requirement development average time for each ontology version

these two metrics is 0.99. Thus, it can be stated that **the number of requirements influences the number of defined tests** and, consequently, hypothesis H1 is validated.

To calculate the development time spent in the five mentioned ontologies and to check whether the number of requirements influences their development time, the information about the commits done by the ontology developers in the GitHub repositories and the information stored in the spreadsheets with the ontology requirements were retrieved. In this project, the developers only used the master branch of the GitHub repository to develop the ontology and the online spreadsheets store the modifications done over the ORSD during the development iterations. Therefore, each sprint starts when the spreadsheet is modified to indicate which are the requirements planned for the sprint, and ends with the commit that releases a new version of the ontology. Due to the fact that these software tools calculate the spent time and they can only calculate it in days, the precision of the results is not as accurate as it would be if it had been calculated in hours. Figure 3 shows the total time spent per requirement and sprint (*VReqTime*) for each ontology version.

In addition to the development time, it was also calculated the effort spent in each ontology during the sprints. Thus, the number of commits that modify the ontologies has been extracted from the GitHub repository. Figure 4 shows the total effort spent per sprint (*VDevEffort*) to develop each ontology, where the requirements are implemented. Moreover, Fig. 4 also illustrates the effort spent during the maintenance phase of the ontology, where bugs were corrected and small changes were proposed.

From the comparison between Table 7, Figs. 3 and 4, it could be concluded that **the number of requirements does not influence directly the time of development**. The ontology developers implemented 171 requirements for the Adapters ontology in only one sprint, while for the 33 initial requirements of the WoT ontology they needed 4 sprints. The reason for this is that the requirements defined for the Adapters ontology were easier to implement or they were already implemented in other ontology as already mentioned, due to the fact that part of the Adapters ontology was taken from previous Core module. However, with the gathered

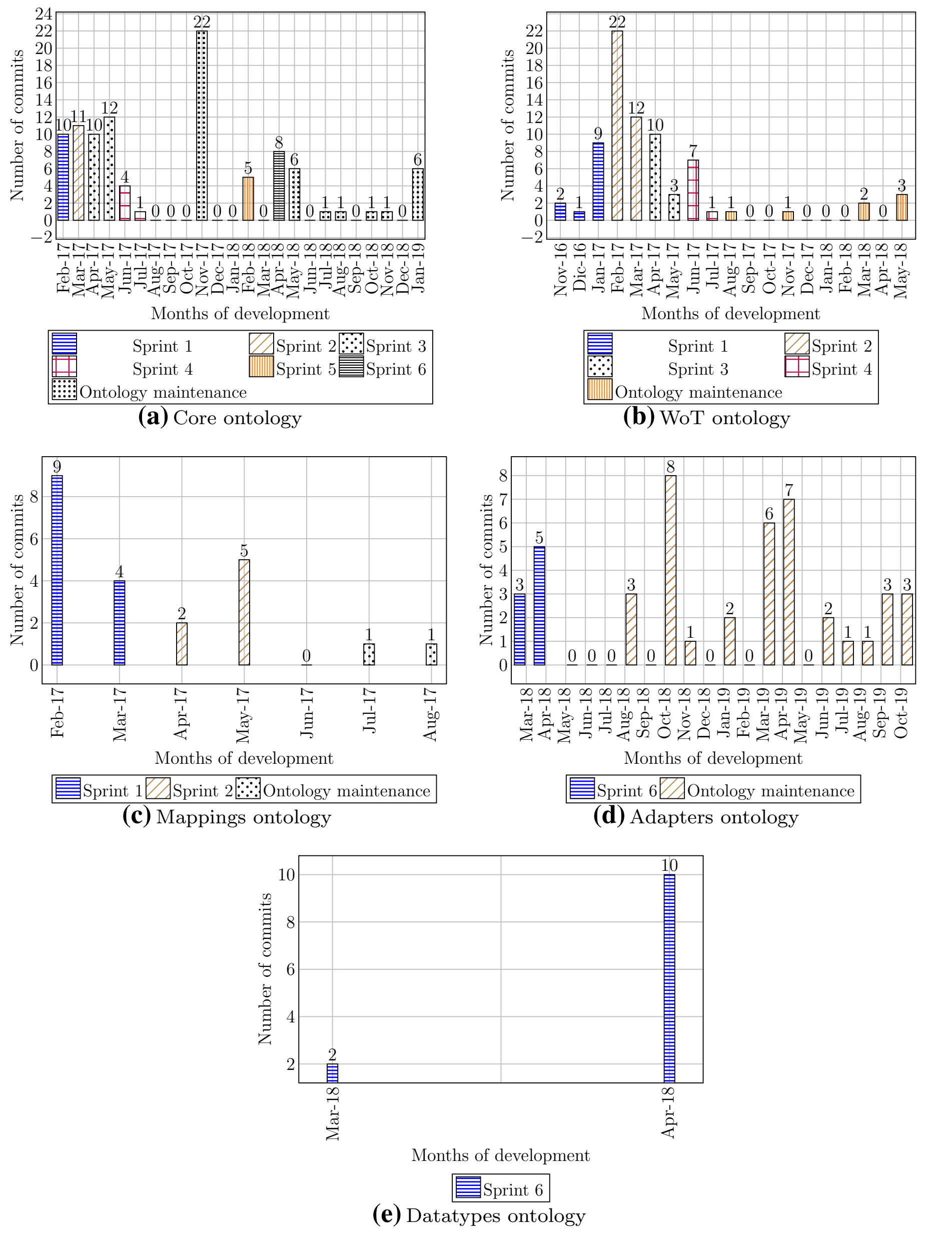


Fig. 4 Distribution of the required effort in each ontology

information related to the development time and effort, the correlation coefficient between these two metrics is -0.33 and, therefore, hypothesis H2 is rejected.

The size of the ontologies during the sprints was also calculated in order to be compared with the number of requirements. Figures 5 and 6 illustrate the evolution of the size of the ontologies during their development process. The former indicates the size without con-

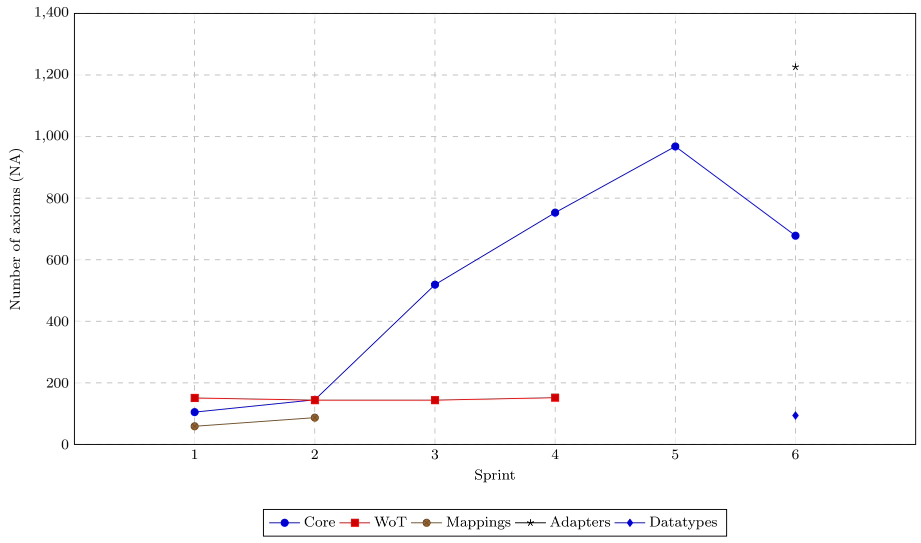


Fig. 5 Size of the ontologies during their development process not including imported ontologies

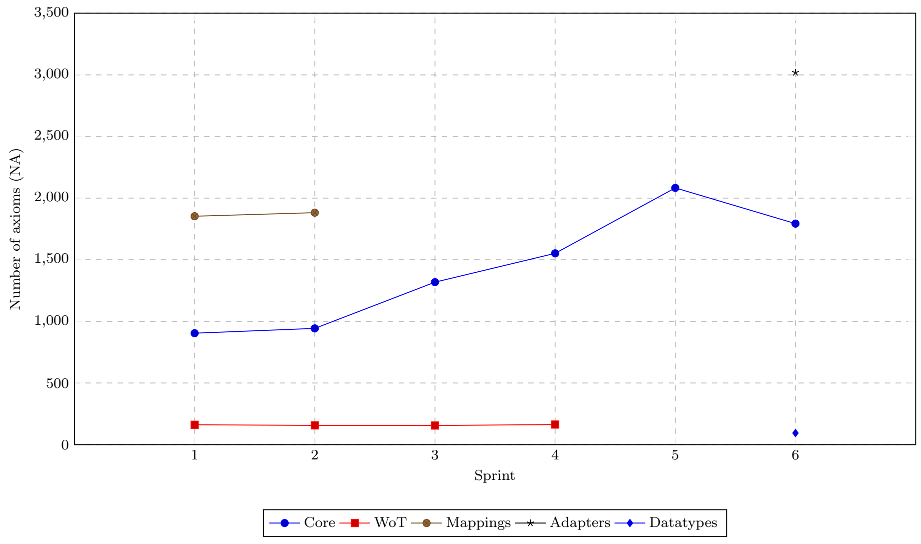


Fig. 6 Size of the ontologies during their development process including imported ontologies

sidering the imported ontologies, while the latter indicates the size including the imported ontologies.

Figure 5 shows that while the size of the WoT ontology and the Mappings ontology have small variations, the size of the Core ontology increases from sprint 2. All these results concur with the values obtained from Fig. 4, because the effort spent in the WoT ontology and the Mappings ontology is insignificant compared to the effort spent in the Core ontology. This is due to the fact that the Core ontology, considering that several domain experts from different domains are involved in the development process, receives more requirements than

Table 8 DL expressivity of the ontologies during each sprint

Ontology	Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Sprint 6
Core	SRIF(D)	SRIF(D)	SRIF(D)	SRIF(D)	SRIQ(D)	SRIQ(D)
WoT	ALCHIF(D)	ALCHIF(D)	ALCHIF(D)	ALCHIF(D)	–	–
Mappings	SRIQ(D)	SRIQ(D)	–	–	–	–
Adapters	–	–	–	–	–	SRIQ(D)
Datatypes	–	–	–	–	–	ALF(D)

the other ontologies during the sprints. Moreover, in the last sprint the ontology that has more defined requirements is the Adapters ontology, which is also the largest ontology. Therefore, with the information obtained from Fig. 5 and Table 7, the correlation coefficient is 0.73. Thus, hypothesis H3 is validated since **the number of requirements affects the size of the ontologies**. It should be mentioned that in this situation the imported ontologies are not included in the analysis, since only the axioms that were added by the developers involved in the project were taken into account.

Nevertheless, Fig. 6 illustrates the size the ontologies considering their imports. In this figure, it can be observed that the Mappings ontology has increased its size, since it imports several ontologies (due to the needs of the application using it) even though it is not indicated in the requirements. The evolution of the rest of the ontologies remains similar, and the Adapters ontology is still the largest one.

Finally, the DL expressivity [2] of the ontologies during the sprints was calculated and is summarised in Table 8. The comparison between Table 7 and Table 8 concluded that **the number of requirements does not affect the expressivity of the ontology**. The Mappings ontology, which has only 13 and 15 requirements in Sprint 1 and 2, has the same expressivity as Adapters, which has 171 requirements. This occurs because the Mappings ontology imports, among others, the Core ontology, increasing its DL expressivity. The correlation coefficient between these two metrics is 0.53 and, thus, hypothesis H4 is rejected.

5.2 Team leader and Ontology developers: analysis of requirements complexity

The analysis of the complexity of the requirements may help team leaders along with ontology developers in estimating the effort needed to implement a requirement. The more complex a requirement is, the more complex may be to implement it in the ontology. The Requirements Axiom Complexity (*ReqACo*) metric was used to obtain the requirements complexity for the five ontologies, due to the fact that it measures the axioms needed to implement each requirement. The number of axioms was obtained by counting the number of axioms to be added to the corresponding ontology for each requirement.

Figure 7 shows the distribution of the requirements axiom complexity in the last sprint of the five analysed ontologies, where the requirements are up to date and, as a consequence, are valuable to make effort estimations. For the sake of clarity, the graphs in Fig. 7 consider a maximum of 50% ReqACo, which represents that the requirement includes 11 axioms out of the 22 considered in the metric as stated in Sect. 4. As an example of the information that can be retrieved from Fig. 7, the figure shows that the Mappings ontology has 12 requirements with a ReqACo between 15 and 20% (e.g. 4 axioms out of 22), 1 requirement with a ReqACo between 25 and 30% (e.g. 6 axioms out of 22), 1 requirement with a ReqACo between 35

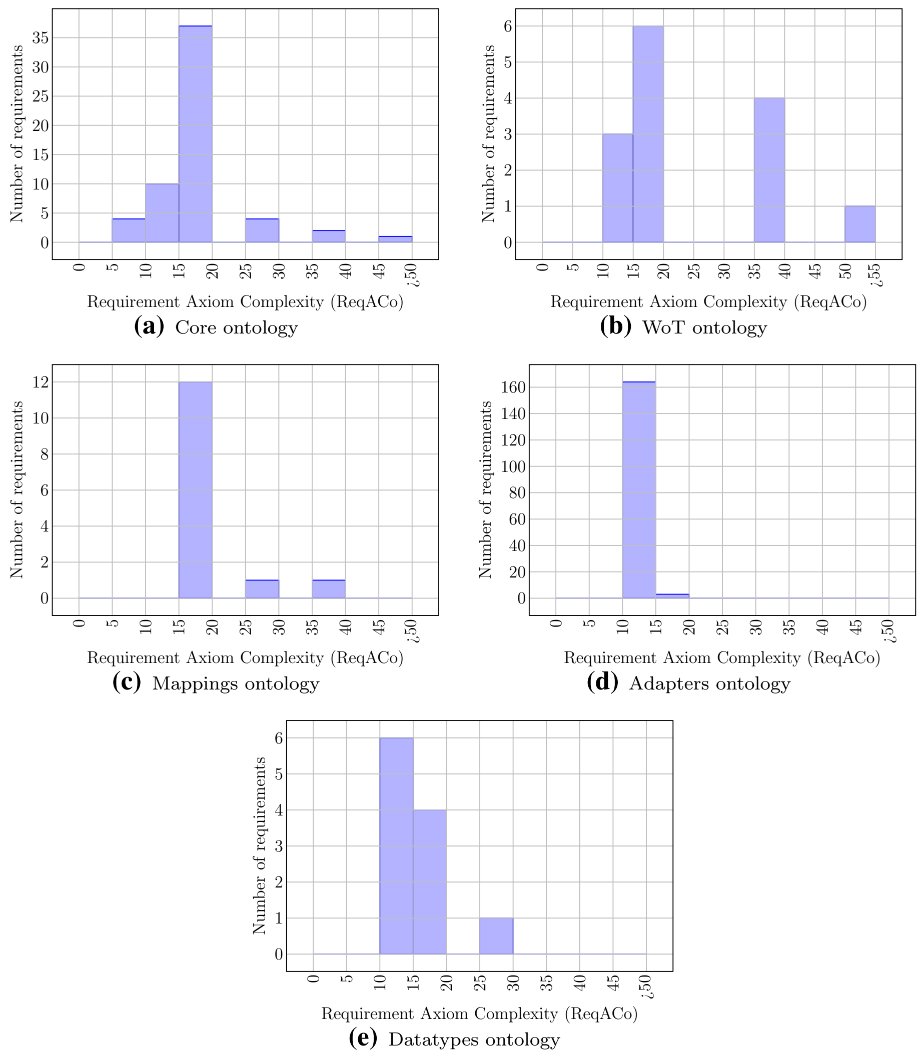


Fig. 7 Distribution of the axiom complexity in each ontology in the last sprint

and 40% (e.g. 8 axioms out of 22) and 1 requirement with more than 50% (e.g. 11 axioms out of 22).

To analyse the effect of the complexity of requirements regarding other aspects of the development process, it was compared with the number of defined tests, the development time and the size of the ontologies during the sprints.

First, the complexity of the requirements was compared with the number of tests defined for each ontology in order to determine whether the complexity of requirements influences the number of defined tests. However, from the information gathered from Fig. 7 and in Table 7, it can be deduced that **the complexity of requirements does not influence the number of tests**, since the most complex requirements are associated with the Core ontology, but no additional tests were defined. Additionally, the Datatypes ontology, which has simple requirements, is the ontology with most additional tests. Furthermore, the correlation coefficient between the average complexity of requirements and the average tests defined during the development process for each ontology is -0.5. Thus, hypothesis H4 is rejected.

The complexity of requirements was also compared with the time and effort spent in the development process. Figure 7 illustrates that the ontology with the most complex requirements is the WoT ontology while the ontology with the highest required effort (Fig. 4) was the Core ontology. The correlation coefficient between the average of complexity of requirements and the average time spent per requirement is − 0.7. Therefore, the complexity of requirements influences negatively the development time of ontologies, i.e. the higher the requirement complexity, the lower the development time per requirement. The hypothesis H5 is validated since **the complexity of requirements influences the development time of the ontology**. These results sound counter-intuitive, establishing that the higher the complexity, the lower the development time. Therefore, intending to analyse whether other factors affect this relationship, the complexity of requirements results were joined with the number of requirements to determine whether its combination affects the development time.

If this complexity of requirements result is joined with Table 7, it can be observed that the number of defined requirements in the Core ontology outnumbered the defined requirements of the almost the rest of ontologies. Moreover, the Core ontology have also requirements with significant complexity. Therefore, the joined information obtained by the complexity and the number of requirements also influences the development time of the ontology.

Finally, the complexity of requirements was also compared with the size of the ontologies, which are shown in Fig. 5. This figure shows that the size of both the Core ontology and the Adapters ontology exceed the size of the other ontologies. However, from Fig. 7 it can be noticed that the WoT ontology is the ontology with most complex requirements. The correlation coefficient between the average of complexity of requirements and the average size of each ontology is -0.4. Thus, with all this information hypothesis H7 is rejected since **the complexity of the requirements does not influence the size of the ontology**.

5.3 Team leader and Ontology developers: analysis of the volatility

Volatility of the ORSD refers to the modifications, additions or deletions of the requirements over time. Figure 8 presents the number of additions of requirements (*NAddedReq*), the number of rejections of requirements (*NRejectedReq*) and the number of acceptances of requirements(*NAcceptedReq*) for the five ontologies during their development iterations.

Figure 8 illustrates that in the WoT and Mappings ontologies the majority of the requirements were identified at the beginning of the development process, even though several of them were rejected during the development. Nevertheless, in the Core ontology the requirements changed over time, adding or deleting at least one requirement in each iteration. In the case of the Adapters and the Datatypes ontologies, no additional information could be extracted since they were created in a single sprint.

If these results are compared with Figs. 5 or 6, it can be observed that the ontology with more volatility in their requirements is the Core ontology, which is also the ontology with more volatility in the size of the ontology. Figure 5 shows that the Core ontology starts with 151 axioms and ends with 678 axioms, reaching 968 in the fifth sprint. Therefore, **it can be concluded that the volatility of the requirements influences the volatility of the ontologies**. Moreover, the correlation coefficient between the changes in the requirements and in the ontologies is 0.64 and, therefore, hypothesis H8 is validated.

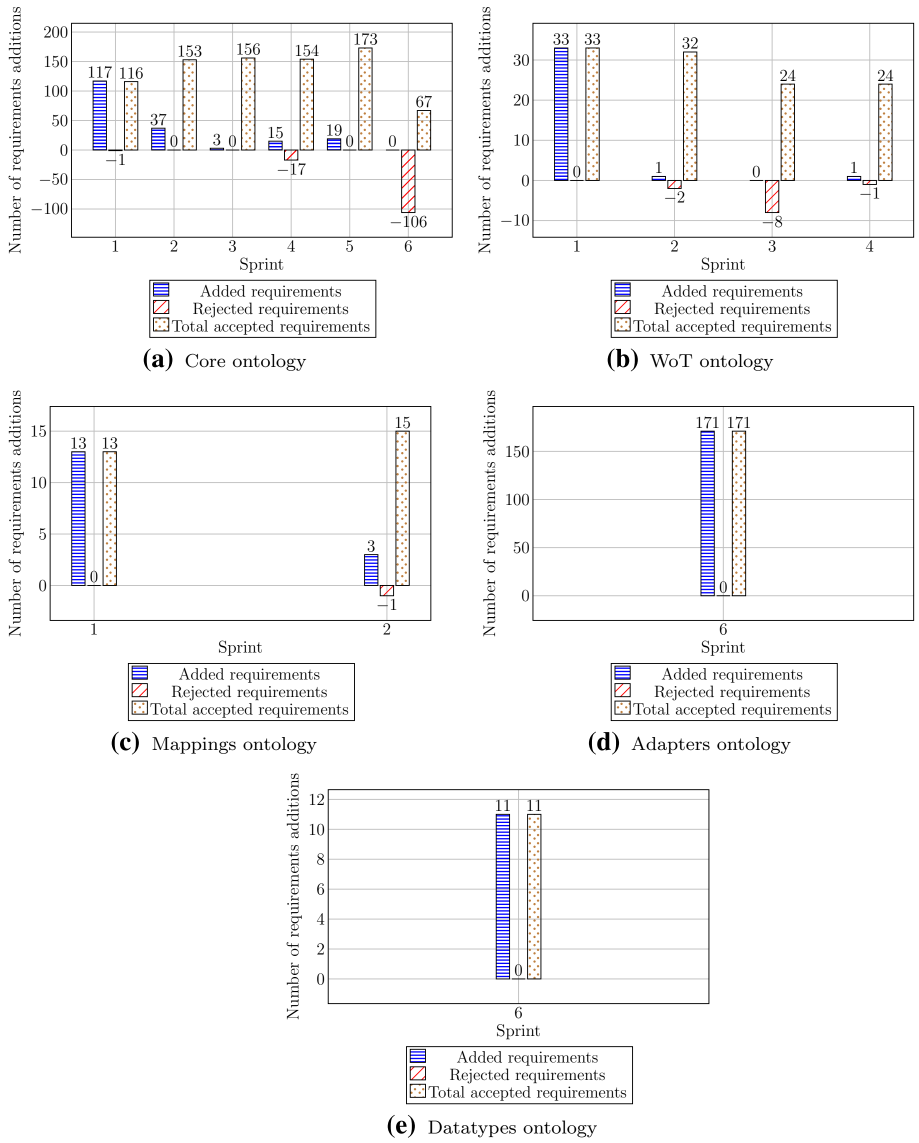


Fig. 8 Distribution of the volatility of the requirements in each ontology in the last sprint

5.4 Team leader, ontology developers and stakeholders: analysis of the defined tests

Team leaders and ontology developers need to analyse the ontology coverage, in order to be aware of the requirements planned for each iteration and to determine whether they are satisfied by the ontology or if the generated tests cover all the requirements. Stakeholders also make use of this information in order to be aware of whether the ontologies satisfy their needs. In this project, a set of tests have been defined in order to verify whether the ontological requirements are satisfied by the analysed ontologies. Table 7 summarises the number of defined tests (*NTests*) per sprint. As it can be observed, the defined tests for the Core and the Adapters ontologies outnumber the tests defined for the rest of the ontologies.

To analyse the effect of the number of defined tests in the development process, it has been compared with the number of tested terms and the coverage of the ontologies during their sprints.

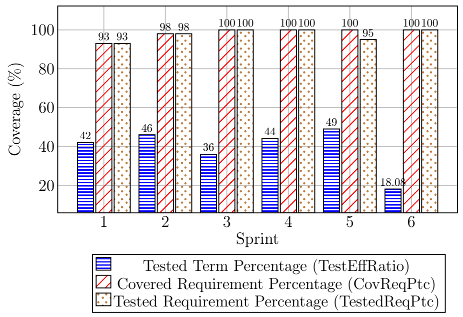
Figure 9 shows the number of tested terms (*TestEffRatio*) during the sprints for each analysed ontology. Only the Adapters ontology has the 100% of its terms tested, while the values in the Core and the WoT ontologies do not exceed 53%. These results are normal due to the fact that there are terms that are not defined in the requirements. These terms can be created from the addition of ontology design patterns [18], the creation of hierarchies, the creation of n-ary relations [35], or the reuse of terms from other ontologies. However, a high value of *TestEffRatio* shows that the ontology is generated almost directly from the requirements, without many modelling decisions of the ontology engineers. In the case of Mappings, Fig. 9 shows that the *TestEffRatio* values are extremely low. After analysing the cause of these results, it was deduced that the Mappings ontology imports the WoT and Core ontologies, but their terms are not tested in the Mappings test suite because they are not identified in the requirements.

As mentioned before, the ontologies with the highest number of tests are the Core and the Adapters ontologies, with 173 defined tests in Sprint 5 and 171 defined tests in Sprint 6, respectively. However, their *TestEffRatio* metric differs significantly, being 49% for the Core ontology and 100% for the Adapters ontology. Thus, with all these results and a correlation coefficient of 0.36, it can be concluded that the **number of tests does not influence the tested terms in the ontology**. Thus, hypothesis H9 is rejected.

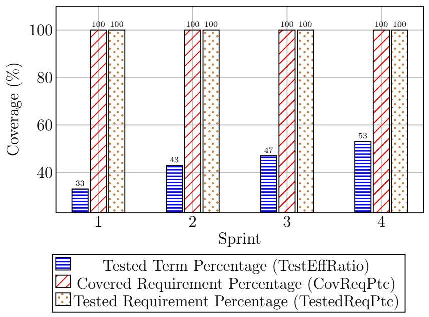
Finally, the *TestEffRatio* metric was also compared with the coverage of the ontology (*CovReqPtc*). Figure 9 depicts the *CovReqPtc* for the ontologies in each sprint. It can be observed that the Core and the WoT ontology, even though the tested terms do not exceed the 53%, reach the 100% of the coverage of the ontology. Similarly, the Adapters ontology has the 100% of its terms tested and also reaches the 100% of the coverage of the ontology. This could occur because due to modelling decisions or ODPs new terms were added to the ontology but avoided in the requirements. Moreover, the correlation coefficient between these two metrics is 0.002. Thus, it was concluded that **the number of tested terms does not influence the coverage of the ontology** and hypothesis H10 is rejected.

6 Discussion

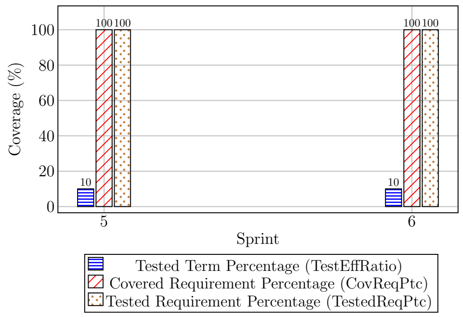
This paper proposes a set of metrics related to different artefacts produced during the development process to provide information about the status of the ontology and the process itself. For doing so, a literature review was performed, reusing existing metrics where possible. However, this literature review focused on metrics that can be directly adopted within the development process, e.g. the *Number of classes* metric described by Zhe and colleagues [58]. Metrics that could be potentially obtained from formulas presented in ontology engineering approaches were not considered in this paper. As an example, a new metric could be extracted from the identification of conflicts described in DOGMA-MESS [8] to indicate the number of concepts with conflicts, or from the list of pitfalls gathered by OOPS! [39] to indicate the total number of pitfalls in an ontology. Although these approaches indeed presented metrics that could be extracted, they are not considered in this analysis.



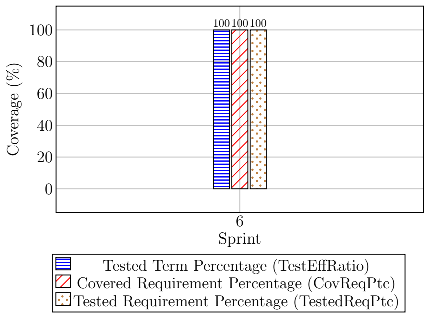
(a) Core ontology



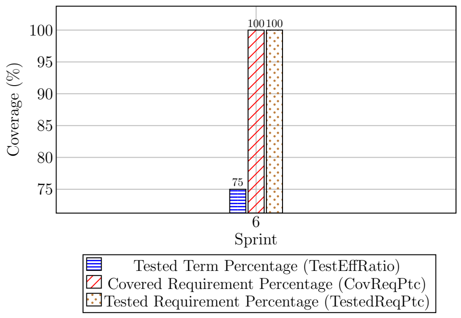
(b) WoT ontology



(c) Mappings ontology



(d) Adapters ontology



(e) Datatypes ontology

Fig. 9 Distribution of the coverage in each ontology in the last sprint

Besides, during the use case analysis several limitations have been found, which are crucial to understand the results presented in Sect. 5. The following paragraphs describe such limitations and their effects.

Firstly, it should be considered that the information about development time and effort depends on a third-party tool, i.e. GitHub. Moreover, the developers only used the master branch to submit their work, and they were not asked to measure the time or effort spent, since the tool does not allow to do that. Consequently, the results regarding the development time are approximated values. Moreover, the time could only be calculated in days and, thus, the precision of the results was not as accurate as expected. However, the results are still valuable for the analysis of the metrics in the use case.

With regards to the project context, in this use case all the sprints have the same goal: to implement a set of planned requirements. Additional effort that could appear during the first sprint in the development process to set up the ontology has not be considered, since it could not be obtained from the tools used in the project.

Finally, it should be mentioned that the methodology followed in this use case does not allow to have an automated way to calculate several metrics, such as the Pending Formalised Requirements Percentage (*PendFReqPtc*) metric, which would measure the percentage of pending to implemented requirements in the ORSD that are already included in the test suite and implemented in the ontology. To be able to automate this calculation, it is needed to have in every sprint all the requirements formalised in the test suite. In this particular case, the formalisation is iterative based on the schedule of the requirements, formalising only those that are planned to be implemented.

7 Conclusions and future work

This paper introduces a set of 25 metrics for ontology engineering related not only to the ontology implementation but also to the ontology development process and other artefacts generated from it, i.e. the requirements specification document and the ontology requirements test suite. These metrics include the combination of the data extracted from two or more of these artefacts. Team leaders, ontology developers and stakeholders are expected to be able to analyse the status of the development process and of ontology verification by means of gathering information extracted from the presented metrics. The metrics were applied in a real use case with the aim of validating the hypotheses stated in Sect. 1.

During the application of the metrics in the use case, it has been observed that computing the metrics is time consuming and requires ontology developers to keep track of the time and effort spent in each activity. Therefore, it is essential to have software that supports the automatic generation of metrics. In addition to this, ontology developers need to invest time to provide an accurate ontology requirement specification document and to generate the ontology test suite to extract the metrics. Nevertheless, this investment of time is worth in an scenario where ontology development involves several roles, e.g. different domain experts and ontology developers, working in collaboration with software developers, and where the ontology is a critical component of a broader project.

If more metrics could be calculated from the presented use case (*TestDevTime* and *PendFReqPtc* could not be calculated with the information obtained from the use case) and their granularity could be increased, then these metrics could be used to answer research questions such as "*What factors influence the quality of the ontology?*" or "*What factors influence the complexity of the requirements?*", since the effect of more variables in the development process, such as the development time of tests or requirements, could be analysed and compared. These research questions could lead to more research insights.

In this work, the proposed metrics were demonstrated in a real use case, providing the information needed for each actor involved in the development process in order to be able to monitor it. The use case was also used to compare the effects of different metrics in the development process. Although some of the hypotheses were rejected, the metrics calculated in the use case provided insights about the development process of the analysed ontologies. As an example, it was found that the number of requirements or the complexity of requirements do not influence individually the time of the development; however, they do it jointly. If the complexity and the number of requirements is high, then the time increases. However, if the number of requirements is high, then the time is not influenced.

It is worth noting that the interpretation of the metrics depends on the use case in which they are used. As an example, the interpretation of the size of the ontology can vary depending on the aspect to be analysed: in the case of completeness the more axioms the better, while in the case of understandability the less axioms the better. Additionally, during the validation of the metrics, it was found that the quality of the requirements is essential for the metrics. The more accuracy in the definition of ontological requirements, the more accuracy in the metrics.

Future work will be directed to define a unique metric which combines data obtained from the available artefacts with the aim of providing ontology developers with a measurement of the *health condition* of the ontology development process, i.e. the quality of the process considering all these different artefacts. An ideal situation in an ontology development process includes a low number of pending requirements, a high level of formalised requirements coverage and a high level of test coverage. This ideal situation indicates that in a particular sprint, the ontology is complete (or almost complete) regarding the requirements defined by the domain experts, that the ontology test suite covers a high percentage of the requirements identified in the ORSD, and that the ontology implementation passed a high percentage of the tests cases defined in the test suite. In addition, future work will also be directed to the implementation of a tool to support computing these metrics in any ontology development project. Moreover, the analysis between more variables in other projects could be also carried out, in order to extract more insights about the development process.

Finally, new metrics which take into account ontology instances or datasets should be defined in future work. New metrics related to the user documentation should be considered, due to the fact that there are still no metrics related to such artefact. Moreover, a deeper analysis of ontology engineering methodologies should be performed to identify metrics that could be extracted from the information provided by such methodologies.

Acknowledgements This work is partially supported by the H2020 project VICINITY: Open virtual neighbourhood network to connect intelligent buildings and smart objects (H2020-688467) and by a Predoctoral grant from the I+D+i program of the Universidad Politécnica de Madrid. We are very grateful to María Navas-Loro for her formula revisions and comments.

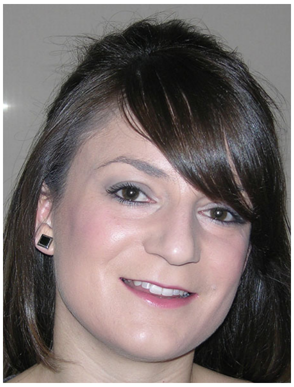
References

1. Albrecht AJ (1979) Measuring application development productivity. In: Proceedings of the joint SHARE/GUIDE/IBM application development symposium, pp 83–92
2. Baader F, Horrocks I, Sattler U (2008) Description logics. Found Artif Intell 3:135–179
3. Blomqvist E, Sepour AS, Presutti V (2012) Ontology testing-methodology and tool. In: Proceedings of the 18th international conference on knowledge engineering and knowledge management, Galway City, Ireland, October 8–12. Springer, Berlin, pp 216–226
4. Boehm B, Clark B, Horowitz E, Westland C, Madachy R, Selby R (1995) Cost models for future software life cycle processes: COCOMO 2.0. Ann Softw Eng 1(1):57–94
5. Brickley D, Guha RV, McBride B (2014) RDF Schema 1.1. W3C Recommendation 25 February 2014. Available at https://www.w3.org/TR/rdf-schema/
6. Costello RJ, Liu DB (1995) Metrics for requirements engineering. J Syst Softw 29(1):39–63
7. Davis A, Overmyer S, Jordan K, Caruso J, Dandashi F, Dinh A, Kincaid G, Ledeboer G, Reynolds P, Sitaram P, Ta A, Theofanos M (1993) Identifying and measuring quality in a software requirements specification. In: Proceedings of the 1st international software metrics symposium, Baltimore, MD, USA, May 21–22. IEEE, pp 141–152

8. De Leenheer P, Debruyne C (2008) DOGMA-MESS: a tool for fact-oriented collaborative ontology evolution. In: Proceedings of the 2008 international conference on On the move to meaningful internet systems: OTM 2008 Workshops, Monterrey, Mexico, November 9–14. Springer, Berlin, pp 797–806
9. De Moor A, De Leenheer P, Meersman R (2006) DOGMA-MESS: a meaning evolution support system for interorganizational ontology engineering. In: Proceedings of the 14th international conference on conceptual structures, Aalborg, Denmark, July 16–21. Springer, Berlin, pp 189–202
10. De Nicola A, Missikoff M, Navigli R (2005) A proposal for a unified process for ontology building: Upon. In: Proceedings of the 16th international conference on database and expert systems applications, Copenhagen, Denmark, August 22–26. Springer, Berlin, pp 655–664
11. Debruyne C, Tran TK, Meersman R (2013) Grounding ontologies with social processes and natural language. J Data Semant 2(2–3):89–118
12. DeMarco T (1979) Structured analysis and system specification. Yourdon Press, Berlin
13. Duque-Ramos A, Fernández-Breis JT, Iniesta M, Dumontier M, Aranguren ME, Schulz S, Aussenac-Gilles N, Stevens R (2013) Evaluation of the OQuaRE framework for ontology quality. Expert Syst Appl 40(7):2696–2703
14. Fenton N, Bieman J (1997) Software metrics: a rigorous and practical approach. PWS Publishing Company, Boston
15. Fenton NE, Neil M (2000) Software metrics: roadmap. In: Proceedings of the conference on the future of software engineering, Limerick, Ireland, June 04–11. ACM, pp 357–370
16. Fernández-López M, Gómez-Pérez A (2002) The integration of OntoClean in WebODE. In: Proceedings of the OntoWeb-SIG3 workshop at the 13th international conference on knowledge engineering and knowledge management, Siguenza, Spain, 30th September. CEUR-WS.org, CEUR Workshop Proceedings, vol 62, pp 38–52
17. Fernández-López M, Gómez-Pérez A, Juristo N (1997) Methontology: from ontological art towards ontological engineering. In: Proceedings of the ontological engineering AAAI97 spring symposium series. Stanford University, EEUU, March 24–26. AAAI Press, pp 33–40
18. Gangemi A, Presutti V (2009) Ontology design patterns. Handbook on ontologies. Springer, Berlin, pp 221–243
19. Gangemi A, Catenacci C, Ciaramita M, Lehmann J (2006) Modelling ontology evaluation and validation. In: Proceedings of the 3rd European Semantic Web Conference, Budva, Montenegro, June 11–14. Springer, Berlin, pp 140–154
20. García-Ramos S, Otero A, Fernández-López M (2009) OntologyTest: A tool to evaluate ontologies through tests defined by the user. In: Proceedings of the 10th international work-conference on artificial neural networks on artificial neural networks, Salamanca, Spain, June 10–12. Springer, Berlin, pp 91–98
21. Gunning R (1952) The technique of clear writing. McGraw-Hill, New York
22. Hitzler P, Krötzsch M, Parsia B, Patel-Schneider PF, Rudolph S (2009) OWL 2 Web Ontology Language Primer (Second Edition) W3C Recommendation 11 December 2012. Available at https://www.w3.org/TR/owl2-primer/
23. Iqbal S, Naeem M, Khan A (2012) Yet another set of requirement metrics for software projects. Int J Softw Eng Its Appl 6(1):19–28
24. Kan SH (2002) Metrics and models in software quality engineering. Addison-Wesley, London
25. Kang YB, Li YF, Krishnaswamy S (2012) Predicting reasoning performance using ontology metrics. In: Proceedings of the 11th international semantic web conference, Boston, MA, USA, November 11–15, Springer, Berlin, pp 198–214
26. Keet CM, Ławrynowicz A (2016) Test-driven development of ontologies. In: Proceedings of 13th European semantic web conference, Heraklion, Crete, Greece, May 29–June 2. Springer, Berlin, pp 642–657
27. Kirch W (ed) (2008) Pearson's correlation coefficient. Springer, Netherlands
28. Kotis K, Vouros GA, Alonso JP (2004) HCOME: A tool-supported methodology for engineering living ontologies. In: Proceedings of the 2nd international workshop on semantic web and databases, Toronto, Canada, August 29–30. Springer, Berlin, pp 155–166
29. Lantow B (2016) OntoMetrics: application of on-line ontology metric calculation. In: Joint proceedings of the BIR 2016 workshops and doctoral consortium co-located with 15th international conference on perspectives in business informatics research, Prague, Czech Republic, September 14–16
30. Ma Y, Jin B, Feng Y (2010) Semantic oriented ontology cohesion metrics for ontology-based systems. J Syst Softw 83(1):143–152
31. Mall R (2014) Fundamentals of software engineering. PHI Learning Pvt Ltd, Delhi
32. McCabe TJ (1976) A complexity measure. IEEE Trans Softw Eng 4:308–320
33. Moser R, Pedrycz W, Succi G (2008) A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: Proceedings of the 30th international conference on software engineering, Leipzig, Germany, May 10–18. ACM, pp 181–190

34. Muller JZ (2018) The Tyranny of Metrics. Princeton University Press, Princeton
35. Noy N, Rector A, Hayes P, Welty C (2006) Defining n-ary relations on the semantic web. W3C working group note 12(4). Available at https://www.w3.org/TR/swbp-n-aryRelations/
36. Orme AM, Tao H, Etzkorn LH (2006) Coupling metrics for ontology-based system. IEEE Softw 23(2):102–108
37. Peroni S (2016) A simplified agile methodology for ontology development. In: Proceedings of the 13th OWL: experiences and directions workshop and 5th OWL reasoner evaluation workshop, Bologna, Italy, November 20. Springer, Berlin, pp 55–69
38. Pinto HS, Staab S, Tempich C (2004) DILIGENT: Towards a fine-grained methodology for Distributed, Loosely-controlled and evolvInG. In: Proceedings of the 16th European conference on artificial intelligence, Valencia, Spain, August 22–27, vol 110, p 393
39. Poveda-Villalón M, Gómez-Pérez A, Suárez-Figueroa MC (2014) OOPS! (OntOlogy Pitfall Scanner!): an on-line tool for ontology evaluation. Int J Seman Web Inform Syst 10(2):7–34
40. Pressman RS (2005) Software engineering: a practitioner's approach. Palgrave Macmillan, London
41. Presutti V, Daga E, Gangemi A, Blomqvist E (2009) eXtreme design with content ontology design patterns. In: Proceedings of the workshop on ontology patterns, collocated with the 8th international semantic web conference, Washington DC, USA, 25 October, CEUR Workshop series, pp 83–97
42. Rahman F, Devanbu P (2013) How, and why, process metrics are better. In: Proceedings of the 35th international conference on software engineering, San Francisco, USA, May 18–26. IEEE, pp 432–441
43. Ren Y, Parvizi A, Mellish C, Pan JZ, van Deemter K, Stevens R (2014) Towards competency question-driven ontology authoring. In: Proceedings of the 11th European semantic web conference, Crete, Greece, May 25–29. Springer, Berlin, pp 752–767
44. Schober D, Tudose I, Svatek V, Boeker M (2012) OntoCheck: verifying ontology naming conventions and metadata completeness in protégé 4. J Biomed Semant 3(S-2):S4
45. Shatnawi R, Li W (2008) The effectiveness of software metrics in identifying error-prone classes in post-release software evolution process. J Syst Softw 81(11):1868–1882
46. Smith B, Ashburner M, Rosse C, Bard J, Bug W, Ceusters W, Goldberg LJ, Eilbeck K, Ireland A, Mungall CJ et al (2007) The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. Nat Biotechnol 25(11):1251
47. Sommerville I (2010) Software engineering, 9th edn. Addison-Wesley, London
48. Suárez-Figueroa M, Gómez-Pérez A, Villazón-Terrazas B (2009) How to write and use the ontology requirements specification document. In: Proceedings of the international conference on on the move to meaningful internet systems, Ilamoura, Portugal, November 1–6. Springer, Berlin, pp 966–982
49. Suárez-Figueroa MC, Gómez-Pérez A, Fernández-López M (2012) The NeOn methodology for ontology engineering. In: Ontology engineering in a networked world. Springer, Berlin, pp 9–34
50. Suárez-Figueroa MC, Aguado de Cea G, Gómez-Pérez A (2013) Lights and shadows in creating a glossary about ontology engineering. Terminology 19(2):202–236
51. Tartir S, Arpinar IB, Moore M, Sheth AP, Aleman-Meza B (2005) OntoQA: Metric-based ontology quality analysis. In: Proceedings of IEEE workshop on knowledge acquisition from distributed, autonomous, semantically heterogeneous data and knowledge sources at 2005 IEEE international conference on data mining, Houston, USA, November 27, pp 45–53
52. Uschold M, Gruninger M (1996) Ontologies: Principles, methods and applications. Knowl Eng Rev 11(2):93–136
53. Vrandečić D, Gangemi A (2006) Unit tests for ontologies. In: Proceedings of the 2006 international conference on the move to meaningful internet systems: OTM 2006 workshops, Montpellier, France, October 29–November 3. Springer, Berlin, pp 1012–1020
54. Vrandečić D, Krötzsch M (2014) Wikidata: a free collaborative knowledgebase. Commun ACM 57(10):78–85
55. Vrandečić D, Sure Y (2007) How to design better ontology metrics. Springer, Berlin, pp 311–325
56. Wilsdon J (2016) The metric tide: independent review of the role of metrics in research assessment and management. Sage, Thousand Oaks
57. Yao H, Orme AM, Etzkorn L (2005) Cohesion metrics for ontology design and application. J Comput Sci 1(1):107–113
58. Zhe Y, Zhang D, Chuan Y (2006) Evaluation metrics for ontology complexity and evolution analysis. In: Proceedings of the IEEE international conference on e-business engineering, Shanghai, China, October 24–26. IEEE Computer Society, pp 162–170

Alba Fernández-Izquierdo is a Ph.D. student at the Artificial Intelligence Department of the Computer Science Faculty of Universidad Politécnica de Madrid, in the Ontology Engineering Group. Previously she finished her studies in Computer Science (2015) by Universidade de Santiago de Compostela, and she holds a Master degree in Artificial Intelligence Research (2016) from the Universidad Politécnica de Madrid. Her research activities focus on ontological engineering and ontology verification.



María Poveda-Villalón is an assistant professor in the Artificial Intelligence Department of the Universidad Politécnica de Madrid and is also part of the Ontology Engineering Group research lab. Her research activities focus on Ontological Engineering, Ontology Evaluation, Knowledge Representation and the Semantic Web. Previously she finished her studies in Computer Science (2009) by Universidad Politécnica de Madrid. She has worked in the context of Spanish research projects as well as in European projects such as ETSI STF for SAREF extensions, BIMERR, VICINITY, READY4SmartCities, easyTV and NeOn. She has contributed to the organization of the "Linked Data in Architecture and Construction Workshop" since 2015, the "13th OWL: Experiences and Directions Workshop and 5th OWL reasoner evaluation workshop" in 2016, the "Linked Energy Data Vocamp" in 2015 and the "Catching up with ontological engineering: To git-commit and beyond" tutorial at EKAW2018.

Asunción Gómez-Pérez is Vice-Rector for Research, Innovation and Doctoral Studies (2016-). She is Fellow of the European Academy of Science (2018) and Full professor on Artificial Intelligence. She is member of the group of experts advising in the national committee R&D Spanish Strategy in Artificial Intelligence. She has received the Award ARITMEL—National Prize of Computer Science 2015, the Annual Award of Investigation of the UPM (2015), the National Prize Ada Byron for the Technologist Woman in its second edition (2015) and the Know Square Prize for her Dissemination trajectory in Artificial Intelligence (2018). Her research areas include: Ontological Engineering, Semantic Web, Linked Data, Natural Language Processing, Multilingualism in Information and knowledge management. She directed significant amount I + D + i research and engineering projects in Spain and Europe. She acts as reviewer of European projects in the European Commission (including ERC), and several European, International and national agencies.

Raúl García-Castro is Associate Professor at the Computer Science School at Universidad Politécnica de Madrid (UPM), Spain. In 2008 he obtained a Ph.D. in Computer Science and Artificial Intelligence at UPM, which obtained the Ph.D. Extraordinary Award. His research focuses on ontological engineering, semantic interoperability and ontology-based data and application integration. He regularly participates in standardization bodies and in the program committees of the conferences and workshops that are most relevant in his field, having also organised several international conferences and workshops.