




- ORIGINAL ARTICLE -

# Specifying and Analyzing a Software Testing Ontology at the Top-Domain Ontological Level

## Especificando y Analizando una Ontología de Pruebas de Software en el Nivel Ontológico de Dominio Superior

Guido Tebes , Luis Olsina , Denis Peppino  and Pablo Becker   
 GIDIS\_Web, School of Engineering, UNLPam, General Pico, La Pampa, Argentina  
 {guido\_tebes, olsinal, beckerp}@ing.unlpam.edu.ar; denispeppino92@gmail.com

### Abstract

One of the Software Engineering areas that supports quality assurance is testing. Given that specific processes, artefacts, methods and ultimately strategies for software testing involve a large number of domain concepts, it is valuable to have a robust conceptual base, that is, a software testing ontology that defines the terms, properties, relationships and axioms explicitly and unambiguously. Ontologies for instance foster a clearer terminological understanding of process and method specifications for strategies, among many other benefits. After analyzing both the results of a conducted Systematic Literature Review of primary studies on conceptualized software testing ontologies and the state-of-the-art of testing-related standards, we decided to develop a software testing top-domain ontology named TestTDO that fits our goals. Therefore, this article specifies development, verification and validation aspects of the TestTDO, which was built following the Design Science Research approach.

**Keywords:** ontologies, software testing, terminologies, top-domain ontological level, vocabularies.

### Resumen

Un área de la Ingeniería del Software que da soporte al aseguramiento de la calidad es *testing*. Dado que los procesos, artefactos, métodos y, en última instancia, estrategias específicas para pruebas de software involucran una gran cantidad de conceptos de dominio, es valioso tener una base conceptual robusta, es decir, una ontología de pruebas de software que defina los términos, propiedades, relaciones y axiomas explícitamente y sin ambigüedades. Las ontologías, por ejemplo, fomentan una comprensión terminológica clara de las especificaciones de procesos y métodos para las

estrategias, entre muchos otros beneficios. Después de analizar los resultados de una Revisión Sistemática de Literatura de estudios primarios sobre ontologías de pruebas de software conceptualizadas y el estado del arte de los estándares relacionados al área, decidimos desarrollar una ontología de dominio superior de pruebas de software llamada TestTDO que se ajuste a nuestros objetivos. Por lo tanto, este artículo especifica aspectos de desarrollo, verificación y validación de TestTDO, que fue construida siguiendo el enfoque de *Design Science Research*.

**Palabras claves:** Nivel Ontológico de Dominio Superior, Ontologías, Terminologías, Pruebas de Software, Vocabularios.

### 1. Introduction

Software testing is a critical process for software quality assurance. It is also a complex domain since testing has a large number of specific methods, processes and strategies. All of them involve many specific domain concepts. Hence, it is valuable to have a robust conceptual base, i.e., a conceptualized software testing ontology that explicitly and unambiguously defines the terms, properties, relationships and axioms or constraints.

A benefit of having a suitable testing ontology is to improve the software testing-related information exchange between agents avoiding ambiguity problems. Furthermore, one desirable feature of a software testing ontology is that it covers concepts related to static and dynamic testing since software testing standards as ISO 29119-1 [1] and International Software Testing Qualifications Board (ISTQB) [2] consider these kinds of testing. Additionally, the software testing ontology should be linked to Non-Functional Requirements (NFRs) and Functional Requirements (FRs) ontologies because software testing strategies are useful to verify and validate both

types of requirements. In particular, we aim to have a suitable software testing ontology that terminologically nourishing specifications of methods and processes for a family of testing strategies to be developed.

To adopt or adapt an existing testing ontology, or to develop a new one, we have followed the Design Science Research (DSR) approach [3], [4]. This is a rigorous and high-level research approach, which proposes the construction of artefacts to provide useful and effective solutions to a relevant problem in a given domain. Artefacts must be innovative and useful solutions to a non-trivial problem. The artefact development implies a cycle of design-construction-verification and validation activities, which should iterate as many times as necessary before the artefact (already verified and validated) is communicated for its use.

Firstly, to find out existing solutions (i.e., conceptualized software testing ontologies) to our problem, we conducted a Systematic Literature Review (SLR) [5]. We selected 12 primary studies documenting conceptualized testing ontologies, which were evaluated from the ontological quality standpoint. This includes characteristics such as structural quality, terminological coverage quality, among others [5], [6].

In general, we have observed that most of them have a lack of non-functional testing and static testing terminological coverage. Moreover, the 12 retrieved ontologies present opportunities to improve their structural quality for different reasons such as: i) they do not have all their terms, non-taxonomic relationships and properties defined as well as axioms specified; ii) they do not have non-taxonomic relationships or, if they do, they do not have well-balanced taxonomic and non-taxonomic relationships. Furthermore, all of them are not directly linked with NFRs and FRs concepts.

Since current test ontologies are not fully suitable for our aim, that is, to terminologically nourish specifications of methods and processes of a family of testing strategies to be developed, we have decided to build a new software testing ontology named TestTDO (i.e., a Software Testing Top-Domain Ontology). Note that we have used the DSR approach as a generic framework to carry out this research work as a whole, i.e., from the conducted SLR to the verification, validation and communication of the ontology. We have also used METHONTOLOGY [7] but only in one DSR activity to build TestTDO.

It is worth mentioning that TestTDO is placed into an ontological architecture called FCD-OntoArch (*Foundational, Core, and Domain Ontological Architecture for Sciences*) [8]. It is a four-layered ontological architecture that considers foundational, core, domain and instance levels. In FCD-OntoArch, ontologies at the same level can be related to each

other. Also, ontologies at lower levels can be semantically enriched by ontologies at higher levels. For example, TestTDO at the domain level is enriched by concepts of the ProcessCO ontology placed at the core level. In turn, the latter is enriched by concepts of ThingFO at the foundational level, as addressed later on.

In summary, the contribution of this work is to specify and discuss aspects of the TestTDO conceptualization (i.e., its terms, properties, relationships and axioms), and its ontological quality evaluation. Also, by using different black-box and white-box testing methods, we analyze aspects of its static and dynamic verification. Moreover, to validate TestTDO, we use a human assessment approach.

It is important to remark that this article thoroughly documents TestTDO in its version 1.2. In [9], we showed summarized results of the TestTDO conceptualization in version 1.0, without being implemented. Conversely, this manuscript contains new aspects of the TestTDO development and implementation, as well as its dynamic verification by using test cases.

The remaining sections of this paper are arranged as follows. Section 2 gives a summary of related work on conceptualized testing ontologies, which were identified by conducting the abovementioned SLR. Section 3 provides an overview of FCD-OntoArch, which contains some ontologies that are part of the TestTDO context, such as ProcessCO and SituationCO. Section 4 documents and analyzes the main concepts, properties, relationships and axioms included in TestTDO. Section 5 illustrates how TestTDO was verified and validated using different approaches. Finally, Section 6 summarizes conclusions and future work.

## 2. Why one more Software Testing Ontology?

We previously mentioned that to achieve our goal, that is, to adopt or adapt an existing testing ontology, or develop a new one, we follow the DSR approach [3], [4]. According to the DSR process [4], the first activity to perform is to define the problem/solution in which the solution requirements artefact is produced. We preliminarily define 9 Solution Requirements (SRs), which are documented at <http://bit.ly/SWTestingOnto-SolReqs>. As an example, SR#2 states that “*the ontology must contain testing concepts that can be related to functional and non-functional requirements concepts*”. Therefore, the testing component must be related to the FRsTDO and NFRsTDO components, as shown on the right side of Fig. 1.

Using as input the SRs artefact, the next DSR activity to be carried out is investigating current

solutions through literature research and/or getting expert feedback. To do this, that is, to find existing solutions (conceptualized software testing ontologies) to our problem and that fit the established SRs, we conducted a SLR [5]. As a result, we have selected, analyzed and evaluated 12 ontologies that document a conceptualization of the software testing domain.

Recall that the main goal of TestTDO is to terminologically nourish specifications of methods and processes of a family of testing strategies to be developed. In other words, TestTDO has to serve as the common vocabulary for this family. Many testing strategies (also named frameworks, approaches, methodologies) contain a set of processes and/or methods (or techniques) that give support testers to achieve some testing-related objectives. In general, these approaches provide a glossary as a vocabulary. Thus, the ISO 29119 standard contains a set of testing processes [10], testing techniques [11] and a glossary of testing terms [1]. On the other hand, ISTQB also provides a glossary for the testing domain [2]. However, a glossary does not have the semantic and structural richness that an ontology has. A glossary only contains a set of terms and their definitions, and therefore does not explicitly describe what are the relationships between terms and what are the properties of these terms. Also, a graphic conceptual model is missing in these glossaries, which are plain text only. These issues could lead to ambiguities and make it difficult to understand the test domain. Therefore, we chose to use an ontology instead of a glossary.

None of the 12-selected ontologies analyzed in [5] was built with the same aim that TestTDO pursues. Most of them had been built to achieve a very specific aim. So most selected ontologies do not have similar scope as TestTDO, which is broader and top domain. For instance, in [12] the authors present PTOntology, which models the performance testing domain; or the ontology presented in [13], which focuses on scenario-based testing.

To some extent, the ontologies with the closest objective to TestTDO are ROoST [14] and the ontology presented in [15]. ROoST was developed for establishing a common conceptualization about the software testing domain, focusing on the testing process, but its scope only reaches the dynamic and functional testing, without considering static and non-functional testing. On the other hand, the ontology documented in [15] was built to represent general software testing knowledge. Regarding its scope, this ontology only has terms for the formal review process, and not consider generic terms for the testing process. Also, it does not contain top-domain terms as test basis, test result, actual result, in addition to terms related to testing project, goals, requirements and environment. Besides, some terms do not share

the core-level ontological vision of FCD-OntoArch. For example, testing methods are types of test goals, while in FCD-OntoArch, a method has different semantics than a goal.

In addition, the selected ontologies in the SLR were evaluated from the ontological quality standpoint. To do this, we developed a NFRs tree (1<sup>st</sup> column of Table 4 in sub-section 5.2) that specifies characteristics and attributes related to the Ontological Quality, which is the root of the NFRs tree (with code 1). The sub-characteristics are Ontological Structural Quality (1.1), Domain-specific Terminological Coverage Quality (1.2) and Compliance to other Vocabularies (1.3). This tree was built taking into account some quality practices described by [6] for ontology design, for which they identify dimensions and features for “beautiful ontologies”. Two (out of three) dimensions are formal structure and conceptual coverage, which are characterized by if the ontology is designed in a principled way; it is formally rigorous; it implements also non-taxonomic relations; it has a good domain coverage; it implements an international standard; and it reuses foundational ontologies, among others.

Besides, we have developed a set of metrics and indicators, some of them documented in [5], to evaluate the selected ontologies. The evaluation results for the 12-selected ontologies are shown in Table 11 of [5]. In that study, we have used the metaphor of the three-coloured semaphore to identify the satisfaction acceptability level achieved. The red square (■) with values within the [0;60] range, in the percentage scale, indicates an “unsatisfactory” acceptability level; the orange rhombus (◆) [60;85] indicates a “marginal” level; and the green circle (●) [85;100] indicates a “satisfactory” level.

In summary, the best-ranked ontology regarding the Ontological Quality (1) was [14], although it did not achieve a satisfactory level. This ontology, called ROoST, reached 79.54% (◆). It lacks NFRs (1.2.4) and static (1.2.1) testing terminological coverage and there is no direct link with FR and NFR terms –recall the SR#2 mentioned above. We needed a top-domain ontology with higher coverage since we plan to develop more specific testing domain ontologies and strategies, e.g., for performance testing, inspections, among others.

Furthermore, ROoST is the only selected ontology that is based on a foundational ontology. Although ROoST is embedded in a network of ontologies whose root is the UFO foundational ontology [16], we consider the ontologies used to enrich ROoST (i.e., UFO and mainly its derived process core ontology) are somewhat difficult to adapt and harmonize with the ontological components that we present in Section 3. We argue that UFO is a bit complex since it is made up of a set of ontologies, namely: UFO-A (endurants), UFO-B

(perdurants or events) and UFO-C (social entities, built on top of UFO-A and B). Instead, FCD-OntoArch includes just one foundational ontology, i.e., ThingFO, which has a small set of terms that makes it easy to reuse and specialize in lower-level ontologies.

According to the DSR process, the above-summarized SLR analysis is part of the information synthesis artefact produced in the “investigate current solutions” activity. Using as input this synthesis and the SRs, the next activity to carry out is to analyze the problem/solution relevance. As a result, the relevance report is produced, which describes the relevance of the problem/solution to decide whether it will be addressed by the DSR approach.

Therefore, we have stated in the relevance report artefact the main reasons why to address the construction of TestTDO through the DSR approach, namely: i) there is no SLR-selected ontology that satisfactorily fulfils all the stated SRs and also meets a high level of acceptability of ontological quality, ii) to build/adapt an ontology is not a routine task, conversely, it is a complex task due to: 1) it is necessary to consider different sources of definitions of terms for the testing domain (such as international standards, other ontologies, etc.) to obtain higher coverage than the current solutions; 2) the new ontological design must allow the linkage with FRsTDO and NFRsTDO components in the context of the FCD-OntoArch architecture; and 3) the ontology should be based on (enriched with) terms of higher-level ontologies at the core and/or foundational level.

Finally, considering the abovementioned relevance report, we have decided to build a software testing ontology that fits our goal. To develop TestTDO, we have considered the best-ranked features of the 12-selected ontologies. TestTDO is a top-domain ontology for software testing, which is semantically enriched with higher-level ontologies, both core and foundational. It also serves as the basis for the development of new lower-level domain ontologies. Ultimately, this will permit us to build specific software testing strategies. In other words, TestTDO will provide us with the semantics to develop specifications of software testing processes and methods, and their grouping into a family of testing strategies for achieving testing purposes.

### 3. Overview of the Four-layered Ontological Architecture and some of its Ontologies

This Section aims to briefly illustrate some of the ontologies that belong to FCD-OntoArch as TestTDO is related to them. Also, we apply a static verification method to TestTDO in order to verify its integration in the context of FCD-OntoArch. This testing method is described in Section 5. Note that the interested reader can access more descriptions of some FCD-OntoArch's ontologies in the following references: ThingFO [8]; ProcessCO<sup>1</sup> (or its predecessor [17]); SituationCO<sup>2</sup>; NFRsTDO<sup>3</sup> and FRsTDO<sup>4</sup>.

As commented in the Introduction Section, TestTDO is placed at the top-domain level into FCD-OntoArch. This is a four-layered ontological architecture, which considers foundational, core, domain and instance levels. In turn, the domain level is split down into two sub-levels, namely: top-domain and low-domain. As depicted in Fig. 1, ontologies at the same level can be related to each other, except for the foundational level where there is only the ThingFO ontology. Ontologies at lower levels can be semantically enriched by ontologies at upper levels. For example, TestTDO placed at the top-domain level is mainly enriched by terms, properties and relationships of the SituationCO and ProcessCO ontologies placed at the core level. In turn, both are enriched by the concepts of ThingFO.

ThingFO terms such as Thing, Thing Category and Assertion semantically enrich terms of components at lower levels. Thing represents a particular or concrete, tangible or intangible object of a given particular world, but not a universal category or class –which is modelled by the term Thing Category.

Additionally, the term Assertion is defined as “*positive and explicit statement that somebody makes about something concerning Things, or their categories, based on thoughts, perceptions, facts, intuitions, intentions, and/or beliefs that is conceived with an attempt at furnishing current or subsequent evidence*”. To be valuable, actionable and ultimately useful for any science, an Assertion should to a great extent be verified and/or validated by theoretical and/or empirical evidence. Assertions can be represented and modelled using informal, semiformal or formal specification languages.

Concerning a Thing and using assertions, we can specify aspects of its substance, relations, structure, behaviour, constraints, intention, situation, quantity and quality, among other aspects. For example, the

<sup>1</sup> <http://dx.doi.org/10.13140/RG.2.2.27140.55688>

<sup>2</sup> <http://dx.doi.org/10.13140/RG.2.2.21065.36968>

<sup>3</sup> <http://dx.doi.org/10.13140/RG.2.2.34457.65129>

<sup>4</sup> <http://dx.doi.org/10.13140/RG.2.2.31659.26400>

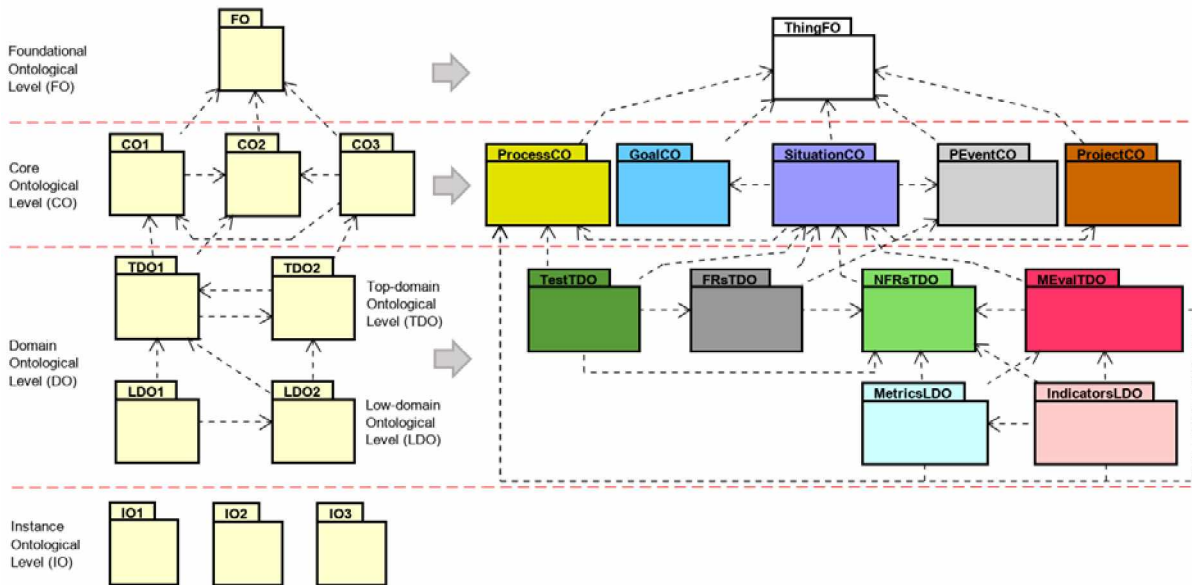


Fig. 1 Four-layered ontological architecture, which considers Foundational, Core, Domain and Instance levels. Also, some conceptual components or modules are shown at the corresponding level. Note that NFRs stands for Non-Functional Requirements, FRs for Functional Requirements, MEval for Measurement and Evaluation, and PEvent for Particular Event.

conceptualization of an ontology as an artefact (e.g., TestTDO in Fig. 2) represents primarily a combination of substance-, relation-, structure-, intention-, and situation-related assertions. The axioms of an ontology can be considered constraint-related assertions.

SituationCO includes terms –some borrowed from other core components- such as Human Agent, Organization, Project, Target Entity, Context Entity with semantic of Thing, and the term Goal with semantic of Assertion. In turn, a Goal can be Specific or Generic. Briefly, a Human Agent/Organization conceives/establishes Goals that are operationalized by Projects. A Goal implies a Situation, which can be specified by a Situation Model. Therefore, a Situation Model represents an Artefact that specifies and models Situations of a given particular or generic world.

Furthermore, a Situation can be Particular or Generic. A Particular Situation is a Situation-related Assertion on Particulars that explicitly states and specifies the combination of particular circumstances, episodes and relationships/events embracing Target Entities and their surrounding Context Entities, which is of interest and relevant to be represented by a Human Agent/Organization with an established Specific Goal. Depending on the Specific Goal's purpose, Target Entities can be for instance Developable Entity (e.g., a document, a source code, etc.), Evaluable Entity (e.g., a work product, a system, etc.), or Testable Entity, which has the semantic of Developable or Evaluable in a given Particular Situation.

The ProcessCO ontology is a core ontology that specifies the terms, relationships and properties for

Work Processes. ProcessCO includes terms with semantic of Thing such as Work Entity (which can be, in turn, a Work Process, Activity or Task), Product Entity (Work/Natural Product, Artefact, Outcome, Service) and Resource Entity (Agent, Method, Strategy, Tool, among others). Any well-specified process for a given domain should document aspects such as specific tasks and activities, artefacts, agents, methods, among other resources. Moreover, any well-specified method should document its procedure and rules. Therefore, having an ontology that explicitly defines these terms and their relationships and properties, is very useful to avoid ambiguities, inconsistencies and incompleteness in a process specifications of a certain application domain as for example, in the testing domain. Since TestTDO aims to terminologically nourish the specifications of methods and processes for a family of testing strategies to be developed, it is very important to consider ProcessCO terms for enriching TestTDO domain terms.

The terms Work Process, Activity and Task are kinds of Work Entities. Work process is composed of sub-processes or activities. In turn, an Activity is formed by sub-activities or tasks. A Task is an atomic, fine-grained Work Entity that cannot be decomposed. Also, they involve common Roles, consume Product Entities, produce Work Products and have Conditions –both preconditions and postconditions. Furthermore, a Work Entity has assigned Work Resources, such as Methods, Tools, Strategies, among others.

Another important concept of ProcessCO is Work Product, which is a Product Entity. In turn, Outcome, Artefact and Service are kinds of work products. For example, Outcome is defined as “*Work Product that*

is intangible, storable and processable”, while Artefact “is a tangible or intangible, versionable Work Product, which can be delivered”.

A Work Entity has a description (or work description), which specifies the steps for achieving its objective. It represents “what” should be done instead of “how” should be performed. The “how” is represented by the Method term, i.e., the specific and particular way to perform the specified steps for instance in a task. Note that the Method concept has the procedure and rules properties. A procedure is an arranged set of method instructions or operations, which specifies how the steps of a description of a work entity must be performed. Whereas a rule is a set of principles, conditions, heuristics, axioms, etc., associated with the procedure.

#### 4. TestTDO: A Top-Domain Ontology for Software Testing

As previously commented, we use DSR [4] as a baseline research approach to build our software testing ontology. In a nutshell, this approach is a rigorous research strategy, proposing the construction of artefacts to provide useful and effective solutions to a relevant problem in a given domain. Artefacts must be innovative and useful solutions to a non-trivial problem. The artefact development implies a cycle of design-construction-verification and validation (V&V) activities, which should iterate as many times as necessary before the artefact (already verified and validated) is communicated for its use. In the following sub-sections, we describe the conceptualization of TestTDO while using some DSR activities.

##### 4.1. Artefact Requirements, Competency Questions and TestTDO in the framework of DSR

Following the DSR process, we specify research questions (RQs) and Artefact Requirements (ARs). To produce them, we use as input the abovementioned SRs (<http://bit.ly/SWTestingOnto-SolReqs>), and the information synthesis produced in the cited SLR. The right formulation of the RQs is paramount in any research study as they conduct the design-development-V&V research cycle, and transmit its essence. In our case, to build the software testing ontology, we have formulated three main RQs, which in turn were divided into sub-RQs.

Once the RQs were established, we also used them to produce the ARs, which were taken into account in the development and V&V cycle of TestTDO. Note that all RQs and ARs are documented in [http://bit.ly/TestTDO-RQs\\_ARs](http://bit.ly/TestTDO-RQs_ARs).

Just to mention a few examples, RQ#1.1 states

“Should the ontology to be developed be of low-domain or top-domain level?”. Since we plan to develop a set of testing strategies for some specific kinds of testing as performance testing, security testing, static testing by reviews, among others, we have established the AR#1: “Design and build a top-domain ontology”. Besides, we have considered the SR#8 (“The software testing ontology should be at the top-domain ontological level”) to specify this requirement.

RQ#1.2 is “What are the most robust and rich documented terminologies (structured as glossaries, taxonomies or ontologies) for the software testing domain?”. We have formulated this RQ since an ontology is a shared conceptualization and therefore it is very important to consider and reuse others terminologies, in particular standard glossaries. When we conducted the SLR, we found and analyzed a set of robust software testing ontologies in addition to software testing international standard glossaries. Taking into account this information, we stated AR#2: “Consider mainly: i) international standard glossaries documented in ISO 29119-1 [1] and ISTQB [2]; and ii) the ROoST [14] domain ontology and the [15] top-domain ontology, which were the two-best ranked among the 12-selected and evaluated ontologies in the before cited SLR”.

As a final example, RQ#1.5 establishes “What is the suitable methodology for ontology development to be used?”. We have formulated this RQ since it is important to consider some engineering methodology to build an ontology in the A2 activity (Design and Develop the Solution) of the DSR process. Related to RQ#1.5 we stated AR#6 “Use METHONTOLOGY [7] for the development of the ontology until its conceptualization stage. Also consider its evaluation and documentation activities”. We selected METHONTOLOGY since it is a well-structured methodology used for developing ontologies from scratch and provides good guidelines for organizing the activities during ontology development [15].

In this situation, we have specified 25 Competency Questions (CQs), which are requirements related to the specific scope of the ontology to be developed. All of them are documented in <http://bit.ly/TestTDO-CQuestions>. For example, CQ1 states “What are the work products produced by a testing design activity?”. These CQs cover the necessary and sufficient aspects of the ontology in order to be extended by lower-domain ontologies. For this, it must consider terms related to static and dynamic testing, as well as functional and non-functional testing. Besides, it must consider concepts of testing work entity, test work product, testing method, testing agent, project, goal, requirement and entity, which should be semantically enriched with concepts from other ontologies at the core and foundational levels.

Once the ARs were produced, we have designed and built the ontology in the A2 activity of the DSR process. Recall that we selected METHONTOLOGY as a development methodology for this activity. Regarding its process, it contains a set of activities to be completed without implying an order of execution of such activities. In short, the main activities are specification, knowledge acquisition, conceptualization, integration, implementation, evaluation and documentation. The reader can find the details of each activity in [7].

At this point, it is important to highlight that we only performed the conceptualization, integration and implementation activities through METHONTOLOGY when we carry out activity A2 (Design and Develop the Solution) of the DSR process. We have not considered the other activities of the METHONTOLOGY process as, to a large extent, they were covered by other activities of the

DSR process. For example, the METHONTOLOGY evaluation activity is covered by DSR activity A3 (Execute V&V). Note that we do not describe a comparison between the activities of DSR and METHONTOLOGY because it is outside the scope of this work.

Moreover, we decided to use DSR as the main approach for developing our ontology, since DSR is a broad approach that is useful for developing any artefacts and METHONTOLOGY is devoted solely to ontology development. Our goal is to develop a set of artefacts, including TestTDO, by using DSR, such as testing strategies and other lower-level domain testing ontologies. Finally, please note that it is possible to partially or fully use other methodologies as DSR activities are conducted, as in this case we partially use METHONTOLOGY. Next, we describe the activities of METHONTOLOGY applied to build TestTDO.

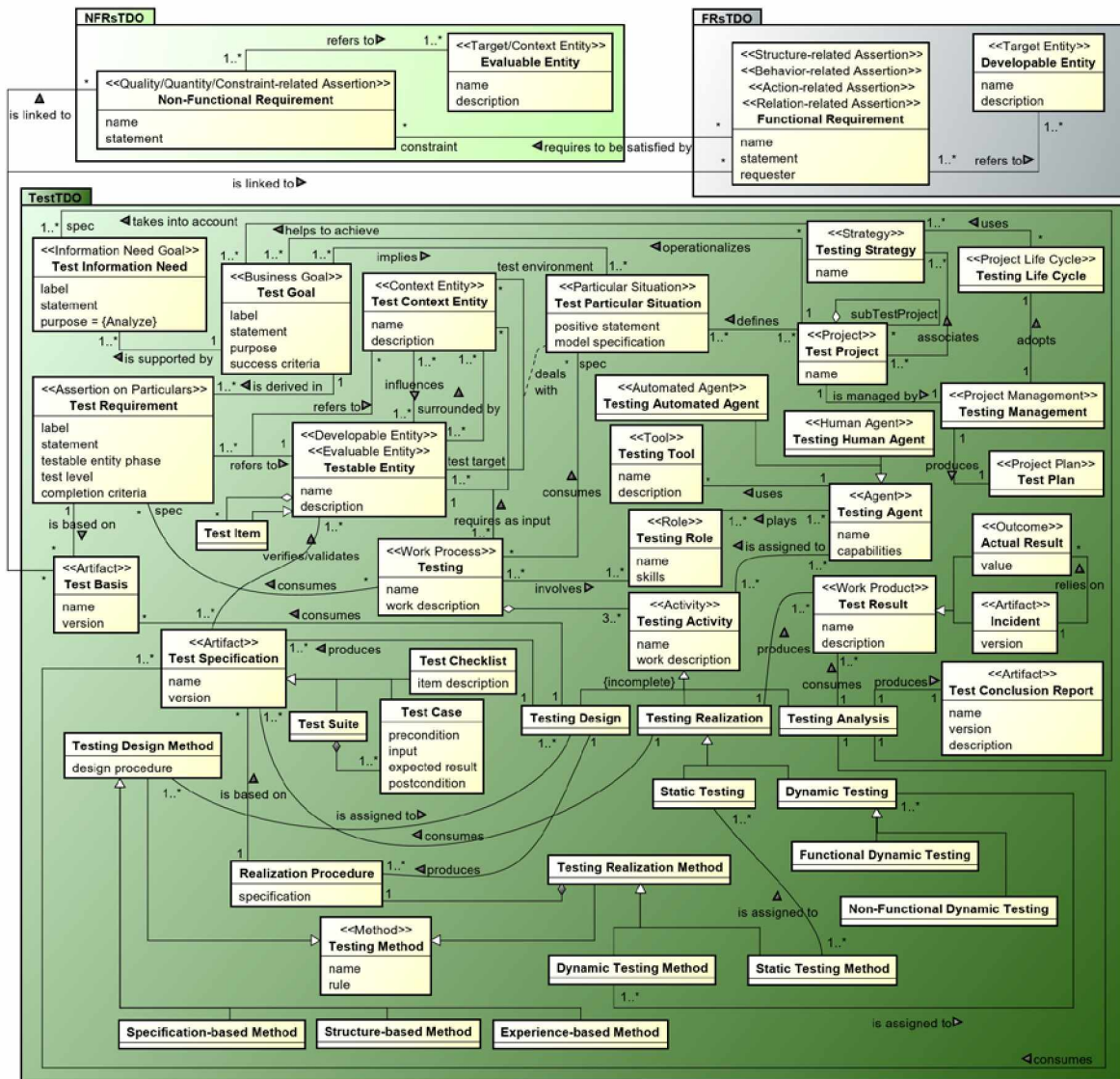


Fig. 2 Main terms, properties and relationships of the TestTDO ontology and its relation with Non-Functional Requirement and Functional Requirement terms.

In the conceptualization activity, we have obtained the conceptual model of TestTDO. As a result, TestTDO has defined 44 terms, 50 properties, 42 non-taxonomic relationships as well as 17 axioms specified in first-order logic. Fig. 2 shows all the concepts of TestTDO (that is, the whole picture), as well as its relation with the Non-Functional Requirement and Functional Requirement terms.

To get the TestTDO conceptualization, METHONTOLOGY proposes first to build a glossary, i.e., concepts with their definitions. This glossary should include terms, properties and non-taxonomic relationships. To develop this glossary, as a starting point we have used the ISO 29119 and ISTQB standard glossaries to gather the set of primary concepts to be included in TestTDO. Note that we have selected the concepts for this glossary keeping in mind the conceptual patterns of the FCD-OntoArch core ontologies and the scope of TestTDO represented in the 25 CQs.

After the construction of this glossary, we have analyzed the semantics provided by the standard glossaries and the existing ontologies against the concepts included in the glossary, and we have established a semantic correspondence with the terms of the FCD-OntoArch ontologies at the core level. For example, we identified that Testing Design is an Activity of ProcessCO.

Next, we have identified and established generalizations (a taxonomy) between the concepts. We did this by analyzing the semantics of the concepts. In this step, we have identified 'kind of' and 'whole-part' relationships. Moreover, the conceptual patterns of the FCD-OntoArch ontologies at the core level were very useful to perform this task. For example, the Work Entity pattern of ProcessCO involves a whole-part relationship in which a Work Process has one or more Activities.

The next step we took was to consider the reuse of some properties and non-taxonomic relationships that belong to the FCD-OntoArch ontologies at the core level. For example, we reused the relationship "Project operationalizes Goals" in TestTDO between Test Project and Test Goal. Finally, we have obtained a conceptual model by using a UML class diagram that represents the glossary of terms.

Regarding the integration activity, we have reused, either partially or fully, definitions of the ISO 29119 and ISTQB standard glossaries as well as the definitions of the FCD-OntoArch ontologies at the core level that semantically enrich TestTDO concepts. On the other hand, in the implementation activity, we have obtained the OWL version of TestTDO. More details of the TestTDO implementation will be covered in sub-section 5.4.

In the following sub-sections, we describe the conceptualization of TestTDO in parts using the following text convention: ontology terms begin with

capital letters, properties are in italics, and relationships are underlined. Note that this article doesn't cover all concepts of TestTDO for space reason, although the reader can access all definitions of terms, properties and relationships, as well as the axioms' specifications at <http://arxiv.org/abs/2104.09232>.

Note that sub-sections 4.2, 4.3 and 4.4 represent a significant extension of Section 4 in [9]. Thus, we have included TestTDO in parts to better illustrate the ontology conceptual blocks due to the number of terms; we show TestTDO in its latest version 1.2, which includes new concepts and axioms with respect to its initial version documented in [9]; and we explicitly clarify some details that involve understanding a set of related concepts, for example, when a Testable Entity has the semantics of Developable or Evaluable Entity. Lastly, we intertwine some discussions accordingly to enrich the ontology documentation.

#### 4.2. TestTDO concepts related to Project/Goal/Requirement/Entity

To cover the test requirement- and entity-related scope, TestTDO has terms such as Test Requirement, Test Basis, Testable Entity, Test Item, Test Context Entity and Test Particular Situation as shown in Fig. 3. These terms are semantically enriched with ThingFO, ProcessCO, and SituationCO terms as mentioned in Section 3. A Test Requirement states, taking into account the Test Goal's *purpose*, what must be verified/validated of a Testable Entity (and/or Test Item) based on the Test Basis, if any. Therefore, a Test Requirement has a *statement* that refers to a Testable Entity. Additionally, a Test Requirement can include details of test environment requirements, which always refers to Test Context Entities. Also, a Test Requirement must include the *test level*, which represents a kind of test that delimits the scope of the Testable Entity and its context. Examples of kinds of *test levels* commonly cited in the literature for Dynamic Testing are "unit", "integration", "system" and "acceptance". We can also include the "document" *test level* for Static Testing.

Test Basis is an Artefact used by Testing Design Methods for designing Test Cases and Checklists. So, the Test Basis represents a thing that may come from development and/or maintenance such as requirements specification, architectural design, documented source code, etc., which in turn could be linked to NFR and/or FR (terms with semantic of) Assertions.

Furthermore, a Testable Entity is a concrete object able to be tested. A Testable Entity always is surrounded by Test Context Entities, which influence it. Test Context Entity represents the concrete Context Object in which the Testable Entity is



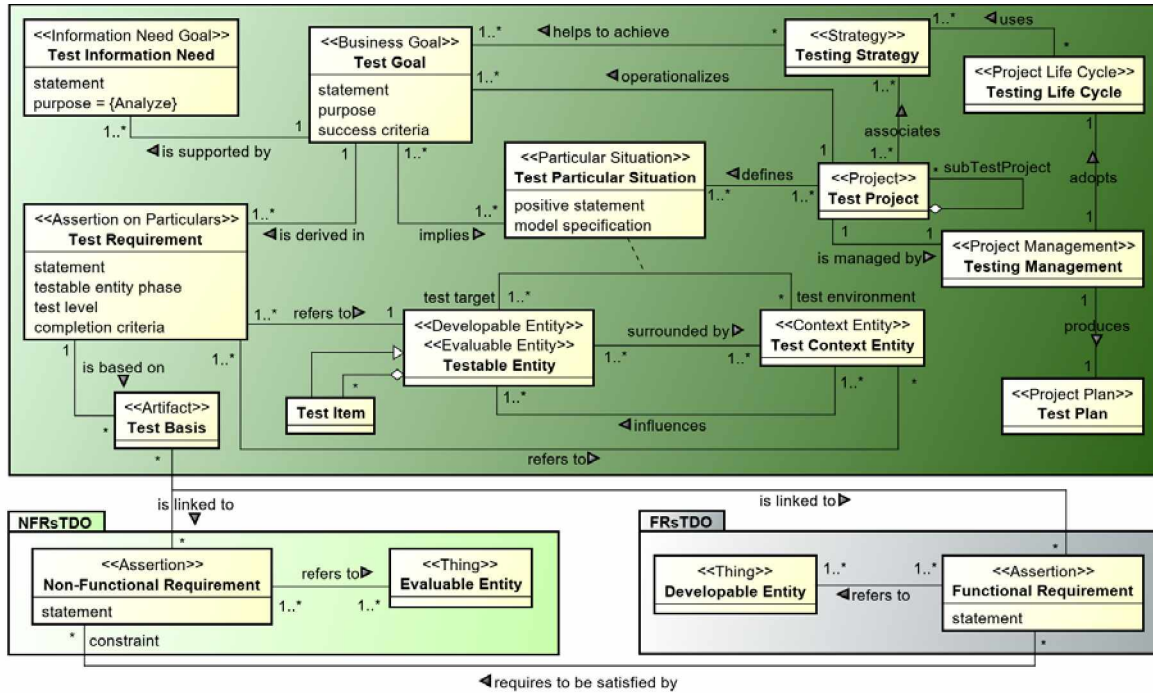


Fig. 3 Fragment of TestTDO terms, relationships and properties related to Project/Goal/Requirement/Entity and its relation with Non-Functional Requirement and Functional Requirement terms.

situated. Although there are always Test Context Entities that surround the Testable Entity –since a Thing is never isolated-, depending on the Test Particular Situation defined by the Test Project, Testing Activities may consider the test environment or not. Therefore, a Test Particular Situation (with semantic of Particular Situation from SituacionCO) represents an association between one or more Testable Entities in the role of test target and none or many Test Context Entities in the role of test environment.

Additionally, depending on the Test Particular Situation implied by the Test Goal, Testable Entity has semantic of Developable Entity or Evaluable Entity. A Test Particular Situation in which the Testable Entity has semantic of Evaluable Entity is when the Test Requirement that refers to the Testable Entity is linked to a NFR through the associated Test Basis. On the other hand, when the Test Requirement that refers to the Testable Entity is linked to a FR through the associated Test Basis, the Testable Entity has semantic of Developable Entity. To formally establish the above statements intended to rule out unwanted interpretations, we specify the following axioms:

**A\_I** description: Any Testable Entity is an Evaluable Entity iff the Test Requirement that refers to this Thing is linked to a Non-Functional Requirement.

**A\_I** specification:  $\forall te, \exists tr, \exists tb, \exists nfr: [TestableEntity(te) \wedge EvaluableEntity(te) \leftrightarrow TestRequirement(tr) \wedge TestBasis(tb) \wedge$

$NonFunctionalRequirement(nfr) \wedge refersTo(tr,te) \wedge isBasedOn(tr,tb) \wedge isLinkedTo(tb,nfr)]$

**A\_II** description: Any Testable Entity is a Developable Entity iff the Test Requirement that refers to this Thing is linked to a Functional Requirement.

**A\_II** specification:  $\forall te, \exists tr, \exists tb, \exists fr: [TestableEntity(te) \wedge DevelopableEntity(te) \leftrightarrow TestRequirement(tr) \wedge TestBasis(tb) \wedge FunctionalRequirement(fr) \wedge refersTo(tr,te) \wedge isBasedOn(tr,tb) \wedge isLinkedTo(tb,fr)]$

Lastly, it is worth highlighting for this sub-section that TestTDO has a conceptual block or pattern, which is inherited from the core ontologies. This involves the relationship between Test Projects, Test Goals, Testing Strategies and Test Particular Situations. As shown in Fig. 3, a Test Project operationalizes a Test Goal. To do this, a Test Project associates (or uses) a Testing Strategy that helps to achieve the Test Goal. Additionally, this Test Goal implies a Test Particular Situation which is defined by the Test Project.

We have observed that none of the 12-selected ontologies in the SLR has this particular conceptual pattern. This pattern recently allowed us to specify a scenario-based and Specification-based Method [18] in which different Test Particular Situations are defined in order to verify and validate Testable Entities surrounded by Test Context Entities.

### 4.3. TestTDO concepts related to Work Product/Activity

To cover the work product- and activity-related scope, TestTDO has terms such as Test Basis, Test Specification, Test Result, Test Conclusion Report and Testing Activity as shown in Fig. 4. The first 4 terms are Work Products or more specifically Artefacts (recalling that an Artefact is a Work Product in ProcessCO) that are consumed/produced by Testing Activities. A Testing Activity can be Testing Design, Testing Realization or Testing Analysis. Note that the Testing process is composed of at least the three abovementioned Testing Activities. These 3 Activities are the minimum and necessary set for all Testing process. Other activities and sub-activities can be considered, but we only made the generic activities explicit since the ontology is at the top-domain level.

A Testing Design is a Testing Activity aimed at designing (i.e., produces) a set of Test Specifications as well as Realization Procedures. Test Specification has semantic of Artefact and there are three types of it, namely: Test Checklist, Test Case and Test Suite. Test Cases contain the necessary information (e.g. *preconditions, inputs, expected results and postconditions*) to perform mainly Dynamic Testing.

Note that Test Cases with common constraints on their realization can be grouped into Test Suites. On the other hand, Test Checklists contain a list of *item descriptions* to be checked for performing mainly Static Testing.

Considering the Realization Procedure, it is an arranged set of Testing Realization Method's instructions or operations, which specifies how the Testing Realization activity must be performed using or based on the Test Specifications. Mainly when Dynamic Testing that involves Test Cases execution is carried out, this term (Realization Procedure) can be synonymous with Test Procedure. A Test Procedure as per ISO/IEC/IEEE 29119-1 [1] is a “*sequence of test cases in execution order, and any associated actions that may be required to set up the initial preconditions and any wrap up activities post execution. Test procedures include detailed instructions for how to run a set of one or more test cases selected to be run consecutively, including set up of common preconditions, and providing input and evaluating the actual result for each included test case*”.

Another Testing Activity is Testing Realization, which consumes one or more Test Specifications to produce one or more Test Results. Test Result has semantic of Work Product and there are two types of

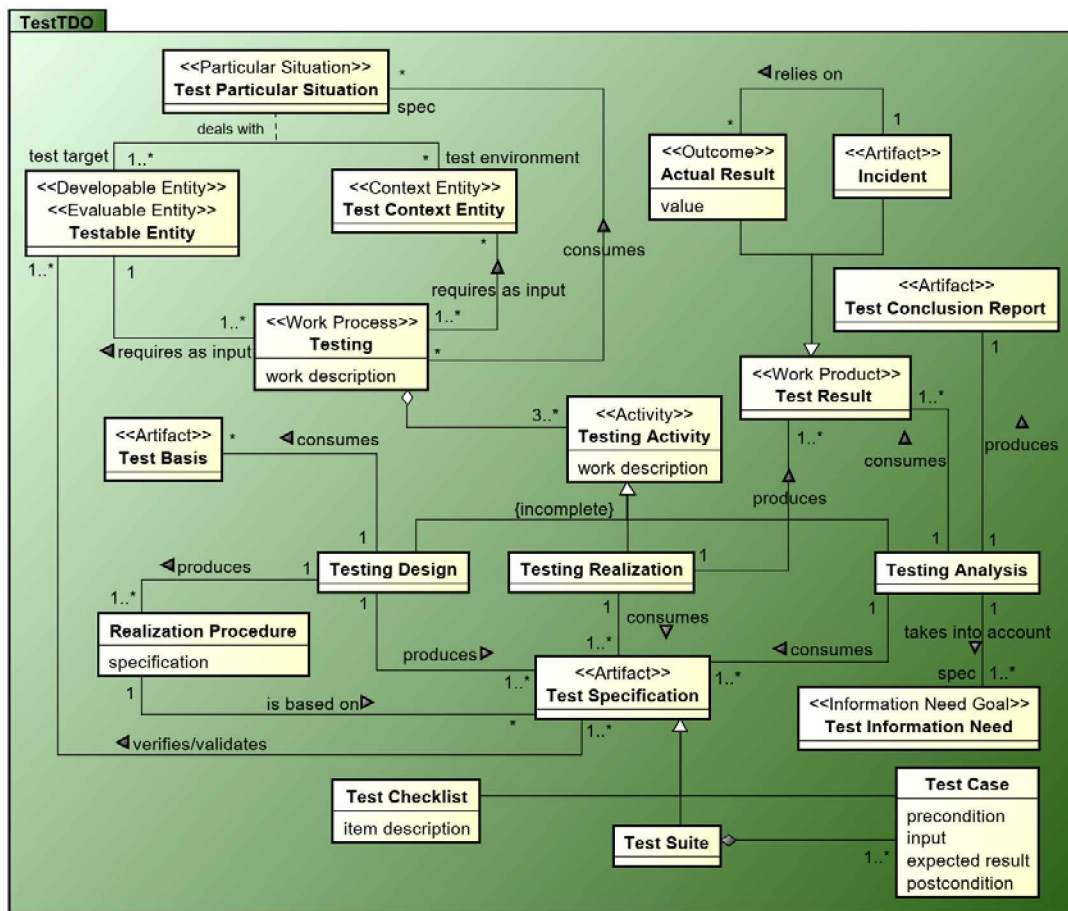


Fig. 4 Fragment of TestTDO terms, relationships and properties related to Work Product/Activity.

it, namely: Actual Result and Incident. The former is an Outcome that represents a numerical or categorical value –expected or unexpected. Instead, an Incident is an Artefact or document, which reports deviations (e.g., between the Test Case's *expected result* and the Actual Result), anomalies (e.g., an error or a failure) or other arisen issues during the Testing Realization. When the Incident occurs since there is a mismatch between the Test Case's *expected result* and the Actual Result, then this Incident relies on the corresponding Actual Result.

On the other hand, Testing Analysis is a Testing Activity that takes into account the specific Test Information Need to produce a Test Conclusion Report by consuming one or more Test Results and Test Specifications. The Test Conclusion Report is an Artefact that documents the analysis of all Test Results. For example, this Artefact could contain details about the degree to which the Test Goals were achieved by analyzing Test Information Need goals, the coverage level achieved by the executed Test Cases, among other analysis. Note that the reader can check related axioms (A1-2 and A10-17) in <http://arxiv.org/abs/2104.09232>.

#### 4.4. TestTDO concepts related to Activity/Method/Agent

To cover the activity- and method- related scope, TestTDO has terms such as Static/Dynamic Testing,

Testing Realization Method and Testing Design Method as depicted in Fig. 5. A Testing Realization is a Testing Activity aimed at enacting a Static or Dynamic Testing. The former has the objective of checking a Testable Entity against one or more Test Specifications without the execution of its software code, if any. Instead, Dynamic Testing aims at verifying/validating a Testable Entity against one or more Test Specifications with the execution of its software code.

As shown in Fig. 5, a Static/Dynamic Testing Method is assigned to one or more Static/Dynamic Testing activities. Examples quoted in the literature of Static Testing Methods are Walkthrough, Technical Review, Inspection, among others. These static and dynamic methods are kinds of Testing Realization Method. Note that a Testing Realization Method is a Testing Method for a task included in a Testing Realization activity, which includes a Realization Procedure.

Additionally, we define the Testing Method (or Testing Technique) as a specific and particular way to perform the specified steps for a task included in a Testing Activity. The specific and particular way of a Testing Method –i.e., how the specified steps in a testing task should be made- is represented by a *procedure* (e.g., *design procedure* or Realization Procedure) and *rules*.

At this point it is important to remark that TestTDO adopts an important conceptual pattern by ProcessCO which most of the 12-selected ontologies

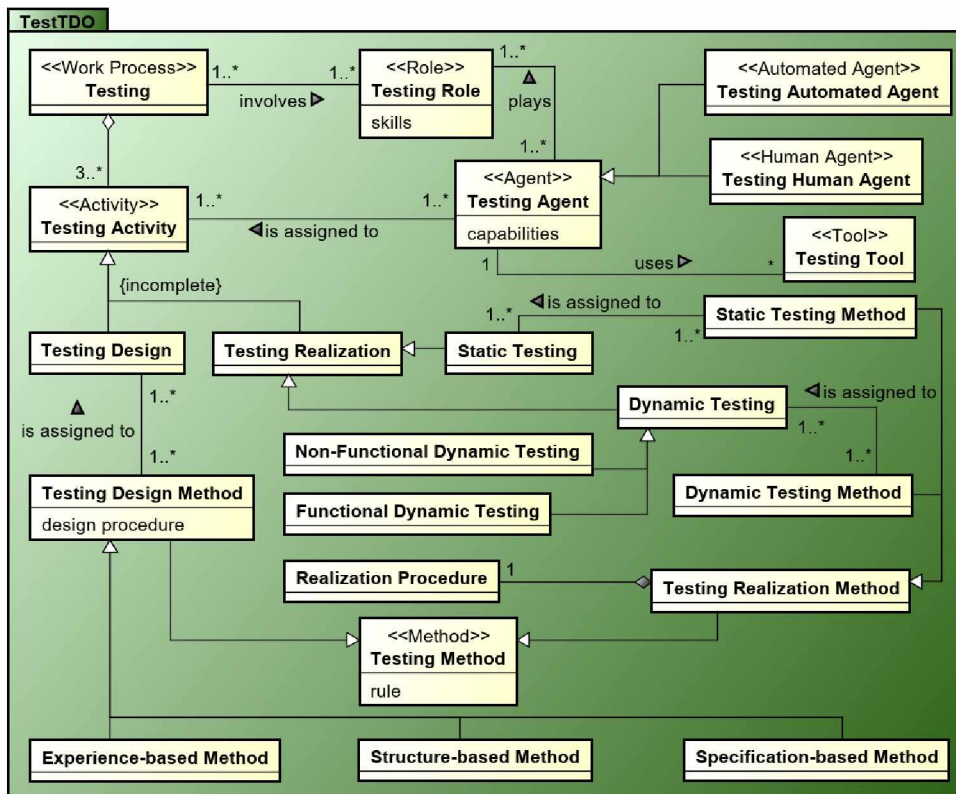


Fig. 5 Fragment of TestTDO terms, relationships and properties related to Activity/Method/Agent.

do not have except ROoST. This pattern attempts to make a clear separation of concerns between ‘the what’ (activities and tasks) and ‘the how’ (methods and procedures). Note that in TestTDO, each Testing Activity has assigned a Testing Method (except for the Testing Analysis which we did not include it in the fragment so as not to overload the model).

Like the Testing Realization Method, the Testing Design Method is another kind of Testing Method, but it is assigned to Testing Design activities. Also, this one has a *design procedure* that specifies how must be performed the Testing Design activity using the Test Basis, if any. There are three kinds of Testing Design Methods, namely: Specification-based Method (also known as black-box), Structure-based Method (also known as white-box), and Experience-based Method.

A Specification-based Method always uses a Test Basis when enacting the Testing Design activity to derive Test Specifications without referring to the internal structure of the Testable Entity. More specific types of Specification-based Methods are: Classification Tree Method, Scenario Testing, Random Testing, State Transition Testing, among others [11], which are not shown in Fig. 5 since TestTDO terms are at the top-domain level.

Also, a Structure-based Method is a Testing Design Method that uses the internal structure of the Testable Entity, and sometimes also uses a Test Basis, while enacting the Testing Design activity for deriving Test Specifications. Examples of it are: Statement Testing, Decision Testing, Modified Condition Decision Coverage Testing, among others [11].

Finally, an Experience-based Method is also a Testing Method but uses the Testing Human Agent's knowledge, expertise and intuition when enacting the Testing Design activity to derive Test Specifications. Some kinds of Experience-based Methods quoted in the literature are Error Guessing [11] and Exploratory Testing [2]. The reader can check related axioms (A3-4) in <http://arxiv.org/abs/2104.09232>.

## 5. Verifying and Validating TestTDO

V&V activities are similar, but they tackle different issues. Validation aims at demonstrating that the artefact or system fulfils its intended use, while verification aims at checking whether the artefact properly mirrors the specified requirements. In other words, validation ensures that “you built the right thing”, while verification ensures that “you built it right”.

In Fig. 6 we depict a hierarchy for V&V activities, which we have slightly adapted from Fig. A.1 of [1]. This figure shows three activities at the first level, namely: Testing, Formal Verification and V&V Analysis. In turn, Testing has two sub-activities: Static and Dynamic Testing. For the latter, we have added Functional and Non-Functional Dynamic Testing sub-activities. On the other side, V&V Analysis has sub-activities such as Demonstration, Assessment, Simulation, and Measurement & Evaluation. Note that we have performed the V&V activities highlighted with dashed lines in the figure to verify and validate TestTDO.

In the following sub-sections, we describe the V&V performed activities on TestTDO. In short, once we have obtained the conceptualization of TestTDO, we have executed the following activities: i) two static verifications (Static Testing in Fig. 6) described in sub-section 5.1, one on the CQs, and the other on the semantic consistency between the relationships of TestTDO and the ontologies of the FCD-OntoArch that enrich them; ii) an evaluation (Measurement & Evaluation) using a strategy named GOCAME [17], [19], which is illustrated in sub-section 5.2; and iii) a proof of concept (Assessment), shown in sub-section 5.3, that instantiates the TestTDO terms for an academic test project on a geometrical figure application to validate whether TestTDO was able to represent concrete world situations, plus a validation by 3 experts in the test domain. Besides, in sub-section 5.4 we describe aspects of the TestTDO implementation and the dynamic verification (Functional Dynamic Testing) of the CQs using test cases. Finally, in sub-section 5.5

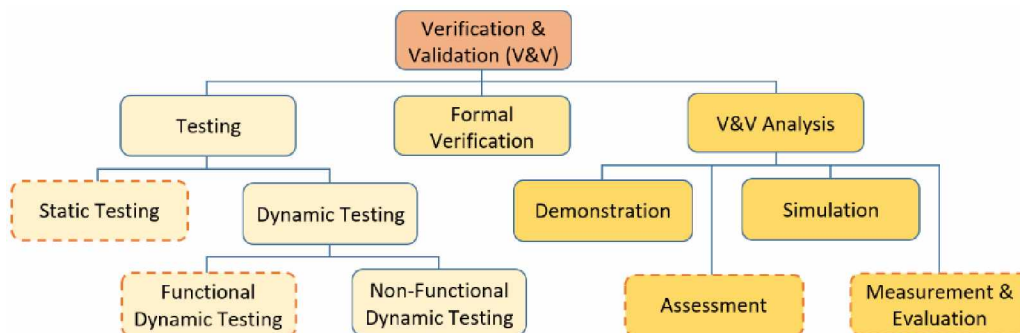


Fig. 6 Hierarchy of V&V activities, which we have adapted slightly from Fig. A.1 of [1].

we discuss some V&V approaches used in ontologies.

### 5.1. Static Testing on TestTDO

The first V&V activity that we carried out was the static verification of the TestTDO conceptualization against the CQs, which represent the scope-related requirements as presented in sub-section 4.1. The purpose was to verify that all CQs were addressed by some of the terms, properties, relationships and/or axioms. In this direction, we produced a verification matrix (see Table 1) using a Specification-based (black-box) Method. To design the matrix, we used as a Test Basis the CQs and the elements that any heavyweight ontology has such as terms, properties, relationships, and axioms. This matrix is a Test Checklist that contains one row per each CQ, in which we record what term, property, relationship or axiom correspond to the CQ.

to achieve a Test Goal, then exists a Test Project, associated with the Testing Strategy, which operationalizes the Test Goal.

On the other hand, we have used another Testing Design Method to statically verify the semantic consistency of the TestTDO relationships against the corresponding relationships from higher-level ontologies that belong to the FCD-OntoArch architecture. In this case, we have performed an integration testing in which we have the following Test Particular Situation: i) the Testable Entities are the dependencies between TestTDO and the SituationCO and ProcessCO ontologies of FCD-OntoArch; and ii) both SituationCO and ProcessCO are Test Context Entities in this situation. Note that a dependency between TestTDO and other core ontology represents a relationship of TestTDO which is inherited by other relationship of the high-level ontology.

Table 1 Excerpt from the TestTDO Verification Matrix. The entire verification matrix with all checked CQs can be accessed at <http://bit.ly/TestTDO-VerifMatrix>.

CQ	Terms, <b>relationships</b> and <b>properties</b>	Axioms
CQ3. What are the work products produced by a testing realization activity?	Testing Realization <b>is-a</b> Testing Activity Testing Realization <b>produces</b> Test Result Actual Result <b>is-a</b> Test Result Incident <b>is-a</b> Test Result	A1, A7, A11
CQ8. What is the minimum set of testing activities included in a testing process?	Testing Activity <b>is-part-of</b> Testing process Testing Design <b>is-a</b> Testing Activity Testing Realization <b>is-a</b> Testing Activity Testing Analysis <b>is-a</b> Testing Activity	A2
CQ25. For a test project that operationalizes a test goal, has the test project an associated testing strategy that helps to achieve the test goal purpose?	Test Project <b>operationalizes</b> Test Goal Test Project <b>associates</b> Testing Strategy Testing Strategy <b>helps to achieve</b> Test Goal Test Goal has the property named <b>purpose</b>	A8

For example, the CQ3 states “What are the work products produced by a testing realization activity?”. If we see Fig. 4, we can note that a Testing Realization is a Testing Activity that produces Test Results (with the semantics of Work Product). Also, Actual Result and Incident are specific kinds of Test Result, therefore a Testing Realization produces Actual Results or Incidents. Furthermore, if we analyze the TestTDO axioms, we can observe that A1, A7 and A11 are related to CQ3. For instance, A1 establish that any Test Result produced by a Testing Realization activity is an Actual Result or an Incident, but not both at the same time.

Additionally, the CQ25 specifies “For a test project that operationalizes a test goal, has the test project an associated testing strategy that helps to achieve the test goal purpose?”. To cover the CQ25 scope, TestTDO has a conceptual pattern that is inherited from the ontologies at the core level. This pattern relates the Testing Strategy, Test Project and Test Goal terms. Besides, TestTDO has the A8 axiom which states the following: if a Testing Strategy helps

For the Testing Design activity, we have used a customized white-box Testing Design Method in order to produce the Test Checklist shown in Table 2. For designing the Checklist, this white-box method uses as a Test Basis the internal structure of the Testable Entities (recall that the Testable Entities are the dependencies between TestTDO and SituationCO and ProcessCO). We have identified these dependencies by analyzing the conceptualizations of TestTDO, SituationCO and ProcessCO. In the first three columns of the table shown in <http://bit.ly/TestTDO-VerCheckRel> we have documented all identified relationships (i.e., the dependencies) and the related concepts with their stereotypes. Note that these relationships belong to the TestTDO v1.0 initially documented in [9] and in this paper we present TestTDO in its version 1.2.

After identifying all dependencies, in the Static Testing (Realization) activity we have inspected each of them to verify its semantic matching with some relationship of higher-level ontologies. Table 2 shows an example in which an Incident was generated and

Table 2 Excerpt from the Test Checklist for inspecting the semantic consistency of TestTDO v1.0 relationships against the relationships that enrich them from higher-level ontologies, e.g. the SituationCO ontology.

TestTDO v1.0 Relationship	TestTDO Term 1 <<Stereotype>>	TestTDO Term 2 <<Stereotype>>	Inspect	Actual Result (pass/fail)	Incident description (if it fails)
is in a particular situation with	Testeable Entity <<Evaluable/Developable Entity from SituationCO>>	Test Context Entity <<Context Entity from SituationCO>>	Is there a semantic match between the relationship named "is in a particular situation with" of TestTDO with some relationship of the Evaluable/Developable Entity and Context Entity concepts of SituationCO?	fail	The TestTDO relationship named ...

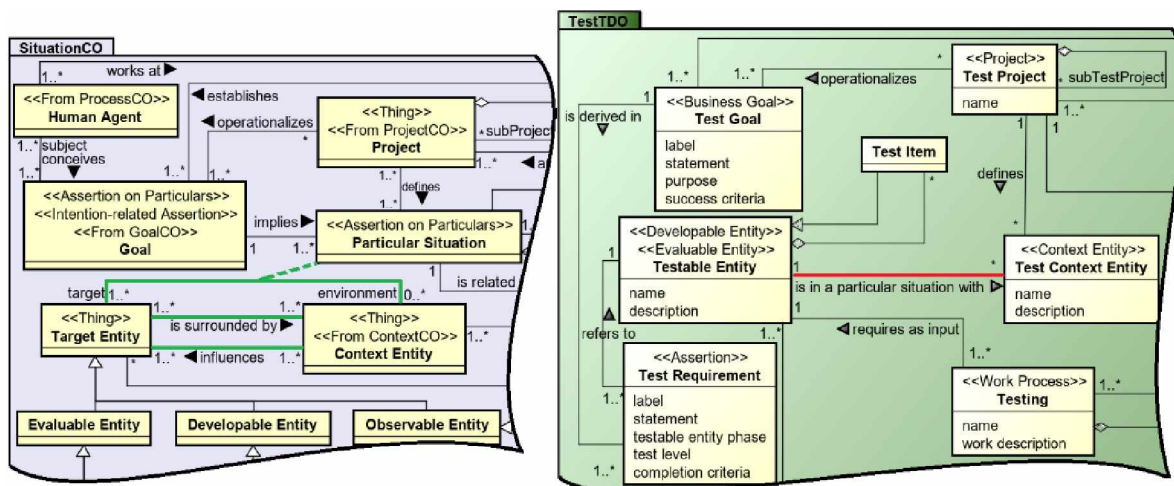


Fig. 7 Semantic mismatch of the TestTDO v1.0 red-highlighted relation against SituationCO one.

therefore the Actual Result was labelled with the 'fail' value. In Fig. 7 we have highlighted in red the TestTDO v1.0 relation named is in a particular situation with, that has the semantic mismatch with the SituationCO green-highlighted one named is surrounded by. Additionally, we detected that the influences relation was missing in TestTDO v1.0.

Analyzing the definitions of both relationships, we observed a semantic nonconformity, which in fact led us to be aware of a deeper issue. Let us look at the conceptual pattern of the Particular Situation term in SituationCO. Particular Situation represents an association between Target Entity and Context Entity in the role of the environment. In turn, the Target Entity is surrounded by Context Entities and the latter influences the former. However, this conceptual (ontological) pattern is not mirrored in TestTDO v1.0.

In summary, we have detected 7 additional Incidents in the TestTDO v1.0 relationships by using this Testing Design Method. It is important to remark that this verification activity allowed us to partially verify AR#7, which states: "...some terms should also be related to SituationCO and ProcessCO sub-ontologies at the core level...". Besides, we have

conducted this Static Testing activity to verify enriched concepts (i.e., with stereotypes). This activity involved verifying each TestTDO concept stereotyped with terms of higher-level ontologies of FCD-OntoArch. We have verified that the semantics between the enriching term and the enriched term correspond appropriately. However, we have conducted this activity informally and therefore we do not document it.

### 5.2. TestTDO Measurement & Evaluation

We show in Table 3 the evaluation results for TestTDO, including the 2 best-ranked ontologies during the cited SLR we performed. Recall that we use the metaphor of the three-coloured semaphore to identify the satisfaction acceptability level achieved, as we mentioned in Section 2.

To evaluate TestTDO we used as NFRs tree the same one that was used in the SLR for the 12-selected software testing ontologies [5], whose root is the Ontological Quality characteristic. Therefore, we included the quality characteristics and attributes shown in Table 4. Besides, we used the same metrics

Table 3 Evaluation results of TestTDO and its comparison with the 2 best-ranked ontologies in the conducting SLR. The green circle indicates “satisfactory” acceptability level (●); orange rhombus “marginal” (◆) and red square “unsatisfactory” (■). Indicators' values are in [%].

Characteristics / Attributes	ROoST [14]	Asman and Srikanth [15]	TestTDO
<b>1. Ontological Quality</b>	79.54 ◆	66.71 ◆	98.11 ●
<b>1.1 Ontological Structural Quality</b>	79.08 ◆	61.92 ◆	96.22 ●
<i>1.1.1 Defined Terms Availability</i>	82.20 ◆	100 ●	100 ●
<i>1.1.2 Defined Properties Availability</i>	0 ■	0 ■	100 ●
<i>1.1.3 Specified Axioms Availability</i>	100 ●	0 ■	100 ●
<b>1.1.4 Balanced Relationships Availability</b>	87.32 ●	73.07 ◆	87.42 ●
<i>1.1.4.1 Balanced Non-Taxonomic Relationships Availability</i>	95.65 ●	66.34 ◆	84.27 ◆
<i>1.1.4.2 Defined Non-Taxonomic Relationships Availability</i>	54 ■	100 ●	100 ●
<b>1.2 Domain-specific Terminological Coverage Quality</b>	50 ■	100 ●	100 ●
<i>1.2.1 Static Testing Terms Availability</i>	0 ■	100 ●	100 ●
<i>1.2.2 Dynamic Testing Terms Availability</i>	100 ●	100 ●	100 ●
<i>1.2.3 Functional Testing Terms Availability</i>	100 ●	100 ●	100 ●
<i>1.2.4 Non-Functional Testing Terms Availability</i>	0 ■	100 ●	100 ●
<b>1.3 Compliance to other Vocabularies</b>	100 ●	52.50 ■	100 ●
<i>1.3.1 Terminological use of Int'l Standard Glossaries</i>	100 ●	85 ●	100 ●
<i>1.3.2 Terminological Compliance to other Domain/Core Ontologies</i>	100 ●	100 ●	100 ●
<i>1.3.3 Terminological Compliance to Foundational Ontologies</i>	100 ●	0 ■	100 ●

and indicators to perform the measurement and evaluation activities as well as the same aggregation scoring model using the GOCAME strategy's process and methods [17], [19]. Note that we have used the same NFRs tree as well as the same metrics and indicators since our objective was, in addition to evaluating TestTDO, to compare the Ontological Quality of TestTDO with the 12-selected ontologies.

Regarding the used metrics and indicators, we have designed some more elaborated and others simpler. For example, the procedure of the metric used to quantify the “Balanced Non-Taxonomic Relationships Availability (1.1.4.1)” attribute contains a formula that is used to obtain the percentage of taxonomic relationships (i.e., “kind-of/is-a” and “whole-part/part-of” relationships) with regard to the total relationships (see Fig. 4 of [5]). In addition, we have designed an elementary indicator (shown in Fig. 6 of [5]) to interpret and evaluate this attribute. These metric and indicator were the more elaborated we used.

On the other hand, to evaluate other attributes as “Terminological Compliance to other Domain/Core Ontologies (1.3.2)” or “Terminological Compliance to Foundational Ontologies (1.3.3)”, we have used simpler metrics and indicators. For example, to evaluate 1.3.2, we have read the documentation associated with an ontology in order to find some evidence if the authors had considered using some other core or domain ontologies to build their ontology, as we did with TestTDO which is integrated with the core ontologies of the FCD-OntoArch. So, the metric we use counts the number of domain/core ontologies considered and the indicator simply interpretes this value as follows: if no ontology is

considered then the indicator value is 0 (■); if 1 is considered then the value is 85 (●); and if 2 or more are considered, the value is 100 (●). Note that we don't perform any further analysis regarding this metric.

Likewise, the metric and indicator used to the 1.3.3 attribute are also simple. The metric only captures the number of foundational ontologies on which the ontology to be measured is based, without going beyond the analysis, i.e., without analyzing the quality of the foundational ontology used. The indicator interprets the measure as follows: if no foundational ontology is considered then the indicator value is 0 (■); and if 1 or more are considered, the value is 100 (●). More details about the metrics and indicators used to measure and evaluate TestTDO and the other 12-selected ontologies can be checked in [5].

### 5.3. TestTDO Assessment

In the Assessment activity (Fig. 6), in order to validate if TestTDO was able to represent concrete situations of the world, we instantiated its terms, properties and relationships using a geometrical figure application for an academic project. It is based on the running testing example introduced by Myers et al. [20]. In this example, we have a program that receives 3 integers, where each value represents the side of a triangle and taking into account these input values the program returns as result the triangle type, i.e., isosceles, scalene or equilateral.

Also, we have instantiated terms related to Testable Entity, Test Project, Testing Strategy, Test Goal, among others, in addition to 2 kinds of Testing

Table 4 NFRs tree (1<sup>st</sup> column) to evaluate the Ontological Quality. In the 2<sup>nd</sup> column are all definitions of characteristics and attributes.

<b>Characteristics / Attributes</b>	<b>Definition:</b> Degree to which...
<b>1. Ontological Quality</b> (root)	...an ontology is well structured, has good terminological coverage and adheres to other vocabularies.
<b>1.1 Ontological Structural Quality</b>	...an ontology is well structured, i.e., has defined terms availability, defined properties availability, specified axioms availability and it is properly balanced with regard to types of relationships.
<i>1.1.1 Defined Terms Availability</i>	...an ontology has defined terms.
<i>1.1.2 Defined Properties Availability</i>	...an ontology has defined properties.
<i>1.1.3 Specified Axioms Availability</i>	...an ontology has specified axioms.
<b>1.1.4 Balanced Relationships Availability</b>	...an ontology has a balance between the amount of non-taxonomic and taxonomic relationships in addition to the former are defined.
<i>1.1.4.1 Balanced Non-Taxonomic Relationships Availability</i>	...an ontology has a balance between the amount of non-taxonomic and taxonomic relationships.
<i>1.1.4.2 Defined Non-Taxonomic Relationships Availability</i>	...an ontology has defined non-taxonomic relationships.
<b>1.2 Domain-specific Terminological Coverage Quality</b>	...an ontology has good terminological coverage of the domain.
<i>1.2.1 Static Testing Terms Availability</i>	...a software testing ontology has terms related to Static Testing.
<i>1.2.2 Dynamic Testing Terms Availability</i>	...a software testing ontology has terms related to Dynamic Testing.
<i>1.2.3 Functional Testing Terms Availability</i>	...a software testing ontology has terms related to Functional Testing.
<i>1.2.4 Non-Functional Testing Terms Availability</i>	...a software testing ontology has terms related to Non-Functional Testing.
<b>1.3 Compliance to other Vocabularies</b>	...an ontology adheres its terminology with other vocabularies.
<i>1.3.1 Terminological use of Int'l Standard Glossaries</i>	...an ontology uses or refers to international standard glossaries.
<i>1.3.2 Terminological Compliance to other Domain/Core Ontologies</i>	...an ontology adheres its terminology to other domain or core ontologies.
<i>1.3.3 Terminological Compliance to Foundational Ontologies</i>	...an ontology adheres its terminology to a foundational ontology.

Design Methods, namely: statement testing method, which is a Structure-based Method, and the Specification-based Method named equivalence partitioning method. All details are available at [http://bit.ly/TestTDO\\_Val](http://bit.ly/TestTDO_Val). Note that this validation activity was carried out in TestTDO v1.0. As a result of this validation, we have concluded that TestTDO can represent this rather simple situation. In this direction, we are starting to use TestTDO in an industrial project dealing with a more complex real-world situation. This should contribute to more extensive validation.

Additionally, we have conducted informal interviews regarding TestTDO with 3 external testing domain experts. They examined TestTDO and gave us their feedback on it. In short, their comments gave us evidence of the potential utility of TestTDO and

they considered, from their expert point of view, that the terminology of the ontology is suitable.

#### 5.4. Functional Dynamic Testing on TestTDO

To be able to dynamically test our software testing ontology, we fully implemented it in OWL using Protégé. Also, we used some guidelines to transform from UML to OWL [21], [22] that were very useful to produce the OWL version of TestTDO. The reader can access it at [http://bit.ly/TestTDO\\_OWL](http://bit.ly/TestTDO_OWL). In summary, we have used a Specification-based (black-box) Method to design Test Cases. To produce them, the CQs and TestTDO conceptualization were used as Test Basis. Additionally, as Test Cases' *input* data, we use instances of the geometrical figure application



Table 5 Examples of Test Cases (TC) used in a Functional Dynamic Testing Activity for TestTDO.

TC#	CQ	Input	Expected Result
TC#1	CQ03.01		Three instances of Actual Result (AR) with the following data (name; value): {"AR#1; "Equilateral"}; {"AR#2; "Scalene"}; {"AR#3; "Isosceles"}
TC#2	CQ03.02	Data from <a href="http://bit.ly/TestTDO_Val">http://bit.ly/TestTDO_Val</a>	One instance of Incident (name; description): {"I#1"; "The Actual Result doesn't match with the expected result, which is "Invalid triangle"}}
TC#3	CQ25		One instance of Test Project (name), one of Testing Strategy (name), one of Test Goal (label; purpose), which are all of them related: {"TP#1"}; {"dynamic testing strategy"}; {"TG#1", "verify"}}

documented in [http://bit.ly/TestTDO\\_Val](http://bit.ly/TestTDO_Val). All designed Test Cases can be accessed at [http://bit.ly/TestTDO\\_TestCases](http://bit.ly/TestTDO_TestCases) although we illustrate some of them in Table 5.

Note that, since CQ3 (see its specification in Table 1) is a little generic to be verified, it was split into 2 sub-CQs, namely: CQ03.01. What are the Actual Results produced by a Testing Realization activity?; and CQ03.02. What are the Incidents produced by a Testing Realization activity? In addition, it was not necessary to do this with the CQ25 (also shown its specification in Table 1).

Besides, we considered the following preconditions:

TC#1's precondition: That there must be implemented individuals of Actual Results and Testing Realization activities that produced them.

TC#2's precondition: That there must be implemented individuals of Incidents and Testing Realization activities that produced them.

TC#3's precondition: That there must be implemented individuals of Test Project, Test Goal and Testing Strategy, and they are related to each other.

Once Test Cases were obtained, we also produced the Realization Procedure. This procedure contains, for each Test Case, a SPARQL query (as seen in Fig. 8) for coding the designed Test Cases. It is important to remark that we executed the queries in the Protégé environment and all Test Cases passed.

### 5.5. Discussion about V&V approaches for ontologies

In summary, looking at the V&V approaches used in each of the 12-selected ontologies in the conducted SLR, only a couple of them are explicitly documented such as in [14], [15], and [23].

We have performed a black-box Testing Design Method for Static Testing (sub-section 5.1) using the TestTDO terminology. A similar verification approach was used by authors (Souza *et al.*) in [14], but they called it "Assessment by human approach" since they consider that testing is just for dynamic V&V.

Considering V&V approaches for Functional Dynamic Testing, we have performed a black-box Testing Design Method for designing Test Cases

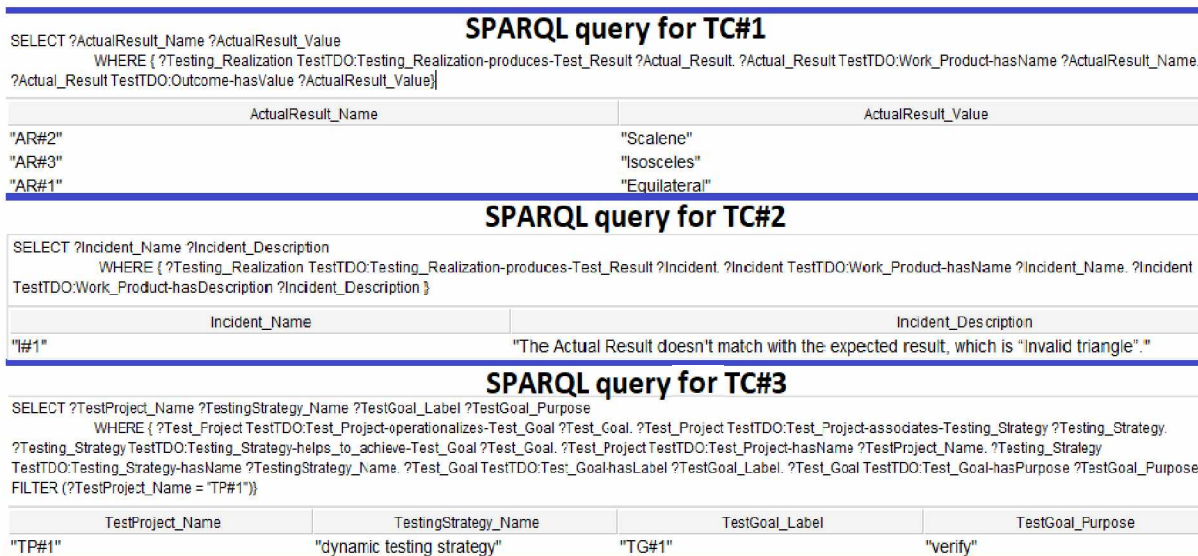


Fig. 8 Implemented SPARQL queries for the designed Test Cases of Table 5.

(sub-section 5.4). We have produced one Test Case for each CQ. Furthermore, we have commented on the Realization Procedure used by the Testing Realization activity. Souza *et al.* used this verification approach as well. They called it “Competency question-driven approach for ontology testing”. Note that previously, [24] introduced the dynamic (unit) testing approach for ontologies too.

To support the V&V Analysis activity (Fig. 6), we have developed an Ontological Quality tree (Table 4) to quantitatively evaluate the structural quality, terminological coverage quality, and compliance to other vocabularies including some quality features described by [6]. The evaluation results, using metrics and indicators, were also yielded for TestTDO following the quoted GOCAME evaluation strategy. Note that Souza *et al.* used just a descriptive and qualitative evaluation approach for ROoST.

Regarding evaluation strategies, the authors in [25] analyze different evaluation approaches for ontologies such as the [26] multi-criteria evaluation approach called Ontometric. However, [27] considered Ontometric has limited usability due to its complexity. Then, they propose an evaluation strategy. In our humble opinion, the metrics and indicators used in both strategies are weakly specified. This makes the results less justifiable and reproducible.

## 6. Concluding Remarks and Future Work

As indicated in the Introduction Section, after analyzing both the results of the conducted SLR of primary studies on software testing ontologies and the state-of-the-art of test-related standards, we decided to develop a new top-domain software testing ontology (i.e., TestTDO) that fits our aim and scope. We have confirmed that there was heterogeneity, ambiguity, and incompleteness for concepts dealing with test goals and requirements as well as with testing work products, activities and methods in the 12-selected ontologies. Furthermore, there was no software testing ontology directly linked with NFRs and FRs ontological concepts.

Following the DSR process, TestTDO, its resulting artefact is a software testing ontology placed at the top-domain level in the context of the four-layered architecture called FCD-OntoArch. It was purposely designed at the top-domain level so that it can be extended by other lower-level software testing domain ontologies. Therefore, as future work, we plan to develop other more specific testing domain ontologies, whether for dynamic and/or static testing, e.g., for performance testing, inspections, reviews, among others. To this end, TestTDO can enrich lower-level software testing ontologies, providing the foundation to support their construction and the

development of new strategies.

To extensively verify and validate TestTDO, we have performed several V&V activities. On the one hand, to verify TestTDO we have carried out two kinds of static verification activities, one to verify the TestTDO scope (i.e., the CQs) and the other to inspect the semantic consistency between the relationships of TestTDO and the ontologies of the FCD-OntoArch that enrich them. We have also dynamically verified the TestTDO scope by running a set of test cases. On the other hand, we have validated TestTDO by instantiating its terms in an academic test project related to a geometrical figure application in addition to its validation by 3 experts in the test domain. Finally, we have performed measurement and evaluation activities following the GOCAME [17], [19] strategy to evaluate the Ontological Quality of TestTDO.

As a result, TestTDO has 44 terms in total, of which 32 are enriched by ProcessCO using stereotypes (see Fig. 2). This means that approximately 70% of TestTDO's terms are enriched by this core ontology. The reader may wonder why this amount of process-related terms is so significant. This is mainly due to TestTDO's aim is to terminologically nourish specifications of methods and processes of a family of testing strategies to be developed. This is also ongoing research. A well-documented process specification should show what activities need to be carried out, what roles are involved, what work products are consumed/produced by activities and what resources are used for the different tasks [17], [28]. Likewise, a well-documented method should specify the procedure to be followed and the associated rules [17]. Therefore, process/method related terms are very important to achieve our goal. However, they are not the only important ones, but other terms related to situations, projects, goals, entities are necessary, since they are cross-cutting concerns for many domains.

As a future work, we will perform a validation between the semantics provided by the TestTDO concepts and the concepts provided by the UML Testing Profile (UTP) [29]. In other words, we will analyze the degree to which the semantics of TestTDO matches or adheres with the semantics provided by UTP.

### Competing interests

The authors have declared that no competing interests exist.

### Authors' contribution

All authors have participated in the construction of TestTDO conceptualization; GT has implemented TestTDO and conducted the dynamic verification; LO and

PB have performed the static verification of TestTDO; GT and DP have evaluated TestTDO by using metrics and indicators and they analyzed the results; GT has written the manuscript; LO, DP and PB have revised the manuscript. All authors read and approved the final manuscript.

### Acknowledgements

This work and line of research had the partial support of the Science and Technology Agency of Argentina, in the PICT 2014-1224 project at UNLPam. Also, it is supported partially by the Engineering School at UNLPam, in the project named “Family of Strategies for Functional and Non-Functional Software Testing considering Different Test Goal Purposes”.

### References

- [1] ISO, “ISO/IEC/IEEE 29119-1, Software and systems engineering - Software Testing – Part 1: Concepts and definitions.” 2013.
- [2] ISTQB, “International Software Testing Qualifications Board, Standard Glossary of Terms used in Software Testing, Version 3.2.” 2019, [Online]. Available: <https://www.istqb.org/>.
- [3] A. R. Hevner, S. T. March, J. Park, and S. Ram, “Design science in information systems research,” *MIS Quarterly: Management Information Systems*, vol. 28, no. 1, pp. 75–105, 2004, [Online]. Available: <https://doi.org/10.2307/25148625>.
- [4] G. Tebes, B. Rivera, P. Becker, M. F. Papa, D. Peppino, and L. Olsina, “Specifying the design science research process: an applied case of building a software testing ontology,” in *XXIII CibSE’ 20*, 2020, pp. 378–391.
- [5] G. Tebes, D. Peppino, P. Becker, G. Maturro, M. Solari, and L. Olsina, “Analyzing and documenting the systematic review results of software testing ontologies,” *Information and Software Technology*, vol. 123, 2020, doi: 10.1016/j.infsof.2020.106298.
- [6] M. D’Aquino and A. Gangemi, “Is There Beauty in Ontologies?,” *Applied Ontology*, vol. 6, no. 3, pp. 165–175, 2011.
- [7] M. Fernandez, A. Gómez-Pérez, and N. Juristo, “Methontology: from ontological art towards ontological engineering,” in *Proceedings of the AAI197 Spring Symposium Series on Ontological Engineering*, 1997, pp. 33–40.
- [8] L. Olsina, “Analyzing the Usefulness of ThingFO as a Foundational Ontology for Sciences,” in *Proceedings of ASSE’20, Argentine Symposium on Software Engineering*, 49 JAIIO, 2020, pp. 172–191.
- [9] G. Tebes, L. Olsina, D. Peppino, and P. Becker, “TestTDO: A Top-Domain Software Testing Ontology,” in *XXIII CibSE’ 20*, 2020, pp. 364–377.
- [10] ISO, “ISO/IEC/IEEE 29119-2: Software and systems engineering - Software Testing – Part 2: Test processes.” 2013.
- [11] ISO, “ISO/IEC/IEEE 29119-4: Software and systems engineering - Software Testing – Part 4: Test techniques.” 2015.
- [12] A. Freitas and R. Vieira, “An ontology for guiding performance testing,” in *Proceedings - 2014 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Workshops, WI-IAT 2014*, 2014, vol. 1, pp. 25–36, doi: 10.1109/WI-IAT.2014.62.
- [13] P. G. Sapna and H. Mohanty, “An Ontology Based Approach for Test Scenario Management,” in *Information Intelligence, Systems, Technology and Management*, 2011, pp. 91–100, doi: 10.1007/978-3-642-19423-8\_10.
- [14] É. F. De Souza, R. De Almeida Falbo, and N. L. Vijaykumar, “ROoST: Reference ontology on software testing,” *Applied Ontology*, vol. 12, no. 1, pp. 59–90, 2017.
- [15] A. Asman and R. M. Srikanth, *A Top Domain Ontology For Software Testing*. Master Thesis, Jönköping University, 2015.
- [16] G. Guizzardi, *Ontological foundations for structural conceptual models*. Ph.D. Thesis, Netherlands, Universal Press, 2005.
- [17] P. Becker, F. Papa, and L. Olsina, “Process Ontology Specification for Enhancing the Process Compliance of a Measurement and Evaluation Strategy,” *CLEI electronic journal*, vol. 18, no. 1, pp. 1–26, 2015, doi: 10.19153/cleiej.18.1.2.
- [18] D. Peppino, G. Tebes, P. Becker, and L. Olsina, “Designing Context-Aware Test Cases for Particular Situations,” in *Congreso Nacional de Ingeniería Informática/Sistemas de Información (CoNaIISI)*, 2020, pp. 49–62.
- [19] L. Olsina and P. Becker, “Family of strategies for different evaluation purposes,” in *XX CibSE’ 17*, 2017, pp. 221–234.
- [20] G. J. Myers, T. Badgett, and C. Sandler, *The Art of Software Testing*, 3rd ed. Wiley, 2012.
- [21] M. H. L. Vo and Q. Hoang, “Transformation of UML class diagram into OWL Ontology,” *Journal of Information and Telecommunication*, vol. 4, no. 1, pp. 1–16, Jan. 2019, doi: 10.1080/24751839.2019.1686681.
- [22] R. E. Y. Haasjes, “Metamodel transformations between UML and OWL, Master Thesis,” University of Twente, 2019.
- [23] S. Vasanthapriyan, J. Tian, D. Zhao, S. Xiong, and J. Xiang, “An Ontology-Based Knowledge Sharing Portal for Software Testing,” in *Proceedings - 2017 IEEE International Conference on Software Quality, Reliability and Security Companion, QRS-C 2017*, 2017, pp. 472–479, doi: 10.1109/QRS-C.2017.82.
- [24] D. Vrandečić and A. Gängemi, “Unit tests for ontologies,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Oct. 2006, vol. 4278 LNCS, pp. 1012–1020, doi: 10.1007/11915072\_2.
- [25] J. Brank, M. Grobelnik, and D. Mladenic, “A survey of ontology evaluation techniques,” in *In Proceedings of the Conference on Data Mining and Data Warehouses (SiKDD 2005)*, 2005, pp. 166–169.
- [26] A. Lozano-Tello and A. Gómez-Pérez, “ONTOMETRIC: A Method to Choose the Appropriate Ontology,” *Journal of Database Management*, vol. 15, no. 2, pp. 1–18, 2004, doi: 10.4018/jdm.2004040101.

- [27] A. Duque-Ramos, J. T. Fernández-Breis, R. Stevens, and N. Aussenac-Gilles, "OQuaRE: A square-based approach for evaluating the quality of ontologies," *Journal of Research and Practice in Information Technology*, vol. 43, no. 2, pp. 159–176, 2011.
- [28] B. Curtis, M. Kellner, and J. Over, "Process Modeling," *Communication of ACM*, vol. 35, no. 9, pp. 75–90, 1992, [Online]. Available: <https://doi.org/10.1145/130994.130998>.
- [29] OMG, "UML Testing Profile 2 (UTP 2), Version 2.1." 2020, [Online]. Available: <https://www.omg.org/spec/UTP2>.

**Citation:** G. Tebes, L. Olsina, D. Peppino and P. Becker. *Specifying and Analyzing a Software Testing Ontology at the Top-Domain Ontological Level*. *Journal of Computer Science & Technology*, vol. 21, no. 2, pp. 126-145, 2021.

**DOI:** 10.24215/16666038.21.e12

**Received:** January 7, 2021 **Accepted:** May 31, 2021.

**Copyright:** This article is distributed under the terms of the Creative Commons License CC-BY-NC.