

TestTDO: A Top-Domain Software Testing Ontology

Guido Tebes, Luis Olsina, Denis Peppino and Pablo Becker

GIDIS_Web, Facultad de Ingeniería, UNLPam, General Pico, LP, Argentina
guido.tebes92@gmail.com; olsinal@ing.unlpam.edu.ar;
denispeppino92@gmail.com; beckerp@ing.unlpam.edu.ar

Abstract. One of the Software Engineering (SE) areas that supports quality assurance is testing. Given that specific processes, artifacts, 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 in an explicit and unambiguous way. 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 (SLR) of primary studies on software testing ontologies and the state-of-the-art of test-related standards, we decided to develop a top-domain ontology that fits our goals. TestTDO is built in the framework of a four-layered ontological architecture, which considers foundational, core, domain and instance ontologies. In this paper, we discuss aspects of the development, evaluation, verification and validation of the TestTDO conceptualization.

Keywords: Software Testing, Top-Domain Ontology, Ontological Architecture.

1 Introduction

Companies commonly establish and reach business goals for different types of purposes. Business goals are the main goals that an organization tries to achieve. In the statement of it always lies a purpose or intentionality. The goal's purpose is the rationale to achieve it. Purposes can be classified into four categories such as evaluation, testing, development and maintenance [6]. Examples of evaluation goal purposes may include to understand, monitor, control, improve, etc. while examples of test goal purposes may entail to find defects, review, verify, validate, among others.

To achieve the purposes of business goals, strategies may be used. A strategy is a core resource of an organization that defines a specific course of action to follow. It specifies what to do and how to do it. Consequently, strategies should integrate 1) process specifications, 2) method specifications, and 3) a robust domain conceptual base – such as an ontology. This principle of integratedness promotes, therefore, knowing what activities are involved, and how to carry them out by means of methods in the framework of a common domain vocabulary. In [17], to achieve evaluation purposes, a family of strategies integrating the three-abovementioned capabilities is discussed.

Bearing in mind that already there are integrated strategies that provide support for achieving evaluation purposes, the reader can surmise that strategies that provide sup-

port for achieving test purposes are feasible to be developed as well. Given that a strategy should integrate a domain terminology, then any well-specified software testing strategy should also have this capability for the software testing domain.

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.

A benefit of having a suitable testing ontology is to minimize the heterogeneity and ambiguity problems that we currently observe in the different concepts dealing with testing methods, processes and artifacts. On the other side, one desirable feature is if the existing testing ontologies cover concepts related to static and dynamic testing as well as their linking with Non-Functional Requirements (NFRs) and Functional Requirements (FRs) sub-ontologies. Having this feature may turn out useful for method and process specifications to our aim, i.e., for developing a family of testing strategies.

In order to adopt or adapt an existing testing ontology, or to develop a new one, we have followed the Design Science Research (DSR) [12, 22] approach. It is a rigorous research methodology, which proposes the construction of artifacts for providing useful and effective solutions to a relevant problem in a given domain. Artifacts must be innovative and useful solutions to a non-trivial problem. The artifact development implies a cycle of design-construction-evaluation activities, which should iterate as many times as necessary before the artifact (already verified and validated) is ready for its use.

To start the DSR process, the Identify the Problem/Solution (A1) activity should firstly be performed [22]. In order to find out existing solutions (i.e., conceptualized software testing ontologies) to our problem, we conducted a Systematic Literature Review (SLR) [20], from the end of May 2018 until the beginning of March 2019. 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 [9, 21].

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

As a result, we have confirmed that there exists heterogeneity, ambiguity, and incompleteness for concepts dealing with testing activities, artifacts and methods. Furthermore, they are not directly linked with NFRs and FRs concepts. So the current testing ontologies are not suitable to our aim, i.e., for terminologically nourishing specifications of methods and processes for a family of testing strategies to be developed. For this reason, after analyzing the relevancy of the problem/solution, we have developed TestTDO, a top-domain software testing ontology by performing two activities of DSR, namely: Design and Develop the Solution (A2) and Execute Verification and Validation (A3). This endeavor took us seven months, which ended by the end of Sept., 2019.

In summary, this work documents aspects of the TestTDO conceptualization and its ontological quality evaluation. In addition, by using the stated set of competency questions and a proof of concept, we illustrate aspects of its verification and validation.

It is worth mentioning that TestTDO is placed into an ontological conceptual architecture called FCD-OntoArch (*Foundational, Core, and Domain Ontological Architecture for Sciences* [6]). 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 upper levels. For example, TestTDO at domain level is enriched by concepts of the ProcessCO [5] ontology placed at the core level. In turn the latter is enriched by concepts of ThingFO at the foundational level, as we see later on.

The rest of the paper is organized as follows. Section 2 provides a summary of related work on conceptualized testing ontologies. Section 3 describes where TestTDO is placed into FCD-OntoArch. Section 4 analyzes the main concepts, properties and relationships included in TestTDO. Section 5 describes how TestTDO was verified, validated and evaluated. Finally, Section 6 summarizes conclusions and future work.

2 Why is another Software Testing Ontology needed?

From the SLR performed in [20], we have selected 12 ontologies. Next, we summarize the aim of each ontology and then show aspects related to their ontological quality.

In [8], Campos *et al.* use a domain ontology named RTE-Ontology (*Regression Tests Execution*) and a provenance ontology model (PROV-O) to capture and provide regression tests data to support the continuous improvement of software testing processes. In [24, 25], Vasanthapriyan *et al.* document a software testing ontology designed to represent the necessary testing knowledge within the software testers' context. Although [24] and [25] share common terms, [25] has more terms related to static testing, while [24] adds the functional and non-functional testing terms, among others. Souza *et al.* document ROoST (*Reference Ontology on Software Testing*) [19], which aims at defining a shared vocabulary to this domain to be used in Knowledge Management initiatives. It was developed for establishing a common conceptualization about the software testing domain focusing on the testing process in order to support the communication between the stakeholders involved in such a process. In [2], Asman *et al.* present a top-domain software testing ontology that contains general software testing knowledge. They developed this ontology to serve as a basis for the development of new lower level domain ontologies.

In addition, Freitas *et al.* present PTOntology (*Performance Testing*) [11] where semantic technologies are explored to demonstrate the practical feasibility of developing ontology-based applications for assisting testers with performance test planning and management. Arnicans *et al.* [1] propose a methodology for semi-automatically obtaining a lightweight ontology to the software testing domain based on the ISTQB glossary [15]. However, they only built a taxonomy of testing techniques rather than a lightweight ontology. Sapna *et al.* [18] use a testing ontology to represent and manage use cases and scenarios and then to enumerate test scenarios in order to produce a test suite. Cai *et al.* [7] present a construction method of software testing ontologies based on SWEBOK [13] and one software testing classification ontology based on the ISO 9126 software quality model. Just like Arnicans *et al.*, the documented ontology is rather a taxonomy of testing techniques. Bai *et al.* developed TOM (*Test Ontology Model*) [3] that is compatible with the U2TP [23]. Besides, it enriches the semantics of the U2TP

model with class properties and constraints using ontology information. Barbosa *et al.* built OntoTest [4], which aims at supporting acquisition, organization, reuse and sharing of knowledge on the testing domain. OntoTest has a Main Software Testing Ontology and 5 sub-ontologies that address specific concepts of it. Unfortunately, authors only documents in a graphic way the conceptualization of the Main Software Testing Ontology and the Testing Resource sub-ontology. Also, the Testing Resource sub-ontology is rather a taxonomy. Finally, Zhu *et al.* developed a system prototype for testing web-based applications that uses an ontology of software testing named STOWS [26]. It aims at facilitating the communications among agents and between agents and human developers and testers.
























































The abovementioned ontologies were evaluated in [20] from the ontological quality standpoint. The summarized results are shown in Table 1, while the full requirements tree (i.e., the included characteristics, sub-characteristics and attributes), and the outcomes are in <http://bit.ly/OntoQualityEval>. Notice that we use the metaphor of the three-colored semaphore to identify the satisfaction acceptability level achieved. The red color with values within the [0;60] range, in the percentage scale indicates an “unsatisfactory” acceptability level; yellow [60;85] indicates a “marginal” level, and green [85;100] indicates a “satisfactory” level. According to Table 1, the two best scores for the Ontological Structural Quality (1.1) sub-characteristic were obtained by ROoST [19] (79.08% ) followed by Asman *et al.* [2] (61.92% ). The remainder ontologies, none reached even the marginal level. In general, this problem is because most ontologies do not have all their terms, non-taxonomic relationships and properties defined as well as axioms specified. Only [19, 26] have fully specified axioms.

Table 1. Summary of the evaluation results of 12-selected ontologies through SLR regarding their Ontological Quality (1), which includes Ontological Structural Quality (1.1), Domain-specific Terminological Coverage Quality (1.2) and Compliance to other Vocabularies (1.3) sub-characteristics. The green color indicates “satisfactory” acceptability level () , yellow “marginal” () , and red “unsatisfactory” (). Indicators' values are expressed in [%].

	[8]	[25]	[24]	[19]	[2]	[11]	[1]	[18]	[7]	[3]	[4]	[26]
1	40.05 	46.62 	51.28 	79.54 	66.71 	36.41 	32.75 	30.95 	25 	32.79 	42.36 	39.52 
1.1	55 	33.24 	32.56 	79.08 	61.92 	22.81 	0 	16.40 	0 	20.08 	28.72 	59.03 
1.2	50 	75 	100 	50 	100 	50 	100 	50 	50 	50 	50 	50 
1.3	8.50 	50 	50 	100 	52.50 	50 	42.50 	42.50 	50 	42.50 	60 	0 

Looking at Table 1, only three ontologies reached 100% () in the Domain-specific Terminological Coverage Quality (1.2) sub-characteristic, namely: Arnicans *et al.* [1], Asman *et al.* [2], and Vasanthapriyan *et al.* [24]. This means that these ontologies have at least one term related to static testing, one related to dynamic testing, one related to functional testing and one related to non-functional testing. The remainder ontologies only achieved 50% () of coverage, except [25], which met 75% ().

Regarding the Compliance to other Vocabularies (1.3) sub-characteristic, almost all ontologies adhere their terminology to international standard glossaries. The only ones that do not specify the use of international standard glossaries are RTE-Ontology [8] and STOWS [26]. On the other hand, very few ontologies take into account the terminology of other core or domain ontologies, namely: Asman *et al.* [2], OntoTest [4],

RTE-Ontology [8] and ROoST [19]. Finally, ROoST is the only one that was built on a foundational ontology, named UFO. Therefore, ROoST is the only ontology that reached 100% (●) in Compliance to other Vocabularies (1.3) sub-characteristic.

Taking into account the above-summarized analysis, the best-ranked ontology with regard to the Ontological Quality (1) is ROoST (79.54% ◆)—although it did not achieve a satisfactory level. It has a lack of NFRs and static testing terminological coverage and there is no linking with FR and NFR terms. We need a top-domain ontology with higher coverage since we plan to develop more specific testing domain ontologies for dynamic testing such as performance testing, among others. Furthermore, although ROoST is embedded in a network of ontologies whose root is the UFO foundational ontology, we consider that the ontologies (i.e., UFO and mainly its derived process core ontology) used to enrich ROoST are a bit complex in their terminology and therefore they are hard to adopt and harmonize their terms into the architectural components that we present in Section 3. On the other side, even if Asman *et al.* [2] document a top-domain ontology, a foundational ontology as well as its specified axioms are missing.

Therefore, the existing ontologies are not suitable enough to our aim. We have built thus TestTDO taking into account many of their best-ranked features. 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. This ultimately will permit us to build specific software testing strategies and their grouping into a family for test purposes.

3 Overview of the Four-layered Ontological Architecture

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 in two sub-levels, namely: Top-domain and Low-domain ontological levels. 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 one ontology. In addition, ontologies at lower levels can be semantically enriched by ontologies at upper levels. For example, TestTDO placed at the top-domain level is enriched by concepts of the SituationCO and ProcessCO [5] ontologies placed at the core level. In turn, both are enriched by the concepts of ThingFO, which is at the foundational level.

ThingFO terms such as Thing, Thing Category and Assertion semantically enrich terms of components at lower levels. Thing represents a particular or individual, tangible or intangible object of a given particular world, but not a universal category—which is modeled by the term Thing Category. A Thing is not a particular object without its Properties and its Powers, therefore this triad emerge simultaneously to form a unity [10]. A Thing cannot exist or be in spatiotemporal isolation from other Things, so in a particular situation, a target Thing is always surrounded by other context Things.

Besides, we define Assertion as a “*positive and explicit statement that one or more persons make about something concerning Things, their categories, contexts or situations based on thoughts, perceptions, facts, intuitions, intentions, and/or beliefs that may be conceived with an attempt at furnishing current or ulterior evidence*”. In order to be valuable, actionable and ultimately useful for any science, an Assertion should

largely be verified and/or validated by theoretical and/or empirical evidence. Assertions can be represented by informal, semiformal or formal specifications. There are Assertion on Particulars for Thing, and Assertion on Universals for Thing Category. Concerning Thing, by means of assertions, we can specify aspects of its substance, relations, structure, behavior, intention, quantity and quality, among others. For example, the conceptualization of an ontology as an artifact (e.g., TestTDO in Fig. 2) represents primarily a mixture of substance-, relation-, structure- and intention-related assertions. (Notice that the axioms of an ontology can be thought as constraint-related assertions).

As commented above TestTDO is enriched mainly by concepts of the SituationCO and ProcessCO ontologies. In turn, SituationCO includes terms –some borrowed from other components- with semantic of Thing such as Person, Organization, Project and Context Entity. In addition, it includes terms with the semantic of Assertion on Particulars such as Goal and Particular Situation. Briefly, a Person/Organization conceives/establishes Goals that are operationalized by Projects. A Goal implies a Particular Situation of interest to be represented. This situation means an association between Target Entities and none or many Context Entities. Hence, a Target Entity is surrounded by context objects. Depending on the goal purpose, Target Entities can be for instance a Developable Entity (e.g., a document, a source code, etc.), an Evaluable Entity (e.g., a work product, a system, etc.), or a Testable Entity, which has the semantic of Developable or Evaluable in a given Particular Situation.

The ProcessCO ontology [5] includes terms with semantic of Thing such as Work Entity (Work Process, Activity, Task), Work Product (Artifact, Outcome), and Work Resource such as Agent, Method, Strategy, Tool, among others.

Most of the conceptual components of Fig. 1 are documented in [6] and its complementary material named “*Populating the Four-layered Ontological Conceptual Architecture*” in Research Gate. Additionally, we make available some of these ontologies as well as the whole documentation of TestTDO at <http://bit.ly/TestTDO-Doc>.

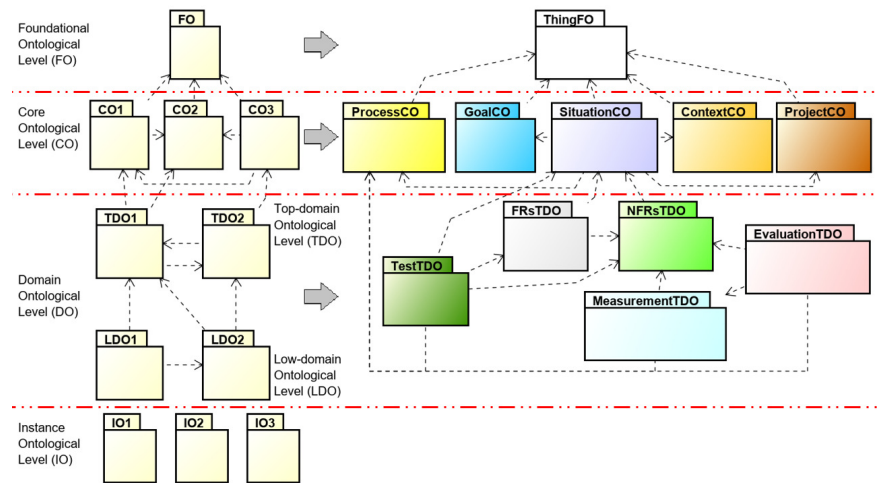


Fig. 1. Four-layered ontological architecture, which considers Foundational, Core, Domain and Instance levels. Also, some conceptual components are shown at the corresponding level. Note that NFRs stands for Non-Functional Requirements while FRs, for Functional Requirements.

4 TestTDO: A Top-Domain Ontology for Software Testing

Following the DSR process, while carrying out the Identify the Problem/Solution (A1) activity, the Artifact Requirements was produced –after performing the Specify Artifact Requirements task [22]. As part of this specification, 25 Competency Questions (CQ) to cover the TestTDO scope were agreed upon. As indicated in the Introduction Section, the artifact development implies a cycle of design-construction-evaluation activities, which must iterate as many times as necessary before the artifact is communicated and is ready for its use. Thus, we have enacted the Design and Develop the Solution (A2) and Execute Verification and Validation (A3) activities. As a result of this cycle, the developed TestTDO ontology comprises 43 defined terms, 48 defined properties, 36 defined non-taxonomic relationships as well as 14 axioms specified in first-order logic. The present section and the next one describe aspects of the CQs and the resulting artifact as well as analyze TestTDO verification, validation and evaluation respectively.

With regard to the 25 CQs, we have formulated questions related to Test Project, Goal, Requirement, Entity and Work Product aspects as well as to Testing Process, Agent, Activity and Method ones. For example, below, the CQ03 is a Work product-related question; CQ14 is a Method-related question; CQ19 is an Entity-related question; CQ24, a Test Requirement-related question; and CQ25, a Project-related question.

CQ03. What are the work products produced by a testing realization activity?

CQ14. What are the types of testing methods assigned to a testing design activity?

CQ19. In which particular situation is a testable entity considered a developable entity?

CQ24. Is a test requirement related to functional and nonfunctional requirements?

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?

We will use these CQs for the verification matrix in Section 5. The remainder CQs can be accessed at <http://bit.ly/TestTDO-CQuestions>. As a result of considering the functional and non-functional requirements specified in the Artifact Requirements document, the TestTDO conceptualization was produced as depicted in Fig. 2. It represents its key terms, properties and relationships in addition to its relation with Non-Functional Requirement and Functional Requirement terms, which are included accordingly in the NFRsTDO and FRsTDO components in Fig. 1. The reader can access all definitions of terms, properties and relationships at <http://bit.ly/TestTDO-Defs>. Also, the 14 axioms' specifications can be accessed at <http://bit.ly/TestTDO-AxiomSpecs>.

In the sequel, we describe aspects of the conceptualization of TestTDO using the following convention in the text: the ontology terms begin with capital letters, the properties are italicized and relations underlined.

In order to cover project-related competency questions, TestTDO has terms such as Test Project, Testing Management, Test Plan, Testing Lyfe Cycle and Testing Strategy. These terms are semantically enriched with terms of the ProjectCO module depicted in Fig. 1. Likewise, to cover goal-related competency questions, TestTDO has terms such as Test Goal and Test Information Need, which are 'extended' from the GoalCO component (Fig. 1). In a nutshell, a Test Project (and its subTestProject, if any) operationalizes one or more Test Goals. It associates one or more Testing Strategies, which help to achieve the Test Goals' *purposes*. In addition, a Test Goal is supported by one or more Test Information Needs. Notice that the Testing Analysis activity takes into account the *statement* of a given Test Information Need goal.

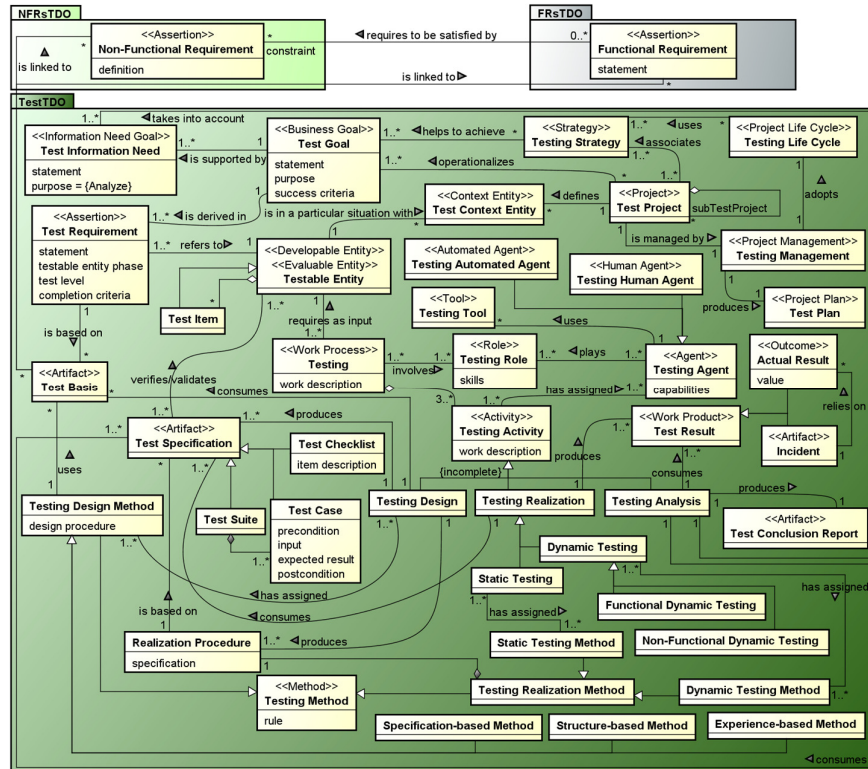


Fig. 2. Main terms, properties and relationships of the TestTDO ontology and its relation with Non-Functional Requirement and Functional Requirement terms.

In order to cover the test requirement- and entity-related scope, TestTDO has terms such as Test Requirement, Test Basis, Testable Entity, Test Item and Test Context Entity. These terms are semantically enriched with terms of ThingFO, ProcessCO, SituationCO and ContextCO modules (Fig. 1). Test Requirement “states, taking into account the Test Goal *purpose*, what must be verified/validated of a Testable Entity (and/or Test Item) based on the Test Basis, if any”. Note that a Test Requirement must include the *test level* (e.g., unit, integration, system, acceptance, etc.) and the *testable entity phase* (e.g., development, operative, maintenance, etc.). While Test Basis is an “artifact used by Testing Design Methods for designing the Test Cases and Checklists”. So the Test Basis represents an Artifact 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 Non-Functional Requirement and/or Functional Requirement (terms with semantic of) assertions. Furthermore, Testable Entity is “a concrete object able to be tested”. A Testable Entity may have none or many Test Items, which in turn are testable. Note that depending on the particular situation, Testable Entity has semantic of Developable Entity (from the FRsTDO component) or Evaluable Entity (from the NFRsTDO component). Lastly, looking at Fig. 2, we can say that Test Goal is derived in one or more Test Requirements. In turn, Test Requirement refers to

a Testable Entity, which can be in a particular situation with Test Context Entities.

In order to cover the process-, activity- and agent-related scope at the top-domain level, TestTDO has terms such as Testing with semantic of Work Process, Testing Activity, Testing Design (Activity), Testing Realization, Static Testing (Realization Activity), Dynamic Testing, Functional Dynamic Testing (Realization Activity), Non-Functional Dynamic Testing, and Testing Analysis (Activity). Additionally, there are terms such as Testing Agent and Testing Role among others. These terms, their properties and relations are fully enriched with terms, properties and relations of the ProcessCO module depicted in Fig. 1. Testing “is a Work Process that is composed of at least three interrelated Testing Activities conducted to facilitate the discovery of defects and/or the assessment of Characteristics and Attributes of a Testable Entity”. (Note that in this definition, the terms Characteristic and Attribute –which are kind of Non-Functional Requirement- are included in the NFRsTDO component. On the other side, notice that “...at least three...” means that we may foresee another activity such as Testing Context Set-up, knowing beforehand that the design of the context can be a sub-activity of the Testing Design activity).

In order to cover the work product-related scope, TestTDO has terms such as Test Basis, Test Specification, Test Result, and Test Conclusion Report. Particularly, Test Specification has a semantic of Artifact and there are three types of it, namely: Test Checklist, Test Case and Test Suite. On the other side, Test Result has semantic of Work Product and there are two types of it, namely: Actual Result (as Outcome) and Incident (as Artifact or document, which reports deviations –e.g. between the *expected result* and the Actual Result-, anomalies –e.g. an error or a failure- or other arisen issues during the Testing Realization). These terms, their properties and relations are fully enriched with terms, properties and relations of the ProcessCO module [5].

Regarding Test Case, it “is a Test Specification that contains the necessary information (e.g. *preconditions*, *inputs*, *expected results* and *postconditions*) to perform mainly Dynamic Testing”. While Test Checklist “is a Test Specification that contains a list of *items (descriptions)* to be checked in order to perform mainly Static Testing”.

We define Testing Design as a “Testing Activity aimed at designing a set of Test Specifications (i.e., Test Cases, Test Suites and/or Test Checklists) as well as Realization Procedures”. This activity consumes none or more Test Basis and produces Test Specifications and Realization Procedures. Besides, we define Testing Realization as a “Testing Activity aimed at enacting a Static or Dynamic Testing”. This activity consumes one or more Test Specifications and produces one or more Test Results. Lastly, we define Testing Analysis as a “Testing Activity that takes into account the specific Test Information Need in order to produce a Test Conclusion Report by consuming one or more Test Results and Test Specifications”.

It is worth mentioning at this point that a Work Process or Activity primarily represents ‘what’ to do rather than indicate ‘how’ to do it by using a particular Method and Tool applied to a *work description*. Method (in ProcessCO) has two properties: procedure and rules.

To cover the method-related scope at the top-domain level, TestTDO has terms such as Testing Method with two sub-types like Testing Design Method and Testing Realization Method. For the former there are three kinds of design methods, namely: Specification-based Method (also known as black-box), Structure-based Method (white-box), and Experience-based Method. For the latter there are two kinds of realization

methods, namely: Dynamic Testing Method and Static Testing Method.

We define Specification-based Method as “a Testing Design Method that always uses a Test Basis for deriving Test Specifications without referring to the internal structure of the Testable Entity”. More specific types of Specification-based Methods are Boundary Value Analysis, Equivalence Partitioning, State Transition Testing, and Decision Table Testing, among others, which are not shown in Fig. 2 since TestTDO terms are a top-domain level. In addition, we define Structure-based Method as “a Testing Design Method that uses the internal structure of the Testable Entity, and sometimes also uses a Test Basis, for deriving Test Specifications”. Examples of it are Branch Testing, Statement Testing, Condition Testing, and Data Flow Testing, among others.

It is important to highlight that the Testing Realization Method has a Realization Procedure, which represents the “arranged set of Testing Realization Method’s instructions or operations which specifies how must be performed the Testing Realization activity using the Test Specification”.

Lastly, the reader can observe that a Testing Design (Activity) has assigned one or more Testing Design Methods, while a Dynamic/Static Testing (Realization Activity) has assigned one or more Dynamic/Static Testing Methods.

Next, we first illustrate aspects of the verification of the TestTDO coverage, which embraces specified axioms as well. Then, we describe some validation and evaluation issues.

5 Verifying, Validating and Evaluating TestTDO

In order to cover the established scope in the A1 (DSR) activity, TestTDO should be able to answer all the CQs. In this direction, in the previous section we made to some extent descriptions of most of the TestTDO terms, properties and relations. In this section, in Table 2, we present an excerpt from the verification matrix with the TestTDO terms, relationships, properties and axioms that answer the five CQs listed previously, in Section 4. Note that Souza *et al.* [19] applied a similar approach for ROoST. The whole verification matrix can be accessed at <http://bit.ly/TestTDO-VerMatrix>.

In the cycle of DSR design-construction-evaluation activities, we continuously checked if the CQs were being answered. Consequently, when we verified that all the CQs were addressed by the terms, properties, relationships and axioms defined in the TestTDO artifact, this cycle ended. This verification matrix allowed us to check not only if the CQs were answered, but also whether there were unnecessary elements for the TestTDO scope.

On the other hand, in order to validate initially 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. This proof of concept is based on the running testing example used by G. J. Myers in [16], and can be accessed at <http://bit.ly/TestTDO-Val>. As a result of this validation, we can conclude that TestTDO is able to represent this rather simple situation. However, we need to implement TestTDO in industrial projects that deal with more complex real-world situations. It is worth mentioning that the third co-author of this paper currently works –playing the tester role- in a company that provides software solutions mainly to abroad customers. So far, TestTDO has covered all the terminological situations he has had to tackle.

Table 2. Excerpt from the TestTDO's Verification Matrix. Note that CQ stands for Competency Question. Also note that at the bottom of this table the highlighted **A**xioms are specified in first-order logic.

CQ	Terms, relationships and properties	A xioms
CQ3	Testing Realization is-a Testing Activity Testing Realization produces Test Result Actual Result is-a Test Result Incident is-a Test Result	A1 , A7, A11
CQ14	Testing Design is-a Testing Activity Testing Design has assigned Testing Design Method Testing Design Method is-a Testing Method Specification-based Method is-a Testing Design Method Structure-based Method is-a Testing Design Method Experience-based Method is-a Testing Design Method	A10 , A12
CQ19	Test Requirement refers to Testable Entity Test Requirement is based on Test Basis Test Basis is linked to Functional Requirement	A6
CQ24	Test Requirement is based on Test Basis Test Basis is linked to Non-Functional Requirement Test Basis is linked to Functional Requirement	A5 , A6
CQ25	Test Project operationalizes Test Goal Test Project associates Testing Strategy Testing Strategy helps to achieve Test Goal Test Goal has the property named purpose	A8

A1) For any Testing Realization activity that produces a Test Result, this result is therefore an Actual Result or an Incident, but not both at the same time.

$$\forall trn, \exists tr: \text{TestingRealization}(trn) \wedge \text{TestResult}(tr) \wedge \text{produces}(trn, tr) \rightarrow \text{ActualResult}(tr) \vee \text{Incident}(tr)$$

A5) Any Testable Entity is an Evaluable Entity iff the Test Requirement that refers to this Thing is linked to a Non-Functional Requirement.

$$\forall te: \text{TestableEntity}(te) \wedge \text{EvaluableEntity}(te) \leftrightarrow \exists tr, tb, nfr: \text{TestRequirement}(tr) \wedge \text{TestBasis}(tb) \wedge \text{NonFunctionalRequirement}(nfr) \wedge \text{refersTo}(tr, te) \wedge \text{isBasedOn}(tr, tb) \wedge \text{isLinkedTo}(tb, nfr)$$

A6) Any Testable Entity is a Developable Entity iff the Test Requirement that refers to this Thing is linked to a Functional Requirement.

$$\forall te: \text{TestableEntity}(te) \wedge \text{DevelopableEntity}(te) \leftrightarrow \exists tr, tb, fr: \text{TestRequirement}(tr) \wedge \text{TestBasis}(tb) \wedge \text{FunctionalRequirement}(fr) \wedge \text{refersTo}(tr, te) \wedge \text{isBasedOn}(tr, tb) \wedge \text{isLinkedTo}(tb, fr)$$

A8) All Test Project operationalizes a Test Goal and associates a Testing Strategy iff this Testing Strategy helps to achieve the operationalized Test Goal.

$$\forall tp, \exists tg, ts: \text{TestProject}(tp) \wedge \text{TestGoal}(tg) \wedge \text{TestStrategy}(ts) \wedge \text{operationalizes}(tp, tg) \wedge \text{associates}(tp, ts) \leftrightarrow \text{helpsToAchieve}(ts, tg)$$

A10) If a Testing Design activity has assigned a Specification-based Method and produces a Test Specification, then always consumes a Test Basis, which is used by the Specification-based Method without using the internal structure of the Testable Entity.

$$\forall td, \exists spbm, ts: \text{TestingDesign}(td) \wedge \text{Specification_basedMethod}(spbm) \wedge \text{TestSpecification}(ts) \wedge \text{hasAssigned}(td, spbm) \wedge \text{produces}(td, ts) \rightarrow \exists tb, te: \text{TestBasis}(tb) \wedge \text{TestableEntity}(te) \wedge \text{uses}(spbm, tb) \wedge \text{consumes}(td, tb) \wedge \text{requiresAsInput}(td, te) \wedge \neg \text{uses}(spbm, te)$$

Table 3. Summary of the evaluation results of the 3 best-ranked ontologies of Table 1, and their comparison with the TestTDO evaluation results. The green color indicates “satisfactory” acceptability level (●); yellow “marginal” (◆) and red “unsatisfactory” (■). Indicators' values are expressed in [%].

Characteristics / Attributes	Vasant-hapriyan <i>et al.</i> [24]	ROoST [19]	Asman <i>et al.</i> [2]	TestTDO
I. Ontological Quality	51.28 ■	79.54 ◆	66.71 ◆	98.71 ●
1.1 Ontological Structural Quality	32.56 ■	79.08 ◆	61.92 ◆	97.43 ●
<i>1.1.1 Defined Terms Availability</i>	15.60 ■	82.20 ◆	100 ●	100 ●
<i>1.1.2 Defined Properties Availability</i>	0 ■	0 ■	0 ■	100 ●
<i>1.1.3 Specified Axioms Availability</i>	50 ■	100 ●	0 ■	100 ●
1.1.4 Balanced Relations Availability	54.40 ■	87.32 ●	73.07 ◆	91.43 ●
<i>1.1.4.1 Balanced Non-Taxonomic Relationships Availability</i>	68 ◆	95.65 ●	66.34 ◆	89.29 ●
<i>1.1.4.2 Defined Non-Taxonomic Relationships Availability</i>	0 ■	54 ■	100 ●	100 ●
1.2 Domain-specific Terminological Coverage Quality	100 ●	50 ■	100 ●	100 ●
...
1.3 Compliance to other Vocabularies	50 ■	100 ●	52.50 ■	100 ●
<i>1.3.1 Terminological Compliance to International Standard Glossaries</i>	100 ●	100 ●	85 ●	100 ●
<i>1.3.2 Terminological Compliance to other Domain/Core Ontologies</i>	0 ■	100 ●	100 ●	100 ●
<i>1.3.3 Terminological Compliance to Foundational Ontologies</i>	0 ■	100 ●	0 ■	100 ●

Last but not least, since we have conducted the ontological quality evaluation of the 12 SLR-selected ontologies, we also planned in the DSR A3 activity to evaluate TestTDO and compare its ontological quality with them. To this end, we have used the same non-functional requirements tree (i.e., all the same sub-characteristics and attributes for the Ontological Quality characteristic), but now including the TestTDO outcomes (see <http://bit.ly/OntoQualityEvalwithTestTDO>). Table 3 shows a fragment of this evaluation including only the 3 best-ranked ontologies of Table 1.

For the 1.1 (Ontological Structural Quality) sub-characteristic and its 3 attributes, TestTDO has met 100% (●) since all the terms and properties are explicitly defined, and the axioms specified. The reader can see and compare this situation with the other three ontologies in Table 3.

Regarding the 1.1.4 sub-characteristic, we define it as “Degree to which an ontology has a balance between the amount of non-taxonomic and taxonomic relationships in addition to the former are defined”. Note that non-taxonomic relationships are those which are not ‘kind of’ (is_a) or ‘whole-part’ (part_of). Non-taxonomic relationships should therefore be defined. We have specified in [21], for the “Balanced Non-Taxonomic Relationships Availability” (1.1.4.1) attribute, its metric that quantifies it, and its elementary indicator that interprets it. As a result of the 1.1.4 sub-characteristic, TestTDO met the satisfactory acceptability level (91.43% ●).

Looking at the Compliance to other Vocabularies (1.3) sub-characteristic, almost all ontologies adhere their terminology to one or more international standard glossaries, as commented in Section 2. TestTDO was built considering official and *de facto* international standards such as ISO 29119 [14] and ISTQB [15], which are widely adopted by

professional testers. On the other hand, ROoST is the unique domain software testing ontology that was built on a foundational ontology. We have considered this attribute of ROoST as a strength to be adopted. Therefore, TestTDO was conceived at the top-domain level considering also our foundational and core ontologies in the framework of the four-layered (FCD-OntoArch) architecture depicted in Fig. 1.

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 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 selected 12 ontologies. Furthermore, there was no software testing ontology directly linked with NFRs and FRs ontological concepts.

TestTDO was created for terminologically nourishing specifications of methods and processes to a family of testing strategies to be developed from now on. TestTDO is a top-domain level ontology in the framework of the four-layered architecture, which was designed to be extended by lower-level software testing domain ontologies.

Therefore, in the present work, we have discussed aspects of the TestTDO development, verification, validation and evaluation for its conceptualization. Depending on the research aim, a conceptualized ontology may also be implemented in a formal semantic language such as OWL. Obviously that an implemented ontology further allows it can be verified automatically by using a query language for OWL such as SQWRL for instance. Hence, CQs can be implemented in SQWRL for verification. We envision to implement TestTDO even this was not our primary aim.

Acknowledgments. This work and line of research are supported by the Science and Technology Agency of Argentina, in the PICT 2014-1224 project at UNLPam.

References

1. Arnicans G., Romans D., Straujums U.: Semi-automatic Generation of a Software Testing Lightweight Ontology from a Glossary Based on the ONTO6 Methodology, *Frontiers in Artificial Intelligence and Applications*, V.249, pp. 263-276 (2013)
2. Asman A., Srikanth R. M.: A Top Domain Ontology for Software Testing, Master Thesis, Jönköping University, Sweden, pp. 1-74 (2016)
3. Bai X., Lee S., Tsai W. T., Chen Y.: Ontology-Based Test Modeling and Partition Testing of Web Services, *IEEE Int'l Conference on Web Services (ICWS'08)*, pp. 465-472 (2008)
4. Barbosa E. F., Nakagawa E. Y., Riekstin A. C., Maldonado J. C.: Ontology-based Development of Testing Related Tools, *20th International Conference on Software Engineering and Knowledge Engineering (SEKE'08)*, pp. 697-702 (2008)
5. Becker P., Papa F., Olsina L.: Process Ontology Specification for Enhancing the Process Compliance of a Measurement and Evaluation Strategy, *CLEI eJnal.*, 18:(1), pp. 1-26 (2015)

6. Becker P., Tebes G., Peppino D., Olsina L.: Applying an Improving Strategy that embeds Functional and Non-Functional Requirements Concepts. *Journal of Computer Science & Technology*, 19:(2), pp. 153-174 (2019)
7. Cai L., Tong W., Liu Z., Zhang J.: Test Case Reuse Based on Ontology, 15th IEEE Pacific Rim International Symposium on Dependable Computing, pp. 103-108 (2009)
8. Campos H., Acácio C., Braga R., Araújo M. A. P., David J. M. N., Campos F.: Regression Tests Provenance Data in the Continuous Software Engineering Context, In 2nd Brazilian Symp. on Systematic and Automated Software Testing (SAST), Paper 10, pp. 1-6 (2017)
9. D'Aquin M., Gangemi A.: Is there beauty in ontologies? *Applied Ontology*, 6:(3), pp. 165-175 (2011)
10. Fleetwood S.: The ontology of things, properties and powers. *Journal of Critical Realism*, 8:(3), pp. 343-366, Available at <http://eprints.uwe.ac.uk/15967> (2009)
11. Freitas A., Vieira R.: An Ontology for Guiding Performance Testing, IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), (WI-IAT'14), V.1, pp. 400-407 (2014)
12. Hevner A. R., March S. T., Park J., Ram S.: *Design Science in Information Systems Research. Management Information Systems Quarterly*, 28:(1), pp. 75-105 (2004)
13. IEEE: Computer Society, SWEBOK. A Guide to the Software Engineering Body of Knowledge. Available at <https://www.computer.org/education/bodies-of-knowledge/software-engineering>, (2004)
14. ISO/IEC/IEEE 29119-1: Software and systems engineering - Software Testing – Part 1: Concepts and Definitions (2013)
15. ISTQB. International Software Testing Qualifications Board, Standard Glossary of Terms used in Software Testing, Version 3.2. Available at <https://www.istqb.org/>, (2019)
16. Myers G.J., Sandler C., Badgett T.: *The Art of Software Testing* (3rd ed.). Wiley Publishing, ISBN 978-1118031964 (2011)
17. Olsina L., Becker P.: Family of Strategies for different Evaluation Purposes, XX CIBSE'17, CABA, Argentina, Published by Curran Associates, pp. 221-234 (2017)
18. Sapna P. G., Mohanty H.: An Ontology Based Approach for Test Scenario Management, 5th International Conference on Information Intelligence, Systems, Technology and Management (ICISTM'2011), V.141, pp. 91-100 (2011)
19. Souza E. F., Falbo R. A., Vijaykumar N. L.: ROoST: Reference Ontology on Software Testing, *Applied Ontology Journal*, 12:(1), pp. 1-30 (2017)
20. Tebes, G., Peppino, D., Becker, P., Maturro, G., Solari, M., Olsina, L.: A Systematic Review on Software Testing Ontologies, 12th International Conference on the Quality of Information and Communications Technology, Springer book, CCIS V.1010, M. Piattini et al. (Eds.): QUATIC 2019, Ciudad Real, Spain, pp. 144-160 (2019)
21. Tebes, G., Peppino, D., Becker, P., Maturro, G., Solari, M., Olsina, L.: Analyzing and Documenting the Systematic Review Results of Software Testing Ontologies. Under review in an International Journal, pp. 1-34 (2020)
22. Tebes, G., Peppino, D., Rivera M.B., Becker, P., Papa, M.F., Olsina L.: Especificación del Proceso de Design Science Research: Caso Aplicado a una Ontología de Testing de Software. 7^{mo} CoNaISI'19, La Matanza, Argentina, Nov.14-15, pp. 1-10 (2019)
23. UML Testing Profile (U2TP), V2.0, Available at <https://www.omg.org/spec/UTP2/2.0/PDF>
24. Vasanthapriyan S., Tian J., Xiang J.: An Ontology-Based Knowledge Framework for Software Testing, In Comm. in Computer and Information Science, V.780, pp. 212-226 (2017)
25. Vasanthapriyan S., Tian J., Zhao D., Xiong S., Xiang J.: An Ontology-Based Knowledge Sharing Portal for Software Testing, In IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C'17), pp. 472-479 (2017)
26. Zhu H., Huo Q.: Developing A Software Testing Ontology in UML for a Software Growth Environment of Web-Based Applications, In Software Evolution with UML and XML, IDEA Group, pp. 263-295 (2005)