Specifying the Design Science Research Process: An Applied Case of Building a Software Testing Ontology

Guido Tebes, Belen Rivera, Pablo Becker, Maria Fernanda Papa, Denis Peppino, and Luis Olsina
GIDIS_Web, Facultad de Ingeniería, UNLPam, General Pico, LP, Argentina
[guido.tebes92, denispeppino92]@gmail.com
[riveramb, beckerp, pmfer, olsinal]@ing.unlpam.edu.ar;

Abstract. Design Science Research (DSR) is a rigorous investigation approach that promotes the development of artifacts to meet a useful solution to a relevant domain problem. The artifact should be an innovative solution for a non-trivial problem. After determining the relevance of a given problem/solution, the development of the artifact involves a cycle of design-development-evaluation activities, which iterates as many times as needed before the artifact is finally verified, validated and communicated for its use. The cutting-edge literature about the DSR approach permitted us to observe that, at the moment of this work, the perspectives of its process model are weakly specified. Aimed at contributing to this aspect, this work represents the DSR process specification using the SPEM language. Moreover, we illustrate its main activities by applying them in the construction process of an artifact, i.e., the software testing ontology named TestTDO. This top-domain ontology was recently developed, verified, validated and evaluated in order to afterward building strategies that help achieve diverse test goals' purposes when carrying out test projects.

Keywords: DSR, Artifact, Top-domain Testing Ontology, DSR process, SPEM

1 Introduction

A strategy is an organizational resource that defines a specific course of action to be followed for achieving goal purposes. Organizations arrange work through projects, which help operationalize goal purposes. These can be diverse and, mainly for engineering, may be classified into evaluation, testing, development, and maintenance.

So far, we have developed evaluation strategies that integrate three capabilities, namely: process specifications, method specifications, and domain conceptual bases structured in ontologies [3, 17]. Having strategies that integrate these three capabilities allows engineers to understand what to do (processes) and how to do it (methods and tools) in the context of a reliable comprehension of the key concepts for a particular domain, for example, to the measurement and evaluation domain. In [17], we discussed and illustrated a family of strategies that help achieve evaluation goal purposes.

Taking into account that integrated strategies we have developed give support for achieving evaluation goal purposes, it is possible to envision other strategies for test goal purposes using well-established testing activities and methods in the context of

common terminology. Therefore, for the testing domain, we should also have a software testing ontology, which explicitly and unambiguously defines terms, properties, relationships and axioms.

Building the software testing ontological artifact is not a trivial endeavor since testing is a complex area that involves a large number of domain-specific concepts for testing activities, methods, agents, roles, as well as test goals, entities and artifacts, among others. Moreover, many of these concepts may be semantically enriched by terms of core ontologies for the process, project, goal, among others. Hence, a sound research approach may prove to be useful in the artifact development process as well.

Since Design Science Research (DSR) is a rigorous investigation approach that supports the design and construction of useful and innovative artifacts to solve non-trivial problems [11], we follow this approach to build the software testing ontology. Importantly, a non-trivial problem is a difficult problem to solve because it could be incomplete, contradictory or have changing requirements that are often hard to recognize and elicit. DSR indicates that the designed artifact must be evaluated to demonstrate that it not only solves the problem but that it does so in an effective and efficient way, giving utility to the user [11]. An artifact can be defined as an object created by a person, with the intention of later use. This definition emphasizes two key aspects of an artifact: that it does not occur naturally and that it has a certain utility [15]. According to [14] four types of artifacts can be distinguished: constructors (vocabularies), models (abstractions), methods (algorithms) and instantiations (implementations). The software testing ontology we present in this paper could be considered in both the constructor and model category.

To build any artifact, DSR proposes a set of activities that include design, construction and evaluation. These activities are repeated as many times as necessary to generate an artifact that solves the problem effectively. The literature specifies that to apply the DSR approach, it is necessary to go through a first stage of identifying the problem and its relevance. Then, the artifact that solves that problem must be designed, constructed, verified and validated to ensure that the solution constitutes an improvement for a given domain problem. Lastly, the artifact is communicated to the intended audience and the acquired knowledge should be disseminated. But so far, many authors [8, 9, 15, 16, 19, 20] agree to indicate that there are weaknesses in the current DSR process specification. To bridge this gap, it is needed to have a well-specified DSR process to allow repetitiveness and reproducibility through a set of activities with inputs and outputs, interdependencies, among other aspects.

Curtis *et al*. [5] propose four perspectives (or views) to model a process, namely: functional, informational, behavioral and organizational. The former depicts what activities are performed and which are the artifacts consumed and produced by activities and tasks. The informational view includes the structure and interrelationships among the artifacts that activities use. On the other hand, the behavioral perspective indicates in which order the activities are performed. The latter view deals with agents (in roles compliance) that perform the activities. Modeling a process taking into account these perspectives fosters to have its activities descriptions well-understood and without ambiguity, which makes it easier to communicate.

Regarding the above issues, this paper has two main contributions. On one hand, due to the weaknesses detected in the DSR process, we contribute by enhancing the DSR process specification by showing in this work just the functional and behavioral

perspectives using the SPEM language [18]. On the other hand, we discuss the applicability of this process by illustrating the construction cycle of the top-domain software testing ontology named TestTDO. This ontology will serve as the conceptual basis for the development of integrated strategies that help achieve test goal purposes.

It is worth mentioning that TestTDO is integrated into the FCD-OntoArch (*Foundational, Core, and Domain Ontological Architecture for Sciences* [4]) architecture, as shown in Fig. 1. Moreover, all the previously developed ontologies [3, 4] that give terminological support to evaluation strategies are also placed into FCD-OntoArch. 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. Ontologies at lower levels can be semantically enriched by ontologies at upper levels. For example, TestTDO, the built artifact illustrated in this paper is placed at the top-domain level. In turn, TestTDO is enriched by concepts of the ProcessCO [3] ontology placed at the core level. And ProcessCO is enriched by terms of ThingFO at the foundational level.

On the other hand, this work was initially documented in [23], in Spanish. But the present paper is a meaningful extension of [23] since we have already finished the target artifact (TestTDO), so we can now illustrate the different activities of the DSR process with all yielded artifacts during the design-build-evaluate cycle. Furthermore, in [23], we did not consider the solution requirements that TestTDO should be included in the context of the FCD-OntoArch architecture. This additional requirement had an important impact on the final version of TestTDO produced by the DSR process. Also, it helped us to exemplify issues that the DSR process must entail.

The rest of the paper is organized as follows. Section 2 shows the proposed specification for the DSR process, which is illustrated with TestTDO, the built artifact. Section 3 analyzes related work regarding DSR processes. Finally, Section 4 summarizes conclusions and future work.

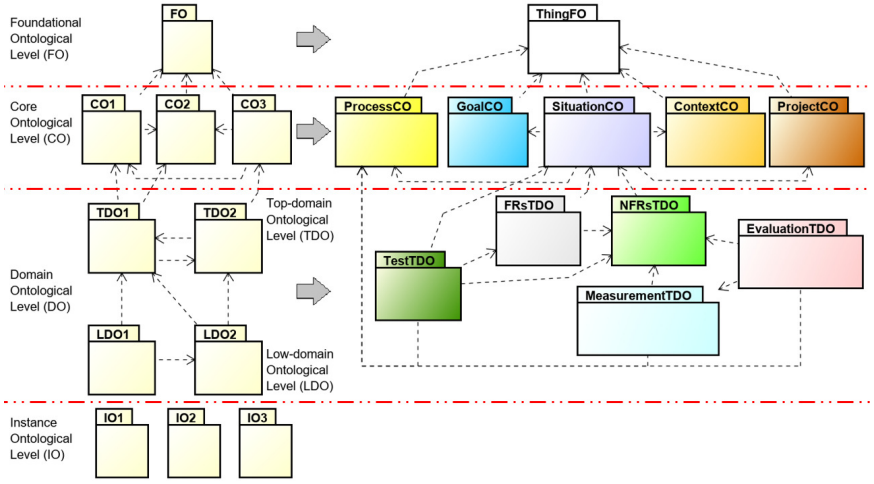

Fig. 1. Four-layered ontological architecture, which considers Foundational, Core, Domain and Instance levels. Also, some conceptual components are shown at the corresponding level. Note that NFRs stands for Non-Functional Requirements while FRs for Functional Requirements

2 Building the TestTDO Artifact using our DSR Process Model

In this Section, we deal with the two contributions stated in the Introduction Section. Thus, from the functional and behavioral perspectives, the proposed DSR process model is specified, and the illustration of each of its activities to build a software testing ontology is described.

As per [10], the DSR approach consists of three research cycles, namely: Relevance, Design and Rigor. In the Relevance cycle, state-of-the-art research is carried out to determine the relevance of the target problem to be addressed. During the Design cycle, activities to design and build the artifact (solution) are performed. This also includes selecting the design process, methods and heuristics to build the artifact, as well as determining how the design and artifact will be evaluated. Finally, in the Rigor cycle, it must be ensured that the design and construction are based on scientific theories and well-established methods stemming from the knowledge base (KB). If the artifact is useful for solving a relevant problem, it can be incorporated into the KB as a contribution.

Taking into account these three cycles in addition to related work as we analyze in Section 3, Fig. 2 shows the DSR process, which is modeled considering the functional and behavioral perspectives [5]. The activities/tasks that correspond to the Relevance cycle are included in the main activity named "A1 Identify Problem/Solution". Furthermore, the activities included in the Design cycle are considered into the main activities named "A2 Design and Develop the Solution" and "A3 Execute Verification and Validation (V&V)". Lastly, the Rigor cycle is related to the information flow from the KB to some sub-activities that belong to A1, A2 and A3. And, after knowledge dissemination ("A4 Communicate Research" activity), if the artefact is useful and adopted, KB could be fed.

Next, we describe these activities for the DSR process while illustrating its applicability through the development of a software testing ontology as an artifact.

A1 Identify Problem/Solution: as shown in Fig. 2 (light blue box), A1 is the first activity that embraces: "Define Problem/Solution", "Investigate Current Solutions" and "Specify Design Requirements" sub-activities. The former, in turn, has two tasks viz. "Define Business Goal/Problem" and "Define the Solution". To define the problem or business goal, the 'Business/Information Need Goals' artifact is required as input. That is, the organization's knowledge about the problem or objective to be achieved. To our case, we considered the following business goal: "*the GIDIS_Web research group requires an ontological conceptualization of the software testing domain to be integrated into the FCD-OntoArch architecture. This artifact must firstly be related to the FRsTDO and NFRsTDO components*". The output of this task is the artifact named 'Business Problem Definition'. In order to specify this artifact, we use the problem specification template proposed in [26]:

Obtain: a rigorous conceptualization of the testing domain.

By means of: the structuring of the domain as an ontology.

Such that: the ontology must be related to the FRsTDO and NFRsTDO components of FCD-OntoArch architecture (Fig. 1). In addition, it should be at the top-domain ontological level.

In order to achieve: support for the specification of a family of integrated testing strategies aimed at satisfying goals with test purposes.

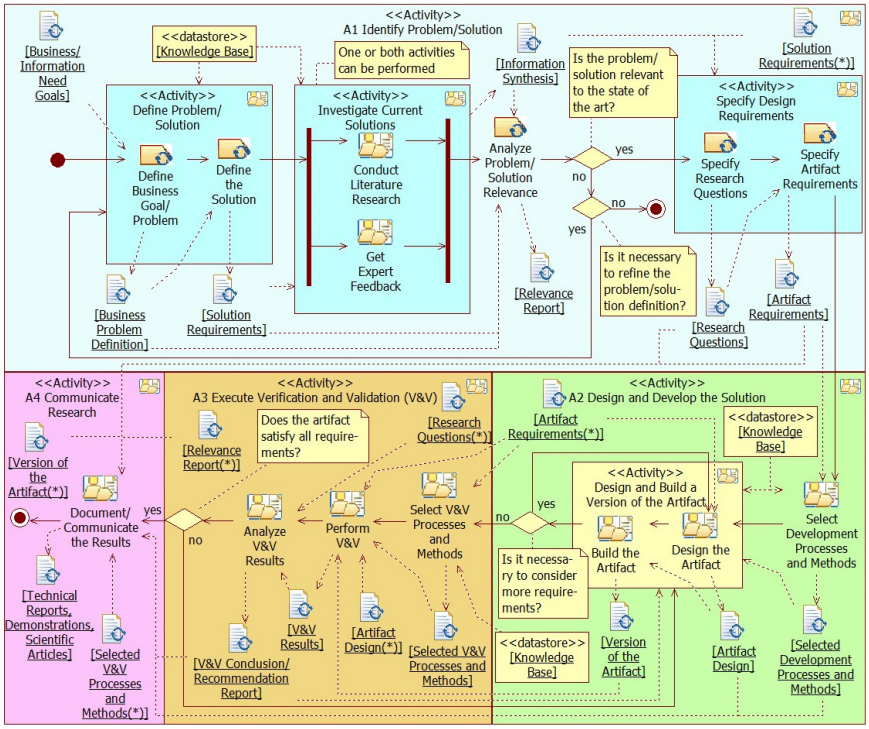

Fig. 2. Functional and behavioral perspectives for the proposed DSR process model. Note that artifacts' names tagged with (*) at the end were replicated, for readability reasons

To define the solution, first of all, it is necessary to inquire about other solutions to similar problems, as well as any other documentation that allows establishing the solution requirements. To do this, we retrieved the documents referenced in [1, 6, 12, 13, 21] from the KB. After revising these documents and considering the initial problem, we established the 'Solution Requirements'. As a result, we define 9 solution requirements (SR). For example: SR#1 *The ontology must be conceptualized but not necessarily implemented*; and SR#9 *Terms of the software testing ontology should be based on (enriched with) terms of higher-level ontologies at the core and/or foundational level*. The list of all solution requirements are at http://bit.ly/SolReqs.

The second activity of A1 is "Investigate Current Solutions", which includes 2 sub-activities: "Conduct Literature Research" and "Get Expert Feedback". Although Fig. 2 shows these activities in parallel, sometimes it is not necessary to perform both or they can be done sequentially in any order. The inputs required to perform them are the KB and the 'Solution Requirements'. We believe that it is important to make an in-depth state-of-the-art analysis of the current situation regarding the established solution requirements and, in addition, ask experts in the domain. In [2] a process that guides the main activities involved in a Systematic Literature Review (SLR) is documented. To our case, we performed the "Conduct Literature Research" activity by a SLR and obtained the 'Information Synthesis' as output. In short, we selected and

evaluated 12 software testing ontologies considering also a NFRs tree for evaluation. The tree was built using the 'Solution Requirements' artifact in addition to some ontological quality practices proposed by D'Aquin *et al*. [6]. The reader can access the conducted SLR in [22], and also the NFRs tree and its evaluation results at http://bit.ly/OntoQualityEval.

According to Fig. 2, the following task to carry out is "Analyze Problem/Solution Relevance". This requires as input the 'Information Synthesis', the 'Solution Requirements' and the 'Business Problem Definition' artifacts in order to produce the 'Relevance Report'. This report describes the relevance of the problem/solution to decide whether it will be addressed by the DSR approach. Fig. 3 illustrates this document, considering the main factors that positively impacted the decision to face the building of the software testing ontology by means of the DSR approach.

Relevance Report
The following are the main reasons why to address the problem/solution through the DSR approach: i) there is no SLR-selected ontology that satisfactorily fulfils all the stated solution requirements that also meet 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.

Fig. 3. 'Relevance Report' artifact for the software testing ontology case

When the problem and its solution are not relevant for the state of the art and the stakeholders, the DSR process ends or the problem/solution definition is redefined, returning to the "Define Problem/Solution" sub-activity. Otherwise, if it is relevant, the next sub-activity is "Specify Design Requirements", which includes the "Specify Research Questions" and "Specify Artifact Requirements" tasks. To perform them, the 'Information Synthesis' and 'Solution Requirements' artifacts are needed as input. As output, the 'Research Questions' and 'Artifact Requirements' are produced.

The right formulation of the RQs constitutes a paramount activity in any research study because they conduct the research and transmit its essence [24]. In our case, we have formulated three main RQs for the software testing ontology to be built:

RQ#1: What are the most appropriate requirements for the software testing ontology in order to support the testing strategies that help achieve test goals? From this, the following sub-RQs were derived:

RQ#1.1: Should the ontology to be developed be of low-domain or top-domain level?

RQ#1.2: What are the most robust and rich documented terminologies (structured as glossaries, taxonomies or ontologies) for the software testing domain?

RQ#1.3: What are the main terms that the software testing ontology should include to achieve good coverage?

RQ#1.4: Should the ontology to be developed be at the stage of conceptualization and/or implementation?

RQ#1.5: What is the suitable methodology for ontology development to be used?

RQ#2: How to integrate the testing ontology with the existing conceptual components (sub-ontologies) and levels of the FCD-OntoArch framework? From this, two sub-RQs follow:

RQ#2.1: With what FCD-OntoArch's sub-ontologies and levels should the new testing ontology be directly related?

RQ#2.2: What terms are necessary to relate and integrate the different sub-ontologies of FCD-OntoArch with the new software testing ontology?

RQ#3: How to assure the software testing ontology quality? From this, two sub-RQs follow:

RQ#3.1: What quality characteristics of the ontology should be evaluated?

RQ#3.2: What verification and validation strategies are best suited to assess the ontology?

After carrying out the "Specify Artifact Requirements" task using as input the above RQs we produced the following (software testing ontology) 'Artifact Requirements' (AR) document:

AR#1) (Related to RQ#1.1). Design and build a top-domain ontology.

AR#2) (Related to RQ#1.2). Consider mainly: i) international standard glossaries documented in ISO 29119-1 [12] and ISTQB [13]; and ii) the ROoST [21] domain ontology and the Asman *et al.* [1] top-domain ontology, which were the two-best ranked among the 12-selected and evaluated ontologies in the conducted SLR [22] (see results at http://bit.ly/OntoQualityEval).

AR#3) (Related to RQ#1.3). The terminological coverage of the top-domain ontology should be the necessary and sufficient 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 definition (work process, activity and task), test work product (artifact, result), 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. For the top-domain software testing ontology, aspects related to its scope were specified in the form of Competency Questions (see the 25 resulting CQs at http://bit.ly/TestTDO-CQuestions).

AR#4) (Related to RQ#1.3). In order to favor good coverage, there must be a balance between the availability of taxonomic and non-taxonomic relationships.

AR#5) (Related to RQ#1.4). The top-domain software testing ontology must be built at the conceptualization stage without the need of implementing it in a formal language since the main goal is to use it as a common vocabulary to enrich process and method specifications for testing strategies to be developed.

AR#6) (Related to RQ#1.5). Use METHONTOLOGY [7] for the development of the ontology until its conceptualization stage. Also consider its evaluation and documentation activities.

AR#7) (Related to RQ#2.1). Link some terms of the software testing ontology with the FRsTDO and NFRsTDO sub-ontologies at the top-domain level (as highlighted in Fig. 1). Additionally, some terms should also be related to SituationCO and ProcessCO sub-ontologies at the core level. (Note that SituationCO needs terms from ProjectCO, GoalCO and ContextCO as well as from ThingFO at the foundational level).

AR#8) (Related to RQ#2.2). The necessary terms to relate and integrate the different sub-ontologies of FCD-OntoArch with the top-domain software testing ontology are

mainly: i) for NFRsTDO: Non-functional Requirement and Evaluable Entity terms; ii) for FRsTDO: Functional Requirement and Developable Entity terms; iii) for GoalCO: Business Goal and Information Need Goal terms; iv) for ProjectCO: Project, Strategy and Project Plan terms; v) for ContextCO: Context Entity term; and vi) for ProcessCO: Work Process, Activity, Work Product, Artifact, Outcome, Method, Role, Tool and Agent terms. Note that SituationCO uses some terms of context, project and goal components adding some others as Particular Situation and Target Entity.

AR#9) (Related to RQ#3.1). The quality characteristics to be evaluated are the same as those used in the SLR for the 12-selected software testing ontologies, namely: Ontological Structural Quality, Balanced Relationship Availability, Domain-specific Terminological Coverage Quality, and Compliance to other Vocabularies.

AR#10) (Related to RQ#3.2). The evaluation strategy to assess the resulting ontology is GOCAME (*Goal-Oriented Context-Aware Measurement and Evaluation*) [3, 17]. Besides, the CQs should be verified with a verification matrix that includes the resulting ontology terms, properties, relations and axioms, which correspond to each CQ. The validation of the resulting ontology should be made by applying their terms to a real-world test project, in addition to being examined with other testing domain experts to get feedback.

A2 Design and Develop the Solution: looking at Fig. 2 (green box), the first sub-activity to carry out is "Select Development Processes and Methods". It has as input the 'Artifact Requirements' and the KB. In our case, according to the AR#6, we select from the KB the METHONTOLOGY approach in order to design and build the software testing ontology.

The next activity to perform is "Design and Build a Version of the Artifact". It includes: "Design the Artifact" and "Build the Artifact" sub-activities. Both activities require the 'Selected Development Processes and Methods' document, as well as the KB. Also, the 'Artifact Requirements' are needed for the "Design the Artifact" activity. As output, the 'Artifact Design' is produced, which serves as input for the "Build the Artifact" activity. For the software testing ontology artifact, the design has been made following the steps that METHONTOLOGY proposes until the conceptualization step.

Then, the "Build the Artifact" is performed to produce a version of the artifact. If it is necessary to do a re-design, e.g. because it is necessary to consider more artifact requirements, then the process has to iterate again to the "Design the Artifact" activity as depicted in Fig. 2. It is worth mentioning that if the design activity is carried out again, the 'Artifact Design' must also be an input to be able to re-design it. But we did not explicitly do so in the process diagram to avoid clutter. Otherwise, the process continues with the A3 activity.

After several iterations in this design-build cycle, we obtained a final version of the artifact, i.e., the software testing ontology, which is named TestTDO. Its conceptualization is depicted in Fig. 4. In addition, the reader can access all definitions of terms, properties and relationships of TestTDO at http://bit.ly/TestTDO-Defs, and its 14 axioms' specifications at http://bit.ly/TestTDO-AxiomSpecs.

A3 Execute Verification and Validation (V&V): looking at Fig. 2 (orange box), the first sub-activity included in A3 is "Select V&V Processes and Methods". This requires as input the 'Artifact Requirements' and the KB in order to produce the 'Selected V&V Processes and Methods' document.

On one hand, to assess TestTDO from the ontological quality standpoint (see AR#10), we selected the GOCAME evaluation strategy [3, 17] and the NFRs tree documented at http://bit.ly/OntoQualityEval. On the other hand, in order to verify that TestTDO covers the established scope in the CQs, we design a verification matrix (see Table 1). This matrix includes the resulting ontology terms, properties, relationships and axioms, which correspond to each CQ. We also decided to verify the integration of TestTDO into the FCD-OntoArch architecture. Finally, we selected validation strategies for our ontology, namely: i) applying their terms to an academic test project; and ii) getting feedback from experts in the testing domain.

Once the V&V strategies were selected, the next activity to carry out is "Perform V&V". This activity requires as input the artifact, its requirements and design, as well as the selected V&V processes and/or methods. It is important to keep in mind that both the artifact and its design can be verified against the requirements.

After performing the V&V, the 'V&V results' are produced. In our case, as above mentioned, we used the GOCAME strategy to obtain all the evaluation results (indicators' values) of TestTDO, found at http://bit.ly/TestTDO_OntoQualityEval. Additionally, we used a verification matrix to verify that TestTDO covers the established scope in the CQs.

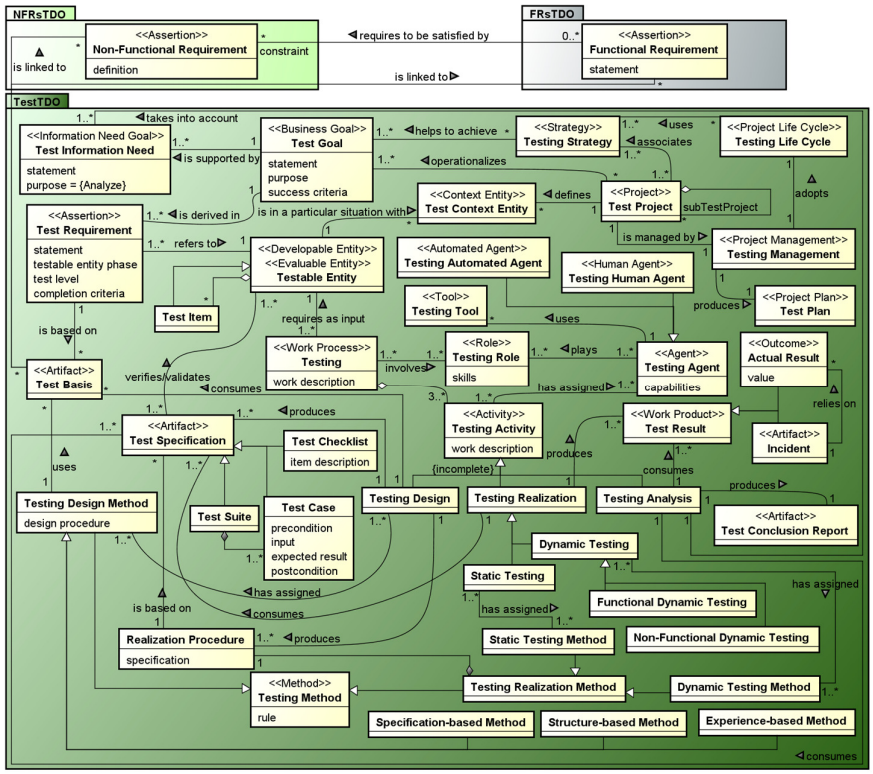

Fig. 4. Main terms, properties and relationships of the TestTDO ontology and its relation with Non-Functional Requirement and Functional Requirement terms

Table 1. Excerpt from the TestTDO's Verification Matrix. Note that at the bottom of this table the axioms are specified in first-order logic

CQ	Terms, relationships and *properties*	Axioms
CQ21	Test Goal is derived in Test Requirement	A9
CQ23	Test Requirement refers to Testable Entity Test Requirement has the property named *test level*	
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

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

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

A9) For all Test Requirements derived from a Test Goal, there is at least one Test Project that operationalizes this Test Goal.

$$\forall tr, \exists tg, tr: TestRequirement(tr) \wedge TestGoal(tg) \wedge isDerivedIn(tg, tr) \rightarrow TestProject(tp) \wedge operationalizes(tp, tg)$$

In Table 1, we illustrate the verification matrix for the CQs #21, #23 and #25. The complete verification matrix can be accessed at http://bit.ly/TestTDO-VerMatrix. In addition, we also verify that, for terms enriched by other higher-level ontologies of FCD-OntoArch, the semantics between the enriching term and the enriched term correspond appropriately. Just to give an example, looking at Fig. 4, the term Actual Result has semantic of outcome and, therefore, is enriched with the Outcome term, and not with the Artifact term, both terms coming from ProcessCO.

In order to validate if TestTDO was able to represent concrete situations of the world, we instantiated its terms, properties and relationship using a geometrical figure application for an academic project. This running testing example can be accessed at http://bit.ly/TestTDO-Val. We are aware that TestTDO should be implemented in projects that deal with more complex real-world situations. Note that the fifth 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.

The last activity of A3 is "Analyze V&V Results", which uses the 'V&V Results' and the RQs as input. As a result, this activity produces the "V&V Conclusion/Recommendation Report". At this point, if the artifact meets all the requirements, then the A4 activity is performed. But, if not, the "Design and Build a Version of the Artifact" sub-activity must be carried out again.

It is important to mention that all the artifacts shown in this work are the final versions yielded after several design-build-evaluation cycles. We are aware that at this point the A1 activity can be performed again to make some changes, e.g., in the RQs or in the artifact requirements. In fact, in the present case, we have done so to update the requirements. But this iteration flow is not shown in Fig. 2 to facilitate the readability of the process.

A4 Communicate Research: this last activity of the DSR process (pink box) includes only one sub-activity called "Document/Communicate the Results". It has as input most artifacts produced in the previous activities as depicted in Fig. 2. As output,

it produces technical reports, demonstrations or scientific articles that allow documenting the performed research for the designed, built and evaluated artifact. This should imply being a valuable digital library resource for the interested community. In our case, the produced ontology should be interesting for the software quality assurance community. Ultimately, if in fact the artifact produces a positive impact on the scientific community, this is, it is interesting and widely adopted due to its usefulness, relevance and innovation, therefore, all documentation must be stored in the KB. This added knowledge will be available to build new artifacts.

3 Related Work

Roel Wieringa is a quoted author when referring to design science and its application in empirical software engineering. He describes design science as the design and research of artifacts in a context. Wieringa recognizes the efforts of Hevner *et al.* [9] in describing a framework for design science and proposed a similar one but containing some main differences, such as the separation of the design and the investigation. Considering this, he described a design science framework, in which two kinds of problem-solving activities are distinguished: solving practical problems through the design of an artifact, and answering RQs by scientific research [28]. The design science framework addresses the issue of building artifacts with the aim of solving practical problems in an organizational environment, thus justifies the use of DSR for investigating the properties of these artifacts. The framework presented in [27] and extended later in [26] does not formally model a design science process that details the main activities to be carried out. Instead, it indicates the relation between the design and investigation, addressing the problem context of an artifact, including for this, the social context and the knowledge context. Therefore, we cannot conclude that there is a formal proposal for a DSR process model specification that is considered similar to the one addressed in the present work.

Other papers raise concerns about DSR process and describe some process approaches to be followed. In [16], authors present a DSR process structured into three main phases: 1) problem identification, 2) solution design, and 3) evaluation, which can interact with each other. This is a relevant work, so these phases were considered when modeling our DSR process proposal. The process suggested in [16] includes the problem identification and the literature revision for determining the problem relevance. Then it takes place the design of the artifact which will give a solution to the detected problem and its subsequent evaluation to ensure its quality. Finally, results must be communicated in order to obtain feedback from the interested community. Note that our proposal is close to that described in [16]. However, they did not use a process specification language (e.g., SPEM) for the DSR process model specification. In addition, they only consider the behavioral process perspective and, therefore, did not model the produced and consumed work products in each phase as we did.

In the Hevner *et al.*'s Information System Research Framework [9, 10], 7 guidelines are presented that contain prerequisites for effectively using DSR considering the roles and techniques that can be used in information system research. In summary, these guidelines argue that DSR must produce a viable artifact that constitutes an effective solution to a relevant business problem. To determine the viability, utility, quality and

efficiency, the artifact should be assessed through rigorous methods, e.g. testing, evaluation and simulation. However, as observed in the abovementioned related work, in [9, 10], a formal process model that specifies main activities and tasks to be performed is missing, as well as the produced/consumed work products by the DSR activities.

In [19, 20], the authors present a DSR process model that depicts the activities to be carried out to implement the research approach. In this way, some activities are intended to the problem identification for detecting its relevance. Then, the solution objectives are identified. The solution is an artifact to be designed and built. Finally, its effectiveness as a solution to the identified problem must be demonstrated. If it is a viable solution, the research results must be communicated to the concerned community. This proposal is also interesting and served as the basis to the construction of our DSR process. But like the other proposals, the presented process ([19, 20]) is not formally described using a process modeling language, nor does it consider different process modeling perspectives. Therefore, it does not take advantage of the benefits of process modeling, as discussed in the Introduction Section.

In [25], the authors describe a process model for applying the DSR approach with the main activities (which authors named "process steps"). As we did, they considered the behavioral and functional perspectives. Nevertheless, regarding the functional perspective, only the produced artifacts are represented while the inputs to the activities are missing.

Despite the efforts made for a common DSR process, there is still no thorough consensus on what the detailed activities are, their inputs and outputs, as well as the sequences between activities and tasks. Finally, other related work [8, 15] agrees to discuss the lack of a standardized process to implement the DSR approach.

4 Conclusions and Future Work

Computer Science/Information System adopted the DSR approach to the design and construction of artifacts. However, an analysis of the literature makes it possible to determine that the DSR process specification that establishes a flow of activities and tasks, and their corresponding inputs/outputs, is not yet formally modeled. Therefore, with the objective of contributing to this aspect, this work presents a process model specification for DSR considering the functional and behavioral modeling perspectives. The specifications were made using the SPEM language. The proposed DSR process model allows to systematically carry out the activities and tasks involved in the research approach. Having a modeled process helps to understand and communicate it to the interested community, making its application more consistent and repeatable in the execution of the activities and tasks included. On the other hand, another aim this research pursued was the illustration of the DSR process using the construction of a software testing ontology (TestTDO) as a practical case. We also show the different documents produced throughout the application of the DSR process followed.

It is worth remarking that we show a recommended flow for the DSR process since we are aware that in a process instantiation there might be some variation points, such as the parallelization of some tasks. In addition, the life cycle of a given DSR project

should organize activities and tasks considering not only the prescribed DSR process but also the appropriate allocation of resources such as methods and tools, among others, for achieving the proposed goal. Therefore, there are additional activities (to those specified in Fig. 2) in which project planning must also take into accounts such as documenting artifacts in all activities and control their changes and versions. Continuous documentation and version control of artifacts are key factors to ensure consistency, repeatability and auditability of any project.

The proposed DSR process model also provides a good baseline for understanding the details and discussing alternatives or customizations to this process. In fact, the rigor provided by process modeling, where several perspectives are combined (e.g. functional, informational, organizational and behavioral), but can also be independently detached, provides a greater richness of expressiveness in sequences and decision flows, while representing different levels of granularity in the work definitions, such as activity, sub-activity and task.

TestTDO has a primary interest in our research group and may also be welcome in the software testing community. In recent years, our research efforts focused on the design and construction of integrated strategies that allow establishing a specific course of action (in addition to providing appropriate methods and conceptual bases) to achieve evaluation goal purposes. From now on, having a software testing ontology as a built artifact guided by DSR will allow us to develop integrated testing strategies that help achieve test goal purposes. Hence, as a future work, we are planning to use the DSR approach to build a family of integrated testing strategies. This future work will allow us to continue validating the specification for the presented DSR process model as well.

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. Asman A., Srikanth R. M.: A Top Domain Ontology for Software Testing, Master Thesis, Jönköping University, Sweden, pp. 1-74 (2016)
2. Becker P., Olsina L., Peppino D., Tebes G.: Specifying the Process Model for Systematic Reviews: An Augmented Proposal, Journal of Software Engineering Research and Development (JSERD), Vol 7, pp. 1-23, (2019) https://doi.org/10.5753/jserd.2019.460
3. 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)
4. 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)
5. Curtis, B., Kellner, M., Over, J.: Process Modelling, Com. of ACM, 35:(9), pp.75-90 (1992)
6. D'Aquin M., Gangemi A.: Is there beauty in ontologies? Applied Ontology, 6:(3), pp. 165-175 (2011)
7. Fernández-López M., Gómez-Pérez A., Juristo N.: METHONTOLOGY: From Ontological Art Towards Ontological Engineering, Spring Symposium on Ontological Engineering of AAAI, pp. 33–40, Stanford University, California (1997)
8. Geerts G. L.: A design science research methodology and its application to accounting information systems research, International Journal of Accounting Information Systems,

12:(2), pp. 142-151 (2011)
9. 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)
10. Hevner A. R.: A Three Cycle View of Design Science Research, Scandinavian Journal of Information Systems, 19:(2), pp. 87-92 (2007)
11. Hevner A.R., Chatterjee S.: Design Research in Information Systems: Theory and Practice, Springer Publishing Company, Incorporated, 1st Edition (2010)
12. ISO/IEC/IEEE 29119: Software and systems engineering - Software Testing. Parts 1, 2, 3 and 4 (2013)
13. ISTQB. International Software Testing Qualifications Board, Standard Glossary of Terms used in Software Testing, Version 3.2 (2019). Available at https://www.istqb.org/.
14. March S., Smith G.: Design and Natural Science Research on Information Technology, 15:(4), pp. 251-266 (1995)
15. Mckay J., Marshall P.: A Review of Design Science in Information Systems, ACIS 2005 Proceedings - 16th Australasian Conference on Information Systems (2005)
16. Offermann P., Levina O., Schönherr M., Bub U.: Outline of a design science research process, Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, Philadelphia, Pennsylvania, pp 1-11 (2009)
17. Olsina L., Becker P.: Family of Strategies for different Evaluation Purposes, XX Conferencia Iberoamericana en Software Engineering (CIbSE'17), CABA, Argentina, Published by Curran Associates, pp. 221-234 (2017)
18. OMG. Software & Systems Process Engineering Meta-Model (SPEM) Specification, Version 2.0 (2008)
19. Peffers K., Tuunanen T., Gengler C., Rossi, M., Hui W., Virtanen V., Bragge J.: The design science research process: A model for producing and presenting information systems research, 1st International Conference on Design Science Research in Information Systems and Technology, DESRIST, pp. 83-106 (2006)
20. Peffers K., Tuunanen T., Rothenberger M., Chatterjee S.: A design science research methodology for information systems research, Journal of Management Information Systems, 24:(3), pp. 45-77 (2007)
21. Souza E. F., Falbo R. A., Vijaykumar N. L.: ROoST: Reference Ontology on Software Testing, Applied Ontology Journal, 12:(1), pp. 1-30 (2017)
22. Tebes G., Peppino D., Becker P., Matturro G., Solari M., Olsina L.: A Systematic Review on Software Testing Ontologies, 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)
23. Tebes G., Peppino D., Rivera B., Becker P., Papa F., Olsina L.: Especificación del Proceso de Design Science Research: Caso Aplicado a una Ontología de Testing de Software, 7mo Congreso Nacional de Ingeniería Informática – Sistemas de Información (CoNaIISI'19), pp. 1-10 (2019)
24. Thuan N. H., Drechsler A., Antunes P.: Construction of Design Science Research Questions, Communications of the Association for Information Systems (2019)
25. Vaishnavi V., Kuechler B., Stacie P.: Design Research in Information Systems, Association for Inf. Sys. (2004). Available at http://desrist.org/design-research-in-information-systems
26. Wieringa, R. J.: Design science methodology for information systems and software engineering. London: Springer, https://doi.org/10.1007/978-3-662-43839-8 (2014)
27. Wieringa, R. J.: Relevance and problem choice in design science, in Global Perspectives on Design Science Research (DESRIST), 5th International Conference, St. Gallen, Lecture Notes in Computer Science, vol. 6105 (Springer, Heidelberg), pp. 61-76 (2010)
28. Wieringa, R.J.: Design science as nested problem solving, 4th International Conference on Design Science Research in Information Systems and Technology, ACM, New York, pp. 1-12 (2009)