

Alma Mater Studiorum Università di Bologna
Archivio istituzionale della ricerca

SQuAP-Ont: An ontology of software quality relational factors from financial systems

This is the final peer-reviewed author's accepted manuscript (postprint) of the following publication:

Published Version:

Ciancarini P., Nuzzolese A.G., Presutti V., Russo D. (2020). SQuAP-Ont: An ontology of software quality relational factors from financial systems. SEMANTIC WEB, 11(6), 1007-1021 [10.3233/SW-200372].

Availability:

This version is available at: <https://hdl.handle.net/11585/797997> since: 2021-02-10

Published:

DOI: <http://doi.org/10.3233/SW-200372>

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (<https://cris.unibo.it/>).
When citing, please refer to the published version.

(Article begins on next page)

This is the final peer-reviewed accepted manuscript of:

Hitzler, P., et al. "SQuAP-Ont: An Ontology of Software Quality Relational Factors from Financial Systems." *Semantic Web*, vol. 11, no. 6, 2020, pp. 1007-1021.

The final published version is available online at: <https://dx.doi.org/10.3233/SW-200372>

Terms of use:

Some rights reserved. The terms and conditions for the reuse of this version of the manuscript are specified in the publishing policy. For all terms of use and more information see the publisher's website.

This item was downloaded from IRIS Università di Bologna (<https://cris.unibo.it/>)

When citing, please refer to the published version.

SQUAP-ONT: AN ONTOLOGY OF SOFTWARE QUALITY RELATIONAL FACTORS FROM FINANCIAL SYSTEMS

A PREPRINT

Paolo Ciancarini

University of Bologna, Italy and Innopolis University, Russia
paolo.ciancarini@unibo.it

Andrea Giovanni Nuzzolese

STLab, Institute of Cognitive Sciences and Technologies, National Research Council, Rome, Italy
andreagiovanni.nuzzolese@cnr.it

Valentina Presutti

STLab, Institute of Cognitive Sciences and Technologies, National Research Council, Rome, Italy
valentina.presutti@cnr.it

Daniel Russo

Lero - The Irish Software Research Center & School of Computer Science & Information Technology
University College Cork, Cork, Ireland
daniel.russo@lero.ie

September 5, 2019

ABSTRACT

Quality, architecture, and process are considered the keystones of software engineering. ISO defines them in three separate standards. However, their interaction has been scarcely studied, so far. The SQuAP model (Software Quality, Architecture, Process) describes twenty-eight main factors that impact on software quality in banking systems, and each factor is described as a relation among some characteristics from the three ISO standards. Hence, SQuAP makes such relations emerge rigorously, although informally. In this paper, we present SQuAP-Ont, an OWL ontology designed by following a well-established methodology based on the re-use of Ontology Design Patterns (i.e. ODPs). SQuAP-Ont formalises the relations emerging from SQuAP to represent and reason via Linked Data about software engineering in a three-dimensional model consisting of quality, architecture, and process ISO characteristics.

Keywords Ontologies, Ontology Design Patterns, Pattern-based Ontology Modelling, Software Engineering, Software Quality, Software Process, Software Architecture.

1 A three-dimensional view on software quality

Industrial standards are widely used in the software engineering practice: they are built on pre-existing literature and provide a common ground to scholars and practitioners to analyze, develop, and assess software systems. As far as software quality is concerned, the reference standard is the ISO/IEC 25010:2011 (ISO quality from now on), which defines the quality of software products and their usage (i.e., in-use quality). The ISO quality standard introduces eight *characteristics* that qualify a software product, and five characteristics that assess its quality in use. A characteristic is a parameter for measuring the quality of a software system-related aspect, e.g., reliability, usability, performance efficiency. The quantitative value associated with a characteristic is measured employing metrics that are dependent on the context of a specific software project and defined following the established literature.

The ISO quality standard only focuses on the resulting software product without explicitly accounting for the *process* that was followed or the implemented *architecture*. However, there is wide agreement [Pressman (2014)] about the importance of the impact of three combined dimensions: software quality, software development process, and software architecture, on the successful management and evolution of information systems. In this respect, the industrial standard ISO/IEC 12207:2008 defines a structure for the software process life cycle, and outlines the tasks required for developing and maintaining software [Singh (1996)]. Regardless of the chosen methodology (i.e., Agile or Waterfall ones [Pressman (2014)]), this standard identifies the relevant concepts of the life cycle and provides a useful tool for software developers to assess if they have undertaken all recommended actions or not. Each lifecycle concept can be evaluated according to its maturity level through established metrics, e.g., the Capability Maturity Model Integration (CMMI) [Team (2002)]. As for the architectural dimension, the ISO/IEC 42010:2011 standard provides a glossary for the relevant objects of software architecture. Concerning software architecture evaluation, intended as a way to achieve quality attributes (i.e., maintainability and reliability in a system), some approaches have emerged, the most prominent being ATAM, proposed by the Software Engineering Institute [Kazman et al. (1994); Clements et al. (2002); Bengtsson et al. (2004); Bellomo et al. (2015)]. Typical research in this domain is about how architectural patterns and guidelines impact software components and configurations [Garlan and Perry (1995)]. A survey study [Dobrica and Niemela (2002)] analyzes architectural patterns to identify potential risks, and to verify the quality requirements that have been addressed in the architectural design of a system.

The mutual relations among the three dimensions and their impact on the quality of software systems have been barely addressed in the literature, but a recent empirical study [Russo et al. (2017)] in the domain of Software Banking Systems pointed out the importance of those relations. The study involved 13 top managers of the IT Banking sector in the first phase and 124 additional domain experts in a second validation phase. The result of such a study was a model named SQuAP that describes these relations in terms of *quality factors*. According to [Gorla et al. (2010)], the information available to guide and support the management of software quality efforts is a critical success factor for IT domains. Considering the broad coverage of its empirical provenance, a formalization of SQuAP may serve as a reference resource and practical tool for scholars and practitioners of software engineering, to assess the quality of a software system, to drive its development to meet a certain quality level, as well as to teach software engineering.

The SQuAP model builds on the concept of *quality factor*: a n -ary relation between software quality characteristics that cover the three dimensions of software product, process, and architecture, based on the three reference standards ISO/IEC 25010:2011, ISO/IEC 12207:2008, and ISO/IEC 42010:2011 respectively. A SQuAP quality factor can be described as a complex quality characteristic (or *parameter*) that provides a three-dimensional view for assessing software quality. The model identifies twenty-eight quality factors.

Our contribution consists in a resource named SQuAP-Ont, an ontology that formally represents the concept of quality factor by reusing existing ontology design patterns (e.g., Description and Situation [Presutti and Gangemi (2016); Gangemi (2008)]), instantiates all factors identified so far, and axiomatizes them in order to infer measurable factors based on the characteristics available at hand. Besides, the ontology has been annotated with OPLa¹ (Ontology Design Pattern representation language) to increase its reusability. SQuAP-Ont is publicly available online² with accompanying documentation that describes the factors, under a CC-BY-4.0 license.

In the rest of the paper, after discussing relevant related work (Section 2), we provide additional details about the SQuAP model by presenting two sample factors in Section 3. We describe the SQuAP-Ont ontology: its main concepts and axioms, the adopted design methodology and the reused ontology design patterns in Section 4. In Section 5 we provide examples of how to use it; and discuss the resource potential impact (Section 6) before concluding and identifying future developments, in Section 7.

2 Related work

The use of ontologies in the software engineering domain is very common [Calero et al. (2006); Zhao et al. (2009); Kabaale et al. (2017)]. The ISO standards referenced in Section 1 have been the subject of several ontological studies. For example, useful guidelines for their ontological representation are proposed by Henderson *et al.* (2014) and Gonzalez *et al.* (2016) [Henderson-Sellers et al. (2014); Gonzalez-Perez et al. (2016)].

An ontology-based approach to express software processes at the conceptual level was proposed in Liao *et al.* (2015) [Liao et al. (2014)] and implemented in e.g., Soydan & Kokan (2006) [Soydan and Kokar (2006)] for CMMI [Chrissis et al. (2003)]. Software quality attributes have been modeled in Kaye *et al.* (2009) [Kaye et al. (2009)], while an ontology for representing software evaluation is proposed in Castro *et al.* (2010) [Castro et al. (2010)]. Finally, a formalisation of

¹<http://ontologydesignpatterns.org/opla/>

²<https://w3id.org/squap/>

the ISO 42010, describing software architecture elements, is developed in Emery & Hilliard (2009) [Emery and Hilliard (2009)] and in Kumar & Prabhakar (2010) [Kumar and Prabhakar (2010)]; and Antunes *et al.* (2013) [Antunes et al. (2013)] argues that different architecture domains can be integrated and analyzed through the use of ontologies.

Most of the works mentioned above focus on a strict representation of standards in terms of ontologies [Liao et al. (2014); Soydan and Kokar (2006); Chrissis et al. (2003); Emery and Hilliard (2009)]. Other scholars [Kayed et al. (2009); Castro et al. (2010)] provide only preliminary ontological solutions for modelling quality characteristics or software evaluation and, to the best of our knowledge, they overlook the reuse of ontology design patterns. In contrast, our work focuses on the relation between the different ISO standards (system quality, software development process, and software architecture) for supporting the assessment of software system quality, with the added value of following a rigorous pattern-based ontology design approach.

Also at a higher level, an ontology to harmonize different hierarchical process levels through multiple models such as CMMI, ISO 90003, ITIL, SWEBOK, COBIT was presented in Pardo *et al.* (2012) [Pardo et al. (2012)].

Ontologies referred to software quality focus primarily on quality attributes [Kayed et al. (2009)]. One quality evaluation, based on the ISO 25010 standard, is enhanced by taking into consideration several object-oriented metrics [Motogna et al. (2015)]. Similarly, Castro *et al.* (2010) reuse current quality standards and models to present an ontology for representing software evaluations [Castro et al. (2010)].

Similarly, scholars advanced an ontology for the ISO 42010 standard regarding software architecture [Emery and Hilliard (2009)]. An ontological representation of architectural elements has also been expanded by Kumar & Prabhakar (2010) [Kumar and Prabhakar (2010)]. With particular reference to the architecture rationale, some visualization and comparison techniques with semantic web technologies have been proposed in literature [López et al. (2009)]. Moreover, scholars showed that different architecture domains could be integrated and analyzed through the use of ontologies [Antunes et al. (2013)].

Finally, the Semantic Web community proposed also guidelines regarding the representation of ISO standards of software engineering with ontologies [Henderson-Sellers et al. (2014); Gonzalez-Perez et al. (2016)]. These two papers use a domain ontology, proposing the creation of a single underpinning abstract domain ontology, from existing ISO/IEC standards. According to the authors, the adoption of a single ontology will permit the re-engineering of existing International Standards as refinements from this domain ontology so that these variously focused standards may inter-operate.

3 Relational quality factors: the SQuAP Model

The motivation for developing SQuAP is based on the understanding, in the software engineering and information systems communities, that assessing software quality for contemporary information systems requires to take into consideration the relations among different dimensional perspectives, namely: software quality, process, and architecture [Pressman (2014)]. Accordingly, we conducted an empirical study in the banking sector [Russo et al. (2018)]. The financial and banking industry is particularly useful to explore information systems quality issues for several reasons. First, it is mission-critical, i.e., the failure of even one system would lead to unpredictable consequences. Thus, the top manager showed increasing concerns about the quality and sustainability of their information systems [Russo et al. (2017)]. Secondly, most institutions, such as those involved in the study, are multinational, sharing the same systems and the same concerns. Finally, this sector is highly connected and regulated centrally by banking authorities. So, many functionalities and requirements are well defined by such authorities, which means that the entire industry is using similar software products.

Therefore, to develop the SQuAP quality meta-model, we executed our research according to the following steps. In the first phase, based on the Delphi method [Dalkey et al. (1969)], we involved 13 top managers of this sector to express their most significant software quality concerns. The result was a set of distinct 28 quality factors emerging from the elicited concerns, after a consensus-based negotiation phase (part of the Delphi method). In a second phase, we involved 124 domain experts that validated the 28 factors with a high level of agreement. Each factor has been then linked to several characteristics or elements defined in the three different standards, i.e., ISO 25010, ISO 42010, and ISO 12207, for software quality, architecture, and process respectively. We followed the theoretical coding approach by Strauss & Corbin (1997) [Strauss and Corbin (1997)] to map the factors to the ISO standards³.

The selection criteria and demographics of the experts involved in both phases, the inclusion and exclusion criteria, the agreement figures as well as the industry coverage are explained in Russo *et al.* (2018) [Russo et al. (2018)]. In

³Standards are *de facto* second-order theories, built on grounded pre-existing ones and shared among scholar's and practitioner's communities.

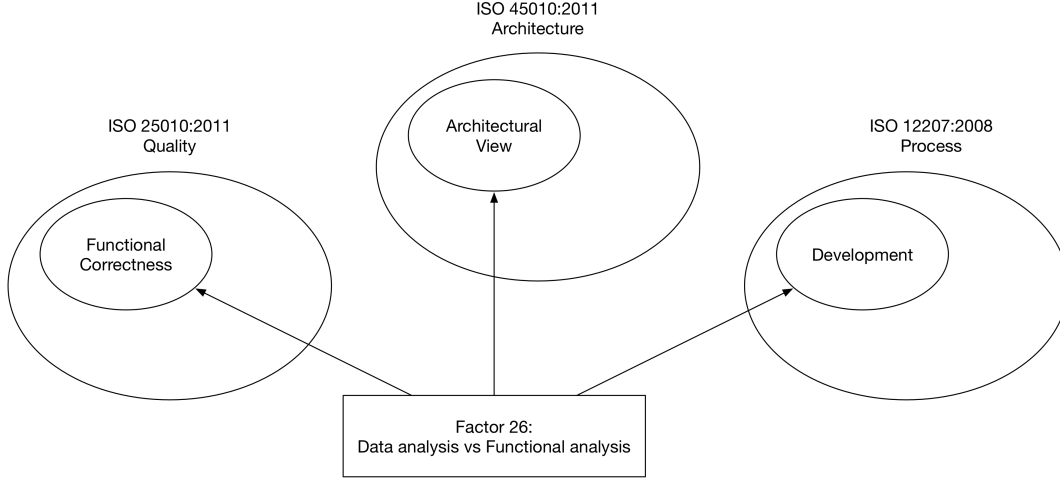


Figure 1: Factor 26: Data analysis vs. Functional analysis. This factor is defined as a relation between three quality characteristics of a software project: Functional Correctness (ISO 25010), Architectural View (ISO 45010), and Development (ISO 12207).

the same paper, we provide further details about the methodological approach and an exhaustive description of the 28 factors. A list of them are also published on the resource website⁴ along with a short textual description, and by tables depicting their mappings to ISO standards.

The 28 SQuAP factors have been rigorous, although informally defined in our previous work [Russo et al. \(2018\)](#). These definitions are the primary input for the development of SQuAP-Ont; hence, it is relevant to report here at least one them. The definition of a factor consists of a set of quotes from the experts involved in the study followed by an analysis (based on theoretical coding) on what are the main characteristics and elements from the standards that emerged as components of the factor. We choose randomly one factor (26) to explain the underlying logic of the factor mapping.

Factor 26: Data analysis vs Functional analysis. This factor explores whenever poor data analysis influences functional analysis and so, system integrity. *“The “functional centric” view is quite misleading since it relegates the importance of data. Data give the “static view” of existing functionalities, which is very important for the functional analysis. One software product may have all possible functionalities required by the user but lacking fundamental data its deployment and system integration becomes impossible”*, said one surveyed expert. Moreover, *“data analysis skills are generally lacking and poorly used in functional analysis”*, affirmed another one. This issue is present market-wide. *“In my opinion, in the market, there is a lacking perception about the importance of data analysis as the preliminary phase of functional analysis”*. However, other experts disagree. *“Saying that poor analysis is due to poor data understanding is a quite generic (it is obvious that data processing is the main IT goal) and old issue”*. Other issues are also relevant to understand the factor. *“Also knowing what different data means is important”*. Also, *“it is not only an issue of poor technical skills”*. Furthermore, *“personally, I saw poorer knowledge of bank’s operation processes”*. There was a shift after 2010, which give fascinating insights. *“It is true for applications developed before 2010. Data governance is now more relevant, and data analysis is performed before the functional one. So I see a clear discontinuity with the past”*.

Theoretical coding analysis: experts stressed the importance of data and functional analysis, impacting on the dimensions of **Functional Correctness** (software quality), **Development** (software process), and **Architecture View** (software architecture). They refer to the functional suitability of applications through correctness. This impacts the development process, which is supported by such an analysis. The architecture view addresses the concern of a suitable system held by the system’s stakeholders. Figure 1 shows a graphical representation of Factor 26.

⁴<https://w3id.org/squap/documentation/factors.html>

4 SQuAP-Ont: an OWL formalisation of SQuAP

In this section we first provide details about the ontology design methodology adopted (cf. Section 4.1), then we describe the SQuAP Ontology (cf. Section 4.3) and we provide its formalisation (cf. Section 4.3). Finally, we provide the implementation details of the ontology (cf. Section 4.4).

4.1 Design methodology

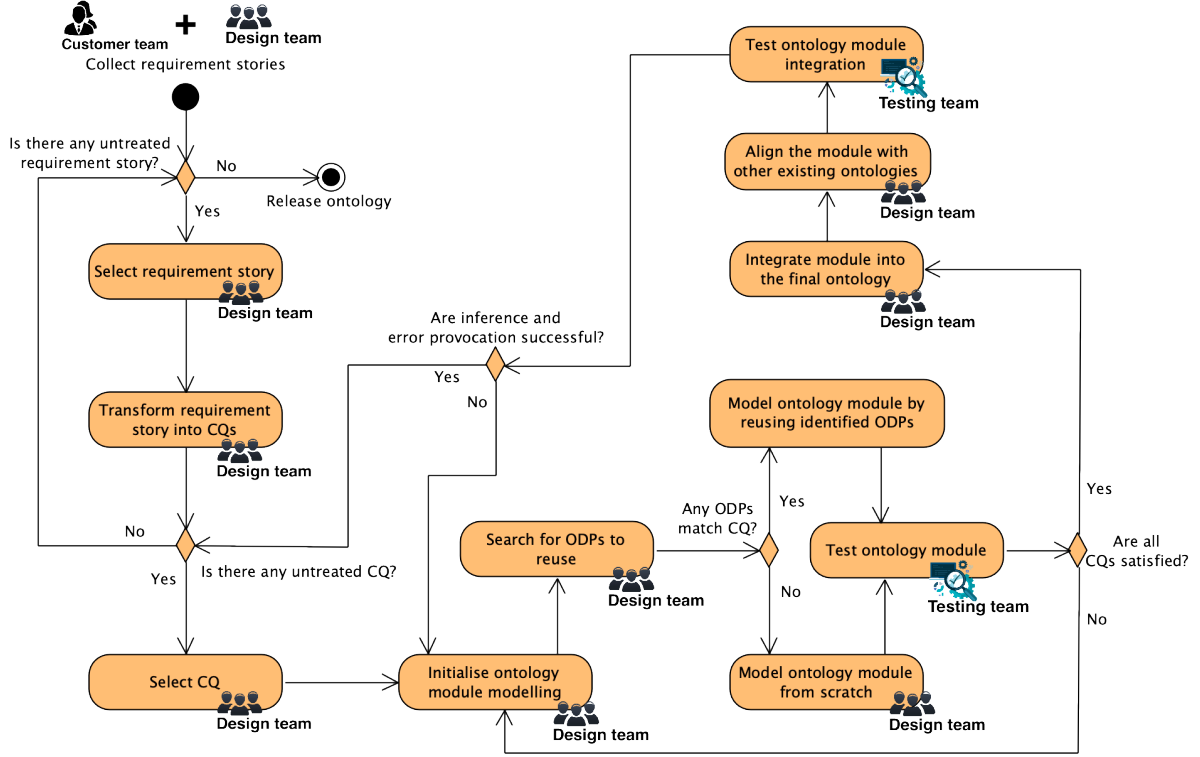


Figure 2: The XD methodology as implemented for modelling the SQuAP ontology.

The SQuAP Ontology (SQuAP-Ont) is designed by reusing ontology design patterns (ODPs) [Gangemi and Presutti \(2009\)](#) according to an extension of the eXtreme Design methodology [Blomqvist et al. \(2010\)](#). We opt for an ODP-based methodology as there are shreds of evidence [Blomqvist et al. \(2010\)](#) that the reuse of ODPs (i) speeds up the ontology design process, (ii) eases design choices, (iii) produces more effective results in terms of ontology quality, and (iv) boosts interoperability. This extension has been first described in [Presutti et al. \(2016\)](#) and mainly focuses on providing ontology engineers with clear strategies for ontology reuse. According to the guidelines provided by [Presutti et al. \(2016\)](#), we adopt the *indirect re-use*: ODPs are reused as templates. Hence, ODPs are specialized by specific ontology modules developed as components of the final ontologies instead of being directly reused and linked by those modules. For example, if we want to reuse the object property `region:hasParameter` from the Region ODP, then we define the object property `:hasParameter` into the target ontology and we formally annotate such indirect reuse employing the OPLa ontology [Hitzler et al. \(2017\)](#). Nevertheless, the ontology guarantees interoperability by keeping the appropriate alignments with the external ODPs and provides extensions that satisfy more specific requirements.

The XD extension implemented in this work has been successfully adopted in several ontologies and linked open data projects so far. Examples are [Nuzzolese et al. \(2016\)](#); [Peroni et al. \(2016\)](#); [Lodi et al. \(2017\)](#); [Gangemi et al. \(2017\)](#); [Carriero et al. \(2019\)](#). This extension implements an iterative and incremental approach to ontology design that involves three different actors: (i) a *design team*, in charge of selecting and implementing suitable ODPs as well as to perform alignments and linking; (ii) a *testing team*, disjoint from the design team, which takes care of testing the ontology; (iii) a *customer team*, who elicits the requirements that are translated by the design team and testing team into ontological commitments (i.e., competency questions and other constraints) that guide the ontology development. Figure 2 depicts the UML activity diagram that represents the XD extension described in [Presutti et al. \(2016\)](#) as implemented in this work. The diagram is extended by explicitly identifying the actors that are associated

with each action they are involved in. The first action of the methodology is the collection of requirements that involves both the customer and the design team. At this stage, the requirements are recorded as *stories*, which are typically used in agile software development for communicating requirements from the customer to the development team. After requirement stories are recorded, the design team starts to transform them into *competency questions* Gr ninger and Fox (1995) (CQs). CQs represent the ontological commitments that drive the ontology development. Table 1 reports the CQs, identified by analysing the SQuAP model (cf. Section 3) and by discussing with domain experts (i.e., the customer team in our context).

Table 1: Competency questions used for modelling SQuAP-Ont.

ID	Competency question
CQ1	What are the quality characteristics of a software system at software, process, and architectural level?
CQ2	What are the factors, the assessment of which, is affected by a certain quality characteristic?
CQ3	What are the quality characteristics that affect the assessment of a certain factor?
CQ4	What is the unit of measure (i.e., metric) associated with a certain quality characteristic?
CQ5	What is the value computed for assessing a certain quality characteristic?

The next action is about the design team looking for possible ODPs to reuse by analysing the resulting CQs. Those ODPs, if available, are reused for designing the modules of the ontology that addresses the specific CQs. When an ontology module is ready, the testing team performs the validation by assessing its fulfilment to the CQs. This validation is performed by (i) converting the CQs to SPARQL queries and (ii) executing those queries on a data sample, which is modelled according to the target ontology module. If the validation is successful, the design team integrates the ontology module in the final ontology. Additionally, the design team provides alignments with related external ontologies and vocabularies in the Semantic Web for boosting interoperability. Then, the testing team performs a set of integration tests aimed at (i) validating the logical consistency of the ontology and (ii) its ability to detect errors by deliberately introducing contradictory facts. Both checks can be performed by using a DL reasoner, such as Hermit⁵, Pellet⁶ etc. We remark that the testing based on the (i) validation of the CQs, (ii) the check of the logical consistency, and (iii) the error provocation follows the methodology designed by Blomqvist *et al.* (2012) Blomqvist et al. (2012). If the integration tests succeed, then the design team performs another iteration of the process by selecting an untreated CQ. If no untreated CQ is available, then the iteration consists of the selection of an untreated requirement story. Finally, after several iterations and when no untreated requirement story is available, the process ends. Accordingly, the ontology is released.

4.2 Ontology description

Figure 3 shows a diagram of SQuAP-Ont. We use the namespace <https://w3id.org/squap/>. SQuAP-Ont reuses as templates the following ontology design patterns Presutti and Gangemi (2016): Description and Situation (D&S)⁷ Gangemi and Mika (2003), and Parameter Region⁸.

The D&S pattern allows representing the conceptualisation of a n -ary relation (i.e., description) and its occurrences (i.e., situations) in the same domain of discourse. For example, it is used for representing the description of a plan (D) and its actual executions (S), the model of disease (D, e.g., its symptoms) and the actual occurrence of it in a patient (S), etc. SQuAP-Ont reuses this pattern for modelling quality factors with the class `:SoftwareQualityFactor`, a subclass of `:Description`. The actual occurrences of quality factors assessed in a specific software project are modelled with the class `:FactorOccurrence`, a subclass of `:Situation`. Both `:Description` and `:Situation` are core elements of the D&S pattern. According to the D&S pattern a `:Description` defines a set of `:Concepts`. In the context of SQuAP-Ont we say that a `:SoftwareQualityFactor` uses a set of `:SoftwareQualityCharacteristic`. This relation is modelled by the property `:usesQualityCharacteristic`. We model three types of `:SoftwareQualityCharacteristic`: `:SoftwareQuality`, `:ArchitecturalAlignment`, and `:ProcessMaturity`. They classify the characteristics associated with the three different ISO standards and their own perspectives, i.e., software quality, architecture, and process. In a similar way, a set of entities are in the setting provided by a `:Situation`. In the context of SQuAP-Ont we say that a set of `:MeasurementResult` affects the assessment of a `:FactorOccurrence`. We model three types of `:MeasurementResult` with the classes `:SoftwareQualityMeasurementResult`, `:ArchitecturalAlignmentResult`, and `:ProcessMaturityResult` which are instantiated with result measurements computed for assessing the quality characteristics of a specific software system. A `:Measure-`

⁵<http://www.hermit-reasoner.com/>

⁶<https://github.com/stardog-union/pellet>

⁷<http://ontologydesignpatterns.org/cp/owl/descriptionandsituation.owl>

⁸<http://ontologydesignpatterns.org/cp/owl/parameterregion.owl>

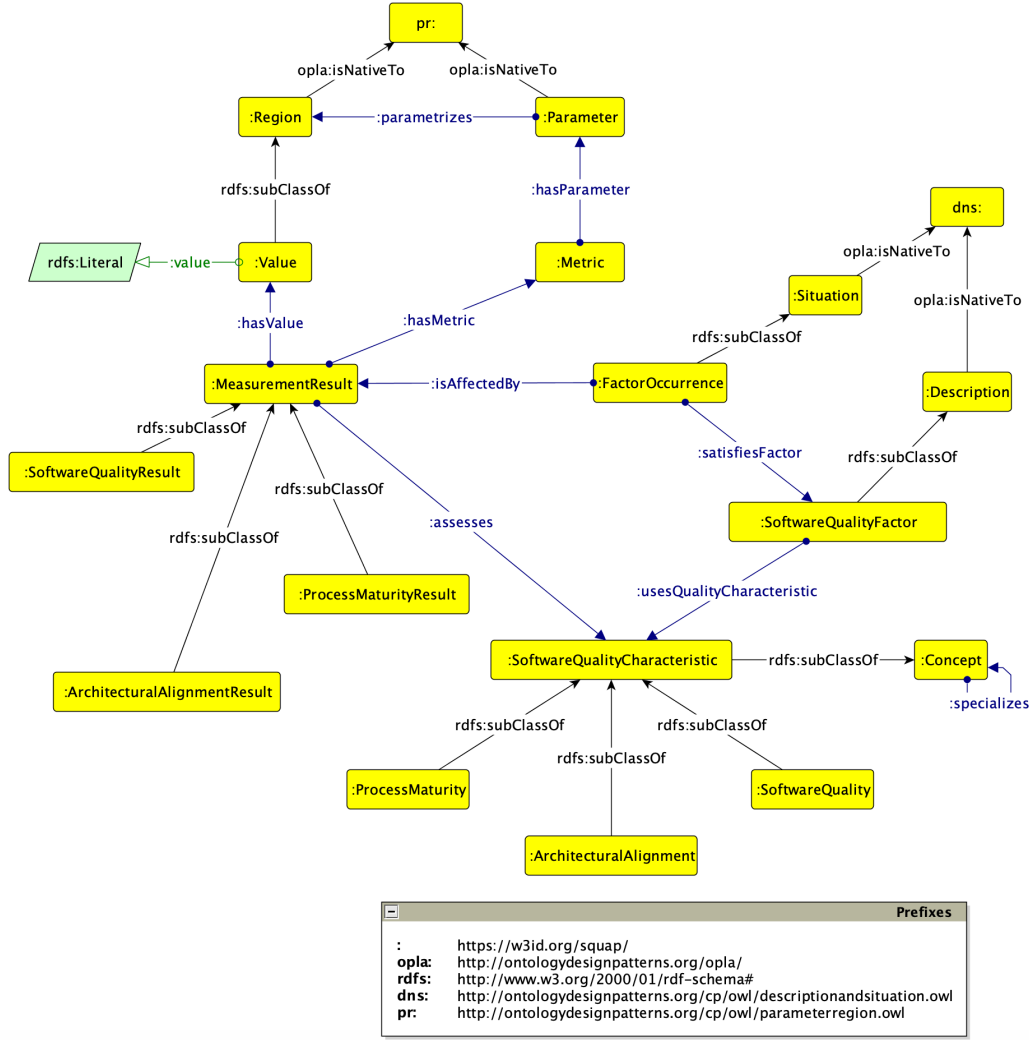


Figure 3: Core classes of SQuAP-Ont.

mentResult has a :Value and a reference :Metric. For example, we may want to represent that the *Reliability* of a software system is associated with a specific degree value according to a certain metric. This part of the model reuses the Parameter Region ontology design pattern as template.

In D&S each entity that is in the setting of a :Situation is classified by a :Concept. In the context of SQuAP-Ont we specialized this relation by saying that a :MeasurementResult assesses a :SoftwareQualityCharacteristic. Based on the :MeasurementResults that compose a :FactorOccurrence it may satisfy one or more :SoftwareQualityFactors (cf. modelled by the property :satisfiesFactor).

:SoftwareQualityFactors are represented in SQuAP both as individuals and classes, by exploiting OWL punning. Punning implements metaclassing in OWL and allows to interpret the same ontology entity either as a class or an instance of a metaclass depending on the syntactic context. This mechanism makes ontology modelling more similar to the way humans communicate knowledge by using natural language. Punning evoke verbal jokes (i.e., pun), which are typically used in natural language to emphasise a particular fact. When quality factors are interpreted as classes, then it is possible to introduce instances of those classes. Hence, it is possible to describe specific quality factors that occur for contextualising the quality of certain software. For example, it is possible to describe the quality resulting from the analysis of a specific software by introducing a specific individual of the factor factor:QualityVsRequirements represented as a class. On the contrary, when quality factors are interpreted as individuals, then it is possible to treat them as instances of the metaclass :SoftwareQualityFactor. Hence, it is possible to predicate them. In the latter case, we can use the SQuAP-Ont to model a knowledge graph that provides facts about the quality factors

from a general perspective. For example, we might state that `factor:QualityVsRequirements` `dul:associated-With` `factor:QualityVsTimeAndBudget`. Additionally, a clear benefit from modelling factors with punning is the possibility to use both DL axioms or rules (e.g., SPARQL CONSTRUCT) to infer new knowledge. All factors identified by the SQuAP model are instantiated in the ontology. SQuAP-Ont models the three types of `:SoftwareQualityCharacteristics` in a similar way: a set of individuals extracted from the SQuAP model, according to the three reference ISO standards (cf. Section 3), are included in the ontology. They are also modelled as classes to make the ontology extensible with possible specific axioms. Furthermore, `:SoftwareQualityCharacteristics` are organized hierarchically through the object property `:specializes`, which is declared as transitive. The use of both `rdfs:subClassOf` and `:specializes` allows to represent hierarchical relations among `:SoftwareQualityCharacteristics` both when they are interpreted as classes and when they are interpreted as individuals. The latter is beneficial for defining and reasoning on software quality characteristics organized taxonomically within a controlled vocabulary, similarly to the taxonomic relations among concepts in SKOS (i.e., `skos:borrower`).

As aforementioned, SQuAP-Ont is annotated with the OPLa ontology [Hitzler et al. \(2017\)](#) for explicitly indicating the reused patterns. We use the property `opla:reusesPatternAsTemplate` to link SQuAP-Ont to the two patterns we adopted as template, i.e., D&S and Parameter Region. Similarly, we use the property `opla:isNativeTo` to indicate that certain classes and properties of SQuAP-Ont are core elements of specific ontology patterns. These annotations enable the automatic identification of the patterns reused by SQuAP-Ont, e.g., with SPARQL queries, hence facilitating the correct reuse of the ontology.

Finally, SQuAP is aligned, using an external file, to DOLCE+DnS UltraLight⁹. Tables 2 and 3 report the alignments axioms between the classes and the properties of the two ontologies, respectively.

Table 2: Alignments between the classes of SQuAP-Ont and DOLCE UltraLight.

SQuAP class	Align. axiom	DOLCE class
<code>:Region</code>	<code>owl:equivalentClass</code>	<code>dul:Region</code>
<code>:Value</code>	<code>owl:subClassOf</code>	<code>dul:Amount</code>
<code>:Parameter</code>	<code>owl:equivalentClass</code>	<code>dul:Parameter</code>
<code>:Concept</code>	<code>owl:equivalentClass</code>	<code>dul:Concept</code>
<code>:Situation</code>	<code>owl:equivalentClass</code>	<code>dul:Situation</code>

Table 3: Alignments between the properties of SQuAP-Ont and DOLCE UltraLight.

SQuAP prop.	Align. axiom	DOLCE prop.
<code>:classifies</code>	<code>owl:equivalentProperty</code>	<code>dul:classifies</code>
<code>:isClassifiedBy</code>	<code>owl:equivalentProperty</code>	<code>dul:isClassifiedBy</code>
<code>:usesConcept</code>	<code>owl:equivalentProperty</code>	<code>dul:usesConcept</code>
<code>:isConceptUsedIn</code>	<code>owl:equivalentProperty</code>	<code>dul:isConceptUsedIn</code>
<code>:satisfies</code>	<code>owl:equivalentProperty</code>	<code>dul:satisfies</code>
<code>:isSatisfied</code>	<code>owl:equivalentProperty</code>	<code>dul:isSatisfied</code>
<code>:specializes</code>	<code>owl:equivalentProperty</code>	<code>dul:specializes</code>
<code>:isSpecializedBy</code>	<code>owl:equivalentProperty</code>	<code>dul:isSpecializedBy</code>
<code>:isSettingFor</code>	<code>owl:equivalentProperty</code>	<code>dul:isSettingFor</code>
<code>:value</code>	<code>owl:subPropertyOf</code>	<code>dul:hasRegionDataValue</code>

4.3 Formalisation

The following is the formalisation of SQuAP-Ont described in Section 4.3. The formalisation is expressed in Description Logics. For brevity, we use the terms `SwQualityChar` for `SoftwareQualityCharacteristic`, `ArchAlign` for `ArchitecturalAlignment`, `ProcMat` for `ProcessMaturity`, `SwQuality` for `SoftwareQuality`, `SwQualityFactor` for `SoftwareQualityFactor`, `MeasureRes` for `MeasurementResult`, `ProcMatRes` for `ProcessMaturityResult`, `SwQualityRes` for `SoftwareQualityResult`, and `MeasureQualityRes` for `MeasurementQualityResult`.

⁹<http://www.ontologydesignpatterns.org/ont/dul/DUL.owl>

Value \sqsubseteq Region	(1)
Value \sqsubseteq lvalue.Literal	(2)
Concept \neq Description	(3)
Concept \neq Description	(4)
SwQualityChar \sqsubseteq Concept	(5)
SwQualityChar \equiv ArchAlign \sqcup ProcMat \sqcup SwQuality	(6)
ArchAlign \sqsubseteq SwQualityChar	(7)
ArchAlign \neq ProcMat	(8)
ArchAlign \neq SwQuality	(9)
ProcMat \sqsubseteq SwQualityChar	(10)
ProcMat \neq ArchAlign	(11)
ProcMat \neq SwQuality	(12)
SwQuality \sqsubseteq SwQualityChar	(13)
SwQuality \neq ArchAlign	(14)
SwQuality \neq ProcMat	(15)
Description \neq Concept	(16)
Description \neq Situation	(17)
SwQualityFactor \sqsubseteq Description	(18)
SwQualityFactor \sqsubseteq \forall usesQualChar.SwQualityChar	(19)
SwQualityFactor \sqsubseteq \exists usesQualChar.SwQualityChar	(20)
MeasureRes \sqsubseteq \exists assess.SwQualityChar	(21)
MeasureRes \sqsubseteq lhasValue.Value	(22)
MeasureRes \sqsubseteq lhasMetric.Metric	(23)
ArcAlignmentRes \sqsubseteq MeasureRes	(24)
ProcMatRes \sqsubseteq MeasureRes	(25)
SwQualityRes \sqsubseteq MeasureRes	(26)
FactorOccurrence \sqsubseteq Situation	(27)
FactorOccurrence \sqsubseteq \exists isAffectedBy.MeasureRes	(28)
FactorOccurrence \sqsubseteq \exists satisfiesFactor.SwQualityFactor	(29)
usesConcept \circ specializes \sqsubseteq usesConcept	(30)

4.4 Implementation details

The namespace <https://w3id.org/squap/> identifies the ontology and enables permanent identifiers to be used for referring to concepts and properties of the ontology. We define individuals' URIs with the name of their types (e.g., ArchitecturalAlignment) preceding their IDs (e.g., ObjectiveCharacteristic). This convention is a common practice in many linked open data projects to define individuals' URIs. For example, `squap:ArchitecturalAlignmentObjectiveCharacteristic` is the URI associated with the individual `ObjectiveCharacteristic` typed as `ArchitecturalAlignment`. All the ontology entities modelled by using OWL punning follow such a convention as they can be interpreted as individuals (or classes) depending on the context. We setup a content negotiation mechanism that allows a client to request the ontology either (i) as HTML (e.g. when accessing the ontology via a browser) or (ii) as one of the possible serialisations allowed (i.e., RDF/XML, Turtle, N-triples).

The alignments with DOLCE+DnS UltraLight (DUL) are published in a separate OWL file¹⁰, which imports both SQuAP-Ont and DUL. This allows one to use either SQuAP-Ont alone or its version aligned with and dependent on DUL. The resource, including the core ontology, the alignments, and the usage examples, is under version control on the CERN Zenodo repository¹¹. SQuAP-Ont is published according to the Creative Commons Attribution 4.0 International (CC-BY-4.0) license¹² and it has been uploaded on Linked Open Vocabularies¹³ (LOV). The license information is included in the ontology by using the `dcterms:license` property.

¹⁰ <https://w3id.org/squap/squap-dul.owl>

¹¹ <http://doi.org/10.5281/zenodo.3361387>

¹² <https://creativecommons.org/licenses/by/4.0/>

¹³ <https://lov.linkeddata.es/dataset/lov/>

5 How to use SQuAP-Ont

Flexibility is among the most relevant characteristic of this ontology. Although the higher levels of this ontology, regarding the standard and the factors mapping, are fixed, its measurement model can be adapted to the most suited scenario. In particular, the proposed way to evaluate information systems characteristics is just an illustrative example, which can be adapted to for any assessment purposes.

As a usage example of the SQuAP ontology, we show a real-world example consisting of the evaluation of a banking application employing the Goal-Question-Metric (GQM) approach [Basili \(1992\)](#). GQM defines a measurement model on three levels, i.e., Conceptual level (Goal), Operational level (Question), Quantitative level (Metric). This method offers a hierarchical assessment framework, where goals are typically defined and stable in time, and metrics may be adapted according to new measurement advances. So, we stress the fact that this paper does not focus on the measurement model, preferably on the knowledge representation of SQuAP for assessment and benchmarking purposes. So, we provide the following synthetic RDF data about the assessment. The data are expressed as RDF serialised in [TURTLE¹⁴](#).

```
@prefix : <https://w3id.org/squap/examples/gqm/> .
@prefix arc:
  <https://w3id.org/squap/ArchitecturalAlignment/> .
@prefix sw:
  <https://w3id.org/squap/SoftwareQuality/> .
@prefix prc:
  <https://w3id.org/squap/ProcessMaturity/> .
@prefix squap: <https://w3id.org/squap/> .

:compatibility-result a
  squap:SoftwareQualityResult ;
  squap:assesses sw:Compatibility ;
  squap:hasMetric :sonarqube-sw-quality ;
  squap:hasValue :sonarqube-value-b .

:correspondencerresult
  a squap:ArchitecturalAlignmentResult ;
  squap:assesses arc:Correspondence ;
  squap:hasMetric :likert-scale-1-7 ;
  squap:hasValue :likert-value-7 .

:documentation-result
  a squap:ProcessMaturityResult ;
  squap:assesses prc:Documentation ;
  squap:hasMetric :likert-based-prc-maturity ;
  squap:hasValue :likert-value-6 .

:sonarqube-sw-quality a squap:Metric ;
  squap:hasParameter :sonarqube-params .

:sonarqube-params a squap:Parameter ;
  squap:parametrizes :sonarqube-value-a ,
    :sonarqube-value-b ,
    :sonarqube-value-c .

:likert-based-prc-maturity a squap:Metric ;
  squap:hasParameter :likert-scale-1-7 .

:likert-scale-1-7 a squap:Parameter ;
  squap:parametrizes
    :likert-value-1 , :likert-value-2 ,
    :likert-value-3 , :likert-value-4 ,
    :likert-value-5 , :likert-value-6 ,
    :likert-value-7 .

:sonarqube-value-b a squap:Value ;
  squap:value "B" .

:likert-value-7 a squap:Value ;
  squap:value 7 .

:likert-value-6 a squap:Value ;
  squap:value 6 .
```

The example describes a banking system associated with three assessments about the dimensions of software quality, architectural alignment, and process maturity. The specific measurement results are: `:compatibility-result`,

¹⁴The RDF is available at <https://w3id.org/squap/examples/gqm>

which assesses the characteristic `sw-quality:Compatibility` (software quality), `:correspondenceresult`, which assesses the characteristic `arc-alignment:Correspondence` (architectural alignment), and `:documentation-result`, which assesses the characteristic `prc-maturity:Documentation` (process maturity). Those measurement results are associated with a value (e.g., `:likert-value-7`, which identifies the value 7 of a Likert scale) and a metric (e.g., `:likert-scale-1-7`, which identifies a Likert scale ranging from 1 to 7). Each value is reported with a literal representation and is associated with a metric. It is possible to use the axioms defined in SQuAP-Ont in order to gather all the factors that can be enabled by the available measured quality characteristics (e.g., `sw-quality:Compatibility`). This can be done, for example, by executing a Protégé DL query, the result of which is shown in Figure 4.

In this example, different standards' items represent the Goals, which are measured with one or several (also concurrent) software quality metrics. To do so, we followed literature recommendations [Wagner et al. \(2012\)](#); [Campbell and Papapetrou \(2013\)](#). The result is the sum of different evaluators, which represent a measurement of the three standards.

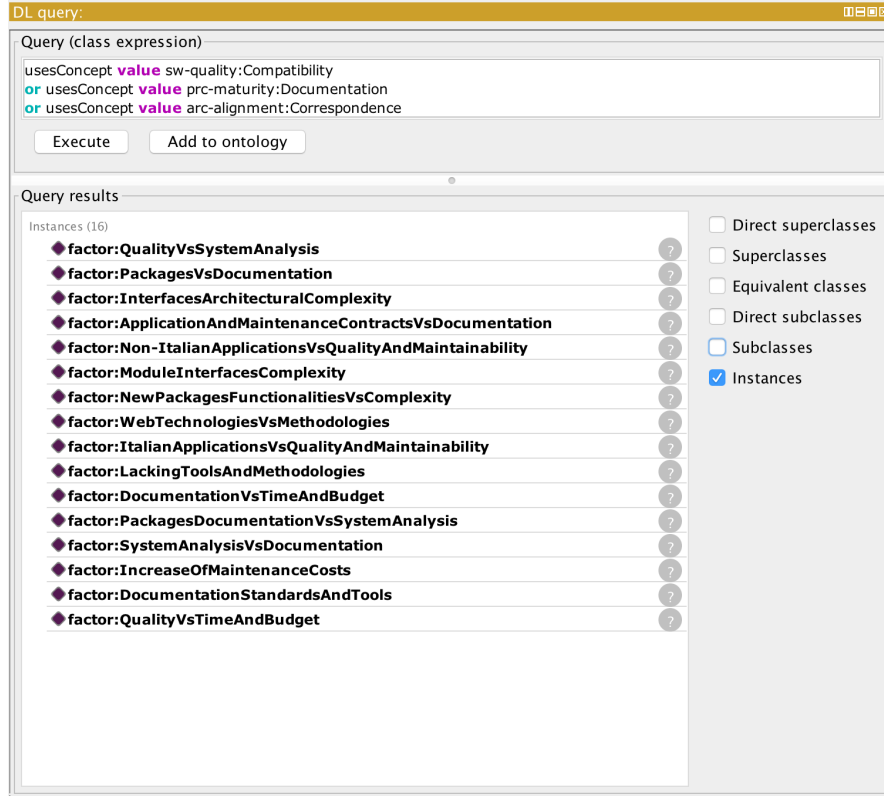


Figure 4: Execution of a DL query on the RDF sample.

Alternatively, it is possible to define productive rules to materialise the factors that are enabled by the available measured quality characteristics. The following SPARQL CONSTRUCT is a possible productive rule for our example.

```
PREFIX squap: <https://w3id.org/squap/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
CONSTRUCT {
  ?measurementResult
    squap:affectsMeasurementOf ?factorOccurrence .
  ?factorOccurrence
    a squap:FactorOccurrence;
    squap:satisfiesFactor ?factor
}
WHERE{
  ?factor
    squap::usesQualityCharacteristic ?char;
    rdfs:label ?factorLabel .
  ?measurementResult
    squap:assesses ?char
  BIND( IRI(
    CONCAT("https://w3id.org/squap/example/gqm/",
```

```

    ?factorLabel))
  AS ?factorOccurrence)
}

```

We remark that factors and quality characteristics are defined in SQuAP-Ont both as classes and individuals through OWL punning. Hence, one can decide to use DL reasoning or rules defined in any other formalism depending by the specific case, e.g., SPARQL CONSTRUCT, Shapes Constraint Language¹⁵ (SHACL), etc.

Another worth presenting example is a *dogfooding* usage scenario. Dogfooding is when an organization uses its product for demonstrating its quality. In this example, we use the SQuAP-Ont to record the metrics, the values, the factors, and the quality characteristics resulting from the measurement of its characteristics. The data of this example are expressed as RDF serialised in TURTLE.

```

@prefix :
  <https://w3id.org/squap/examples/dogfooding/> .
@prefix owl:
  <http://www.w3.org/2002/07/owl#> .
@prefix rdfs:
  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:
  <http://www.w3.org/2001/XMLSchema#> .
@prefix squap:
  <https://w3id.org/squap/> .
@prefix factor:
  <https://w3id.org/squap/Factor/> .
@prefix prc:
  <https://w3id.org/squap/ProcessMaturity/> .

:documentation-measurement-result
  a squap:MeasurementResult ;
  squap:hasMetric
    :protege-ontology-annotations-metric ;
  squap:hasValue
    :documentation-measurement-result-value .

:documentation-measurement-result-value
  a squap:Value ;
  squap:value "233"^^xsd:integer .

:protege-ontology-annotations-metric
  a squap:Metric ;
  squap:assess
    prc:Documentation

prc:Documentation
  a squap:ProcessMaturity .

factor:PackagesVsDocumentation
  a squap:Factor ;
  squap:usesConcept prc:Documentation .

:process-maturity-occurrence
  a squap:FactorOccurrence ;
  squap:isAffectedBy
    :documentation-measurement-result ;
  squap:satisfiesFactor
    factor:PackagesVsDocumentation .

```

In the dogfooding example the ontology is used for recording a measurement result (i.e., `:documentation-measurement-result`) for the SQuAP-ONT based on the Protégé ontology metric that records the number of annotations in an ontology (i.e., `:protege-ontology-annotations-metric`). The value associated with such a measurement result is "233", that is the number of annotations used for documenting the ontology. The aforementioned value can be easily obtained by using the ontology metrics view when opening the SQuAP-ONT with Protégé. The `:protege-ontology-annotations-metric` assesses the `:prc:Documentation`, which is a specific concept defined in SQuAP-ONT for characterising the process maturity (i.e., `squap:ProcessMaturity`) in terms of how much the software is documented. It is worth noticing that, by relying on OWL punning, we use `prc:Documentation` as an individual, though it is also defined as a class in the ontology. The concept of `prc:Documentation` is used by a specif factor, that is `factor:PackagesVsDocumentation`. Accordingly, we have a factor occurrence, that is `:process-maturity-occurrence`.

¹⁵<https://www.w3.org/TR/shacl/>

6 Potential impact

In the last decade, there has been a considerable effort, especially by the Management Information Systems research community, to study the phenomenon of the alignment of business and information systems [Aerts et al. (2004)]. What emerged is the importance of such alignment for both business' competitiveness and technical efficiency. When it comes to integrating new solutions, modules, or interfaces, such alignment is of crucial importance. Several other scholars found similar results, suggesting the importance of standard governance defining key architecture roles, involving critical stakeholders through liaison roles and direct communication, institutionalizing monitoring processes and centralizing IT critical decisions [Boh and Yellin (2006)]. Especially in the financial sector, architectural governance is a crucial issue for IT efficiency and flexibility [Schmidt and Buxmann (2011)]. Generally speaking, this finding is also primarily shared beyond the financial sector [Lange et al. (2016)]. The need for people from different backgrounds (mainly business and technical ones) to align the organization is the most considerable insight into this research stream.

To tackle the issue of information systems quality from an empirical perspective, we started in 2014 to survey banking application maintenance group experts, Chief Executive Officers, Chief Information Officers, IT architects, technical sales accounts, Chief Data Officers, and maintenance managers [Russo et al. (2017)]. This ongoing project is pursued with a leading consultancy firm, according to which we were able to cover with our representative sample the IT banking sector. Consequently, the need for knowledge representation of different measurement models is perceived as contingent and requested by the IT banking community in this significant project on information systems' quality. One crucial insight that emerged from the factors is the difficulty to assess their applications, also due to the diversity and complexity of measurement models. Indeed, the three standards measure three different dimensions. Quality measures the software as a product; Process as a process; and Architecture the alignment to a taxonomy. Accordingly, metrics and predictors reflect these differences. Therefore, the development of this ontology is a direct request from practitioners.

Since this research journey started from an industry's need, an ontology, intended as the knowledge representation of different measurement models is of pivotal importance, and a first tool to systematize the assessment of banking information systems' quality. Thus, this ontology will be used for consultancy purposes to implement the SQuAP quality model. Moreover, it is also useful to trace changes in quality in time and suggest specific improvements. So, this ontology is the knowledge layer over which this quality model is built. Consultancy firms expressed their interest in a knowledge representation tool which can be displayed to customers in the assessment phase, to tailor their consultancy efforts. However, also the bank's IT departments will use it for similar purposes. They can also tailor-made and modify this ontology and the underlying metrics suggested by the literature, according to their specific needs.

For this reason, we used a CC-BY-4.0 license, open to commercial use. Our industrial partners consider the use and reuse of this ontology as an excellent value for the practitioners' community.

7 Conclusion and future development

In this paper, we have described SQuAP-Ont, an ontology to assess information systems of the banking sector. SQuAP-Ont a) guides its users through the (ongoing) assessment phases suggested by software engineering literature; b) helps to identify critical quality flaws within applications; and c) extends and integrates existing work on software ISO ontology terms, diagram visualizations and ontology revisions. SQuAP-Ont has been developed for commercial use, within an industrial project on quality of banking information systems. Nevertheless, like all ontologies, it is an evolving effort, and we are open to suggestions proposed by the broad researchers' and practitioners' communities. We have addressed several issues raised in previous studies, and according to the industry's expectations.

Our future work aims to facilitate enrichment and refine the ontology continuously along with standards and literature recommendation changes. The enrichment is also about the introduction of specific annotations based on reference vocabularies for tracking provenance and versioning (e.g., PROV-O). Another important aspect is to validate and monitor the application of SQuAP in domains and software projects different from the banking system context. This may lead to the model's enrichment and improvement.

Acknowledgments

This work was partially funded by the Consorzio Interuniversitario Nazionale per l'Informatica (CINI) and the Science Foundation Ireland grant 15/SIRG/3293 and 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero—the Irish Software Research Centre. (www.lero.ie).

References

- A.T.M. Aerts, J.B.M. Goossenaerts, D.K. Hammer, J.C. Wortmann, Architectures in context: on the evolution of business, application software, and ict platform architectures. *Information & Management* **41**(6), 781–794 (2004). doi:10.1016/j.im.2003.06.002
- G. Antunes, A. Caetano, M. Bakhshandeh, R. Mayer, J. Borbinha, Using ontologies to integrate multiple enterprise architecture domains, in *Proceedings of the 16th International Conference on Business Information Systems*, ed. by W. Abramowicz Lecture Notes in Business Information Processing, vol. 160, Springer, 2013, pp. 61–72. Springer. doi:10.1007/978-3-642-41687-3_8
- V. Basili, Software modeling and measurement: the Goal Question Metric paradigm, Technical report, 1992
- S. Bellomo, I. Gorton, R. Kazman, Toward agile architecture: Insights from 15 years of ATAM data. *IEEE Software* **32**(5), 38–45 (2015). doi:10.1109/MS.2015.35
- P. Bengtsson, N. Lassing, J. Bosch, H. van Vliet, Architecture-level modifiability analysis (alma). *Journal of Systems and Software* **69**(1), 129–147 (2004). doi:10.1016/S0164-1212(03)00080-3
- E. Blomqvist, A. Seil Sepour, V. Presutti, Ontology Testing - Methodology and Tool, in *Proceedings of the 18th International Conference on Knowledge Engineering and Knowledge Management*, ed. by A. ten Teije, J. Völker, S. Handschuh, H. Stuckenschmidt, M. dAcquin, A. Nikolov, N. Aussenac-Gilles, N. Hernandez Lecture Notes in Computer Science, vol. 7603 (Springer, ???, 2012), pp. 216–226. doi:10.1007/978-3-642-33876-2_20
- E. Blomqvist, V. Presutti, E. Daga, A. Gangemi, Experimenting with eXtreme design, in *Proceedings of the 17th International Conference on Knowledge Engineering and Knowledge Management*, ed. by P. Cimiano, H.S. Pinto Lecture Notes in Computer Science, vol. 6317, Springer, 2010, pp. 120–134. Springer. doi:10.1007/978-3-642-16438-5_9
- W.F. Boh, D. Yellin, Using enterprise architecture standards in managing information technology. *Journal of Management Information Systems* **23**(3), 163–207 (2006). doi:10.2753/MIS0742-1222230307
- C. Calero, F. Ruiz, M. Piattini, *Ontologies for software engineering and software technology* (Springer, ???, 2006). ISBN: 9783540345183
- G. Campbell, P. Papapetrou, *SonarQube in action* (Manning, ???, 2013). ISBN 9781617290954
- V. Carriero, A. Gangemi, M. Mancinelli, L. Marinucci, A.G. Nuzzolese, V. Presutti, C. Veninata, ArCo: the Italian Cultural Heritage Knowledge Graph, in *Proceedings of the 18th International Semantic Web Conference 2019 (to appear)*. Lecture Notes in Computer Science (Springer, ???, 2019). ArXiv: 1905.02840
- R.G. Castro, M.E. Gutiérrez, M. Kerrigan, S. Grimm, An Ontology Model to Support the Automated Evaluation of Software, in *Proceedings of the 22nd International Conference on Software Engineering and Knowledge Engineering* (Knowledge Systems Institute Graduate School, ???, 2010)
- M. Chrissis, M. Konrad, S. Shrum, *CMMI guidelines for process integration and product improvement* (Addison-Wesley Longman, ???, 2003). ISBN 9780321711502
- P. Clements, R. Kazman, M. Klein, *Evaluating Software Architectures* (Addison-Wesley Professional, ???, 2002). ISBN 9780201704822
- N.C. Dalkey, B.B. Brown, S. Cochran, *The Delphi method: An experimental study of group opinion*, vol. 3 (Rand Corporation Santa Monica, CA, ???, 1969)
- L. Dobrica, E. Niemela, A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering* **28**(7), 638–653 (2002). doi:10.1109/TSE.2002.1019479
- D. Emery, R. Hilliard, Every architecture description needs a framework: Expressing architecture frameworks using ISO/IEC 42010, in *Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture*, IEEE, 2009, pp. 31–40. IEEE. doi:10.1109/WICSA.2009.5290789
- A. Gangemi, P. Mika, Understanding the semantic web through descriptions and situations, in *Proceedings of the OTM Confederated International Conferences “On the Move to Meaningful Internet Systems”*, ed. by R. Meersman, Z. Tari, D.C. Schmidt Lecture Notes in Computer Science, vol. 2888, Springer, 2003, pp. 689–706. Springer. doi:10.1007/978-3-540-39964-3_44

- A. Gangemi, V. Presutti, Ontology design patterns, in *Handbook on Ontologies*, ed. by S. Staab, R. Studer International Handbooks on Information Systems (Springer, ???, 2009), pp. 221–243. Chap. Ontology Engineering. doi:10.1007/978-3-540-39964-3_44
- A. Gangemi, R. Lillo, G. Lodi, A.G. Nuzzolese, V. Presutti, A Pattern-based Ontology for the Internet of Things., in *Proceedings of the 8th Workshop on Ontology Design and Patterns of the 16th International Semantic Web Conference*, ed. by E. Blomqvist, O. Corcho, M. Horridge, D. Carral, R. Hoekstra CEUR Workshop Proceedings, vol. 2043 (CEUR-WS.org, ???, 2017)
- A. Gangemi, Norms and plans as unification criteria for social collectives. *Autonomous Agents and Multi-Agent Systems* **17**(1), 70–112 (2008). doi:10.1007/s10458-008-9038-9
- D. Garlan, D.E. Perry, Introduction to the special issue on software architecture. *IEEE Transaction on Software Engineering* **21**(4), 269–274 (1995). doi:10.1109/TSE.1995.10003
- C. Gonzalez-Perez, B. Henderson-Sellers, T. McBride, G.C. Low, X. Larrucea, An Ontology for ISO software engineering standards: 2) Proof of concept and application. *Computer Standards & Interfaces* **48**, 112–123 (2016). doi:10.1016/j.csi.2016.04.007
- N. Gorla, T. Somers, B. Wong, Organizational impact of system quality, information quality, and service quality. *The Journal of Strategic Information Systems* **19**(3), 207–228 (2010). doi:10.1016/j.jsis.2010.05.001
- M. Grüninger, M. Fox, The role of competency questions in enterprise engineering, in *Benchmarking—Theory and practice*, ed. by A. Rolstadås (Springer, ???, 1995), pp. 22–31. doi:10.1007/978-0-387-34847-6_3
- B. Henderson-Sellers, C. Gonzalez-Perez, T. McBride, G. Low, An ontology for iso software engineering standards. 1) creating the infrastructure. *Computer Standards & Interfaces* **36**(3), 563–576 (2014). doi:10.1016/j.csi.2013.11.001
- P. Hitzler, A. Gangemi, K. Janowicz, A.A. Krisnadhi, V. Presutti, Towards a simple but useful ontology design pattern representation language, in *Proceedings of the 8th Workshop on Ontology Design and Patterns of the 16th International Semantic Web Conference*, ed. by E. Blomqvist, O. Corcho, M. Horridge, D. Carral, R. Hoekstra CEUR Workshop Proceedings, vol. 2043 (CEUR-WS.org, ???, 2017)
- E. Kabaale, L. Wen, Z. Wang, T. Rout, An Axiom Based Metamodel for Software Process Formalisation: An Ontology Approach, in *Proceedings of the 17th International Conference on Software Process Improvement and Capability Determination*, ed. by A. Mas, A. Mesquida, R.V. O’Connor, T. Rout, A. Dorling Communications in Computer and Information Science, vol. 770, Springer, 2017, pp. 226–240. Springer. doi:10.1007/978-3-319-67383-7_17
- A. Kayed, N. Hirzalla, A.A. Samhan, M. Alfayoumi, Towards an ontology for software product quality attributes, in *Proceedings of the 4th International Conference on Internet and Web Applications and Services*, ed. by M. Perry, H. Sasaki, M. Ehmann, G. Ortiz, O. Dini, IEEE, 2009, pp. 200–204. IEEE. doi:10.1109/ICIW.2009.36
- R. Kazman, L. Bass, G. Abowd, M. Webb, SAAM: A method for analyzing the properties of software architectures, in *Proceeding of the 16th International Conference on Software Engineering*, ed. by B. Fadini, L. Osterweil, A. van Lamsweerde, IEEE, 1994, pp. 81–90. IEEE. doi:10.1109/ICSE.1994.296768
- K. Kumar, T.V. Prabhakar, Pattern-oriented knowledge model for architecture design, in *Proceedings of the 17th Conference on Pattern Languages of Programs*, ed. by C. Kohls, ACM, 2010, pp. 23–12321. ACM. doi:10.1145/2493288.2493311
- M. Lange, J. Mendling, J. Recker, An empirical analysis of the factors and measures of enterprise architecture management success. *European Journal of Information Systems* **25**(5), 411–431 (2016). doi:10.1057/ejis.2014.39
- L. Liao, Y. Qu, H. Leung, A software process ontology and its application, in *Semantic Web Enabled Software Engineering*, ed. by J.Z. Pan, Y. Zhao Studies on the Semantic Web, vol. 17 (IOS-Press, ???, 2014), pp. 207–217. doi:10.3233/978-1-61499-370-4-207
- G. Lodi, L. Asprino, A.G. Nuzzolese, V. Presutti, A. Gangemi, D.R. Recupero, C. Veninata, A. Orsini, Semantic web for cultural heritage valorisation, in *Data Analytics in Digital Humanities*, ed. by S. Hai-Jew (Springer, ???, 2017), pp. 3–37. doi:10.1007/978-3-319-54499-1_1
- C. López, P. Inostroza, L.M. Cysneiros, H. Astudillo, Visualization and comparison of architecture rationale with semantic web technologies. *Journal of Systems and Software* **82**(8), 1198–1210 (2009). doi:10.1016/j.jss.2009.03.085

- S. Motogna, I. Ciuciu, C. Serban, A. Vescan, Improving software quality using an ontology-based approach, in *Proceedings of the OTM Confederated International Conferences*, ed. by I. Ciuciu, H. Panetto, C. Debruyne, A. Aubry, P. Bollen, R. Valencia-Garca, A. Mishra, A. Fensel, F. Ferri, Springer, 2015, pp. 456–465. Springer. doi:10.1007/978-3-319-26138-6_49
- A. Nuzzolese, A.L. Gentile, V. Presutti, A. Gangemi, Conference Linked Data: The ScholarlyData Project, in *Proceedings of the 15th International Semantic Web Conference*, ed. by P.T. Groth, E. Simperl, A.J.G. Gray, M. Sabou, M. Krötzsch, F. Lécué, F. Flöck, Y. Gil Lecture Notes in Computer Science, vol. 9982, 2016, pp. 150–158. doi:10.1007/978-3-319-46547-0_16
- C. Pardo, F.J. Pino, F. García, M. Piattini, M.T. Baldassarre, An ontology for the harmonization of multiple standards and models. *Computer Standards & Interfaces* **34**(1), 48–59 (2012). doi:10.1016/j.csi.2011.05.005
- S. Peroni, G. Lodi, L. Asprino, A. Gangemi, V. Presutti, FOOD: FOod in Open Data, in *Proceedings of the 15th International Semantic Web Conference*, ed. by P.T. Groth, E. Simperl, A.J.G. Gray, M. Sabou, M. Krtzsch, F. Lécué, F. Flck, Y. Gil Lecture Notes in Computer Science, vol. 9982, 2016, pp. 168–176. doi:10.1007/978-3-319-46547-0_18
- R. Pressman, *Software Engineering: a Practitioner's Approach* (McGrawHill, ???, 2014). ISBN 9780078022128
- V. Presutti, A. Gangemi, Dolce+D&S Ultralite and its main ontology design patterns, in *Ontology Engineering with Ontology Design Patterns - Foundations and Applications*, ed. by P. Hitzler, A. Gangemi, K. Janowicz, A. Krisnadhi, V. Presutti Studies on the Semantic Web, vol. 25 (IOS Press, ???, 2016), pp. 81–103. doi:10.3233/978-1-61499-676-7-81
- V. Presutti, G. Lodi, A. Nuzzolese, A. Gangemi, S. Peroni, L. Asprino, The role of ontology design patterns in linked data projects, in *Proceedings of the 35th International Conference on Conceptual Modeling*, ed. by I. Comyn-Wattiau, K. Tanaka, I.-Y. Song, S. Yamamoto, M. Saeki Lecture Notes in Computer Science, vol. 9974, Springer, 2016, pp. 113–121. Springer. doi:10.1007/978-3-319-46397-1_9
- D. Russo, P. Ciancarini, T. Falasconi, M. Tomasi, Software Quality Concerns in the Italian Bank Sector: The Emergence of a Meta-quality Dimension, in *Proceedings of the 39th International Conference on Software Engineering*, ed. by N. Juristo, D. Shepherd, IEEE, 2017, pp. 63–72. IEEE. doi:10.1109/ICSE-SEIP.2017.10
- D. Russo, P. Ciancarini, T. Falasconi, M. Tomasi, A meta-model for information systems quality: A mixed study of the financial sector. *ACM Transactions on Management Information Systems* **9**(3), 1–38 (2018). doi:10.1145/3230713
- C. Schmidt, P. Buxmann, Outcomes and success factors of enterprise it architecture management: empirical insight from the international financial services industry. *European Journal of Information Systems* **20**(2), 168–185 (2011). doi:10.1057/ejis.2010.68
- R. Singh, International Standard ISO/IEC 12207 software life cycle processes. *Software Process Improvement and Practice* **2**(1), 35–50 (1996). doi:10.1002/(SICI)1099-1670(199603)2:1;35::AID-SPIP29;3.0.CO;2-3
- G.H. Soydan, M. Kokar, An OWL ontology for representing the CMMI-SW model, in *Proceedings of the Workshop on Semantic Web Enabled Software Engineering*, 2006
- A. Strauss, J. Corbin, *Grounded theory in practice* (Sage, ???, 1997). ISBN: 9780761907480
- C.P. Team, Capability Maturity Model® Integration (CMMI), Version 1.1–Continuous Representation, Technical report, 2002
- S. Wagner, et al., The Quamoco product quality modelling and assessment approach, in *Proceedings of the 34th International Conference on Software Engineering*, ed. by M. Glinz, G. Murphy, M. Pezzè, IEEE, 2012, pp. 1133–1142. IEEE. doi:10.1109/ICSE.2012.6227106
- Y. Zhao, J. Dong, T. Peng, Ontology classification for semantic-web-based software engineering. *IEEE Transactions on Services Computing* **2**(4), 303–317 (2009). doi:10.1109/TSC.2009.20