Available online at www.sciencedirect.com

ScienceDirect

Advances in Space Research 70 (2022) 485–495

ADVANCES IN SPACE RESEARCH (a COSPAR publication)

www.elsevier.com/locate/asr

Knowledge tools to organise software engineering Data: Development and validation of an ontology based on ECSS standard

Ricardo Eito-Brun [a,*], Juan Miguel Gómez-Berbís [b], Antonio de Amescua Seco [b]

[a] *Universidad Carlos III de Madrid. Department of Information Science, Spain*
[b] *Universidad Carlos III de Madrid. Computer Science and Engineering Department, Spain*

Received 24 August 2021; received in revised form 20 April 2022; accepted 23 April 2022
Available online 6 May 2022

Abstract

This paper describes an OWL-based ontology to manage the data and artifacts created as part of software development projects based on the ECSS-E-ST-40C standard for SW development. ECSS standards are the main reference in aerospace projects. ECSS-E-ST-40C establishes the process, activities, requirements, and work products that must be generated. As part of its requirements, ECSS-E-ST-40C implicitly refers to specific data that must be generated, collected, and managed. The ontology identifies these data and the relationships that are inferred from the requirements in the standard, intending to set up the basis for building advanced information systems where data coming from different projects can be integrated and connected into a coherent set. The definition of this ontology provides the conceptual framework to develop data exchange processes and interfaces between the different tools and information systems used by the different players in the aerospace industry.
© 2022 COSPAR. Published by Elsevier B.V. All rights reserved.

Keywords: Software Development; Aeropace Software engineering; ECSS Standards; Ontologies and taxonomies; Data exchange; Data aggregation

1. Introduction

Software development is a key activity in the development and implementation of complex systems in different areas: automotive, aerospace, etc. To ensure the quality, consistency, and traceability of the software product, engineering and managerial teams need to gather, share, and report data generated by different processes and heterogeneous software tools. In the area of software engineering, different efforts to standardise the exchange of specific data can be identified, like the OMG's ReqIf model to exchange requirements data or the XMI language for UML models. In the context of specific industries, like automotive, additional efforts can be observed, like the QDX (Quality Data eXchange) model published by the VDA-QMC for quality-related data.

The European Cooperation for Space Standardization (ECSS) maintains a set of standards that cover different aspects of aerospace engineering, management, and quality disciplines; as part of these standards, ECSS-E-ST-40C focuses on software engineering requirements. ECSS standards are embraced by different aerospace agencies: Agenzia Spaziale Italiana (ASI), UK Space Agency, Centre National d'Etudes Spatiales (CNES), Deutsches Zentrum für Luft- und Raumfahrt (DLR), European Space Agency (ESA), Netherlands Space Office (NSO) and Norwegian Space Centre. ECSS standards count also with the Canadian Space Agency (CSA) as an associate member, and with CEN-CENELEC, EUMETSAT, the European Commission (EC), and the European Defence Agency (EDA) as

* Corresponding author at: Universidad Carlos III de Madrid c/ Madrid 126, 28903 Getafe, Madrid, Spain.
E-mail addresses: reito@bib.uc3m.es (R. Eito-Brun), jmgberbi@inf.uc3m.es (J. Miguel Gómez-Berbís), amescua@inf.uc3m.es (A. de Amescua Seco).

https://doi.org/10.1016/j.asr.2022.04.052
0273-1177/© 2022 COSPAR. Published by Elsevier B.V. All rights reserved.

observers. ECSS standards provide companies with clear guidance on the expected characteristics of the processes to be followed and the different artifacts and deliverables to be generated.

ECSS-E-ST-40C - the standard for software development - establishes a process model and requirements that can be enacted in different life cycles (waterfall, incremental, agile, etc.). Chapter 5 of the standard includes the requirements grouped into nine processes: 5.2. Software-related system requirements, 5.3. Software management, 5.4. Software requirements and architecture, 5.5. Software design and implementation, 5.6. Software validation, 5.7. Software delivery and acceptance, 5.8. Software verification, 5.9. Software operation and 5.10. Software maintenance. Each process is divided into different activities (level 3 subsections in the standard), and each activity into one or more tasks (level 4 in the standard) stated in the form clauses and sub-clauses identified by a four digits sequence. Each task results in a set of outcomes that must be reflected and documented in specific work products or documents.

The applicability of requirements in ECSS-E-ST-40C depends on the criticality of the software to be developed, with a distinction between four criticalities, being A the most restrictive and D the less one. ECSS-E-ST-40C includes an annex, annex R, which summarizes the applicability of the requirements to the four types of criticality. The relationships between each requirement and the document types and work products is also defined in annex Q. ECSS-E-ST-40C includes additional annexes, one for each document type, in the form of Document Requirements Definition (DRD) that propose the table of content and sections of each document.

One area of improvement that can be identified after the analysis of the standard refers to the lack of an explicit model on how to represent and manage the data generated by those processes. Although data representation and management requirements can be derived from the standard - they are implicitly stated in the requirements, tasks, and DRD descriptions -, the document-oriented approach followed by the standard jeopardizes this deductive process.

The lack of an explicit data representation model, covering data items and their relationship at a more precise, lower level of abstraction than the one provided by the DRDs, can generate some problems and limitations, among others:

a) Lack of detail and missing information in the documents generated by different partners involved in the software development projects.
b) Difficulties to establish data exchange protocols and interfaces between partners when exchanging information about requirements, design, quality of the software products, traceability data, etc.

To solve these potential issues, the proposed ontology identifies specific data elements that can be derived from the content of the ECSS-E-ST-40C. These data elements are aggregated into different work products or software items, which are later traced to the document types (DRD) defined in the standard.

The resulting software information model (SWIM) is represented as an ontology encoded in the OWL/RDF language. The model serves to represent the data and work products that must be generated and managed in software projects following ECSS-E-ST-40C. The ontology can also be used to define data exchange interfaces for different software items (requirements, problems, change requests, design elements, quality data, etc.) helping companies and agencies gain an effective control on the large volumes of software engineering data created by aerospace projects. The outputs of the different tools used by companies to complete their activities can be mapped to the data elements identified in the ontology, and those data coming from heterogeneous tools can be aggregated and integrated into a coherent set. The availability of a common, conceptual framework covering the different data elements and work products generated during the projects' life cycle, can be used to define data sharing and reuse policies.

2. Purpose of the ontology

The main purpose of the proposed ontology is to provide a common semantic to publish and share data about the items (requirements, design elements, source code files, test specifications, document deliverables, etc.) generated during the software life cycle. The ontology aims to contribute to the overall digitalisation of space projects by solving the difficulties to exploit, reuse and analyse the data hidden in documents.

The capability of sharing this information independently of specific commercial tools will improve engineers' productivity and reduce one of the most complex challenges in software engineering activities: information overload (Blincoe, Valetto and Damian, 2015; Demirsoy and Petersen, 2018; Gomes, Antunes and Furtado, 2015; Micallef and Porter, 2019).

Software engineers need to be aware of the availability of different data and items that have an impact on the activities they execute: getting knowledge on missing traceability's, access to the approved requirements and design, planned test cases, issues detected by static analysis tools, etc., usually require searching the data in different tools and databases. The lack of standard models to ensure a unified view on these data constitutes a handicap for productivity and reliability. The proposed ontology is expected to work as a reference model to support the harvesting and aggregation of data generated by different tools, and to support an efficient management of this complex information ecosystem.

Despite the significant number of discussions on the use of ontologies to support software engineering activities, some sectors have not embraced yet the use of ontologies.

Reasons contributing to the lack of adoption of these technologies include:

a) Formulations of ontologies that are not fully aligned with the standards applicable in specific industries.
b) Lack of evidence of the benefits that can be reached with the use of ontologies to support the software development processes.

To deal with the first issue, this research took the decision of creating the ontology using as the main input the text of the ECSS standard. A similar approach based on the use of the ISO standards was proposed by Henderson-Sellers, B. et al. (2104). The use of standards as the basis of ontology development ensures the availability of a common vocabulary and saves the effort of target users to get familiar with the proposed model. Besides that, as standards represent a shared understanding of the processes, activities, and work products. For the second issue, the capability of searching data typically spread in different data sources and managed with heterogeneous tools justifies the effort to develop conversion pipelines to move data to a representation aligned with the ontology's model.

3. Previous work and related experiences

The design and development of ontologies for software engineering have been a constant line of work in the academic area since the early days of the Semantic Web. An early overview of the first initiatives can be found in Calero et al. (2006). Studies that reached significant visibility include those of Kitchenham, B. A., et al. (1999) and Ruiz, F. et al. (2004) in the area of software maintenance, Wongthongtham, P. et al. (2009) on project development aspects, Korthaus, Schwind and Seedorf (2007) on component specification or Henderson-Sellers (2011) on the use of metamodeling techniques. Recently reported experiences on the application of ontologies in software engineering focus on specific areas like requirements and change management (Alsanad, Chikh, and Mirza, 2019; Duarte, 2018), quality and measurement (Blas, Gonnet and Leone, 2017; Fonseca, Barcellos and de Almeida, 2017). A recent review in the broader area of system engineering is provided by Lan, Cormican, and Yu (2019). In the specific area of aerospace software engineering, previous work includes the contribution of Eito-Brun (2016) and Hennig et al. (2016). The first paper addressed the modelling of document types and project management concepts, paying attention to the items managed in formal reviews (review item discrepancies, objectives, document types, data packages and actions) and the evolution of document baselines. Main difference between that previous work and the current one refers to the purpose, extent and scope of the ontology. That previous paper focused on aspects related to the publication and distribution of the different artifacts following a document-oriented approach: the proposal aimed to facilitate the generation and versioning of documents from projects and products' data, and classes and properties were only derived from the ECSS DRDs (e.g., TypesOfDocumentFiles, TypesOfFolders, DocumentFiles, etc) to represent the relations between documents and formal reviews. On the other side, the work presented in this paper extends that previous work and reaches a more detailed representation: ontology elements (classes, properties and individuals) are also derived from the text of the ECSS requirements (not only from the DRDs), and the ontology covers all the software engineering artifacts used in the standard and their properties, regardless their representation as documents. This updated scope allows additional scenarios for the exploitation of the ontology, beyond the management of formal reviews, documents and baselines. Hennig et al. (2016) proposed an approach to use ontologies to model satellite data and components in the context of Model Based System Engineering (MBSE); their ontology included classes and properties to model aspects related to the electrical architecture, functional and operational design, requirements, etc., as well as quantities, units and dimensions. They also proposed several evaluation cases related to the use of reasoning techniques to derive tests, and the administration of critical items. The use of ontologies in the context of MBSE has been widely discussed and tested in the context of the MODEX project, which formalized the artefacts produced and exchanged along software development process based on models (Tiago et al., 2021).

In addition to the studies and analysis reported above, the most relevant standardization activity in system and software is the Open Services for Lifecycle Collaboration (OSLC). Its purpose was to apply Semantic Web technologies (in particular, Linked Data Platform) to support data exchange and reuse. OSLC was launched in 2009 by IBM, and several companies like Boeing, Dassault-Systèmes, Siemens, Raytheon, etc., joined the project hereafter. The Project was later transferred to OASIS in 2013, and since 2019 is being managed by OSLC Open Project. OSLC establishes a protocol to query and update data about requirements, problems, change requests, test cases, etc. OSLC purpose is to avoid dealing with the proprietary interfaces between applications to access data. The set of specifications currently available includes OSLC Core v3.0, OSLC Query v3.0, OSLC Requirements management v2.1 and OSCL Change management v3.0, and OSLC Quality Management v2.1, among others.

OSLC Core 3.0 was published on September 17th 2020 and includes three parts – 6 to 9 - that define the classes, properties and restrictions of the ontology, being Part 7 the most relevant one from an ontological perspective. The core vocabulary defines classes and properties to manage annotations, comments and discussions on other resources. In addition to the Core Vocabulary, OSLC has published specifications for different domains: Quality management, Change management, Requirements Management, Asset management, etc. (only Change Management, Requirements Management and Quality

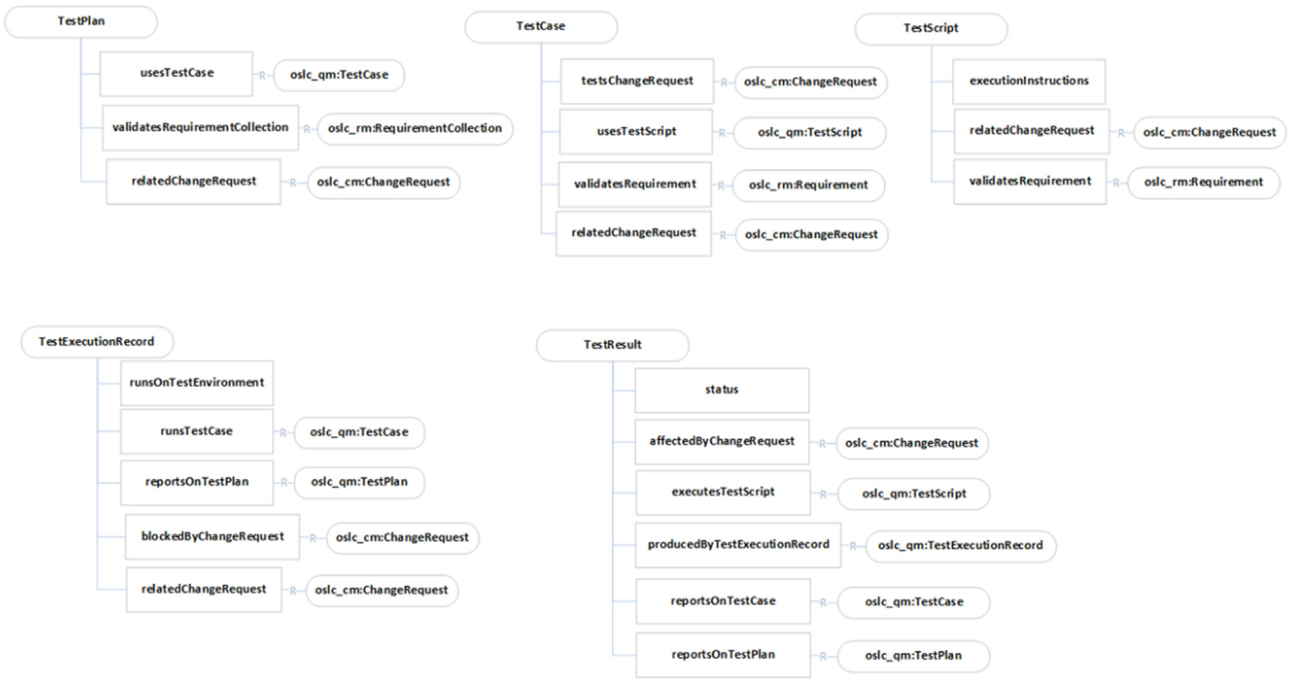


Fig. 1. OSCL Quality Management Ontology.

Management compatible with Core version 3). As an example, OSLC Quality Management part 2 includes a vocabulary made up of six classes and twenty-one properties (see Fig. 1); OSLC Requirements Management part 2 defines two classes and fifteen properties with additional restrictions, and OSLC Change Management nine classes and twenty-seven properties.

OSCL provides a valid ontology and information model to support data exchange and querying between heterogeneous applications, but its current scope does not cover

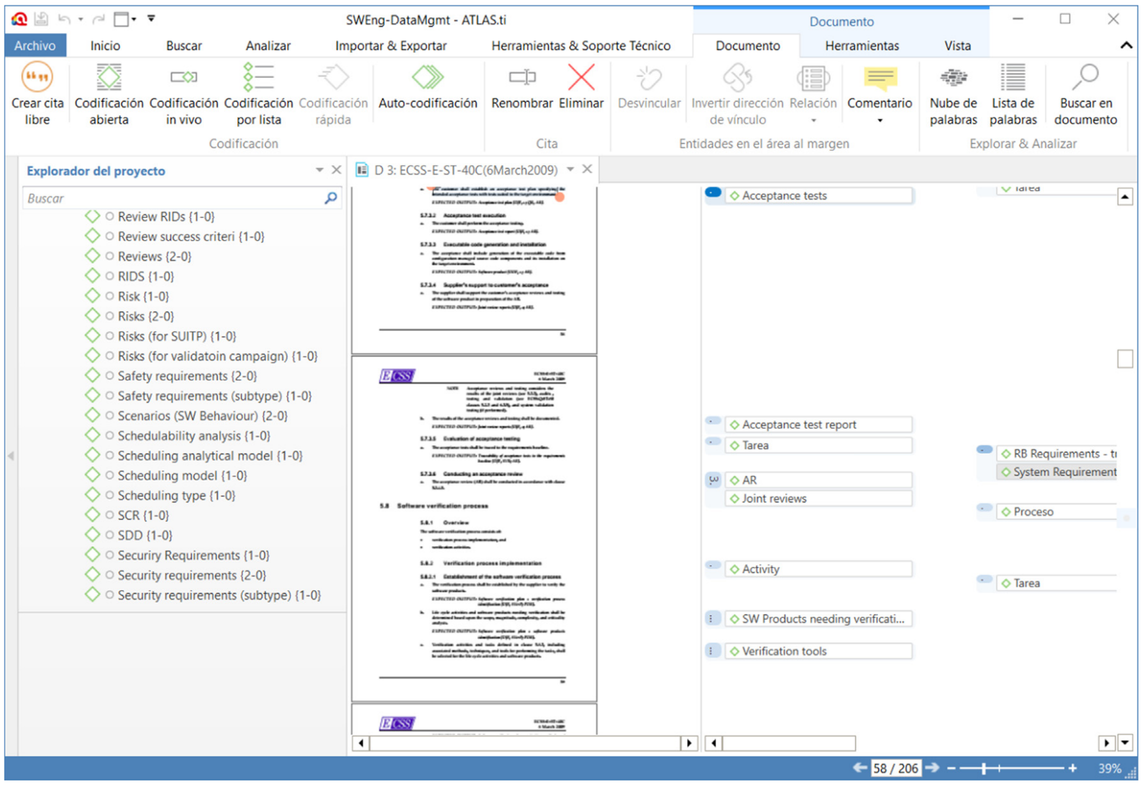


Fig. 2. Analysis of source text with ATLAS.ti.

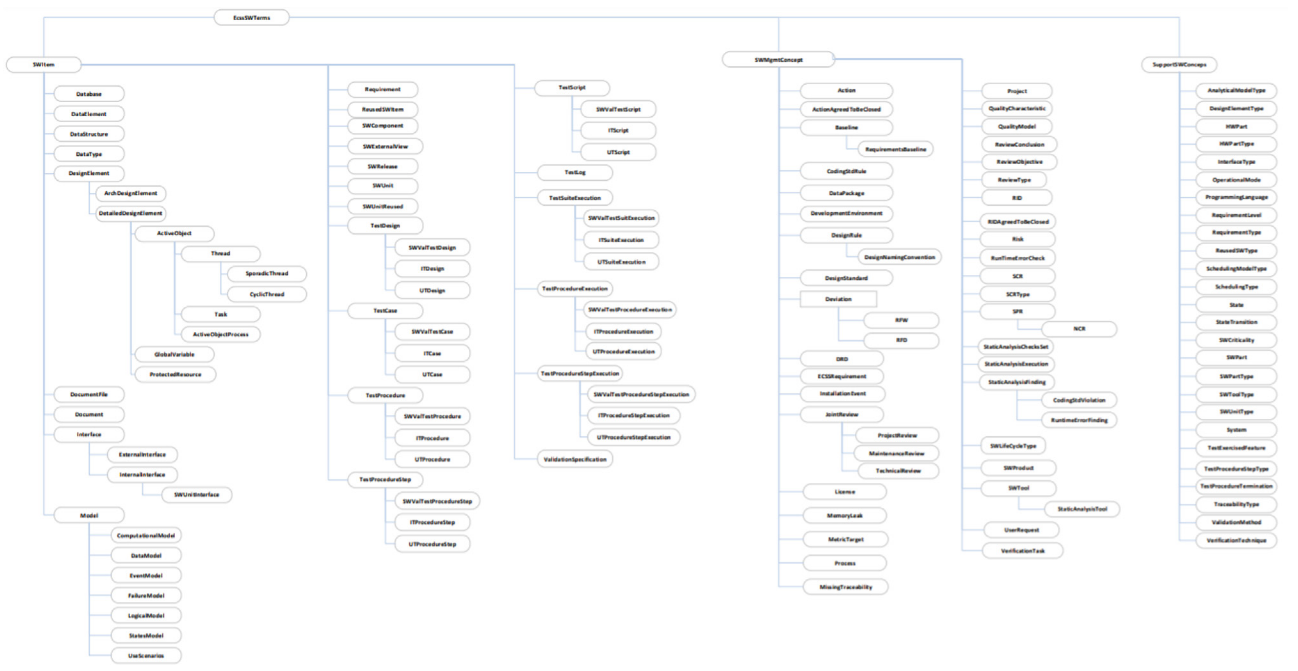


Fig. 3. Ontology class hierarchy.

some essential aspects in software development projects: quality is limited to testing, and other relevant data items like related to process performance, reviews, and source code quality or verification tasks. These limitations lead to the need of building an ontology that provides a major coverage to the information management requirements from aerospace engineering standards. The benefits of the proposed ontology include a wider coverage to the concepts used in the ECSS standard used as a reference, and the definition of a model that can be used as a reference to support the integration and exchange of data managed with heterogeneous tools and formats. The use of a standardized data modelling approach – semantic web data modelling with OWL/RDF) ensures and warranties the mapping between OSCL and the proposed ontology.

4. Methodology

The design of the ontology started with two parallel activities:

a) Analysis of the text of the standard to identify all the concepts that correspond to work products, data elements, and their characteristics.
b) Formulation of a set of competency questions with software engineers involved in ECSS-based projects to identify their information needs.

The analysis of the ECSS text was completed with a qualitative analysis tool: ATLAS.ti (see Fig. 2). The tool provides the capability of tagging terms, fragments, and sentences where concepts appear. These concepts are referred to as *codes*, which can be later combined and organised hierarchically to create a schema of concepts. Each code maintains the link to the fragments of the text where they have been identified. The tool also supports the automatic identification of codes (by searching the words used to represent the code across the full text of the document). The analysis of the ECSS standard with ATLAS.ti led to the identification of 660 codes. These codes – and the fragments of the standard where they appeared - were later analysed to establish the classes and properties of the target ontology. By doing that, all the elements in the ontology are derived and traced to the text of the standard, and properly justified.

The second activity performed to define the ontology – the formulation of *competency questions* - is one of the most popular and recognized methods for building ontologies (Corcho 2005; Iqbal 2013) and has been successfully applied in different scenarios. In this case, the identified competency questions referred, in most of the cases, to three different categories:

a) Characteristics of the work products: requirements, design elements, validation cases and procedures, etc.
b) Traceability and consistency between the different work products, with two separate focus: analysis of the impact of changes and identification of inconsistencies between related work products.

As an example, questions in the first group included: what requirements may be affected by a change made in a specific source code file? What test cases need to be reviewed and re-executed after a change in a design

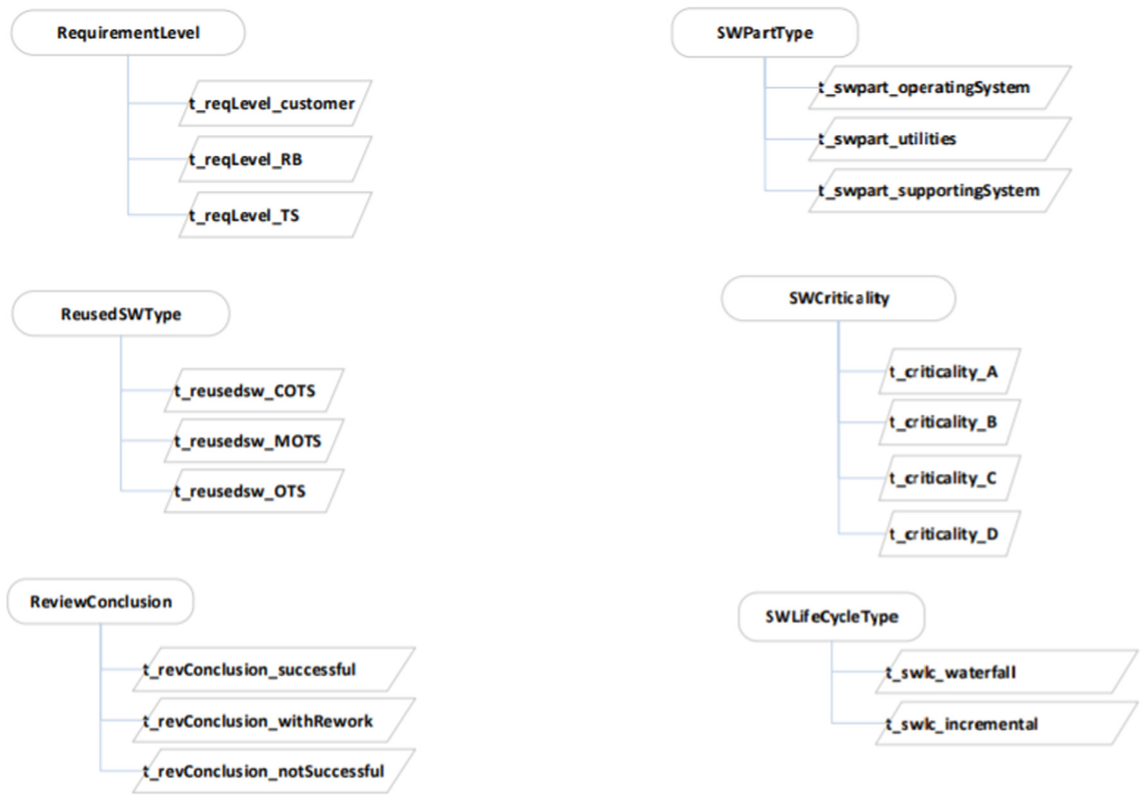


Fig. 4. Definition of instances or individuals (subset).

element? What interfaces may be affected by a change in a data structure? Questions in the second group had the purpose of identifying inconsistencies between the target versions allocated to different work products (e.g. upper-level and lower-level requirements, or requirements and traceability), or discrepancies between the software criticality allocated to requirements at the different levels (system and software).

c) Quality characteristics of the software units (i.e. source code files), including values for metrics, violations to coding rules or findings detected by static analysis tools. In this specific group, interviewed software engineers reported problems related to the need of dealing with different static analysis tools generating outputs in non-compatible formats.

The exercise identified a set of fifty-eight competency questions that were later used to validate the design of the ontology using a set of data extracted from a real project via SPARQL queries (Wiśniewskia, D. et al., 2019).

5. Results and discussion

The resulting ontology, which is publicly available at the link https://thesdoc.uc3m.es/?p=14, was built on the concepts and codes extracted from the analysis of the standard and the information needs identified through the competency questions. It does worth remarking that the ontology includes concepts (class and properties) that were not identified or mentioned in the competency questions. Regarding this, it is understood that - as ECSS standards are the result of an international effort led by subject matter experts and based on their agreements -, the exclusion of the concepts that were only derived from the text of the standard would result in missing information that could be useful in future, unplanned uses of the ontology.

The ontology includes a total of 155 classes. Classes have been hierarchically arranged into three main top-level categories: a) software items, b) software management concepts, and c) support software concepts (see Fig. 3).

This grouping of classes reflects the different aspects covered by the ECSS-E-ST-40C standard, which establishes a process model that includes both software engineering, quality, and management aspects.

- *Software items* refers to the artifacts or data that must be created during the software life cycle, and that – all together – will make the final, deliverable software product: requirements, models, source code files, architectural and detailed design elements, test designs, cases and procedures, documents, reused software items, etc. Classes that represent these items are defined as subclasses of the SWItem class.
- *Software management concepts* refer to aspects needed to manage the development and to ensure the quality of the software items. They include the process, activities, and

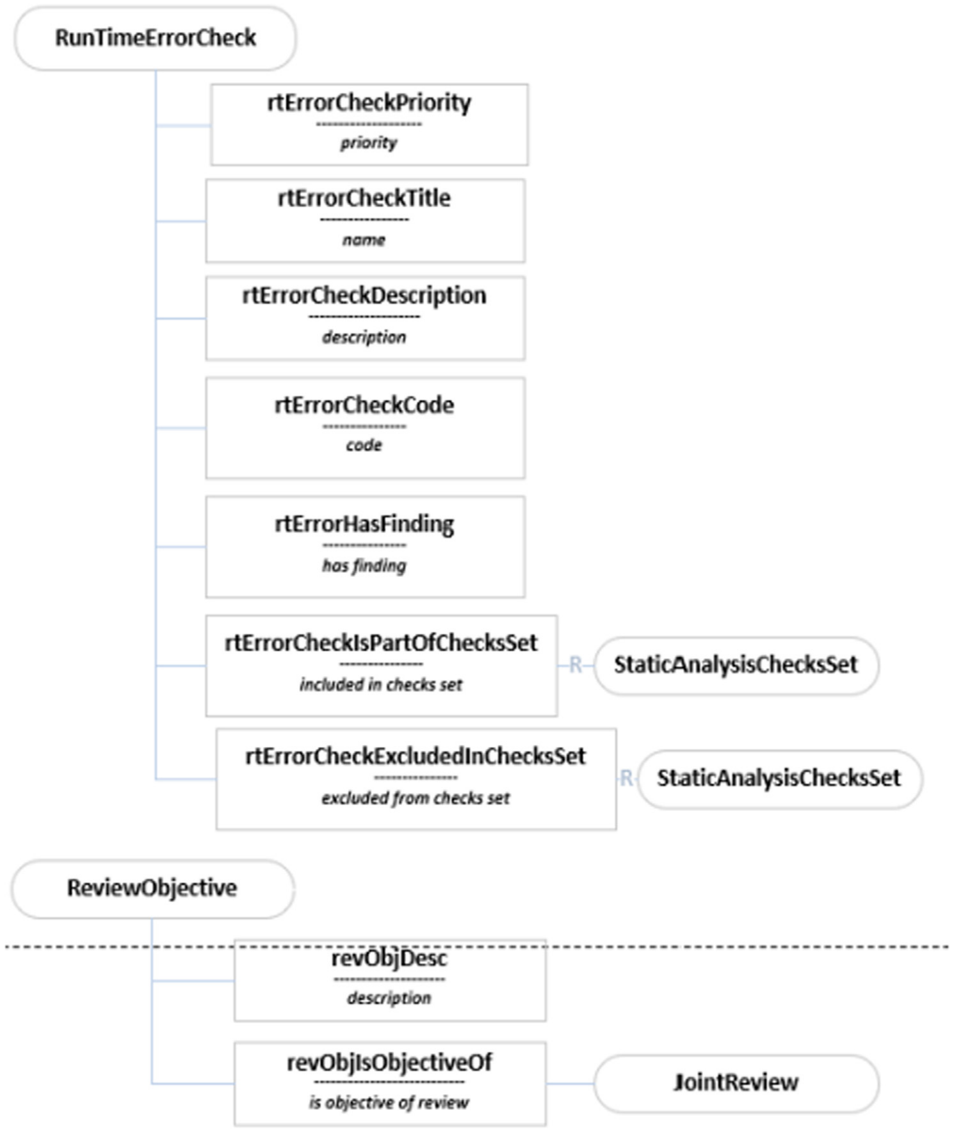


Fig. 5. Extract of properties with range and domain.

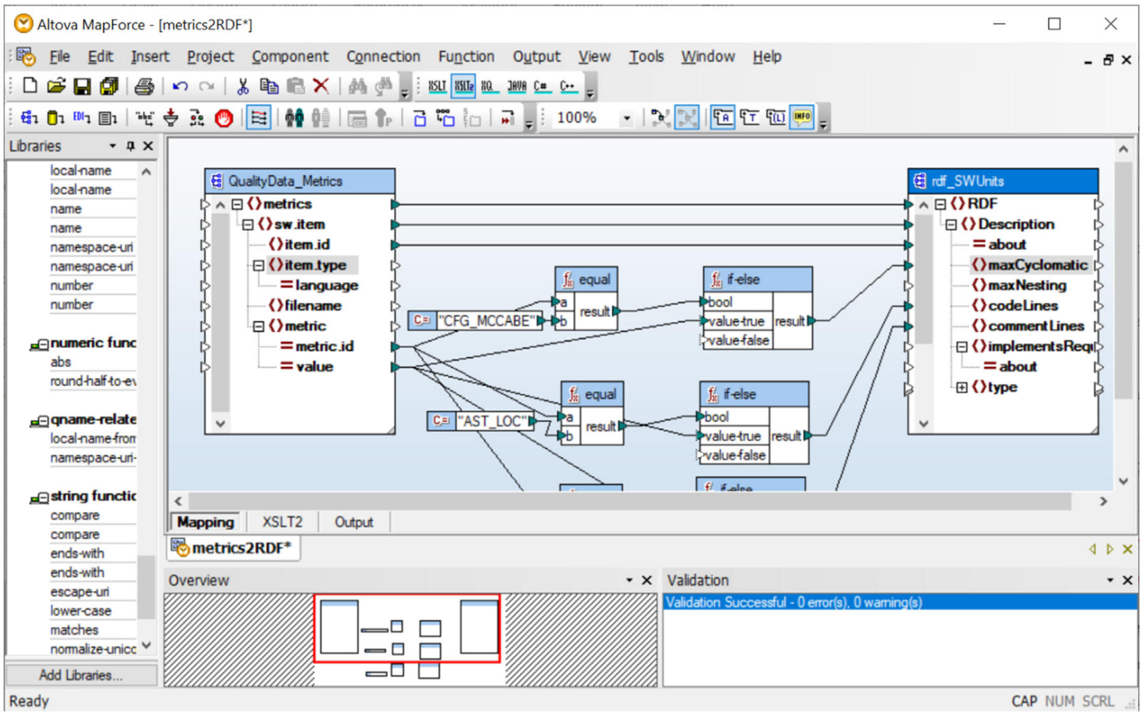
tasks in the model, joint reviews, findings identified during the review of documents (review item discrepancies or RIDs), actions, risks, deviations, change requests, license models, test campaigns, etc., as well as the findings detected by different verification activities that must be solved before the delivery of the software product. All these elements are represented as subclasses of the SWMgmtConcept class.

- Finally, *support software concepts* cover those items needed to characterise the elements in the previous groups, which are not part of the elements that compose the deliverable software. Concepts like types of analytical models or scheduling policies, operation modes, state transitions, features exercised by tests, etc., are represented as subclasses of the SupportSWConcept class.

To support – as much as possible – the standardization of the data in the planned uses of the ontology, specific instances or individuals have been defined for some of these classes (typically, for those that correspond to *support software concepts*). The designations for these instances have been extracted from the ECSS-E-ST-40C terminology. According to this approach, the categorization of types of requirements, types of operational modes, scheduling modes, software tools, verification techniques, features under tests, etc., have been completed through the enumeration of instances (see Fig. 4).

Naming conventions have been defined and applied for the different items in the ontology. In the case of instances that represent types, the applied convention uses the *t_class_value* convention, e.g. *t_reqLevel_RB, t_reqLevelTS, t_swunit_datafile, t_swunit_buildfile*, etc.

The design of the ontology also identified 448 properties (data and object properties). Properties are defined through their domain, range, and tagged as inverse, symmetric,



Validation Successful - 0 error(s), 0 warning(s)

Fig. 6. XSLT transformation of input data into RDF/XML.

transitive, functional, and inverse functional according to their semantics.

The graphical representation of the ontology – used in different figures in this paper – shows classes as ovals and properties as squares linked to the classes they use as domain and range. Fig. 5 shows an extract of all the properties that have the classes *RunTimeErrorCheck* as domain and *StaticAnalysisChecksSet* as range, and *ReviewObjective* as domain, and *JointReview* as the range.

It is highlighted that the resulting ontology does not follow a document-oriented approach: its purpose is neither to model the document types (DRDs) defined in the standard (annexes B to P) nor represent the sections within them; the ontology aims to identify the data and work products referenced in the standard to support the modelling, exchange and aggregation of the information generated and managed in a typical ECSS-based project. The ontology provides the capability of mapping and collecting information about artifacts like requirements, software problem reports, configuration items, design elements, etc., to support software engineering activities.

Regarding the use of patterns, the ontology incorporated classes to deal with aggregations of and relationships between data that do not correspond to entities or items that can be directly deduced from the text of the standard. RDF modelling approach based on the use of triples implies some difficulties to deal with aggregations that require the relationship of more than two individuals. An example of this is the data about missing traceability. Traceability is a key concept in software engineering to ensure the consistency, completeness, and coherence of the software engineering data developed at the different stages of the project. Traceability affects different items (requirements, design elements, software units, test cases, procedures, etc.) and – when missing or not provided – these gaps must be duly justified. To provide a general approach to deal with missing traceability justifications, the ontology incorporated one separate class that refers to the affected item, the traceability type that is not provided, and their justification. Instances of these classes are related to the affected software item whatever its type: requirement, design element, software unit, etc.

A similar approach was followed for the findings detected by static analysis, which require a relation between source code files, their specific version, the rule or control being violated, the tool used to execute the analysis, and the particular execution where the finding was raised. To deal with that, findings were declared as classes linked to different properties that point to the different data.

6. Validation

The ontology was validated in three steps:

a) Definition of a representative dataset with information extracted from a real software project.
b) Creation of SPARQL queries capable of answering the information needs expressed by the lead users' competency questions.

Table 1
SPARQL Queries.

Get all the requirements from the Technical Specification requirements with "Medium" priority.

```
SELECT?lowerReqcode?lowerReqtext?reqPriority
WHERE
{
  ?subject rdf:type < https://thesdoc.uc3m.es/ecss/onto/Requirement >.
  ?subject ecss:reqLevel < https://thesdoc.uc3m.es/ecss/onto/t_reqLevel_TS >.
  ?subject ecss:reqIdentifier?lowerReqcode.
  ?subject ecss:reqText?lowerReqtext.
    ?subject ecss:reqPriority?reqPriority.
  FILTER regex(?reqPriority, "Medium")
    }
```

List all the requirements from the Requirements Baseline with all their versions

```
PREFIX skos: <https://www.w3.org/2004/02/skos/core#>
PREFIX rdf: <https://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ecss: <https://thesdoc.uc3m.es/ecss/onto/>
PREFIX org: <https://www.w3.org/ns/org#>

SELECT?code?text
WHERE
{
  ?subject rdf:type < https://thesdoc.uc3m.es/ecss/onto/Requirement >.
  ?subject ecss:reqLevel < https://thesdoc.uc3m.es/ecss/onto/t_reqLevel_RB >.
  ?subject ecss:reqIdentifier?code.
  ?subject ecss:reqText?text.
```

List all the requirements in the Requirements Baseline that are traced to lower-level requirements in the Technical Specification

```
SELECT?code?text?lowerlevelreq
WHERE
{
  ?subject rdf:type < https://thesdoc.uc3m.es/ecss/onto/Requirement >.
  ?subject ecss:reqLevel < https://thesdoc.uc3m.es/ecss/onto/t_reqLevel_RB >.
  ?subject ecss:reqIdentifier?code.
  ?subject ecss:reqText?text.
  ?subject ecss:reqForwardTraceability?lowerlevelreq.
}
```

Get all the requirements in the Technical Specification that are derived from the RB requirements with code ID_50 and ID_59.

```
SELECT?lowerReqcode?lowerReqtext?reqPriority?upperLevelReq
WHERE
{
  ?subject rdf:type < https://thesdoc.uc3m.es/ecss/onto/Requirement >.
  ?subject ecss:reqLevel < https://thesdoc.uc3m.es/ecss/onto/t_reqLevel_TS >.
  ?subject ecss:reqIdentifier?lowerReqcode.
  ?subject ecss:reqText?lowerReqtext.
  ?subject ecss:reqBackwardTraceability?upperLevelReq.
    {?subject ecss:reqBackwardTraceability < https://thesdoc.uc3m.es/ecss/onto/ID_50> }
      UNION
        {?subject ecss:reqBackwardTraceability < https://thesdoc.uc3m.es/ecss/onto/ID_59> }
      }
```

Get a full traceability matrix between the Requirements Baseline (RB) and the Technical Specification (TS), including the requirements code, text and the target, planned version.

```
SELECT?lowerReqcode?lowerReqtext?lowerReqVersion?upperReqCode?upperReqtext?upperReqVersion
WHERE
{
  ?subject rdf:type < https://thesdoc.uc3m.es/ecss/onto/Requirement >.
  ?subject ecss:reqLevel < https://thesdoc.uc3m.es/ecss/onto/t_reqLevel_TS >.
  ?subject ecss:reqIdentifier?lowerReqcode.
  ?subject ecss:reqText?lowerReqtext.
    ?subject ecss:reqTargetVersion?lowerReqVersion.
  ?subject ecss:reqBackwardTraceability?upperReqCode.
    ?upperReqCode ecss:reqText?upperReqtext.
    ?upperReqCode ecss:reqTargetVersion?upperReqVersion.
}
```

Get the requirements covered by a waiver or deviation (RFD / RFW)

```
SELECT?lowerReqcode?lowerReqtext?reqPriority?deviation
WHERE
{
  ?subject rdf:type < https://thesdoc.uc3m.es/ecss/onto/Requirement >.
  ?subject ecss:reqLevel < https://thesdoc.uc3m.es/ecss/onto/t_reqLevel_TS >.
  ?subject ecss:reqIdentifier?lowerReqcode.
  ?subject ecss:reqText?lowerReqtext.
    ?subject ecss:reqBackwardTraceability?upperLevelReq.
  ?subject ecss:reqCoveredByDeviation?deviation
}
.
```

c) Mapping between the proposed ontology and OSCL classes and properties.

Loading of test data was conducted by exporting existing data from different tools into RDF format. Specific test data files were created for the following categories: requirements, design, software units, software units' verification – including unit tests and static analysis tools -, software validation and testing. Data were collected from these tools: DOORS®, Enterprise Architect®, C++ files, Polyspace Code Prover™, Apache Maven, Bug Finding™, and SciTool Understand™. Specific conversions were defined to convert the output of these tools to a suitable RDF format. These conversions made use of XSLT to transform the output files generated by the tools (typically Excel and HTML files) into RDF/XML. XSLT is a declarative language that takes as an input a text-based file and establishes the rules to map the input file into an output XML file. The conversions were implemented with the Altova® MapForce™ tool, which provides a user interface where transformation rules can be defined visually, using drag-and-drop features. The resulting XSLT files can be executed programmatically on batch mode to automate the conversion of the input data Fig. 6.

Once the sample data were available, the validation of the ontology was conducted through the design and execution of SPARQL queries run against the Ontotext GraphDB™ tool. Both the ontology schema and the sample dataset were loaded into the tool. The proposed schema was considered useful to deal with the information needs expressed by the users, and a set of typical SPARQL queries were defined so they can be reused in future sessions and projects. A subset of the SPARQL queries is provided in Table 1:

The SPARQL queries demonstrate the benefits of the proposed approach and its validity to implement end-user interfaces to explore the data managed in software development projects using a standard query language. Semantic web technologies ensure a greater compatibility between data and applications and simplifies the need of using proprietary query mechanisms.

Finally, the mapping between the proposed ontology and the different ontologies defined in OSCL did not identify gaps in the proposed ECSS-based ontology, but a greater level of detail and granularity. The provision of this mapping also supports the conversion of proprietary data formats from tools already supporting OSCL to the ontology's RDF.

7. Conclusions

ECSS standards represent the agreement reached by the representative, authorised actors of the industry on the recommended way to execute engineering activities and processes. A great number of aerospace projects uses the ECSS standards as the main guideline for their activities. The representation of the ECSS-E-ST-40C data requirements in the form of an ontology establishes the basis for building information systems capable of supporting data exchange and data aggregation requirements. These systems could be deployed in the context of a specific project or company – to ensure a single point of access to data generated by heterogeneous tools – or in an inter-company context to ensure the exchange of data, in a reusable format, in larger projects.

The proposed ontology lays the foundation to gather data derived from software engineering activities in a standardized way. The use of instances for types of items ensures the standard representation of aspects that – today – are not explicitly defined in any standard in place. The ontology also facilitates the explicit representation of the data, using approved, open and independent standards (OWL/RDF). Of course, the conversion into RDF of the original data as extracted from the source tools implies an additional effort, but once these mappings are implemented, the benefits of data integration are evident. The development of data conversion tools cannot be considered a showstopper to the proposed strategy. Today, we have at our disposal consolidated technologies like XSLT supported by a wide array of commercial and open source tools that make the development of these conversions easy. The fact that these mappings and conversions are built on the specific tools' output makes them reusable across different contexts. As a summary, it can be said that the ontology provides a model to which the data generated during the software life cycle can be easily mapped to be further analysed and exploited to answer different questions that – in any other case – would require access to heterogeneous tools, costly manual work and inspections.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Supplementary material

Supplementary data to this article can be found online at https://doi.org/10.1016/j.asr.2022.04.052.

References

Alsanad, A., Chikh, A., Mirza, A., 2019. A Domain Ontology for Software Requirements Change Management in Global Software Development Environment. IEEE Access 7, 49352–49361.

Blas, M.J., Gonnet, S., Leone, H., 2017. An Ontology to Document a Quality Scheme Specification of a Software Product. Expert Systems 34 (5).

Blincoe, K., Valetto, G., Damian, D., 2015. Facilitating coordination between software developers: A study and techniques for timely and efficient recommendations. IEEE Trans. Software Eng. 41 (10), 969–985. https://doi.org/10.1109/TSE.2015.2431680.

Calero, C., Ruiz, F., Piattini, M. (Eds.), 2006. Ontologies for Software Engineering and Software Technology. Springer, Berlin.

Corcho, O., 2005. Building Legal Ontologies with METHONTOLOGY and WebODE. In: Law and the semantic web: legal ontologies, methodologies, legal information retrieval, and applications. Springer, Berlin, pp. 142–157.

Demirsoy, A., Petersen, K., 2018. Semantic knowledge management system to support software engineers: Implementation and static evaluation through interviews at ericsson. E-Informatica Software Engineering Journal 12 (1), 237–263. https://doi.org/10.5277/e-Inf180110.

Duarte, B., Castro-Leal, A., Almeida-Falbo, R., et al., 2018. Ontological Foundations for Software Requirements with a Focus on Requirements at Runtime. Appl. Ontol. 13 (2), 73–105.

ECSS-E-ST-40C Rev. 1. 2009. *Space Engineering. Software*. ESA-ESTEC. ECSS Secretariat.

Eito-Brun, R. 2016. Design of an Ontologies for the Exchange of Software Engineering Data in the Aerospace Industry. In: Ngonga Ngomo, A. C., Křemen, P. (Eds.), *Knowledge Engineering and Semantic Web. KESW 2016*, Springer, https://doi.org/10.1007/978-3-319-45880-9_6.

Fonseca, V., Barcellos, M., de Almeida, R., 2017. An Ontology-Based Approach for Integrating Tools Supporting the Software Measurement Process. Sci. Comput. Program. 135, 20–44.

Gomes, P., Antunes, B.; Furtado, B. 2015. Using context for search, browse and recommendation in software development doi:10.1007/978-3-662-46549-3_18.

Hennig, C., Viehl, A., Kämpgen, B., Eisenmann, H. 2016. Ontology-Based Design of Space Systems. In: Groth, P. et al. (Eds.), *The Semantic Web – ISWC 2016. ISWC 2016. Lecture Notes in Computer Science*, vol 9982. Springer, Cham. https://doi.org/10.1007/978-3-319-46547-0_29.

Henderson-Sellers, B., 2011. Bridging Metamodels and Ontologies in Software Engineering. J. Syst. Softw. 84 (2), 301–313.

Henderson-Sellers, B., Gonzalez-Perez, C., McBride, T., et al. 2104. An ontology for ISO software engineering standards: 1) Creating the infrastructure. *Computer Standards & Interface*s. 36(3): 563-576.

Iqbal, R., 2013. An Analysis of Ontology Engineering Methodologies: A Literature Review. Research journal of applied sciences, engineering, and technology. 6 (16), 2993–3000.

Kitchenham, B.A., Travassos, G., Mayrhauser, A., et al., 1999. Towards an Ontology of Software Maintenance. Journal of Software Maintenance and Evolution 11 (6), 365–389.

Korthaus, A., Schwind, M., Seedorf, S., 2007. Leveraging Semantic Web technologies for business component specification. Journal of Web Semantics 5 (2), 130–141.

Lan, Y., Cormican, K., Yu, M., 2019. Ontology-based systems engineering: A state-of-the-art review. Comput. Ind. 111, 148–171.

Micallef, M., Porter, C., 2019. Towards detecting and managing information anxiety in the ICT industry. In: Paper presented at the Proceedings of the International Conference on Software Engineering and Knowledge Engineering. https://doi.org/10.18293/SEKE2019-095.

Ruiz, F., Vizcaino, A., Piattini, M., et al., 2004. An Ontology for the Management of Software Maintenance Projects. Int. J. Software Eng. Knowl. Eng. 14 (3), 323–349.

Tiago. J., de Ferluc, R., Ávila, I., et al. 2021. Implementation of the ECSS-E-ST-40C Processes Using a Model-Based Paradigm. *Model Based Space Systems and Software Engineering 2021. Available at:* https://indico.esa.int/event/386/contributions/6288/attachments/4284/6394/1605%20-%20implementation%20of%20the%20ecss-e-st-40c%20processes%20using%20a%20model-based%20paradigm.pdf.

Wiśniewskia, D., Potoniecab, J., Ławrynowicz, A., et al., 2019. Analysis of Ontology Competency Questions and their formalizations in SPARQL-OWL. Journal of Web Semantics 59. https://doi.org/10.1016/j.websem.2019.100534.

Wongthongtham, P., Chang, E., Dillon, T., et al., 2009. Development of a Software Engineering Ontology for Multisite Software Development. IEEE Trans. Knowl. Data Eng. 21 (8), 1205–1217.