# Supporting SW-FMEA through an ontology-based methodology


## Irene Bicchierai, Giacomo Bucci, Enrico Vicario

Department of Information Engineering,
University of Florence,
Florence, Italy
E-mail: {irene.bicchierai, giacomo.bucci, enrico.vicario}@unifi.it



**Abstract** Failure Mode and Effects Analysis (FMEA) is a widely used dependability and safety technique aiming at systematically identifying failure modes, their causes, their effects, and the countermeasures that could mitigate their impact. Although FMEA has been thought for hardware, its use is also advocated for software (SW-FMEA). However, this involves several major hurdles such as the complexity of functional requirements and the difficulty to identify failure modes of SW components.

We present an approach for efficient and effective manipulation of data involved in the SW-FMEA process, introducing an ontological model that formalizes concepts comprised in the analysis and provides a common conceptual framework supporting cohesion across different stages of a development lifecycle.

The ontological model opened the way to the implementation of a tool, which automates SW-FMEA providing support for basic and advanced functionalities, fastening the analysis. Use of the tool in a real SW development process is discussed at the end of the paper.



**Keywords:** Failure Mode Effects Analysis (FMEA); V-Model; ontologies; automated reasoning; Reliability Availability Maintainability and Safety (RAMS); SW Engineering; traceability.


**Reference**


**Biographical notes:** Irene Bicchierai received the Bachelor Degree in Informatics Engineering and the Master Degree in Informatics Engineering from the University of Florence in 2005 and 2008 where she is currently working toward the Ph.D. degree in informatics, multimedia, and telecommunications engineering. Her research interests focus on methods for design and testing of real-time embedded systems. Her activity mainly addresses the integration of formal methods in the development life-cycle of safety critical software.

Giacomo Bucci received the degree in electrical engineering from the University of Bologna in 1968. From 1970 to 1982, he was with the University of Bologna. During 1975, he was a visiting researcher at IBM T.J. Watson Research Center, Yorktown Heights, New York. Since 1986, he has been a full professor at the University of Florence, Faculty of Engineering, where he teaches a course in computer architectures and a course in software engineering. Prof. Giacomo Bucci has held several academic positions. Currently, he coordinates the pH doctorate in Informatics, Systems and Telecommunications at the University of Florence. His current research interests include computer architecture, real-time systems, distributed systems, software development methodologies and computer performance evaluation.

Enrico Vicario is a Full Professor of Computer Science at the School of Engineering of the University of Florence, where he received the Doctoral Degree in Electronics Engineering and the Ph.D. in Informatics and Telecommunications Engineering, in 1990 and 1994, respectively. His research is presently focused on formal methods for model driven development, correctness verification of real-time systems, and quantitative evaluation of concurrent non-Markovian models.

## 1 Introduction

Failure Mode Effects Analysis (FMEA) is a widely used dependability and safety technique which aims at the systematic identification of the failure modes of a system, the generating causes, the consequences they might have, and the countermeasures that could mitigate their effects or reduce their likelihood. FMEA was first developed by the Department of Defense of USA and standardized in the Military Standard 1629A (MIL-STD-1629A) [51], and it was then extended to various other industrial domains, some of which developed their own standards [27, 48, 8]. Furthermore, FMEA has been treated in a large number of works in literature [50, 34].

Several tools supporting the analysis are available off-the-shelf (see for instance [40, 45, 11]). Usually, they provide a variety of features to process and organize relevant information, as well as to perform ancillary tasks such as risk analysis.

Unfortunately, FMEA-related information is usually acquired in natural language, implying that interpretation of the terms and the concepts used across the analysis may differ from team to team; even the same team may give different interpretation when reusing an already performed analysis in a later occasion. Due to the lack of reusability, FMEA is often done from scratch. In large systems, it is barely possible to avoid inconsistencies [14].

To address these issues, several authors have proposed to resort to ontologies as a way to formalize the FMEA process, to manipulate concepts and to process data involved in the analysis. In [54], a method and a system aiming at facilitating reuse of knowledge, supporting complete and precise description of processes and products is proposed; the semantic knowledge is hierarchically organized in form of taxonomies, containing typical recurring technical knowledge about systems, functions, failure modes, and actions. A step further is made in [14], where the ontological formalization of FMEA concepts provides the ground for their explicit representation and for their unambiguous interpretation. In [31], a mapping from the concepts of an extended functional ontology to the concepts of a classic FMEA worksheet is introduced. Some guidelines for application of ontologies to FMEA are outlined in [16].

While FMEA has been mainly thought for hardware systems, its use is also advocated for those systems where safety and dependability are strongly affected by SW, resulting in the so-called SW-FMEA. This is explicitly prescribed in several regulatory standards driving the development process in mature application contexts, such as space [20, 35] or railways signalling [12].

However, the application of FMEA to SW faces various major hurdles. Functional requirements allocated to SW are by far more complex than those assigned to hardware; identification of failure modes of SW components cannot rely on data-sheets or operational previous experience; SW faults often elude testing, remaining hidden until some complex

triggering conditions activate them [1, 43]; they cannot be traced back to well identified causes, such as ageing, wearing, or stressing [38].

Complexities affecting SW-FMEA, in conjunction with those described for the classical FMEA, add further motivations to employ ontologies to deal with them, so as to improve the overall process.

In this paper, we propose a methodology that formalizes the semantics of both concepts and data involved in the SW-FMEA process. This provides a common conceptual framework, supporting cohesion across different stages of the development life-cycle, giving a precise semantics to concepts collected in the artifacts of an industrial documentation process [52]. We enhance the preliminary work presented in [2], adding support for functional requirements traceability, providing a technique for the automatic extraction of relations between functional and structural perspectives of a system. Furthermore, once the mapping between these perspectives is obtained, it is possible to automatically check if the system realization satisfies a certain *level of rigor* established by the criticality associated to each functional requirement. The proposed methodology also opens the way to the implementation of a tool for automating the processing of SW-FMEA data by leveraging both the semantics defined by the ontology and well-established technologies [33, 39, 24]. With respect to [2], where we outlined the tool which was still under development, we give a more comprehensive illustration of its architecture, its advanced functionalities, as well as how it is employed in an industrial context.

Without loss of generality, this paper refers to an industrial context where activities and documental artifacts are regulated by the European Cooperation for Space Standardization (ECSS) E-40 standard [18] and organized according to the V-Model framework [10] and the MIL-STD-498 [52]. Specifically, this relies on a concrete experience made at the FinMeccanica site of Selex Galileo in the frame of AASTR project, called *reference context* in the sequel.

The rest of the paper is organized in four sections. In Section 2, the ontology modeling SW-FMEA process is presented giving a general overview of ontologies (Section 2.1) followed by the description of the three ontological fragments covering the three phases of the methodology (Sections 2.2, 2.3, and 2.4). In Section 3, various ways of discovering the concrete data to be inserted in the ontology are described, three alternatives are provided: i) extracting data from documents produced during the SW life-cycle (Section 3.1); ii) discovering relations between functional and structural perspectives, automatically tracing functional requirements on their implementing SW components (Section 3.2); iii) inferring new information in order to evaluate the quality of the implemented SW components (3.3). In Section 4 the tool supporting the methodology is illustrated from basic (Section 4.1) to advanced (Section 4.2) capabilities and through the experience done within the real scenario of a Selex Galileo project (Section 4.3). In Section 5, conclusion based on our experience are drawn and possible future work is envisaged.

## 2 Ontological formalization of SW-FMEA concepts

The SW-FMEA process is naturally decomposed into phases. In [44], SW-FMEA is decomposed into two steps: i) a hazard analysis is initially performed to identify failure modes and their consequences; ii) countermeasures for mitigation or elimination of failures are then identified. In [22], SW-FMEA is also decomposed into two steps: i) *System*

*SW FMEA*, performed early in the design process to minimize the impact of design recommendations resulting from the analysis; ii) *Detailed SW FMEA*, performed later during the system design, with the aim of discovering safety requirements not being met and possible additional requirements. In [5], the process is decomposed into four activities: *Top level functional FMEA*, *Functional SW FMEA*, *Interface SW FMEA*, and *Detailed SW FMEA*, corresponding to different levels of detail of the analysis.

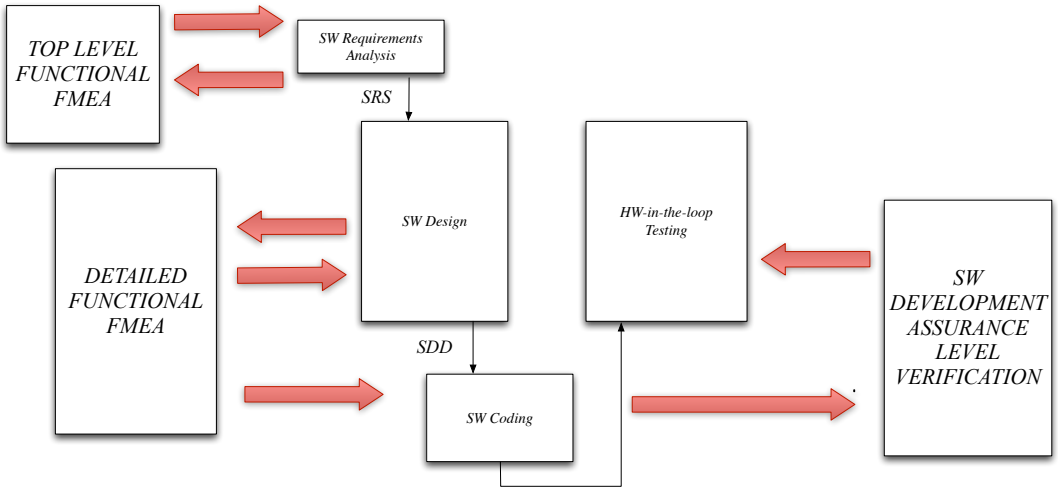


**Figure 1** The stages of the FMEA process mapped on the activities of the development life-cycle characterizing the *reference context*.

Reflecting the practice of our *reference context*, this paper assumes that the SW-FMEA process is decomposed into three phases spanning over the whole SW life-cycle as shown in Fig. 1: the *Top Level Functional FMEA* is mapped on the *SW Requirements Analysis*; the *Detailed Functional FMEA* is mapped on the *SW Design* and *SW Coding* activities; the *SW Development Assurance Level (SW-DAL) Verification* is mapped on the *SW Coding* and *HW-in-the-loop Testing* activities. While the treatment of this paper is focused on this concrete case, the proposed ontological model can be conveniently adapted to other methods and practices like those previously mentioned [44, 22, 5].

In the following, an ontology is used to support both the formal characterization of concepts involved in the SW-FMEA process and the automatic manipulation of their instances. We also establish a connection between concepts and notions mentioned in the standards regulating the development life-cycle and characterizing the context of use.

The next section recalls a few basic notions about ontologies, while the subsequent sections describe the three phases characterizing our methodology.

## 2.1 Ontologies overview

An ontology is an explicit specification of a conceptualization [23]. The fundamental elements of an ontological model are *classes* and *properties*, where classes are categories or sets of elements, and properties specify the internal structure of a class or the relations among classes. Classes and properties represent the *intensional* part of the ontology, while their instantiations represent the *extensional* part: *individuals* are realizations of concepts

described by classes and *attributes* are realizations of properties. These elements are concretely implemented through a stack of standardized technologies of the Semantic Web, comprising: i) Ontology Web Language (OWL) [33], used for describing ontologies and based on Resource Description Framework (RDF) [30] and RDF Schema [7] which, in turn, depend on XML [6] and XML Schema [21]; ii) Simple Protocol and RDF Query Language (SPARQL) [39], used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware; iii) Semantic Web Rule Language (SWRL) [24] used for writing inference rules and based on a combination of OWL-DL with the Rule Markup Language [4]. In addition, off-the-shelf reasoners are available (e.g. [47, 28, 32]). Altogether, they devise a new paradigm for the organization of systems [9]. Ontologies can be visualized using Unified Modeling Language (UML) notation augmented with stereotypes for RDF and OWL concepts as standardized in the Ontology Definition Metamodel (ODM) [36]. In the following, we adhere to the ODM notation to represent (portions of) the ontological model (see Figs. 2, 3, 4).

In the following, an ontology is used to formalize the semantics of concepts and data involved in the SW-FMEA process, providing a conceptual framework which is robust enough to enforce cohesion and consistency among information elements acquired along different phases of the development life-cycle.

## *2.2 Top Level Functional FMEA*

The first phase of the analysis is carried out early in the SW life-cycle, when the impact of changes to the original project are significantly less expensive. The analysis is performed during the allocation of technical requirements to the *Computer SW Configuration Items* (*CSCI*s) and *Hardware Configuration Items* (*HCI*s) in which the system is decomposed.

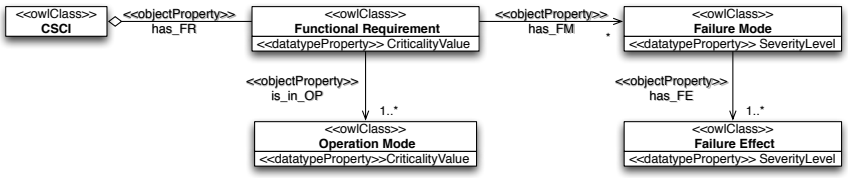


**Figure 2** UML representation of the intensional part of the ontology modeling the concepts involved in the Top Level Functional FMEA.

Fig. 2 shows the ontological concepts involved in this phase. The focus is on the definition of functional requirements associated with the *CSCI*. Each *Functional Requirement* is associated with one or more *Operation Mode*s and with some *Failure Mode*s; the latter are the different ways in which the delivered service deviates from the correct implementation of the system function [1]. Each failure mode is associated with the *Failure Effect*s, representing the consequences of the failure upon the system environment. Consequences are then classified through the *Severity Level* of failure effects. The number, the labeling and the characteristics of the severity levels are application-related and involve the dependability attributes for the considered application [19, 42, 13]. A *Failure Mode* has its own *Severity Level* whose value is taken as the maximum severity level of its associated *Failure Effect*s.

Each functional requirement is characterized by a *Criticality Value* which accounts for the risk of failures associated with the requirement. In our *reference context*, the *Criticality*

*Value* of a functional requirement is defined as a function of both the severity level of its associated failure modes and the criticality value of its associated operation modes. Note that this slightly differs from [19] where the criticality is assigned only to failure modes and is derived from the severity and the probability of the failure mode occurrence. In this phase of development, failure rates are not available, and only severity is thus considered.

The analysis is carried out in early development phases in an iterative manner. At each iteration the criticality of functional requirements is evaluated and, if necessary, further functional requirements are added as countermeasures against the failure modes having high severity levels.

### *2.3 Detailed Functional FMEA*

The second phase of the analysis is accomplished when the SW architecture is almost completed and each CSCI has been associated with a SW structure (see Fig. 1).

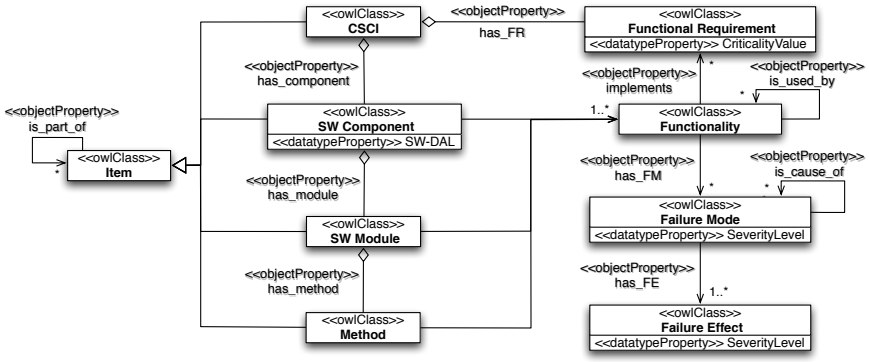


**Figure 3** UML representation of the intensional part of the ontology modeling the concepts involved in the Detailed Functional FMEA.

Fig. 3 shows the involved ontological concepts. The focus is on the definition of the structural SW elements that compose the system, represented by *Item* class. An item can be the entire *CSCI*, a *SW Component*, a *SW Module*, or a *Method*. Structural elements are decomposed in hierarchical manner, from the entire CSCI to the methods, which are the smallest parts of SW with documented functionalities. A CSCI is organized in SW components with their own responsibilities expressed through their methods. Components are physically organized in modules written in some programming language, which, in our *reference context*, are C and assembly. Each *SW Component* has an attribute, *SW-DAL*, representing the level of assurance that must be attained in the development of the component itself.

In this phase of the analysis, functionalities are associated with the structural elements that implement them. The hierarchical decomposition of structural elements is mirrored in a corresponding hierarchy of functionalities. Thus, elements of higher level are associated with more generic functionalities, which in turn are organized in sub-functionalities implemented by elements at a lower level of the hierarchy. Items directly associated with functionalities (see Fig. 3) cooperate in satisfying the functional requirements associated with the CSCI at the root of the hierarchy. Each *Functionality* is associated with one or more *Failure Mode*s, their *Failure Effect*s and their severity, in the same way as in the Top

Level Functional FMEA. A failure mode associated with a functionality decomposed in sub-functionalities (*is_used_by* property in Fig. 3) is likely caused by the failure modes associated with these sub-functionalities.

It is worth noting that there is an indirect association between *SW Component* and *Functional Requirement*, obtained through the *Functionality* class. This provides the path between *SW Component* and *Functional Requirement*, so that the SW-DAL of a SW Component can be taken as the assurance level corresponding to the highest criticality value among implemented functional requirements.

### *2.4 SW Development Assurance Level Verification*

The third and last phase of the analysis is accomplished late in the life-cycle, when coding is completed (see Fig. 1). Regulatory standards as [42, 12, 35] define the assurance levels classifying SW components according to risk associated with their implementation in relation to dependability attributes (e.g. availability, reliability, safety) relevant for the considered application [1, 46]. Standards prescribe the execution of activities and the production of artifacts along the development life-cycle in order to achieve some specified levels of assurance: RTCA [42] defines *Development Assurance Levels (DALs)*, which guide activities applied along the development; CENELEC [12] associates with *Safety Integrity Levels (SILs)* the corresponding intervals of failure rates; NASA [35] specifies a *SW Risk Index* and a *SW Safety Effort* and the mapping between them.

The allocation of the DAL to SW is primarily based on the severity of the effects of a failure mode; however it can be optimized through the use of algorithms and criteria. For example, in [3], methods to check or optimize a DAL allocation are addressed, formalizing a set of allocation rules as a set of constraints, using a constraints solver to find the optimal solution. Differently from [3], in our *reference context* the SW-DAL requested in the development of a SW component is taken as the level corresponding to the highest criticality value among the criticality values associated with requirements implemented by the SW component (Section 2.3).

Fig. 4 shows the ontological concepts involved in this phase. Here, the association between a *Functional Requirement* and a *SW Component* (not depicted in the UML diagram of Fig. 3) is obtained through the *Usage Degree* association class specifying how many of component's functionalities are used to meet the functional requirement. The *Criticality Value* of a functional requirement comes from Top Level Functional FMEA (see Fig. 2) and represents the risk associated with the implementation of the requirement itself. The *SW-DAL* of a *SW Component* is computed during Detailed Functional FMEA and quantifies the level of rigor that has to be attained in the development.

Each SW component is then organized in *SW Module*s which contain *Method*s. A method is associated with *SW Parameter*s and *Prevention Mitigation*s which account, respectively, for code and process metrics. Instances of SW parameters are associated with a method through an instance of the *SW Parameter Accountability* association class representing the value of the specific parameter for that method. In the same way, the association among a method and prevention mitigations is accounted by a value modeled by the *Prevention Mitigation Accountability* association class. *SW Parameter*s represent properties in the structural perspective, examples from our *reference context* are number of Lines Of Code (LOC), level of nesting, and cyclomatic complexity; *Prevention Mitigation*s represent process metrics, examples from our *reference context* are *testing coverage*, *requirements stability* and *number of open and closed non conformances*.

The objective of this phase is to verify if the SW components implementation achieves the required level of assurance, so as to guide the elimination of design and implementation flaws which could impair the system dependability. This is carried out by verifying whether the actual implementation of SW components satisfies the level of assurance defined through information coming from previous phases. This will be shown in Section 3.3. If this verification shows that the required level of assurance is not achieved, additional testing activities or code revisions of the components violating the expected level of rigor can be performed.

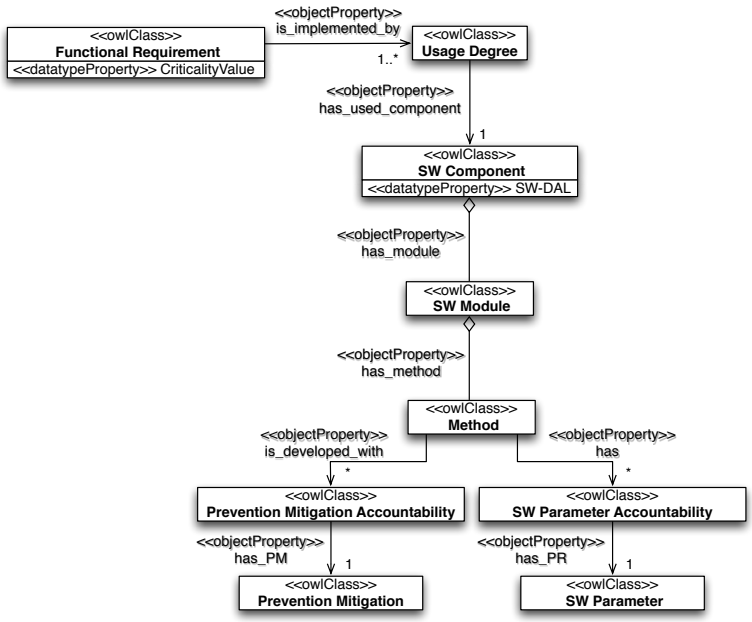


**Figure 4** UML representation of the intensional part of the ontology modeling the concepts involved in the SW Development Assurance Level Verification.

## 3 Casting process data into ontology concepts

Ontological concepts involved in the SW-FMEA become concrete by associating them with actual data derived from artifacts and documents produced along the development lifecycle. This stands for the population of the ontological model with instances of structural elements (e.g. CSCI, SW component), functional elements (e.g. functional requirement, functionality), and relations among them.

### *3.1 Finding instances in documents*

Data related to the Top Level Functional FMEA (see Fig. 2) are captured in the *SW Requirements Specification (SRS)* document, that specializes the IEEE-830 standard [25] for the given industrial context. In the SRS, functional and non-functional requirements of a

CSCI are described along with the failure modes, the effects of failure, and the resulting criticality value associated with each requirement. Functional requirements are prescribed to be organized in groups: they can be divided depending on the operation mode they refer to.

Concepts related to the Detailed Functional FMEA (see Fig. 3) are mostly derived from the design activity of SW life-cycle, where architectural items are defined and associated with functionalities. The organization of the SW structure is usually described in the *SW Design Description (SDD)* document, which specializes the IEEE-1016 standard [26] for the application context. SDD is divided in two parts: the former reports a high level description of the SW items and their functionalities, the latter provides a detailed description of the same items, including explanatory code fragments.

Data related to the SW-DAL Verification (see Fig. 4) come from the SW Coding. The mapping between functional requirements and SW components is usually reported at the end of the SDD while data about code and process metrics, in our *reference context*, are provided in the *Product Assurance Report*, conforming to the standard ECSS-Q-80 [20].

### *3.2 Tracing Functional Requirements*

A crucial point in the analysis of failures is given by the relations between functional and structural perspectives. In fact, the association among structural and functional elements is important with respect to maintainability, since it impacts on the ability of the system in isolating or correcting a defect, as well as on satisfaction of new requirements. In addition, the identification of the SW components that implement a functional requirement is also relevant to verify that the implementation is compliant with design specification. As a matter of fact, the regulatory standards expressly require that documents, such as SRS and SDD, contain the *traceability matrix*.

Traceability of functional requirements has been addressed in various works [17, 56, 55]. In [17], the identification of required computational units is performed through a technique that combines static and dynamic analysis, using concept analysis. The static part of the technique consists in the extraction of the static dependency graph, while the dynamic part of the technique traces the execution of some features, giving the system appropriate inputs. A classification of components is then obtained, analyzing the relevance of a component with regard to a feature. The approach presented in [56] is based on the identification of code invoked during both the execution of the target feature and the execution of the other features. Subtraction of the second from the first gives the desired result. The quantitative evaluation of the relation between a component and a feature is addressed in [55] through the introduction of three metrics: the *concentration* of a feature in a SW component, the *dedication* of a component to a feature, and the *disparity* which measures the closeness between a feature and a component. The authors consider a component as a file composed by basic blocks (i.e. sequences of consecutive statements or expressions containing no transfer of control).

The aim of this paper is not to define or improve a current traceability technique, but rather to use related data within the ontology to help verify the compliance of SW specification with the outcome of its development. Among the possible techniques, we adopted the Aspect Oriented Programming technique [29, 49] as a means to perform requirements traceability. This required the instrumentation of the code so as to obtain the instances of the association between *Functional Requirement*s and *SW Component*s represented in the fragment of the model shown in Fig. 4.

Formally, let $r_i$ be a generic functional requirement and $c_j$ a generic SW component, we look for the relation $\mathcal{T} \subseteq 2^{FR} \times 2^{C}$ where $FR$ is the set of functional requirements and $C$ is the set of SW components. We define the generic element $T \in \mathcal{T}$ as $< T_r, T_c >$ where $T_r \in 2^{FR}$ and $T_c \in 2^{C}$. In doing so, $< T_r, T_c > \in \mathcal{T}$ means that a set of SW components $T_c$ are related to a set of functional requirements $T_r$. Abusing of terms, we call a component $c_i$ *necessary* for a functional requirement $r_j$ if

$$\forall T \in \mathcal{T} : r_j \in T_r \Rightarrow c_i \in T_c,$$

we call a component $c_i$ *potential* for a functional requirement $r_j$ if

$$\exists T', T'' \in \mathcal{T} : r_j \in T'_r \wedge r_j \in T''_r \Rightarrow c_i \in T'_c \wedge c_i \notin T''_c.$$

As in [15], we express a usage degree ($UG$) of a SW component $c_i$ in the implementation of a functional requirement $r_j$, accounting for how many methods $M$ of $c_i$ are executed in realizing $r_j$:

$$UG(c_i, r_j) = \frac{M(c_i, r_j)}{M(c_i)}.$$

Instances of the association of Fig. 4 between functional requirements and SW components are obtained for each functional requirement extracting the components *necessary* for its implementation and then importing them in the ontology. As a result the ontology provides the ground to perform verification of the compliance of SW implementation with its specification.

### *3.3 Inferring new concepts for SW Development Assurance Level Verification*

The aim of the SW Development Assurance Level Verification phase (see Fig. 4) is to provide a verification of the level of assurance attained in the development of SW components, measuring the degree of compliance with the SW-DAL assigned to the SW component during the Detailed Functional FMEA (see Section 2.3). Each level of development assurance is associated to a set of required predicates about the values of code and process metrics. A set of predicates for a functional requirement $r$ with criticality value $c$ has the form

$$\mathcal{P}_{c,r} = \left\{ X_1 \lesseqgtr k_1, \ldots, X_n \lesseqgtr k_n, Y_1 = s_1, \ldots, Y_m = s_m \right\} \text{ with } n, m \in \mathbb{N}$$

where, referring to the fragment of the ontological model shown in Fig. 4, $X_i$ and $Y_j$, with $i = 1 \ldots n$ and $j = 1 \ldots m$, are instances of *SW Parameter* and *Prevention Mitigation* classes, respectively, while $k_i$ and $s_j$ are instances of *SW Parameter Accountability* and *Prevention Mitigation Accountability* classes, respectively. If all the SW components contributing to the realization of $r$ are implemented with values of $X_i$ lower (greater) than $k_i$ and $Y_j$ equal to $s_j$, then $r$ is considered rigorously implemented.

A concrete example coming from our *reference context* of a set of predicates is

$$\mathcal{P}_{c,r} = \{CC < 5, TC = \text{“all edges”}\}$$

where $CC$ stands for the McCabe's cyclomatic complexity and $TC$ stands for the testing coverage, two metrics playing an important role in our case study project.

Predicates can be operatively verified by collecting values of both code and process metrics. Several tools supporting static analysis functionalities can be used to extract values

of code metrics [53, 37, 41], while data regarding process metrics can be extracted directly from the documentation.

Once data relative to metrics are available, the validation process for a functional requirement implementation consists in checking whether each SW component implementing it satisfies the predicates. This will be shown in Section 4.1.

## 4 Practical experience

The ontology abstraction of the proposed methodology can be directly converted into an advanced SW architecture incorporating the ontological model. This has been done by implementing a web application, called *RAMSES* (Reliability Availability Maintainability and Safety Engineering Semantics), built on top of a stack of semantic web technologies and standards.

The web application architecture is shortly sketched in Fig. 5. The *Presentation Layer* represents the interface between the user and the *Domain Layer*, which is in charge of the application logic and the data processing functionalities and is implemented using *Plain Old Java Object* (*POJO*). The *Data Layer* is responsible for data representation and conceptualization and is realized through an *Ontological Model*. The *Object-to-Ontology Mapping Layer* bridges the gap between the *Domain Layer* and the *Data Layer* solving the *impedance mismatch*, i.e. the conceptual distance between the object model and the ontological model. In so doing, the domain logic is captured by the ontological model, enabling the generalization of the application logic to adapt it as the concepts describing data change.

Use of ontologies brings about a number of relevant benefits: i) each SW-FMEA project can be represented in OWL form and then it can be exported and imported through this format, enhancing reusability and interoperability; ii) the construction of SW-FMEA worksheets and other reports is reduced to the extraction of proper data from the result set generated by a SPARQL query, which in turn is automatically resolved by an ontological reasoner; iii) new information can be inferred from the knowledge base by the ontological reasoner by means, for instance, of predefined SWRL rules.

### *4.1 Basic Tool Capabilities*

Fig. 6 shows the Use Case Diagram representing the basic functionalities of the tool. The user is involved in the following cases: a) *CRUD* (Create, Retrieve, Update, Delete) operations on *Entities* involved in the methodology (e.g. functional requirements, SW components, functionalities); b) the addition/removal of an association between *Entities*; c) the generation and the view of SW-FMEA worksheets; d) the visualization of hierarchical views. The administrator is responsible for the activities of management of users.

As mentioned previously, the user populates the extensional part of the ontology with data produced along the development life-cycle, e.g. functional requirements, structural elements and their functionalities. This enables the execution of basic activities such as the production of SW-FMEA worksheets and the generation of hierarchical views of both structural and functional elements. In our *reference context*, the worksheet assumes the generic form of Fig. 7, standardized in ECSS. However, depending on the stage of the analysis, the tool can generate worksheets in which some fields are omitted.

Part of the concepts contained in the ontology stands for data categories contained in the worksheet. For instance, *Item*, *Functionality*, *Failure Mode*, *Operation Modes* and

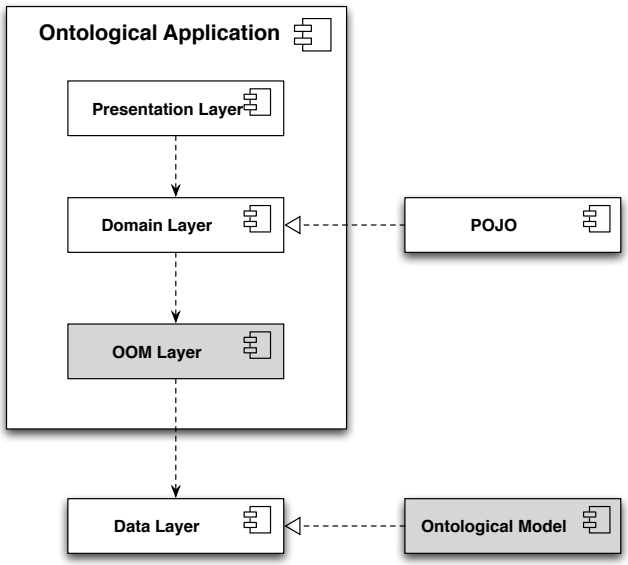


**Figure 5** Three-tier ontological architecture of a web application.

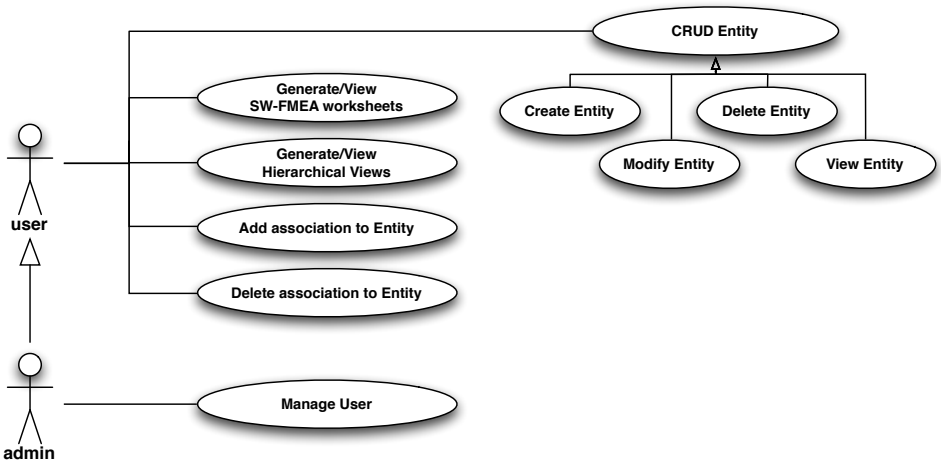


**Figure 6** Use Case Diagram representing the basic functionalities supported by RAMSES.

Failure Modes and Effects Analysis - Worksheets
System _____
Mission _____

| Item Number | Item / Block | Functionality | Failure Mode (+ unique id) | Failure Causes | Mission Phases/Op. Modes | Failure Effects | Severity | Failure Detection Methods | Comp. Provisions | Corrective Actions | Remarks |
|---|---|---|---|---|---|---|---|---|---|---|---|

**Figure 7** The format of a row in the SW-FMEA worksheet, as standardized in ECSS.

*Failure Effects* correspond to classes of the ontology, while *Item Number*, *Failure Causes*, *Severity*, *Failure Detection Methods*, *Compensating Provisions*, *Corrective Actions* and *Remarks* correspond to properties. Thanks to a query language as SPARQL, the ontology can be queried to extract the concepts' instances to fill the worksheet. As far as the ontology is concerned, this is written in OWL and organized as *triples* (or *statements*) in the form of ⟨*subject,predicate,object*⟩, where *subject* is the concept described by the triple, *predicate* describes a relationship between *subject* and *object* which, in turn, is a concept as well. The SPARQL query shown in Listing 1 automatically obtains the generic SW-FMEA worksheet of Fig. 7.

```
SELECT ?idComponent ?component ?functionality ?failureMode ?failureCause
       ?opMode ?failureEffect ?detMethod ?severityLevel ?compProv ?corrAct
       ?remarks
WHERE { ?component rdf:type <urn:ramses#SWComponent> .
        ?component <urn:ramses#hasItemId> ?idComponent .
        ?component <urn:ramses#hasFunctionality> ?functionality .
        ?functReq <urn:ramases#isImplementedBy> ?component .
        ?functReq <urn:ramses#isInOperationMode> ?opMode .
        ?functionality <urn:ramses#hasFailureMode> ?failureMode .
        ?failureCause <urn:ramses#isCausesOf> ?failureMode .
        ?failureMode <urn:ramses#hasEffect> ?failureEffect .
        ?failureMode <urn:ramses#hasSeverityLevel> ?severityLevel .
        ?failureMode <urn:ramses#hasDetectionMethod> ?detMethod .
        ?failureMode <urn:ramses#hasCompensatingProvision> ?compProv .
        ?failureMode <urn:ramses#hasCorrectiveAction> ?corrAct .
        ?failureMode <urn:ramses#hasRemarks> ?remarks }
```

Listing 1 A SPARQL query producing a result set comprising the values for the construction of the SW-FMEA worksheet.

RAMSES can also build hierarchical views of both structural and functional elements, providing a clearer picture of the system, by aggregating data scattered in different documents produced along the life-cycle. Fig. 8 shows the hierarchical view of structural elements obtained through the execution of the SPARQL query of Listing 2. This view corresponds to the left side of the model of Fig. 3, which, however, does not include the concept of *SubProgram*. In fact, the *SubProgram* entity has been added to the ontological class hierarchy between *CSCI* and *SW Component*, in order to adapt the (general) model of Fig. 8 to the needs of our *reference context*. This witnesses the capability of our methodology to adjust to different design styles.

```
SELECT ?CSCI ?SubProgram ?SWComponent ?SWModule
WHERE { ?CSCI rdf:type <urn:ramses#CSCI> .
        ?CSCI <urn:ramses#hasSubProg> ?SubProgram .
        ?SubProgram <urn:ramses#hasSWComponent> ?SWComponent .
        ?SWComponent <urn:ramses#hasSWModule> ?SWModule }
```

Listing 2 A SPARQL query producing a result set comprising the data used to build the hierarchical view.

### *4.2 Advanced Tool Capabilities*

In addition to the previously mentioned functions, the tool provides advanced functionalities to ease and improve the execution of the analysis. Fig. 9 shows the Use Case Diagram

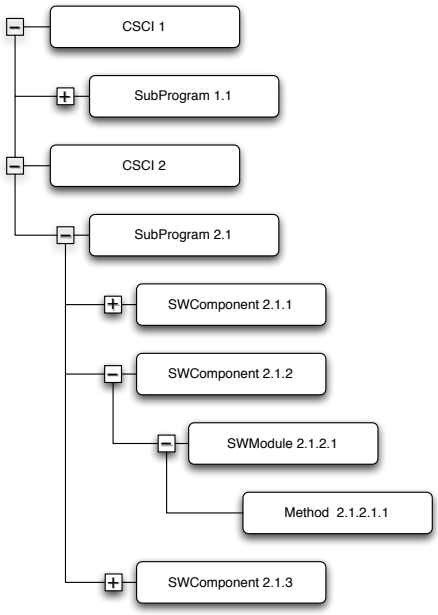


**Figure 8** Sample of structural hierarchy in use in our *reference context*.

representing the advanced functionalities of the tool. The user is involved in the following cases: a) the generation and the view of chains of failure modes; b) the identification of most critical functional requirements; c) the verification of the SW-DAL attained in the development of SW components.

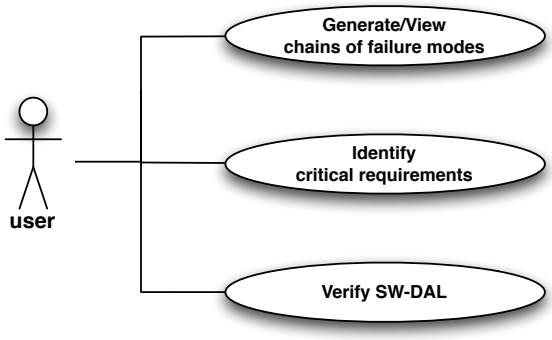


**Figure 9** Use Case Diagram representing the advanced functionalities supported by RAMSES.

The tabular nature of the SW-FMEA worksheets leads to a scattered representation of information about the system. Therefore, the search for a chain of failure modes causing a failure effect compels the analyst to jump from one row to another of the worksheets, following the links contained in cell labeled “Failure Causes” (Fig. 7). On the contrary, the ontology of Fig. 3 makes explicit the chain by the *Failure Mode* and *Failure Effect* classes

and by the *is_cause_of* and *has_FE* properties, simplifying the search of the failure modes leading to a given failure effect. In fact, instances of the entities involved in the chain can be easily retrieved through a SPARQL query. In other words, the ontology provides the tool the capability of gathering information which is hidden (e.g. scattered throughout the worksheets) in a classical SW-FMEA process.

When dealing with a large number of functional requirements, it is desirable that the analyst focuses his attention on the most critical ones. Leveraging the reasoning capability provided by the ontological model, the tool provides automatic identification of the most critical functional requirements. In our case, the criticality value of a functional requirement can be obtained through a set of SWRL rules. For instance, Listing 3 shows an SWRL rule, written in human readable syntax [24], which states that a functional requirement `?z` has criticality value "highly critical" if it is related to an operation mode `?x` with criticality value "highly critical" and it is affected by a failure mode `?y` with severity level equal to 4.

```
ramses:FunctionalRequirement(?z) ∧
∧ ramses:OperationMode(?x) ∧
∧ ramses:hasCriticalityValue(?x,"highly critical") ∧
∧ ramses:FailureMode(?y) ∧
∧ ramses:hasSeverityLevel(?y,4) ∧
∧ ramses:is_In_OP(?z, ?x) ∧
∧ ramses:has_FM(?z, ?y) ⇒
⇒ ramses:hasCriticalityValue(?z,"highly critical")
```

Listing 3 SWRL rule stating the conditions to define "highly critical" the criticality value of a functional requirement.

The reasoning capability is also crucial for the SW-DAL Verification phase. As reported in Section 3.3 a functional requirement is rigorously implemented if the related SW components satisfy a predefined set of predicates. For instance, to verify one of the predicates exemplified at the end of that section, the SWRL rule of Listing 4 can be used. The

```
ramses:hasCriticalityValue(?f,c) ∧
∧ ramses:isImplementedBy(?r,?ud) ∧
∧ ramses:hasUsedComponent(?ud,?swc) ∧
∧ ramses:hasSWModule(?swc,?swm) ∧
∧ ramses:hasMethod(?swm,?m) ∧
∧ ramses:hasSWParameterAcc(?m,?spa) ∧
∧ ramses:hasLinkedParameter(?spa, ?sp) ∧
∧ ramses:hasName(?sp,``cyclomatic complexity'') ∧
∧ ramses:hasParamValue(?spa, ?pv) ∧
∧ swrlb:greaterThan(?pv, 5) ⇒
⇒ ramses:NotRigorous(?f)
```

Listing 4 SWRL rule verifying the satisfaction of a predicate.

rule verifies whether for a functional requirement `?r` with criticality value `c`, there exists a SW component `?swc` which is implemented by a SW module `?swm` containing a method `?m` having a McCabe's cyclomatic complexity `?sp` greater than 5. If the rule is verified, the predicate is violated and the functional requirement is considered not rigorously implemented. Therefore taking advantage of the ontology, RAMSES is able to recommend appropriate actions to the analyst.

### 4.3 Practical experimentation on the AASTR SW

Experimentation of RAMSES tool was done in the context of the AASTR project, which targeted the development of the Active Pixel Sensor (APS) Autonomous Star Tracker for the Bepi Colombo Mission under the control of Astrium Space Deutschland (ASD) and European Space Agency (ESA). The main purpose of the AASTR SW is acquiring data from the APS detector, performing the star clustering and filtering, and then calculating the attitude and the angular rate. Attitude propagation allows tracking of the stars. The AASTR SW incorporates the algorithms for the attitude determination and the on-board stars catalogue with the reference stars and the related patterns (triads).

The application of the methodology starts with the Top Level Functional FMEA phase. The analyst inserts information about the CSCIs composing the system and the operation modes characterizing it. The related functional requirements are then defined and their failure modes with the respective failure effects are inductively discovered; all of them are loaded in the ontological base through the tool interface. Fig. 10 shows the screenshot of the interface for entering *Functional Requirement*s instances. To give an idea of our case study, AASTR SW is composed by a single CSCI, having 8 possible operation modes and 243 functional requirements. Once the previous data have been input, the chain of failure

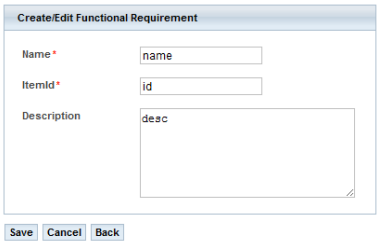


**Figure 10** Screenshot showing the tool interface provided to enter a new instance of *Functional Requirement* in the ontological model.

modes discussed in the previous section can be visualized, helping the analyst in focusing the causes of a certain failure effect. Fig. 11 shows how such chains are displayed by the tool interface. As reported in Section 2.2, the functional requirement criticality value is a function of both the criticality value of the operation mode it refers to and the severity level of the failure modes affecting it. In AASTR SW project, criticality values of operation modes are classified as in Table 1, that is according to both the kind of memory (PROM or EEPROM) the SW is resident in and the possibility of applying patches on that SW.

| | **patch** | **semi-patch** | **no patch** |
|---|---|---|---|
| **PROM** | | critical | highly critical |
| **EEPROM** | lowly critical | | |

**Table 1** Operation mode criticality values

The severity level of a failure mode is classified according to the ECSS standard: *minor* or *negligible*, *major*, *critical*, and *catastrophic*. Each level is mapped to a numerical value

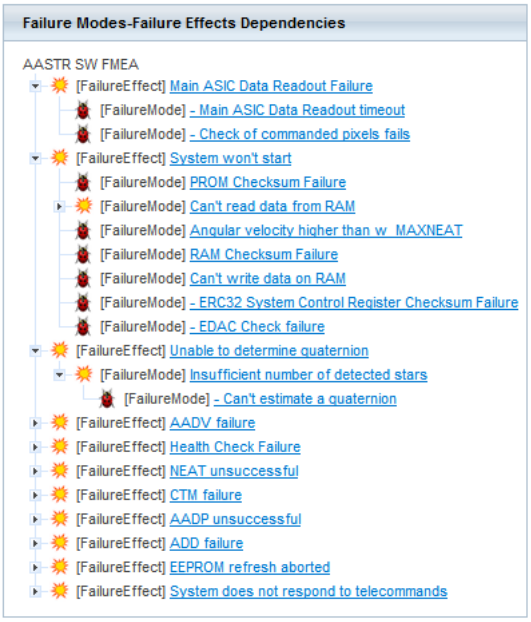


**Figure 11** Failure modes chains leading to failure effects as visualized by the RAMSES interface.

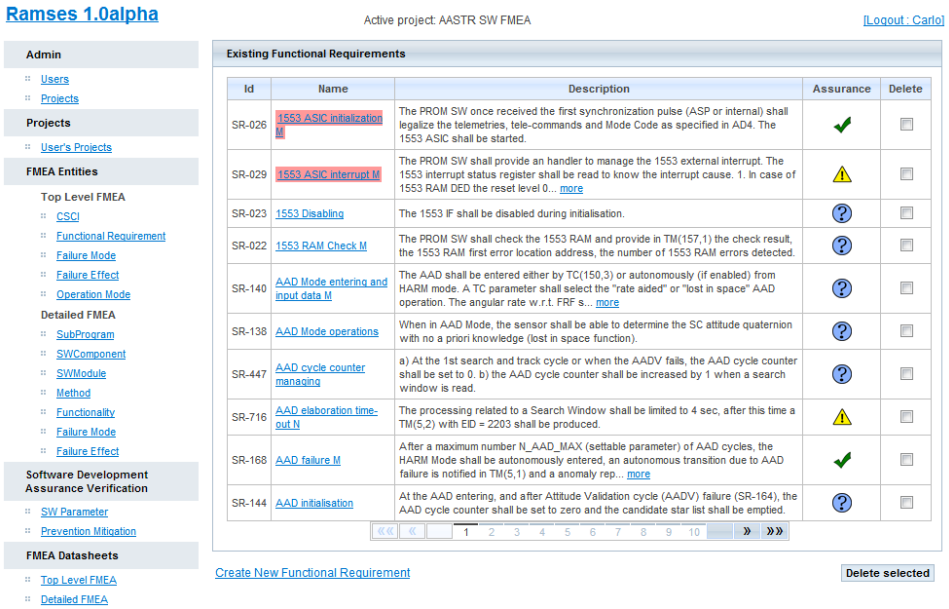


**Figure 12** List of functional requirements.

ranging from 1 to 4 respectively. Using rules such that in Listing 3, the reasoner can identify the most critical functional requirements. These are highlighted in the screen of Fig. 12.

Table 2 shows the functional requirement criticality value as a function of the maximum operation mode criticality value the requirement is related to, and the maximum failure mode severity level the requirement is affected by.

| | | **Operation Mode Criticality Value** | | |
|---|---|---|---|---|
| | | **lowly critical** | **critical** | **highly critical** |
| **Failure mode Severity Level** | **1** | lowly critical | critical | highly critical |
| | **2** | critical | critical | highly critical |
| | **3** | critical | highly critical | highly critical |
| | **4** | highly critical | highly critical | highly critical |

**Table 2** Functional Requirement criticality values.

Fig. 13 shows the ongoing Top Level Functional FMEA worksheet that is automatically generated by the tool at this point of the analysis.

**Top Level Functional FMEA Datasheet**

AASTR SW

Functional Requirement: Attitude coarse check (AADP)

| Failure Mode | Failure Effects | Severity Level | Detection Method / Observable Symptoms | Compensating Provisions | Corrective Actions | Remarks |
|---|---|---|---|---|---|---|
| Insufficient number of predicted stars in AADP | ▪ AADP unsuccessful | 5 | The following events shall be issued: · Not Enough stars have been predicted the event TM(5,3) shall be issued with EID=2316 (NOT_ENOUGH_PREDICTED_STARS_AAD) | | | |
| Insufficient number of segments or clusters in AADP | ▪ AADP unsuccessful | 5 | Telemetry | When the AADP fails, the star acquisition (SR-146) shall be resumed addressing the next search window and continuing star tracking of candidate stars found previously | | |

Functional Requirement: PROM loading and check

| Failure Mode | Failure Effects | Severity Level | Detection Method / Observable Symptoms | Compensating Provisions | Corrective Actions | Remarks |
|---|---|---|---|---|---|---|
| RAM Checksum Failure | ▪ System won't start | 8 | System alert (telemetry) | try again. | | if retrying does not work, there's need of a local repair. |
| PROM Checksum Failure | ▪ System won't start | 8 | system alert (telemetry) | try again. | | if not working, needs substitution. |

**Figure 13** An example of Top Level Functional FMEA worksheet as generated by RAMSES through a SPARQL query.

In the subsequent phase, the Detailed Functional FMEA, the analyst proceeds in a similar manner. Collected information (i.e. instances of *SubProgram*, *SW Component*, *SW Module*, *Method*, and *Functionality*) is added to the ontological base and can be browsed through the tool interface. Furthermore, a Detailed Functional FMEA worksheet, i.e. focusing on failure modes affecting functionalities, can be generated. Also, hierarchical views such the one generated by the query of Listing 2, can be displayed as in Fig. 14.

In the last phase, SW-DAL Verification, information about SW parameters and prevention mitigations is added to the methods. In the AASTR SW project, 24 metrics have been used. Though only the following metrics have been considered: *McCabe cyclomatic*

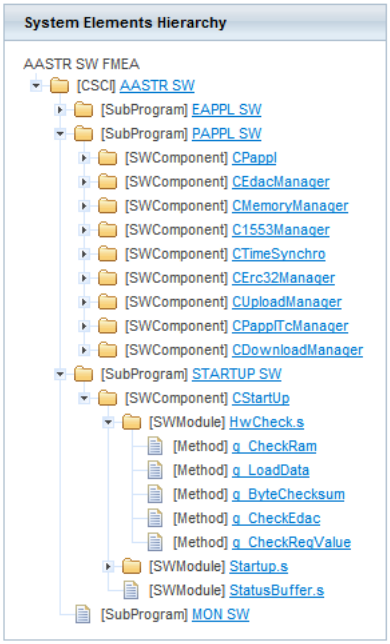


**Figure 14** The system hierarchical view as generated by the RAMSES tool through a SPARQL query on the ontology.

*number*, *number of source code instructions*, *number of nested control structures* and *test coverage*. In any case, new metrics can be dynamically added at any time through the application interface. As the metrics values are inserted in the ontological base, the reasoner controls if the required level of assurance for a functional requirement is reached by the current implementation using a rule such that of Listing 4. Fig. 12 shows the recommendation issued by RAMSES when a violation is detected by the reasoner: highlighted elements correspond to highly critical functional requirements; a warning signal visualized in the "Assurance" column indicates a violation of a required SW-DAL; a tick indicates a rigorously implemented functional requirement; a question mark indicates a functional requirement with SW-DAL not still defined or with components not still implemented. At this point, the analyst can enter in the functional requirement details page, where the methods, the related SW module and the related SW component causing the violation of one or more predicate are listed in the "Methods violating SW-DAL predicates" section (Fig. 15), indicating where a deeper effort should be put (e.g. modifying the code or using better testing strategies) to attain the expected level.

## 5 Conclusions

We proposed a methodology based on an ontological formalization of the SW-FMEA process which is robust enough to enforce cohesion and consistency among information elements acquired along different phases of the development process, possibly contributed by different parties. The methodology is structured into three phases that are related to the stages comprised in a SW life-cycle. The methodology was tuned to fit to the V-Model

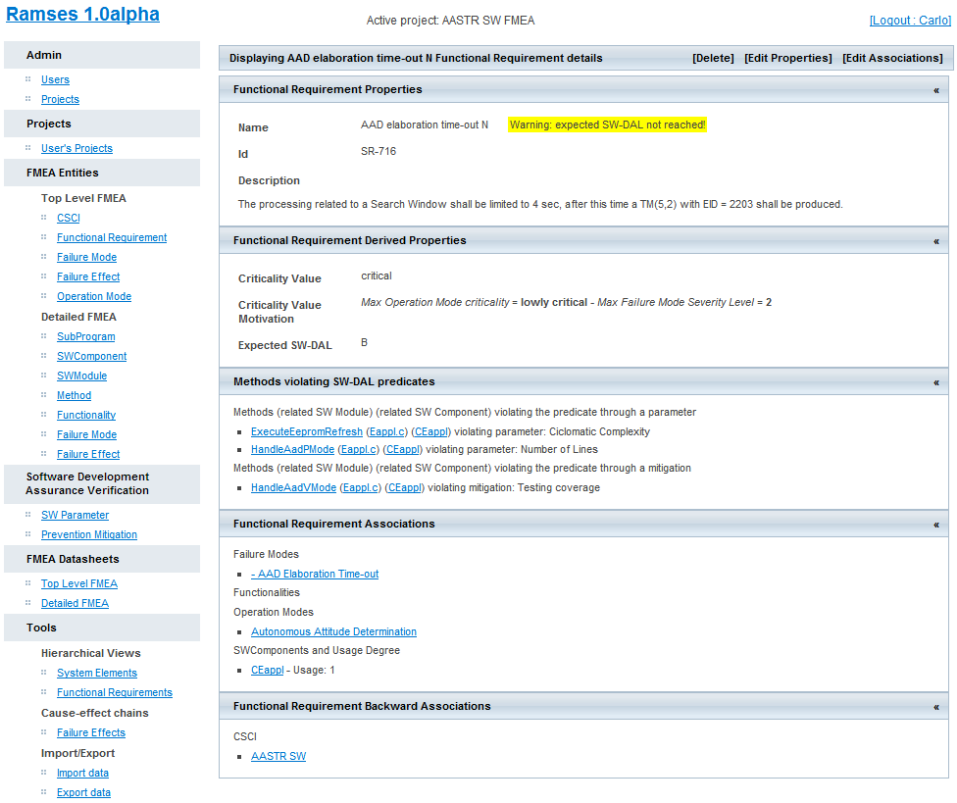


**Figure 15** Not rigorously implemented functional requirement details page.

adopted in our *reference context*; however, it can be effectively tailored to different development policies, redefining the relations between the three phases and the stages of the life-cycle considered. The methodology can also be adapted to different standards, which may prescribe the collection of different information pieces during SW-FMEA process, leading to the generation of different structures for the worksheet. In this paper, we referred to the ECSS standard.

The formalized conceptualization enables effective application of reasoning tools aiding the accomplishment of crucial and effort-expensive activities. This includes basic functionalities such as efficient production of SW-FMEA worksheets, as well as hierarchical views of functional and structural elements. It also provides the ground for advanced functionalities such as: identification of the most critical functional requirements, automated mapping between functional requirements and SW components, visualization of failure mode-failure effect chains, automatic identification of SW components impairing the required quality of service.

The ontological model opened the way to the implementation of a tool (RAMSES) built on well-established semantic web technologies. RAMSES automates the processing of SW-FMEA data by exploiting query capabilities of SPARQL as well as the inference capabilities of an off-the-shelf reasoner. The tool has been tailored to a specific industrial context, however, it can be adapted to different industrial processes/life-cycles, leveraging the adaptability of the ontological model.

The tool was tested and verified with data referring to an already accomplished project (AASTR), and it is being applied in the frame of a new project of Selex Galileo, which is currently under development.

## References

[1] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11 – 33, jan.-march 2004.

[2] I. Bicchierai, G. Bucci, C. Nocentini, and E. Vicario. An ontological approach to systematization of SW-FMEA. In *Proceedings of the 31st Int. Conf. on Computer Safety, Reliability, and Security*, SAFECOMP'12. Springer-Verlag, 2012.

[3] P. Bieber, R. Delmas, and C. Seguin. DALculus: theory and tool for development assurance level allocation. In *Proceedings of the 30th international conference on Computer safety, reliability, and security*, SAFECOMP'11, pages 43–56, Berlin, Heidelberg, 2011. Springer-Verlag.

[4] H. Boley, T. Athan, A. Paschke, S. Tabet, B. Grosof, N. Bassiliades, G. Governatori, F. Olken, and D. Hirtle. *Specification of Deliberation RuleML 1.0*. http://wiki.ruleml.org/index.php/Specification, 2012.

[5] J. Bowles and C. Wan. Software Failure Modes and Effects Analysis for a small embedded control system. In *Reliability and Maintainability Symposium, 2001. Proceedings. Annual*, pages 1 –6, 2001.

[6] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. http://www.w3.org/TR/2008/REC-xml-20081126/, November 2008.

[7] D. Brickley and R. Guha. *RDF Vocabulary Description Language 1.0: RDF Schema*. http://www.w3.org/TR/rdf-schema/, February 2004.

[8] BSI - British Standard Institution. BS5760 Reliability of Systems, Equipment and Components Part 5. Guide to Failure Modes, Effects and Criticality Analysis (FMEA and FMECA). Technical report, British Standard Institution, 1991.

[9] G. Bucci, V. Sandrucci, and E. Vicario. An Ontological SW Architecture Supporting Agile Development of Semantic Portals. In *Software and Data Technologies*, volume 22 of *Communications in Computer and Information Science*, pages 185–200. Springer Berlin Heidelberg, 2009.

[10] BWB - Federal Office for Military Technology and Procurement of Germany. *V-Model 97, Lifecycle Process Model-Developing Standard for IT Systems of the Federal Republic of Germany. General Directive No. 250*, June 1997.

[11] Byteworx. Byteworx FMEA official website. http://www.byteworx.com/.

[12] CENELEC European Committee for Electrotechnical Standardization. *CENELEC EN 50128 Railway applications - Communications, signalling and processing systems - Software for railway control and protection systems*, March 2001.

[13] CENELEC European Committee for Electrotechnical Standardization. *CENELEC EN 50126 Railway applications - The Specification and Demonstration of Reliability, Availability,Maintainability and Safety (RAMS)*, February 2007.

[14] L. Dittmann, T. Rademacher, and S. Zelewski. Performing FMEA Using Ontologies. In *Proceedings of 18th International Workshop on Qualitative Reasoning (QR04)*, pages 209–216, Northwestern University, Evanston, USA, August 2004.

[15] M. Eaddy, A. Aho, and G. C. Murphy. Identifying, assigning, and quantifying crosscutting concerns. In *Proceedings of the First International Workshop on Assessment of Contemporary Modularization Techniques*, ACoM '07, pages 2–, Washington, DC, USA, 2007. IEEE Computer Society.

[16] V. Ebrahimipour, K. Rezaie, and S. Shokravi. An ontology approach to support FMEA studies. *Expert Systems with Applications*, 37(1):671 – 677, 2010.

[17] T. Eisenbarth, R. Koschke, and D. Simon. Locating features in source code. *IEEE Trans. Softw. Eng.*, 29:210–224, March 2003.

[18] European Cooperation for Space Standardization. *ECSS-E-ST-40C Space Engineering -Software*, March 2009.

[19] European Cooperation for Space Standardization. *ECSS-Q-ST-30-02C Space product assurance - Failure modes, effects (and criticality) analysis (FMEA/FMECA)*, March 2009.

[20] European Cooperation for Space Standardization. *ECSS-Q-ST-80C Space product assurance - Software product assurance*, March 2009.

[21] D. C. Fallside and P. Walmsley. *XML Schema Part 0: Primer Second Edition*. http://www.w3.org/TR/xmlschema-0/, February 2004.

[22] P. Goddard. Software FMEA techniques. In *Reliability and Maintainability Symposium, 2000. Proceedings. Annual*, pages 118 –123, 2000.

[23] T. R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[24] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. http://www.w3.org/Submission/SWRL/, May 2004.

[25] IEEE Computer Society. IEEE Guide to Software Requirements Specifications (Std 830 - 1993) . Technical report, IEEE, 1993.

[26] IEEE Computer Society. IEEE Recommended Practice for Software Design Descriptions (Std 1016 - 1998) . Technical report, IEEE, 1998.

[27] International Electrotechnical Commission. *IEC-60812 Analysis techniques for system reliability - Procedure for Failure Mode and Effects Analysis (FMEA)*, 1985.

[28] M. Jang and J.-C. Sohn. Bossam: An extended rule engine for OWL inferencing. In G. Antoniou and H. Boley, editors, *Rules and Rule Markup Languages for the Semantic Web*, volume 3323 of *Lecture Notes in Computer Science*, pages 128–138. Springer Berlin / Heidelberg, 2004.

[29] G. Kiczales, J. Lamping, A. Mehdhekar, C. Maeda, C. V. Lopes, J. Loingtier, and J. Irwin. Aspect-Oriented Programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP)*, pages 53–60. Springer-Verlag, 1997.

[30] G. Klyne and J. J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax*. http://www.w3.org/TR/rdf-concepts/, February 2004.

[31] Y. Koji, Y. Kitamura, and R. Mizoguchi. Ontology-based transformation from an extended functional model to FMEA. In *In Proc. of ICED 05*, 2005.

[32] E. Kozlenkov and M. Schroeder. PROVA: Rule-based java-scripting for a bioinformatics semantic web. In *International Workshop on Data Integration in the Life Sciences DILS*. Springer, 2004.

[33] D. L. McGuinness and F. van Harmelen. OWL Web Ontology Language. http://www.w3.org/TR/owl-features/, February 2004.

[34] R. J. Mikulak, M. Robin, and B. Michael. *The Basics of FMEA*. Productivity Press, 2008.

[35] National Aeronautics and Space Administration. *NASA Software Safety Guidebook NASA-GB-8719.13 - NASA TECHNICAL STANDARD*, March 2004.

[36] Object Management Group. *Ontology Definition Metamodel v1.0*, 2009.

[37] Parasoft. Parasoft. http://www.parasoft.com.

[38] H. Pentti and H. Atte. *Failure Mode and Effects Analysis of software-based automation systems - STUK-YTO-TR 190*. VTT Industrial Systems - STUK, August 2002.

[39] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF. http://www.w3.org/TR/rdf-sparql-query/, January 2008.

[40] PTC Product Development Company. Windchill FMEA official website. http://www.ptc.com/product/windchill/fmea.

[41] QA Systems - The Software Quality Company. Cantata++. http://www.qa-systems.com/cantata.html.

[42] Radio Technical Commission for Aeronautics. *DO-178B, Software Considerations in Airborne Systems and Equipment Certification*, 1992.

[43] E. S. Raymond. *The New Hacker's Dictionary*. The MIT Press, Cambridge, 1991.

[44] D. J. Reifer. Software Failure Modes and Effects Analysis. *Reliability, IEEE Transactions on*, R-28(3):247 –249, aug. 1979.

[45] ReliaSoft. XFMEA official website. http://www.reliasoft.com/xfmea/.

[46] R. A. Sahner, K. S. Trivedi, and A. Puliafito. *Performance and reliability analysis of computer systems: an example-based approach using the SHARPE software package*. Kluwer Academic Publishers, Norwell, MA, USA, 1996.

[47] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *J. Web Sem.*, 5(2):51–53, 2007.

[48] Society of Automotive Engineers'. *SAE J-1739 Potential Failure Mode and Effects Analysis in Design (Design FMEA) and Potential Failure Mode and Effects Analysis in Manufacturing and assembly Processes (Process FMEA) Reference Manual*, 1994.

[49] O. Spinczyk, A. Gal, and W. Schröder-Preikschat. AspectC++: An Aspect-Oriented Extension to C++. In *In Proceedings of the 40th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS) Pacific 2002*, pages 53–60, 2002.

[50] D. H. Stamatis. *Failure Mode and Effect Analysis: FMEA from Theory to Execution*. Amer Society for Quality, 2003.

[51] United States Department of Defense. MIL-STD-1629A Procedures for Performing a Failure Mode, Effects and Criticality Analysis. Technical report, US Department of Defense, November 1980.

[52] United States Department of Defense. MIL-STD-498, MILITARY STANDARD FOR SOFTWARE DEVELOPMENT AND DOCUMENTATION. Technical report, US-DoD, 1994.

[53] USC Center for Software Engineering. UCC: Unified Code Count. http://sunset.usc.edu/research/CODECOUNT/.

[54] R. Wirth, B. Berthold, A. Krämer, and Peter. Knowledge-Based Support of System Analysis for Failure Mode and Effects Analysis. *Engineering Applications of Artificial Intelligence*, 9:219–229, 1996.

[55] W. E. Wong, S. S. Gokhale, and J. R. Horgan. Quantifying the closeness between program components and features. *J. Syst. Softw.*, 54:87–98, October 2000.

[56] W. E. Wong, J. R. Horgan, S. S. Gokhale, and K. S. Trivedi. Locating program features using execution slices. In *Proceedings of the 1999 IEEE Symposium on Application - Specific Systems and Software Engineering and Technology*, ASSET '99, pages 194–203, Washington, DC, USA, 1999. IEEE Computer Society.