Artificial Intelligence and Law (2021) 29:587–627
https://doi.org/10.1007/s10506-021-09285-5

ORIGINAL RESEARCH

Compliance-aware engineering process plans: the case of space software engineering processes

Julieth Patricia Castellanos-Ardila[1] · Barbara Gallina[1] · Guido Governatori[2]

Accepted: 14 February 2021 / Published online: 20 March 2021

Abstract

Safety-critical systems manufacturers have the duty of care, i.e., they should take correct steps while performing acts that could foreseeably harm others. Commonly, industry standards prescribe reasonable steps in their process requirements, which regulatory bodies trust. Manufacturers perform careful documentation of compliance with each requirement to show that they act under acceptable criteria. To facilitate this task, a safety-centered planning-time framework, called ACCEPT, has been proposed. Based on compliance-by-design, ACCEPT capabilities (i.e., processes and standards modeling, and automatic compliance checking) permit to design Compliance-aware Engineering Process Plans (CaEPP), which are able to show the planning-time allocation of standard demands, i.e., if the elements set down by the standard requirements are present at given points in the engineering process plan. In this paper, we perform a case study to understand if the ACCEPT produced models could support the planning of space software engineering processes. Space software is safety and mission-critical, and it is often the result of industrial cooperation. Such cooperation is coordinated through compliance with relevant standards. In the European context, ECSS-E-ST-40C is the de-facto standard for space software production. The planning of processes in compliance with project-specific ECSS-E-ST-40C applicable requirements is mandatory during contractual agreements. Our analysis is based on qualitative criteria targeting the effort dictated by task demands required to create a CaEPP for software development with ACCEPT. Initial observations show that the effort required to model compliance and processes artifacts is significant. However, such an effort pays off in the long term since models are, to some extend, reusable and flexible. The coverage level of the models is also analyzed based on design decisions. In our opinion, such a level is adequate since it responds to the information needs required by the ECSS-E-ST-40C framework.

Keywords Process compliance checking · Software process plan · ECSS-E-ST-40C

✉ Julieth Patricia Castellanos-Ardila
julieth.castellanos@mdh.se

Extended author information available on the last page of the article

1 Introduction

Safety-critical systems manufacturers have the duty of care[1] (Ladkin 2019), i.e., they should follow accepted practices of reasonable care, usually found in industry standards (Generowicz 2013). Failure or inadequate compliance with such standards could lead to legal risks, i.e., penalties (Cusumano 2004) and prosecutions (Ingolfo et al. 2011). For example, in 2015, The Volkswagen "Dieselgate" scandal (Walkinshaw 2017), i.e., emissions levels of the cars were not complying with emission standards, resulted in huge lost to the company (Blackwelder et al. 2016). Compliance with industry standards is relevant evidence for a jury to consider in a product liability action (Schwartz 2000). In England, the Health and Safety Executive has used compliance with IEC 61508 (IEC 2010) as a guideline for bringing legal actions if harm is caused by safety-critical systems (Ladkin 2019).

Industry standards demand documented evidence of responsibilities and agreements (Moyón et al. 2020). Usually, they place requirements on engineering processes (Eastaughffe et al. 1999), which should be planned at the beginning of the engineering activities (Gallina et al. 2018). Compliant engineering process plans are used to coordinate and track engineering progress, support contractual relationships between partners and agreements with certification bodies. In the context of the European project AMASS (Ruiz et al. 2016; de la Vara et al. 2019), a safety-centered planning-time framework, called ACCEPT (Automated Compliance Checking of Engineering Process plans against sTandards) (Castellanos Ardila 2019a, b), has been proposed to facilitate process compliance checking tasks. ACCEPT is based on Compliance-by-design (Lu et al. 2007), an approach aimed at integrating compliance requirements at design time, permitting to resolve compliance violations in engineering process plans before they are executed. ACCEPT is supported by rules-based technologies to automatically check if a compliance-aware engineering process plan (CaEPP) is designed, i.e., if the elements set down by the requirements (e.g., tasks, personnel, work products, techniques, and tools, as well as their properties) are present at given points in the engineering process plan. A CaEPP can show how and when the evidence will be produced, taking into account all the process-related requirements or their tailoring (i.e., adapted to the specific project conditions in a compliant form). A CaEPP is able to demonstrate intentional compliance (Siena et al. 2008), i.e., planning-time allocation of responsibilities, such that if every actor fulfills its duties, then the compliance is ensured.

ACCEPT uses Formal Contract Logic (FCL) (Governatori 2005), which provides a framework that unambiguously represents normative knowledge, i.e., obligations, prohibitions, and permissions. ACCEPT also uses the compliance checker Regorous (Governatori 2015), which provides an algorithm that determines whether an annotated process model is compliant with a specific set of FCL rules. The annotated process models required by Regorous, i.e., process

[1] In tort law, a duty of care is a legal obligation which is imposed on an individual requiring adherence to a standard of reasonable care while performing any acts that could foreseeably harm others (Icheku 2011).

enriched with compliance effects through annotations representing the formalized requirements, is provided via SPEM 2.0 (Systems & Software Process Engineering Metamodel) (OMG 2008). We chose SPEM 2.0, as opposed to other process modeling notations, for several reasons. (1) SPEM 2.0 is a standardized language, based on the Unified Modeling Language (UML) (OMG 2017). (2) SPEM 2.0-like artifacts can be captured freely via Eclipse Process Framework Composer (EPF-C) (Eclipse Foundation 2018) (recently ported to Eclipse Neon 4.6.3 (Javed and Gallina 2018a)). (3) SPEM 2.0 has the ability to capture several types of information. (4) SPEM 2.0 provides variability mechanisms that can be exploited for flexible process derivation. Such mechanisms are currently tool-supported via the composition of EPC-C with BVR (Base Variability Management Tool (SINTEF 2016)) (Javed and Gallina 2018b) included in the AMASS tool platform (de la Vara et al. 2020). (5) SPEM 2.0 elements can also be customized to permit the definition of a variety of artifacts. All these characteristics facilitate the modeling of process-related compliance artifacts, i.e., engineering processes and their elements, as well as standards requirements and their derived rulesets, annotated process plans, and workflows representations, which can be also reused, tailored and explicitly documented. EPF-C models can be ported to other tools, via model-driven transformations. Finally, SPEM 2.0 is widely accepted by the research community (Ruiz-Rube et al. 2012) and industry (Baumgarten et al. 2015).

In this paper, we perform a case study to understand if the ACCEPT produced models could support space manufacturers' needs in planning space software engineering processes. Space software is safety-critical since a failure could cause a mission disaster leading to financial losses, environmental pollution, and people's endangerment in case of manned missions (Rantala et al. 2017). Moreover, space software production is frequently the result of industrial cooperation. For example, the European space context consists of space agencies often acting as customers in projects, and companies, which act as suppliers, or as intermediate customers for subcontractors (Lill 2018). Meeting the highest levels of industry standards helps to coordinate such cooperation. In this context, ECSS-E-ST-40C is the de-facto standard for space software production. Thus, the planning of processes in compliance with project-specific ECSS-E-ST-40C applicable requirements is mandatory during contractual agreements. We have selected a portion of the ECSS-E-ST-40C (ESA 2009a) related to the design of the software items to perform our analysis, which is based on a set of well-defined qualitative criteria defined in Ghanavati et al. (2008). In particular, we target the effort dictated by task demands required to create a CaEPP for software development with ACCEPT. Initial observations show that the effort required to model compliance and processes artifacts is significant. However, such an effort pays off in the long term since models are, to some extend, reusable and flexible. The coverage level of the models is also analyzed based on design decisions. In our opinion, such a level is adequate since it responds to the information needs required by the ECSS-E-ST-40C framework.

The paper is organized as follows. In Sect. 2, we provide essential background. In Sect. 3, we present the case study design. In Sect. 4, we present the data collection. In Sect. 5, we present the case study analysis. In Sect. 6, we discuss the findings. In

Sect. 7, we present related work. Finally, In Sect. 8, we present the conclusion and future work.

2 Background

In this section, we present essential background information required in this paper.

2.1 Compliance with industry standards

Industry standards offer frameworks that encompass adequate practices refined by experts from historically successful experiences (Harkiolakis 2013) as well as knowledge and awareness of public policy, societal norms, and preferences (Leveson 2016). Organizations comply with industry standards (sometimes augmented with internal guidelines) to minimize legal risks (Kienle et al. 2012) since compliance is the demonstration that the organization acts under well-defined and acceptable criteria. In some industries, a compliance certification is mandatory to be able to sell products on a specific market, e.g., medical devices (U.S. FDA 1906). Compliance is also a mark that customers trust. For example, in space, standards requirements are intended to support the contractual negotiation by helping customers to formulate their requirements and suppliers to prepare their responses and to implement the work (ESA 2009b). Contracts are legally binding documents in which development freedom becomes limited. Thus, non-compliance is harmful to the success of organizations. In the remaining part of this section, we recall essential information regarding software process standards, and we focus on the software engineering standard that regulates the European space context.

2.1.1 Software process standards

In the past, software companies vacate liability for software errors by licensing it to a user that agreed that the company would not be liable for damages caused by errors in the code (Denning and Tedre 2019). This policy contributed to enforce the computer revolution. However, the software was limited to provide simple tasks and sometimes computational power for complex systems. Nowadays, the software is used to control most systems (including physical) involving potentially large and even catastrophic loses (Leveson 2020). Consequently, software projects are becoming critical in terms of legal aspects, e.g., software not delivered in time or with ill-defined functionality could lead to legal claims (Kalus and Kuhrmann 2013). In the safety-critical context, legal aspects are also related to each activity performed in its production (Cosgrove 2001; Buglione et al. 2010). The reason is that a well-defined process would make it difficult to exclude significant aspects of the software engineering aspects. Examples of inadequate software engineering process practices have been considered as one of the factors that cause Therac-25 radiotherapy machine's massive overdose (Leveson et al. 1995) and the failed launch of the ARIADNE 5 (Dowson 1997). Choices seem not to be either deliberately planned in the definition of the features created to force the plane BOEING 737-MAX to nose down, causing fatal accidents (Cruz and de Oliveira Dias 2020). Sound engineering processes present a structured collection of practices (SEI 2011). Companies that follow the process-related frameworks prescribed by industry standards tend to achieve more consistent results (O'Regan 2018). Legal risk can also be prevented since proofs of compliance can demonstrate that companies have taken correct steps while performing acts that could foreseeably harm others (Cosgrove 2001; Kienle et al. 2012). Software process standards do not restrict organizations from using a particular development lifecycle. Instead, the process framework focuses on what needs to be done. Sometimes, who should be involved in the process and the recommended techniques and tools to be used to achieve desirable results are also prescribed. Route maps may be indicated, but exact specifications on how the process should be done usually are not provided. In addition, software with high requirements, such as safety, requires detailed documentation according to regulations, which may imply the creation of very formal software processes (Kalus and Kuhrmann 2013). For this reason, a software process engineer is responsible for the selection, composition, and correct documentation of adequate software process elements aimed at achieving the required process goals (Gallina et al. 2016).

2.1.2 ECSS standards: focus on software engineering

The European Cooperation for Space Standardization (ECSS) developed a set of standards for use in all European space activities. The ECSS standard system includes three branches, i.e., Management (M), Engineering (E), and Product Assurance (Q). Handbooks (HB) guide the application of the requirements. The software engineering handbook, ECSS-E-HB-40A (ESA 2013), states that in a space software project, a customer-supplier business agreement should be established. The customer shall produce the project requirements documentation, which could be produced by using the ECSS Applicability Requirements Matrix (EARM). The EARM should have the list of applicable ECSS requirements with identifiers, applicability condition, i.e., applicable without change (A), applicable with modification (M), not applicable (D), and new generated requirement (N). The supplier responds with the ECSS Compliance Matrix (ECM), indicating the compliance for each requirement provided in the EARM. Partial compliance needs to be detailed, such that the customer can assess the extent to which the objective of the ECSS is covered. Noncompliance also needs to be investigated in terms of feasibility and acceptability in the scope of the project. When a space project starts, the supplier has to identify a suitable software lifecycle process. Thus, discussions about the technical specifications based on the requirements baseline must start early in the lifecycle process (Ahmad et al. 2010).

In space software development, the requirements prescribed by the standard ECSS-E-ST-40C (ESA 2009a), which determines mission (non-safety) requirements on how the goals can be achieved, should be applied. Such requirements could be tailored, i.e., adapted for the characteristics of the project. For example, ECSS-E-ST-40C-Annex R, provides a pretailoring based on safety criticality categories, which rank from catastrophic to negligible (prescribed in ECSS-Q-ST-40C (ESA

2017)). Thus, mission requirements have an inherent relationship with safety issues. Further tailoring should be analyzed in the scope of the project and its consequences assessed and documented. If requirements are tailored out, the associated expected outputs are also tailored out. Table 1 recalls a set of requirements from the phase *5.5. Software Design and Implementation Engineering Process*, particularly the activity *5.5.2. Design of Software Items*. The inputs of this activity are the Technical Specification of the Software Components (TSSC), the Architectural Design (AD) the Design Justification (DJ), and the Preliminary Design Review (PDR). During the *detailed design review* the expected items of every requirement are revised and compiled in eight work products, i.e., the DDF (Design Definition File), SDD (Software Design Document), CDR (Critical Design Review), TS (Technical Specification), ICD (Interface Control Document), SUM (Software User Manual), DJF (Design Justification File) and SUITP (Software unit-integration Test Plan).

Table 1 Activity 5.5.2: Design of the software items

<table>
<tr><th>Id.</th><th>Requirement</th><th>Expected Item</th></tr>
<tr><td><b>5.5.2.1</b></td><td><b>Detailed design of each software component</b></td><td>Software components design document (Scdd)</td></tr>
<tr><td colspan="3">a. The supplier shall develop a detailed design for each component of the software and document it.
b. Each software component shall be refined into lower levels
c. It shall be ensured that all the software requirements are allocated to software units.</td></tr>
<tr><td><b>5.5.2.2</b></td><td><b>Development and documentation of the software interfaces detailed design</b></td><td>External and internal interfaces design (update) (Eid /Iid).</td></tr>
<tr><td colspan="3">a. The supplier shall develop and document a detailed design for the interfaces (external and internal).</td></tr>
<tr><td><b>5.5.2.3</b></td><td><b>Production of the detailed design model</b></td><td>Software static-dynamic and -behavioral design model (Ssdm, Sddm, Sbdm)</td></tr>
<tr><td colspan="3">a. The supplier shall produce the detailed design model of the software components (static, dynamic and behavioural aspects).</td></tr>
<tr><td><b>5.5.2.4</b></td><td><b>Software detail design method</b></td><td>Software design method (Sdm).</td></tr>
<tr><td colspan="3">a. The supplier shall use a design method to produce the detailed design including software units, their interfaces, and relationships.</td></tr>
<tr><td><b>5.5.2.5</b></td><td><b>Detailed design of real–time software</b></td><td>Real-time software dynamic design model (R-tddm).</td></tr>
<tr><td colspan="3">a. The dynamic design model shall be compatible with the computational model of the architectural design model.
b. The supplier shall document and justify all timing and synchronization mechanisms.
c. The supplier shall document and justify all the design mutual exclusion mechanisms.
d. The supplier shall document and justify the use of dynamic allocation of resources.
e. The supplier shall ensure protection during the use of dynamic allocation of resources.</td></tr>
<tr><td><b>5.5.2.6</b></td><td><b>Utilization of description techniques for the software behaviour</b></td><td>Software behavioural design model techniques (Sbdmt).</td></tr>
<tr><td colspan="3">a. The behavioural design of the units shall be described by means of techniques (i.e. automata and scenarios.)</td></tr>
<tr><td><b>5.5.2.7</b></td><td><b>Determination of design method consistency for real–time software</b></td><td>Compatibility of real-time design methods with the computational model (CR-tdm).</td></tr>
<tr><td colspan="3">a. It shall be ensured that all the methods utilized for different item of the same software are, from a dynamic stand–point, consistent among themselves and consistent with the selected computational model.</td></tr>
<tr><td><b>5.5.2.8</b></td><td><b>Development and documentation of the software user manual</b></td><td>Software user manual (Sum)</td></tr>
<tr><td colspan="3">a.The supplier shall develop and document the software user manual.</td></tr>
<tr><td><b>5.5.2.9</b></td><td><b>Definition and documentation of the software unit test requirements and plan</b></td><td>Software unit test plan (Sutp).</td></tr>
<tr><td colspan="3">a. The supplier shall define and document responsibility and schedule, control procedures, testing approach, test design and test case specification for testing software units.</td></tr>
<tr><td><b>5.5.2.10</b></td><td><b>Conducting a detailed design review</b></td><td></td></tr>
<tr><td colspan="3">a. The supplier shall conduct a detailed design review.</td></tr>
</table>

2.2 SPEM 2.0

SPEM 2.0 (Software and Systems Process Engineering Metamodel) (OMG 2008) is a modeling language that defines the elements required to plan engineering processes. An engineering process is a sequence of units of work (e.g., tasks) that consume resources (e.g., employee time) to transform inputs (e.g., data, raw material) into outputs (e.g., products) (Boutros and Purdie 2014). SPEM 2.0 concepts are defined in separated UML (OMG 2017) packages that are interrelated. For example, the meta-class *TaskDefinition*, which belongs to the package *MethodContent* is used to describe assignable units of work. Instances of *Task Definition* can be applied in a process breakdown structure by defining a proxy with a *TaskUse*, a meta-class that belongs to the package *ProcessWithMethods* (both meta-classes are highlighted with red in Fig. 1). The same approach is used for the definition and use of roles and work products. Instead, a *tool* definition is used to specify the tool's participation in a Task Definition. *Guidance*, which belongs to the package *Managed Content*, is a describable element that provides additional information to other elements. There are different guidance kinds, e.g., concept and reusable asset. A *Delivery Process*, which belongs to the package *Process Structure*, describes an approach for performing a specific project. A *Category* is used to group elements in a recursive way. SPEM 2.0 supports variability management on breakdown structures representing processes as well as in content elements. In particular, we recall the variability mechanism called *extends*, in which the method content element that extends the base method element inherits the attributes of the extended base element.

SPEM 2.0-like concepts can be modeled with an open-source tool, called Eclipse Process Framework Composer(EPF-C) (Eclipse Foundation 2018). In particular, EPF-C provides a *Method Authoring*, which is used to describe roles, tasks, work products, and guidance. EPF-C also has a *Process Authoring*, which is used to organize reusable process building blocks in the form of delivery processes. EPF-C implements the method plugin package, which defines capabilities for modularization and extensibility. Such plugins, which can contain libraries of method content and processes, are reusable. (see Fig. 2a). Conceptually, a task can be represented as a synergy between different process elements (see Fig. 2b). In EPF-C, the process's partial execution semantics can be modeled with UML activity diagrams (see Fig. 2c).

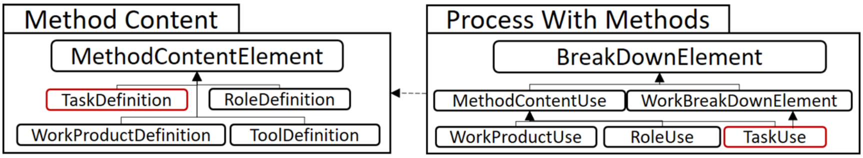
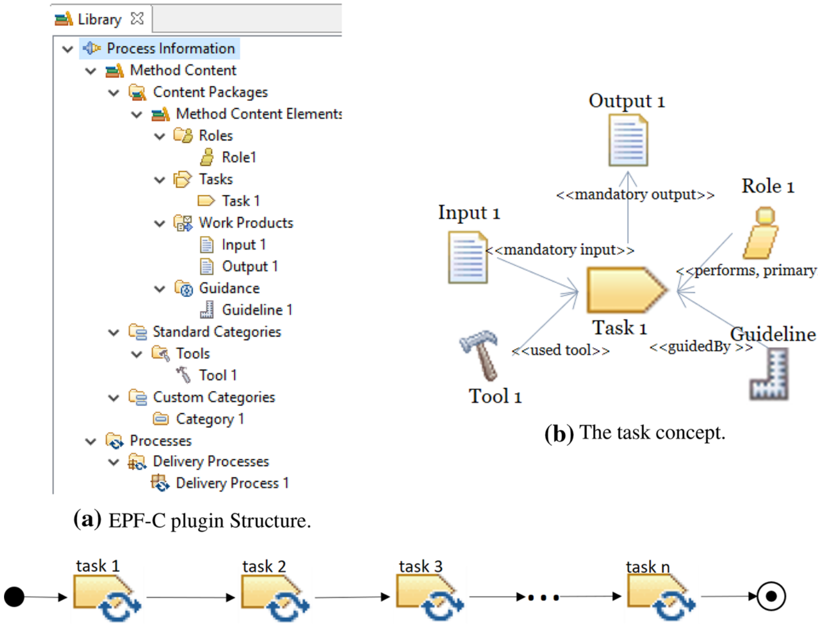


Fig. 1 Partial representation of SPEM 2.0 taxonomy



(a) EPF-C plugin Structure.

(b) The task concept.

(c) Activity diagram representing a process workflow.

Fig. 2 EPF-C environment

2.3 FCL

FCL (Formal Contract Logic) (Governatori 2005) is a language that permits the formalization of normative requirements. An FCL rule has the form $a_1, ..., a_n \Rightarrow c$, where r is the unique identifier of the rule, $a_1, ..., a_n$ are the propositions that represent the conditions of the applicability of such a rule, and c is the conclusion. The conclusion characterizes normative deontic effects, such as obligations, prohibitions, or permissions. FCL does not support contradictory conclusions but seeks to resolve conflicts. For instance, if it is sustainable support to conclude both c and $-c$, FCL does not conclude any of them. However, if the support for c has priority to the support of $-c$, then c is concluded. Thus, an FCL rules designer has to identify pairs of rules with incompatible literals and define superiority relations, as follows:

$$r\,:\,a_1,...,a_n \Rightarrow c, \quad \text{and} \quad r'\,:\,b_1,...,b_n \Rightarrow -c, \quad \text{then} \quad r' > r$$

Obligations and prohibitions are constraints that limit the behaviour of processes. As such, they can be violated. Permissions, which cannot be violated, can be used to determine that there are no obligations or prohibitions to the contrary. Hashmi et al. (2013) proposes the foundations for the normative requirements that constraint processes, which considers different types of obligations (based on the temporal validity of norms and the effects of violating these obligations). Thus, an obligation is in force if the obligation is activated at a particular time point in a time interval. An

Table 2 FCL rule notations

Notation	Description
[P]P	A proposition P is permitted
[OM]P	There is a maintenance obligation for the proposition P
[OM]-P	There is a prohibition for proposition P
[OAPP]P	There is an achievement, preemptive, and non-perdurant obligation for the proposition P
[OANPP]P	There is an achievement, non-preemptive and perdurant obligation for the proposition P
[OAPNP]P	There is an achievement, preemptive and non-perdurant obligation for the proposition P
[OANPNP]P	There is an achievement, non-preemptive and non-perdurant obligation for proposition P

obligation is considered to be non-persistent if it remains in force until it is terminated. Such obligation should be obeyed for the instant it is in force. In opposition, an obligation is considered persistent if it remains in force until it is removed. When a persistent obligation needs to be obeyed for the whole duration within the interval in which it is in force, it is called *maintenance obligation*. If achieving the content of the obligation at least once is enough to fulfill it, it is called *achievement obligation*. An achievement obligation is *preemptive* if it could be fulfilled even before the obligation is in force. Otherwise, it is *non-preemptive*. An achievement obligation is *perdurant* if, after being violated, the obligation is still required to be fulfilled. Otherwise is *non-perdurant*. A *prohibition* corresponds to the negation of the content of an achievement obligation. The types mentioned above are adopted, and notated in FCL, as presented in Table 2.

2.4 Regorous

Regorous (Governatori 2015) is a process compliance checker that implements compliance by design (Sadiq et al. 2007), i.e., check requirements that are propagated into models of process plans. Regorous requires two specifications: (1) a rule base representing the regulation in FCL (recalled in Sect. 2.3), and (2) a state representation of a process, i.e., a process enriched with semantic annotations. Semantic annotations on process elements are literals that record data, resources, and other information used by machines to refer, compute, and align information. The recorded information, which represents the effects caused by the tasks, is used by Regorous to perform compliance analysis. Two types of semantic annotations are necessary. The first one is State (t,n), which semantically annotates the set of facts in the computation to determine which rules fire (get active) for the n-th element in a trace t. A trace is a sequence of tasks in which a process can be executed. Consequently, obligations are in force after rules fire. The second one is Force(t,n+1), which contains the obligations that are in force but are not terminated in n-th element in the trace t. An obligation can be terminated if the deadline is reached, the obligation has been fulfilled, or if the obligation has been violated and is not perdurant. A process is fully compliant if all obligations are fulfilled, or if violated, they are compensated). For example, Fig. 3, shows a fictional FCL rule base and a compliance annotated process. As the figure depicts, the ruleset in FCL contains four rules. The first rule,

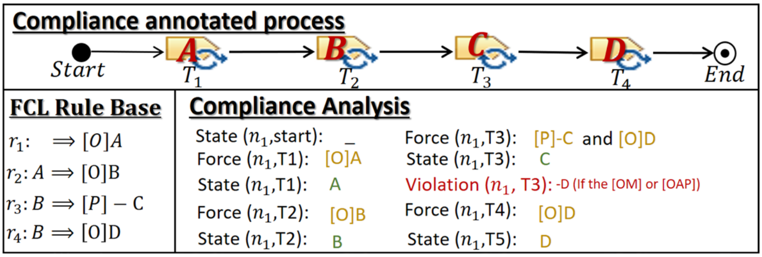


Fig. 3 Analysis of compliance

r1, implies the obligation of providing A. The second rule, r2, implies the obligation of B given the provision of A. The third rule, r3, implies the permission to not provide C given the provision of B. And r4 implies the obligation of D given the provision of B. From the FCL rule base, we have four compliance effects, i.e., A, B, C, and D. As seen, the compliance effects are extracted from the formulas composing the rules. The tasks in the process are annotated with the effects as follows. T1 is annotated with effect A, T2 is annotated with effect B, T3 is annotated with effect C, and T4 is annotated with effect D. To check compliance, we use the functions State and Force, as previously described. The State of the start point is empty because we have not defined any effect. After the start point, the compliance checking process is activated. Thus, the first rule is in force. The first task is expected to provide the effect A since there is the obligation to provide A. Then, we check the State after the task T1. As we see in the figure, T1 produces the effect A. So, the rule is fulfilled. Then, providing A forces the provision of B in T2. In the figure, we can see that T2 provides effect B. So, the second rule is also fulfilled. After B is provided, it implies two normative effects. The first one is the permission to not providing C in T3. Second, it implies the obligation of providing D in T3. When checking T3, we can see that it provides the effect C. However, having C as the produced effect does not imply a violation of rule r3 because the force function has a permit, not an obligation. However, in T3, we should have D, and the tasks T3 is not providing E. If the obligation of providing D is a Maintenance or achievement preemptive, we have a violation. A violation means that the process is not compliant. If the obligation is achievement non-preemptive, it can be fulfilled in T4. In this case, there is no violation, and the process is compliant.

2.5 Process compliance hints and patterns

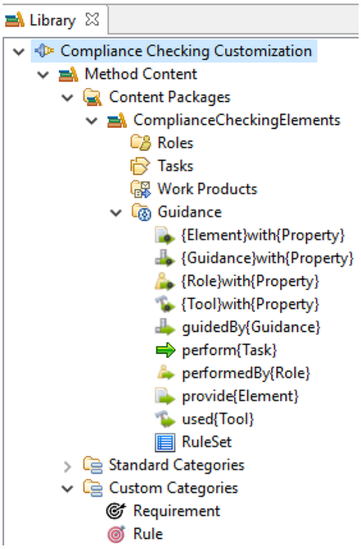
Skillful FCL ruleset design can be reached by applying computational thinking resources, in particular, design hints and patterns (Denning and Tedre 2019). Hints are rules of thumb found in previous FCL formalization experiences, while patterns indicate common situations an FCL designer is likely to encounter. Both process compliance hints and patterns aim at facilitating the formalization of process-related requirements into FCL rules. In the remaining part of this section, we recall these resources in more detail.

2.5.1 Process compliance hints

The divide-and-conquer strategy, adopted in software engineering as a principle to manage complexity (Smith 1985), is a hint that can be applied in the formalization of process-related requirements, as presented in Castellanos Ardila and Gallina (2020) In particular, the aspects that requirements in standards regulate are the tasks, their specific order, the mandatory in/outputs of the tasks, roles performing the tasks, and the tools/recommended techniques used to do the tasks. Thus, the concept of a task is central, to which properties such as the definition of roles, inputs, outputs, tools, and techniques must apply. However, requirements not only define the properties of the tasks. For example, roles and tools should be qualified. This kind of requirements does not directly affect the tasks. They directly affect other elements, which in turn have effects on tasks. Thus, a process can be deemed compliant if we can demonstrate that the process contains the permitted tasks, such tasks have associated the prescribed roles, inputs, outputs, tools, and techniques, and if the associated elements have associated their related properties. With such consideration, dividing requirements in terms of the elements they target as well as the specific properties defined for each element seems to be the natural way in which concerns should be separated. To facilitate the creation of compliance effects, which later can be used to form the propositions of the rules in FCL (recalled in Sect. 2.3), two aspects are proposed (see Fig. 4). The first aspect is the customization of icons, which describe the targeted elements. The second aspect is the definition of templates that facilitate compliance effects creation (fragments between { }, should be replaced by the specific element or its property). Both, icons and templates are based on the concepts described in SPEM 2.0 (recalled in Sect. 2.2, specifically in Fig. 2). Once created, the compliance annotations are performed in the elements that carry out their compliance responsibility.

2.5.2 Process compliance patterns

Process Compliance Patterns (PCP) (Castellanos Ardila and Gallina 2017) are commonly occurring normative requirements on the permissible state sequence of a finite state model of a process. The PCPs description is based on similar (or a combination of) behaviors described for the property specification patterns (Dwyer et al. 1999), which are mapped to the notations provided in FCL (recalled in Table 2). A global scope, which represents the entire process model execution, is defined as a [OM]P. A before scope, which includes the execution of the process model up to a given state, is mapped to a partial [OAP]. An after scope, which includes the execution of the process model after a given state, is mapped to a partial [OANP]. If an obligation admits an exception, e.g., tailoring, the part of the pattern corresponding to the exception is described as [P] since if something is permitted, the obligation to the contrary does not hold. The excepted obligation is modeled as non-perdurant,



Library ☒

- Compliance Checking Customization
  - Method Content
    - Content Packages
      - ComplianceCheckingElements
        - Roles
        - Tasks
        - Work Products
        - Guidance
          - {Element}with{Property}
          - {Guidance}with{Property}
          - {Role}with{Property}
          - {Tool}with{Property}
          - guidedBy{Guidance}
          - perform{Task}
          - performedBy{Role}
          - provide{Element}
          - used{Tool}
          - RuleSet
    - Standard Categories
    - Custom Categories
      - Requirement
      - Rule

Fig. 4 Elements customization

since the permission is not a violation of the obligation. Thus, the obligation does not persist after the permission is granted. In principle, all the requirements could be tailored. Thus, obligations are modeled as [OAPNP] or [OANPNP]. In this case, obligation and permission have contradictory conclusions, but the permission is superior since it represents an exception. Table 3 presents the templates of the PCPs. In all templates {#} should be replaced with the number that identifies the requirement in the standard. When it is described as {#.*i*}, the *i* should be replaced by a, b, ..., n, where n is the number of sub-items, e.g., if there is a requirement with two parts that is identified with the number 5, the rules' identifiers are 5.a and 5.b. Following, we present a more detailed description of the patterns.

Tailoring requirements (PCP 1a and 1b) Tailoring means to adapt (omit or perform differently) the requirements to a specific project in a compliant form. Tailoring requires a rationale (or justification). For being valid, a rationale should always be verified by an expert. The rationale is an input element, and its verification is a property. An expert with specific qualifications should also be appointed. Thus, we use the templates for definitional and property-based propositions described in Fig 4 for in/output elements and roles, i.e., $\mathit{provide}\{\mathit{Rationale}\}$, $\{\mathit{Rationale}\}\mathit{withVerificationByExpert}$, $\mathit{performedBy}\{\mathit{Expert}\}$ and $\{\mathit{Expert}\}\mathit{with}\{\mathit{Qualification}\}$. Providing those four conditions permit to omit the requirement (in other words, permit not to perform the requirement). Any of the definitional and property-based propositions present in Fig. 4 could be the target of such omission. For explanations purposes, we consider omitting a requirement that imposes the definition of a task ($\Rightarrow [P] - \mathit{perform}\{\mathit{Task}\}$). In PCP 1a, $\{\mathit{Rationale}\}$ should be replaced with the title of the required justification. $\{\mathit{Task}\}$ should be replaced with the name of the task that will be omitted. Finally, $\{\mathit{Expert}\}$ should be replaced with the role required and $\{\mathit{Qualification}\}$ with the necessary qualifications. A second rule, i.e.,

Table 3 PCP Templates

PCP	FCL notation
1a	$\mathbf{r}\{\#\}\mathbf{.Omitted:}\textit{provide}\{\textit{Rationale}\},\{\textit{Rationale}\}\textit{withVerificationByExpert},$ $\textit{performedBy}\{\textit{Expert}\},\{\textit{Expert}\}\textit{with}\{Qualification\} \Rightarrow [\textit{OANPP}] - perform\{Task\}$
1b	$\mathbf{r}\{\#\}\mathbf{.ChangedRule:}$ $\textit{-perform}\{Task\} \Rightarrow [\textit{OANPNP}]perform\{DifferentTask\}$
2	$\mathbf{r}\{\#.i\}\mathbf{:}\{optionalTrigeringObligation\} \Rightarrow [\textit{OAPNP}]\textit{provide}\{prerequisite.i\}$
3a	$\mathbf{r}\{\#\}\mathbf{:}$ provide$\{Prerequisite1\}...,\textit{provide}\{Prerequisite.i\} \Rightarrow [\textit{OAPNP}]perform\{TitleClause\}$
3b	$\textit{perform}\{Task\} \Rightarrow [\textit{OANPNP}]perform\{FollowingTask\}$
4	$\{\#.i\}\mathbf{:}\textit{perform}\{Task\},\{Guidance\}with\{Property\} \Rightarrow [\textit{OAPNP}]\textit{guidedBy}\{Guidance\}$
5	$\mathbf{r}\{\#.i\}\mathbf{:}\{providePreviousObligations\},\{WorkProduct\}with\{Property\}$ $\Rightarrow [\textit{OANPNP}]\textit{provide}\{WorkProduct\}$

PCP 1b, is included in case the task is done in a different way, where *[OANPNP] perform*{*DifferentTask*} corresponds to the new task replacing the previous one.

Provide a prerequisite (PCP 2) A prerequisite is an obligatory input element, which should be fulfilled before it is in force (preemptive). PCP 2, {*prerequisite*} should be replaced with the name of the prerequisite. If a previous rule triggers the prerequisite, its conclusion is included in the {*optionalTrigeringObligation*}, e.g., when the prerequisite is produced by a previous task. Prerequisite could have properties. In this case, the {*optionalTrigeringObligation*} could be a list of such properties, using the template {*Element*}*with*{*Property*}. Otherwise, it is left empty.

Perform a unit of work (PCP 3a and 3b) Template PCP 3a represents the performance of a unit of work that can be prescribed in a process (i.e., phase/activity/task). It considers the prerequisites, if any, as the conditions of the applicability of the rule, which normative conclusion is performing a unit of work (e.g., a phase). It could be preemptive ([OAPNP]), if the prerequisites and the task are provided at the same time. It can be non-preemtive ([OANPNP]) as in template 3b, if the prerequisite is another task, that have to be done first. In the example of PCP 3a, {*TitleClause*} should be replaced with the specific clause title.

Provide guidance (PCP 4) Guidance elements may not be required during standards compliance auditing. However, internal policies in a company may impose guidance elements. In that case, guidance elements should be provided at the moment the element guided is created. We create the propositions by using the template for guidance provided in Fig. 4, i.e., *guidedBy*{*Guidance*} and {*Guidance*} *with*{*Property*}. Guidance can be defined for any element in the process (tasks, work product, tool, or role). For explanation purposes, we consider *perform*{*Task*} (see PCP 4).

Provide a work product (PCP 5) Work products are the result of certain requirements. Thus, these requirements are presented as antecedents that oblige the provision of the related work product. PCP 5 presents this aspect in FCL, where {*providePreviousObligations*} should be replaced with the conditions that oblige the work product's production, usually the execution of a task (*perform*{*Task*}). Work product properties may be also required, i.e., ({*WorkProduct*}*with*{*Property*}, where {*WorkProduct*} should be replaced with the work product's name and {*Property*} with the corresponding property.

2.6 ACCEPT

ACCEPT (Automatic Compliance Checking of Engineering Processes against sTandards) (Castellanos et al. 2018b, a), is a safety-centered planning-time framework aimed at facilitating the analysis of the tradeoffs associated with the planning of compliant processes in the safety-critical context. ACCEPT uses state-of-the-art tools and methodologies (see Fig. 5).

In particular, ACCEPT is based on compliance by design (Lu et al. 2007), a preventive approach aimed at integrating compliance requirements into process plans. Such an approach requires the definition of two specifications. The first one is the FCL (recalled in Sect. 2.3) based standards requirements. FCL provides a framework that unambiguously represents the deontic notions required for compliance analysis. The second one is the process plan enriched with compliance effects, which is provided via SPEM 2.0-like artifacts in EPF-C (recalled in Sect. 2.2). SPEM 2.0 is flexible, i.e., concepts can be customized and extended to permit not only the creation artifacts related to processes but also compliance checking artifacts, such as standard requirements, rules and annotated process plans. SPEM 2.0-like artifacts are also reusable since capabilities for modularization and extensibility are implemented in EPF-C (i.e., plugins). With the composition of EPF-C and the Base Variability Management Tool (Javed and Gallina 2018b), tailoring of compliance artifacts and reuse is also facilitated. ACCEPT is equipped with guidance regarding process compliance hints and patterns (recalled in Sect. 2.5) that ease the creation of the required specifications. ACCEPT uses Regorous (recalled in Sect. 2.4), which provides a sound algorithm for the analysis of FCL rules that automatically check if a compliance-aware engineering process plan (CaEPP) is designed, i.e., if the elements set down by the requirements (e.g., tasks, personnel, work products, techniques, and tools, as well as their properties) are present at given points in the engineering process plan. The approach consists of five methodological steps, as shown in Fig. 6.

Step 1: Formalization of Requirements Standard requirements are formalized in FCL by and FCL-trained person supported by a process engineer (or an FCL-trained process engineer). Three inputs are required: the standard requirements, the compliance hints and patterns guidance, and the EPF-C plugin with the customized compliance checking artifacts (see Fig. 4). First, the requirements should be classified in terms of the process elements they target and their properties to create the rules' propositions (see Sect. 2.5.1). Then, PCPs are used to create the FCL rules

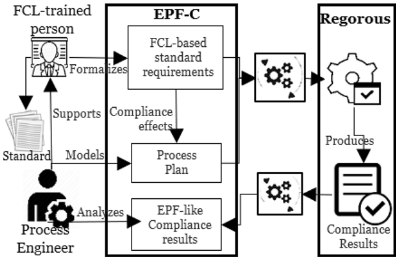


Fig. 5 ACCEPT Framework

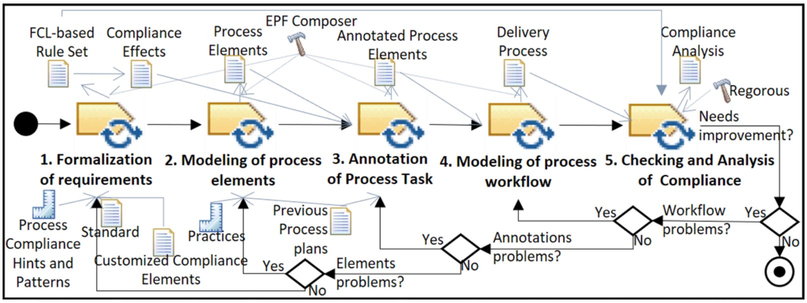

FCL-based Rule Set
Compliance Effects
Process Elements
EPF Composer
Annotated Process Elements
Delivery Process
Compliance Analysis
Regorous
Needs improvement?
1. Formalization of requirements
2. Modeling of process elements
3. Annotation of Process Task
4. Modeling of process workflow
5. Checking and Analysis of Compliance
Process Compliance Hints and Patterns
Standard
Customized Compliance Elements
Practices
Previous Process plans
Yes
No
Elements problems?
Yes
No
Annotations problems?
Yes
No
Workflow problems?
Yes
No

Fig. 6 Methodological steps required for using ACCEPT

(see sect. 2.5.2). The output is an EPF-C plugin with the FCL-based ruleset containing information about the standard, their requirements, the rules derived from the requirements, and the separated set of propositions composing the rules.

Step 2: Modeling of Process Elements Capturing process plan elements is a task performed by the process engineer. The required input is information about process plans, which could steam from the organization's practices and previous process plans. The output is the representation of the process elements in EPF-C, as depicted in Fig. 2a. Detailed guidance regarding the creation of content elements in EPF-C is provided in Tuft (2010)).

Step 3: Annotation of Process Tasks The annotation process, which a process engineer performs manually, consists of assigning the compliance effects to the elements that fulfill them as presented in Fig. 7. For this, the compliance effects modeled in the FCL ruleset (created in step 1) and the process elements (created in step 2), or previous process plans, are the inputs of this step. The output is the annotated process elements in EPF-C.

Step 4: Modeling of Process Workflow The process engineer uses the compliance annotated process elements resulting from step 3 to model the workflow (see Fig. 2c). The output is the delivery process in EPF-C, which contains the process plan checkable for compliance, i.e., the compliance state representation of the process plan.

Step 5: Checking and Analysis Checking and analyzing compliance is a task performed by the process engineer. The required inputs are the FCL-based ruleset and the delivery process. The output is the compliance analysis, which contains information regarding the rules violated by the process, their reparation policies, and the rules that were not activated during the compliance checking analysis. Such information is used to improve the process plans to be checked iteratively. Reasons for such improvements could be workflow problems (error in the placement of tasks), failure in the annotation process (errors in the assignment of the compliance effects), failure in the selection of process elements (e.g., missed elements), or FCL ruleset errors (not applicable rules due to tailoring or standards evolution).

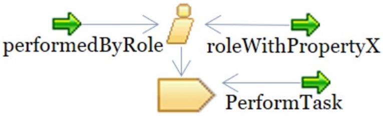


Fig. 7 Annotation

3 Case study design

In this section, we present the essential details regarding the case study design.

3.1 Rationale for the case study

In the European context, space software production is often the result of industrial cooperation. Such cooperation is coordinated using the de-facto standard ECSS-E-ST-40C (recalled in Sect. 2.1.2). Such a standard provides requirements that help customers formulate their project-specific requirements by using the EARM matrix. Suppliers need to prepare their responses by using the ECM matrix, which will help them implement the work. ECSS-E-ST-40C is a process-related standard. Thus, the planning of software engineering processes in compliance with project-specific ECSS-E-ST-40C applicable requirements is mandatory during contractual agreements. Moreover, the tailoring decisions, i.e., A, M, D, and N, should be documented. Thus, we wonder if the current status of the models produced by ACCEPT could support space manufacturers' needs. For this, we perform a case study, according to the guidelines provided in Runeson et al. (2012). In particular, we consider the selected portion of ECSS-E-ST-40C requirements related to the software items' design (recalled in Table 1). Our case study is descriptive (i.e., it portrays the current status of ACCEPT) and exploratory (i.e., it seeks future ACCEPT improvements). The data collected is qualitative involving models and their descriptions. The criteria used for the analysis is described in Ghanavati et al. (2008). In particular, we analyze the *effort to model* (needed to establish a model for managing compliance), the *effort to comprehend* (processes and standards models), the *effort to document compliance*, (needed to verify whether process models comply with standards models), and the *effort to manage evolution*, (needed to find potential instances of non-compliance when standards change). Moreover, we take into account the *level of coverage for the model* (which shows how much of the requirements and engineering processes can be modeled), the *level of coverage for compliance documentation* (which examines the level of success of the approach in terms of documenting the compliance), and the *level of coverage for the evolution management* (which examines the approach's success in handling the changes).

3.2 Goal and research questions of the case study

As presented in Sect. 3.1, we want to analyze ACCEPT in the context of space software engineering processes planning in compliance with ECSS-E-ST-40C. We have also selected specific criteria, which essentially consider two variables: effort and coverage level. The effort, according to Steele (2020), is a variable that could be estimated during task performance in two ways: the actual effort (determined by task demands) and the perception of effort (relative to a subject's capacity to recognize the effort). In this case study, our analysis is based on actual effort (from now on called effort) since, in theory, it can be used to determine the intent to complete a task a priori, independently of any conscious actor. The coverage level is analyzed considering how the models respond to the information that needs to be required by the ECSS-E-ST-40C framework, i.e., information regarding standards, process plans, and compliance (i.e., EARM and ECM matrices). Thus, our goal is **to qualitatively analyze the current effort required to model a CaEPP in ACCEPT for software development processes in compliance with ECSS-E-ST-40C and the coverage level of such models**. Based on this goal, we derive the following research questions:

RQ1: **How could we consider the effort required in designing a CaEPP with ACCEPT for software development?** The answer of this question will be supported by answering the following subquestions:

RQ1.1: How could we consider the effort required to create models?

RQ1.2: How could we consider the effort required to comprehend the models?

RQ1.3: How could we consider the effort required to document compliance?

RQ1.4: How could we consider the effort required to manage evolution?

RQ2: **How could we consider the coverage level of a CaEPP for software development created with ACCEPT?** The answer of this question will be supported by answering the following subquestions:

RQ2.1: How could we consider the coverage level of the models?

RQ2.2: How could we consider the coverage level of the documentation?

RQ2.3: How could we consider the coverage level of the evolution management?

3.3 Unit of analysis and method

To support our goal (defined in Sect. 3.2), we model a CaEPP for space software engineering processes. The standard requirements involved in our models are the ECSS-E-ST-40C, focus on software design (recalled in Sect. 2.1.2). In general, ECSS-E-ST-40C determines mission-critical requirements that have an inherent relationship with safety issues since a software failure could lead to mission loss that could have catastrophic consequences. Thus, such requirements belong to the safety-critical context. The portion selected provides a view to the general structure of such a standard, i.e., prescribes requirements that impose the presence of process elements that can be tailored in a process plan. Thus, we consider such a portion representative of the whole standard. The method selected for conducting the case study is described in the steps required for facilitating automated compliance checking of engineering processes against standards (see Fig. 6). Such a method permits us to collect the data required for the analysis.

3.4 Validity of the study

Case studies in software engineering are conducted to increase knowledge and bringing change in the studied phenomenon (Runeson et al. 2012). Researchers must consider issues that may diminish the results' trustworthiness by demonstrating the extent to which the researchers' subjectiveness does not bias the results. In this study, we consider a scheme of four aspects of validity in case studies in software engineering defined in Runeson et al. (2012). (1) *Construct validity* reflects the extent to which the research represents the theoretical concepts used in the study. (2) *Internal validity* is of concern when causal relations are examined. (3) *External validity* addresses the ability of the research to be generalized. (4) *Reliability* is concerned with the extent to which the data and analysis are dependent on specific researchers. Addressing these four validity aspects is essential since it permits an accurate account of the research by selecting and using acceptable methodological practices that guarantee correct steps for collecting and analyzing the data. The concrete ways in which we addressed the mentioned validity aspects are listed below.

Construct validity To avoid construct validity, we established a chain of evidence by rigorously following our defined methodology (see Fig. 6), reporting the results consistently with such a methodology. However, designed methodologies may be biased. For mitigating this aspect, we review our assumptions against theoretical foundations several times during several sessions to avoid oversimplifications that may confirm our preconceptions. We also got external reviews in previous phases of our work, as well as initial stages of this case study. We use such reviews to improve our methodology, its presentation, and the definition of the case study itself.

Internal validity The manual formalization of the FCL rules may imply a internal validity threat, due to the possibility of typos in the syntax and inconsistencies in the rules statements. For mitigating this aspect, we instantiated the process compliance hints and patterns (see Sect. 2.5) and performed manual syntactic corrections of the FCL specification. In the future, we plan to develop tools for supporting the process of writing and verifying rules.

External validity We have performed an ACCEPT analysis on a limited portion of a software process standard. It is a single case study, but it is not trivial. It shows dilemmas and design choices that are typical in safety-related engineering process plans. In particular, ECSS-E-ST-40C determines mission-critical requirements that have an inherent relationship with safety issues since mission loss could lead to catastrophic consequences. Thus, they belong to the safety context. Moreover, the ECSS-E-ST-40C portion selected contains all the characteristics regarding process-based standards, i.e., the definition of work units, in/outputs, elements properties, and other process-related elements such as guidance, which have the possibility to be tailored. Such characteristics are presented in whole standard. Thus, the selected requirements are representative and can be generalized to the complete ECSS-E-ST-40C standard. However, the outcome of this case study applies to a CaEPP for software development that respond to the mentioned characteristics. Additional challenges may arise when analyzing standards beyond safety and software that also apply in the safety-critical context. Thus, to generalize our framework capabilities, we must carry out case studies beyond the ones we have already performed.

Reliability Reliability threats were mitigated by involving the researchers in peer debriefing, i.e., iterative review of research artifacts (formalization of standard requirements, EPF-C models) during all the process.

4 Data collection

In this section, we collect the data required for the case study.

4.1 Formalization of ECSS-E-ST-40C requirements

As described in Sect. 2.6, we classify the requirements in terms of the process elements and their properties (see Table 4), and create the rules' propositions (see Fig. 8) based on process compliance hints (see Sect. 2.5.1).

Table 4 Process elements required by ECSS-E-ST-40C

Element	Description
Inputs	TSSC, AD, DJ, PDR.
Outputs	Expected items of every requirement are: Scdd, Eid, Iid, Ssdm, Sddm, Sbdm, Sdm, R-tddm (with timing, synchronization and mutual exclusion mechanisms, and dynamic allocation of resources), Sbdmt CR-tdm, Sum, and Sutp (with responsibility, schedule, control procedures, testing approach, test design and test case specification). Final outputs: DDF, SDD, CDR, TS, ICD, SUM, DJF and SUITP.
Tasks	Phase Design of the software items which contains the following tasks (and their properties): 1) Detailed design of each software component, 2) Development and documentation of the software interfaces detailed design, 3) Production of the detailed design model, 4) Software detail design method, 5) Detailed design of real–time software, 6) Utilization of description techniques for the software behaviour, 7) Determination of design method consistency for real–time software, 8) Development and documentation of the software user manual, 9)Definition and documentation of the software unit test requirements and plan, and 10) Conducting a detailed design review
Guide	Guidance for Scdd (req-5-5-2-1), R-tddm (req-5-5-2-5), Sutp (req-5-5-2-9)

The initial part of the ruleset defines the provision of requirements TSSC, AD, DJ, PDR, which are needed to start the activity described in Table 1. The PCP 2 is used to create the rules mandating the requirements (rules r5.5.2.a to r5.5.2.d) and PCP 3a to create the rule mandating the phase definition (rule r5.5.2) as follows:

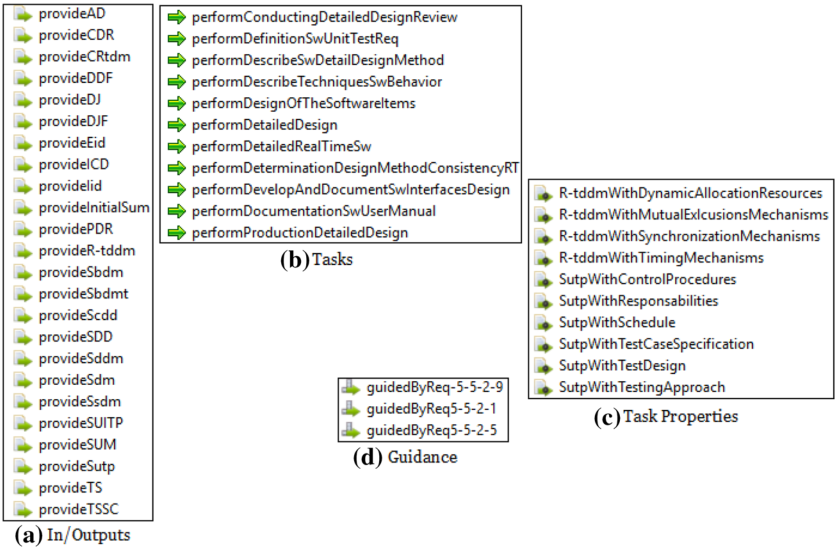
$$\begin{aligned}\mathbf{r5.5.2.a} &:\Rightarrow [OAPNP]provideTSSC\\ \mathbf{r5.5.2.b} &:\Rightarrow [OAPNP]provideAD\\ \mathbf{r5.5.2.c} &:\Rightarrow [OAPNP]provideDJ\\ \mathbf{r5.5.2.d} &:\Rightarrow [OAPNP]providePDR\\ \mathbf{r5.5.2} &: provideTSSC, provideAD, provideDJ, providePDR\\ &\Rightarrow [OAPNP]performDesignOfTheSoftwareItems\end{aligned}$$

We define a custom category in EFP-C (ECSS-E-ST-40C) to create the rule set. For each requirement, we nest a category. In each category, we nest the rule. We assign the compliance effect (the conclusion of the rule) to each rule (see Fig. 9). All requirements and rules are modeled in a similar way.

We use PCP 3b formalize the requirements that define the first task *Detailed design of each software component* (see rule 5.5.2.1), which prerequisite is the activity definition (see rule 5.5.2). Then, we use PCP 5 to define the expected item (ei), which is the work product of this task (see rule r5.5.2.1.ei). Then we used the PCP 4 to define the guidance (see rule r5.5.2.1.guide).

$$\begin{aligned}\mathbf{r5.5.2.1} &: performDesignOfTheSoftwareItems\\ &\Rightarrow [OANPNP]performDetailedDesign\\ \mathbf{r5.5.2.1.ei} &: performDetailedDesign \Rightarrow [OANPNP]provideScdd\\ \mathbf{r5.5.2.1.guide} &: performDetailedDesign\\ &\Rightarrow [OANPP]guidedByReq5-5-2-1\end{aligned}$$

Requisite 5.5.2.2 is the definition of the task *Development and documentation of the software interfaces detailed design*, which produces two expected items (ei) *Eid* and *Iid*. As two ei are created, we further identify the rules by adding a and b to the rules (see rules r5.5.2.2.ei.a and r5.5.2.2.ei.b).



- provideAD
- provideCDR
- provideCRtdm
- provideDDF
- provideDJ
- provideDJF
- provideEid
- provideICD
- provideIid
- provideInitialSum
- providePDR
- provideR-tddm
- provideSbdm
- provideSbdmt
- provideScdd
- provideSDD
- provideSddm
- provideSdm
- provideSsdm
- provideSUITP
- provideSUM
- provideSutp
- provideTS
- provideTSSC

(a) In/Outputs

- performConductingDetailedDesignReview
- performDefinitionSwUnitTestReq
- performDescribeSwDetailDesignMethod
- performDescribeTechniquesSwBehavior
- performDesignOfTheSoftwareItems
- performDetailedDesign
- performDetailedRealTimeSw
- performDeterminationDesignMethodConsistencyRT
- performDevelopAndDocumentSwInterfacesDesign
- performDocumentationSwUserManual
- performProductionDetailedDesign

(b) Tasks

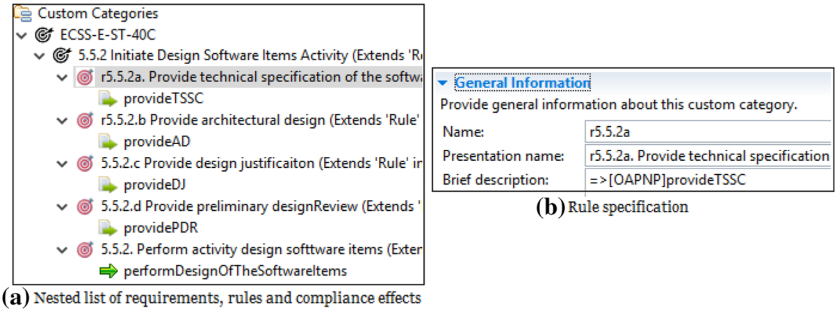
- R-tddmWithDynamicAllocationResources
- R-tddmWithMutualExlcusionsMechanisms
- R-tddmWithSynchronizationMechanisms
- R-tddmWithTimingMechanisms
- SutpWithControlProcedures
- SutpWithResponsabilities
- SutpWithSchedule
- SutpWithTestCaseSpecification
- SutpWithTestDesign
- SutpWithTestingApproach

(c) Task Properties

- guidedByReq-5-5-2-9
- guidedByReq5-5-2-1
- guidedByReq5-5-2-5

(d) Guidance

Fig. 8 Rules propositions



```
Custom Categories
  ECSS-E-ST-40C
    5.5.2 Initiate Design Software Items Activity (Extends 'R
      r5.5.2a. Provide technical specification of the softw
        provideTSSC
      r5.5.2.b Provide architectural design (Extends 'Rule'
        provideAD
      5.5.2.c Provide design justificaiton (Extends 'Rule' ir
        provideDJ
      5.5.2.d Provide preliminary designReview (Extends '
        providePDR
      5.5.2. Perform activity design softtware items (Exter
        performDesignOfTheSoftwareItems
```

(a) Nested list of requirements, rules and compliance effects

General Information

Provide general information about this custom category.

Name:	r5.5.2a
Presentation name:	r5.5.2a. Provide technical specification
Brief description:	=>[OAPNP]provideTSSC

(b) Rule specification

Fig. 9 Rules definition

$$\begin{aligned}\mathbf{r5.5.2.2}\ &:\ \mathit{performDetailedDesign}\\ &\Rightarrow [\mathit{OANPNP}]\mathit{performDevelopAndDocumentSwInterfacesDesign}\\ \mathbf{r5.5.2.2.ei.a}\ &:\ \mathit{performDevelopAndDocumentSwInterfacesDesign}\\ &\Rightarrow [\mathit{OANPNP}]\mathit{provideEid}\\ \mathbf{r5.5.2.2.ei.b}\ &:\ \mathit{performDevelopAndDocumentSwInterfacesDesign}\\ &\Rightarrow [\mathit{OANPNP}]\mathit{provideEid}\end{aligned}$$

Requisite 5.5.2.3 is the definition of the task *Production of the detailed design model* (see rule r5.5.2.3) and three items are expected (see rules r5.5.2.3.ei.a, r5.5.2.3.ei.b and r5.5.2.3.ei.c).

$$\begin{aligned}
\mathbf{r5.5.2.3} &: \mathit{performDevelopAndDocumentSwInterfacesDesign} \\
&\Rightarrow [\mathit{OANPNP}]\mathit{performProductionDetailedDesign} \\
\mathbf{r5.5.2.3.ei.a} &: \mathit{performProductionDetailedDesign} \Rightarrow [\mathit{OANPNP}]\mathit{provideSsdm} \\
\mathbf{r5.5.2.3.ei.b} &: \mathit{performProductionDetailedDesign} \Rightarrow [\mathit{OANPNP}]\mathit{provideSddm} \\
\mathbf{r5.5.2.3.ei.c} &: \mathit{performProductionDetailedDesign} \Rightarrow [\mathit{OANPNP}]\mathit{provideSbdm}
\end{aligned}$$

Requisite 5.5.2.4 is the definition of the task *Software detail design method* (see rule r5.5.2.4) and one item is expected (see rule r5.5.2.4.ei).

$$\begin{aligned}
\mathbf{r5.5.2.4} &: \mathit{performProductionDetailedDesign} \\
&\Rightarrow [\mathit{OANPNP}]\mathit{performDescribeSwDetailDesignMethod} \\
\mathbf{r5.5.2.ei} &: \mathit{performProductionDetailedDesign} \Rightarrow [\mathit{OANPNP}]\mathit{provideSdm}
\end{aligned}$$

Requisite 5.5.2.5 is the definition of the task *Detailed design of real–time software* (see rule r5.5.2.5) and one item is expected . However, this expected item has several properties, which are included in the antecedent of the rule r5.5.2.5.ei. Additionally, guidance is defined for this task. So, we use PCP 4 to define the mandatory guidance (see rule r5.5.2.5.guide).

$$\begin{aligned}
\mathbf{r5.5.2.5} &: \mathit{performDescribeSwDetailDesignMethod} \\
&\Rightarrow [\mathit{OANPNP}]\mathit{performDetailedRealTimeSw} \\
\mathbf{r5.5.2.5.ei} &: \mathit{performDetailedRealTimeSw}, R \\
&-\mathit{tddmWithDynamicAllocationResources}, \\
R &- \mathit{tddmWithMutualExlcusionsMechanisms}, R \\
&-\mathit{tddmWithSynchronizationMechanisms}, \\
R - \mathit{tddmWithTimingMechanisms} &\Rightarrow [\mathit{OANPNP}]\mathit{provideR} - \mathit{tddm} \\
\mathbf{r5.5.2.5.guide} &: \mathit{performDescribeSwDetailDesignMethod} \\
&\Rightarrow [\mathit{OAPNP}]\mathit{guidedByReq}5 - 5 - 2 - 5
\end{aligned}$$

Requisite 5.5.2.6 is the definition of the task *Utilization of description techniques for the software behaviour* (see rule r5.5.2.6) and one item is expected (see rule r5.5.2.6.ei).

$$\begin{aligned}
\mathbf{r5.5.2.6} &: \mathit{performDetailedRealTimeSw} \\
&\Rightarrow [\mathit{OANPNP}]\mathit{performDescribeTechniquesSwBehavior} \\
\mathbf{r5.5.2.6.ei} &: \mathit{performDescribeTechniquesSwBehavior} \\
&\Rightarrow [\mathit{OANPNP}]\mathit{provideSbdmt}
\end{aligned}$$

Requisite 5.5.2.7 is the definition of the task *Determination of design method consistency for real–time software* (see rule r5.5.2.7) and one item is expected (see rule r5.5.2.7.ei).

$$\begin{aligned}\mathbf{r5.5.2.7}\ &:\ \mathit{performDescribeTechniquesSwBehavior}\\ &\Rightarrow [\mathit{OANPNP}]\mathit{performDeterminationDesignMethodConsistencyRT}\\ \mathbf{r5.5.2.7.ei}\ &:\ \mathit{performDeterminationDesignMethodConsistencyRT}\\ &\Rightarrow [\mathit{OANPNP}]\mathit{provideCRtdm}\end{aligned}$$

Requisite 5.5.2.8 is the definition of the task *Development and documentation of the software user manual* (see rule r5.5.2.8) and one item is expected (see rule r5.5.2.8.ei).

$$\begin{aligned}\mathbf{r5.5.2.8}\ &:\ \mathit{performDeterminationDesignMethodConsistencyRT}\\ &\Rightarrow [\mathit{OANPNP}]\mathit{performDocumentationSwUserManual}\\ \mathbf{r5.5.2.8.ei}\ &:\ \mathit{performDocumentationSwUserManual}\\ &\Rightarrow [\mathit{OANPNP}]\mathit{provideInitialSum}\end{aligned}$$

Requisite 5.5.2.9 is the definition of the task *Definition and documentation of the software unit test requirements and plan* (see rule r5.5.2.9). One item, with several properties is expected (see rule r5.5.2.9.ei) as well as guidance (see rule r5.5.2.9.guide).

$$\begin{aligned}\mathbf{r5.5.2.9}\ &:\ \mathit{performDocumentationSwUserManual}\\ &\Rightarrow [\mathit{OANPNP}]\mathit{performDefinitionSwUnitTestReq}\\ \mathbf{r5.5.2.9.ei}\ &:\ \mathit{performDefinitionSwUnitTestReq},\\ &\mathit{SutpWithControlProcedures}, \mathit{SutpWithResponsabilities},\\ &\mathit{SutpWithSchedule}, \mathit{SutpWithTestCaseSpecification},\\ &\mathit{SutpWithTestDesign}, \mathit{SutpWithTestingApproach}\\ &\Rightarrow [\mathit{OANPNP}]\mathit{provideSutp}\\ \mathbf{r5.5.2.9.guide}\ &:\ \mathit{performDocumentationSwUserManual}\\ &\Rightarrow [\mathit{OAPNP}]\mathit{guidedByReq} - 5 - 5 - 2 - 9\end{aligned}$$

Finally, requisite 5.5.2.10 is the definition of the task *Conducting a detailed design review* (see rule r5.5.2.10), Eight items are expected after this task (see rules r5.5.2.10.ei.a to r5.5.2.10.ei.h).

$$
\begin{aligned}
\mathbf{r5.5.2.10} &: \mathit{performDefinitionSwUnitTestReq} \\
&\Rightarrow [\mathit{OANPNP}]\mathit{performConductingDetailedDesignReview} \\
\mathbf{r5.5.2.10.ei.a} &: \mathit{performConductingDetailedDesignReview} \\
&\Rightarrow [\mathit{OANPNP}]\mathit{provideDDF} \\
\mathbf{r5.5.2.10.ei.b} &: \mathit{performConductingDetailedDesignReview} \\
&\Rightarrow [\mathit{OANPNP}]\mathit{provideSDD} \\
\mathbf{r5.5.2.10.ei.c} &: \mathit{performConductingDetailedDesignReview} \\
&\Rightarrow [\mathit{OANPNP}]\mathit{provideCDR} \\
\mathbf{r5.5.2.10.ei.d} &: \mathit{performConductingDetailedDesignReview} \\
&\Rightarrow [\mathit{OANPNP}]\mathit{provideTS} \\
\mathbf{r5.5.2.10.ei.e} &: \mathit{performConductingDetailedDesignReview} \\
&\Rightarrow [\mathit{OANPNP}]\mathit{provideICD} \\
\mathbf{r5.5.2.10.ei.f} &: \mathit{performConductingDetailedDesignReview} \\
&\Rightarrow [\mathit{OANPNP}]\mathit{provideSUM} \\
\mathbf{r5.5.2.10.ei.g} &: \mathit{performConductingDetailedDesignReview} \\
&\Rightarrow [\mathit{OANPNP}]\mathit{provideDJF} \\
\mathbf{r5.5.2.10.ei.h} &: \mathit{performConductingDetailedDesignReview} \\
&\Rightarrow [\mathit{OANPNP}]\mathit{provideSUITP}
\end{aligned}
$$

Requirements tailored as omitted (not applicable (D) in the ECSS Applicability Requirements Matrix (EARM)) can be formalized using PCP 1a. For example, it is defined that requirement 5.5.2.10, which is the definition of the task *Conducting a detailed design review* is omitted (see rule r5.5.2.10.Ommited). If we do not perform the review, their work products are also not required.

$$
\begin{aligned}
\mathbf{r5.5.2.10.Ommited} &: \mathit{provideJustificationNotPerformConductingDetailedDesignReview}, \\
&\mathit{JustificationNotPerformConductingDetailedDesignReviewWithVerificationByExpert}, \\
&\mathit{performedByAssesor}, \\
&\mathit{AsessorWithExperienceECSS} - E - ST - 40 - C \\
&\Rightarrow [P] - \mathit{performConductingDetailedDesignReview} \\
&\mathbf{r5.5.2.10.Ommited} > \mathbf{r5.5.2.10}
\end{aligned}
$$

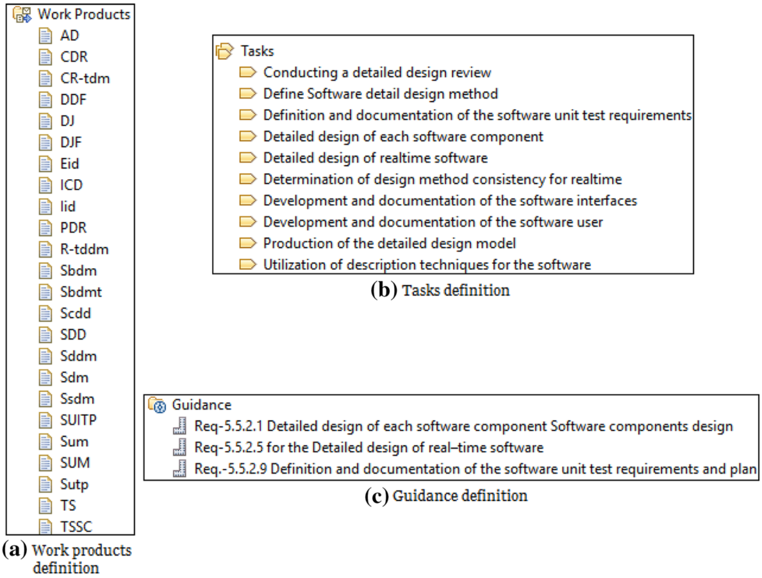
We use the PCP 1b, if the same task is defined in the EARM as applicable with modification (M), or new generated requirement (N). For illustration purpose, we consider that a simple review could be performed instead of the detailed review (see rule r5.5.2.10.ChangedRule).

$$
\begin{aligned}
\mathbf{r5.5.2.10.ChangedRule} &: -\mathit{performConductingDetailedDesignReview} \\
&\Rightarrow [\mathit{OANPNP}]\mathit{performSimpleReview}
\end{aligned}
$$

Propositions used in rules r5.5.2.10.Ommited and r5.5.2.10.ChangedRule should be added to the list of rule propositions presented in Fig. 8.

4.2 Modeling of process elements

Initial process plan elements are extracted from the standard ECSS-E-ST-40C, specifically, the requirements presented in Table 1. The result is process elements depicted in Fig. 10, which contains work products, tasks and guidance artifacts.



Work Products
- AD
- CDR
- CR-tdm
- DDF
- DJ
- DJF
- Eid
- ICD
- Iid
- PDR
- R-tddm
- Sbdm
- Sbdmt
- Scdd
- SDD
- Sddm
- Sdm
- Ssdm
- SUITP
- Sum
- SUM
- Sutp
- TS
- TSSC

(a) Work products definition

Tasks
- Conducting a detailed design review
- Define Software detail design method
- Definition and documentation of the software unit test requirements
- Detailed design of each software component
- Detailed design of realtime software
- Determination of design method consistency for realtime
- Development and documentation of the software interfaces
- Development and documentation of the software user
- Production of the detailed design model
- Utilization of description techniques for the software

(b) Tasks definition

Guidance
- Req-5.5.2.1 Detailed design of each software component Software components design
- Req-5.5.2.5 for the Detailed design of real–time software
- Req.-5.5.2.9 Definition and documentation of the software unit test requirements and plan

(c) Guidance definition

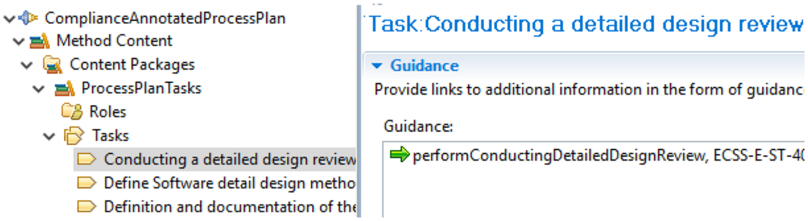
Fig. 10 Process elements plugin

4.3 Annotation of process tasks

A copy of the process elements defined in Sect. 4.2 (depicted in Fig. 10) is created in a new plugin, which we called *ComplianceAnnotatedProcessPlan*. These process elements are also extended to the original by using the content variability type *Extends*. With this extension, we ensure that the information previously defined is also included, such as the assignment of in/output or guidance to the tasks. Then, every process element is annotated with the compliance effect they produce by adding the guidance elements that contains the effect (see Fig. 11). The complete compliance annotation of process elements is presented in Table 5.

4.4 Modeling of process workflow

The tasks annotated in Sect. 4.3 are used to create the delivery process (see Fig. 12).



ComplianceAnnotatedProcessPlan
- Method Content
  - Content Packages
    - ProcessPlanTasks
      - Roles
      - Tasks
        - Conducting a detailed design review
        - Define Software detail design metho
        - Definition and documentation of the

Task:Conducting a detailed design review

Guidance

Provide links to additional information in the form of guidanc

Guidance:

performConductingDetailedDesignReview, ECSS-E-ST-4(

Fig. 11 Annotation of tasks

As depicted in Fig. 13, ACCEPT involves the modeling of four separated models, which are concretized in EPF Composer as plugins. The *ComplianceCheckingCustomization* (highlighted in red in the figure) is provided in the method and the process engineer only needs to use it.

4.5 Checking and analysis of compliance

The ruleset formalized in Sect. 4.1, and the workflow modeled in Sect. 4.4 (located in the plugins *ECSS-E-ST-40C-Requirements* and *ComplianceAnnotatedProcessPlan* in Fig. 13, respectively) are the two specifications required by Regorous (recalled in Sect. 2.4) to perform automatic compliance checking. The results of the checking are presented in Fig. 14.

As Fig. 14a depicts, *the process is non-compliant*. Thus, the plan needs improvement. Such results could point to compliance problems on the workflow, in the annotation process, or missing characteristics in the process plans (e.g., absent tasks or work products). The problems related to the rules (e.g., wrong formalization) need to be analyzed in the context of specific standard with experts, such as safety assessors. For example, there is a violation regarding provideAD (see Fig. 14b). It turns out that rule r5.5.2.b is violated (see Fig. 14c). The reparation policy suggests to *prevent the violation, by performing provideAD after 'Start'*. This means that in the first task, which is called *detailed design of each software component*, we need to add the input *AD* and its corresponding compliance effect *provideAD* (see Fig. 14d). When the compliance checking results are positive, namely the result is *the process is compliant*, it does not mean that there is no further improvement. Instead, we need to perform the analysis, taking into account the rules that did not fire. Once the process is improved for compliance, compliance checking is performed iteratively until the process plan is deemed compliant by Regorous.

5 Case study analysis

In this section, we analyse the case study results presented in Sect. 4 by answering the research questions defined in Sect. 3.2.

5.1 Effort designing a CAEPP for software development (RQ1)

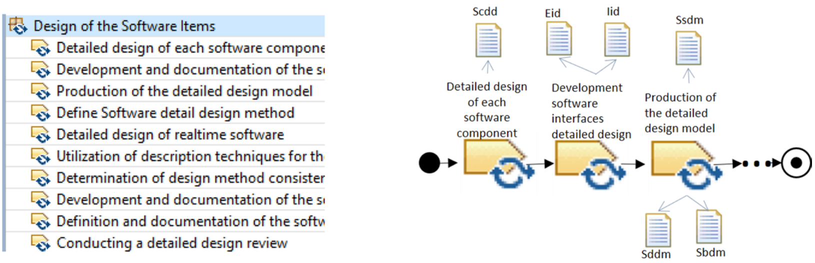
We judge the effort determined by task demand, which is based on design choices. In theory, such effort can be used to determine the intent to complete a task independently of any conscious actor. We refer in this analysis to the effort required to create and comprehend the models and document and manage evolution.

Table 5 Compliance effects annotation on process elements

Process element	Compliance Effect
Detailed design of each software component	performDetailedDesign
Scdd	provideScdd
Development/documentation of the software interfaces	performDevelopAndDocumentSwInterfacesDesign
Eid	provideEid
Iid	provideIid
Production of the detailed design model	performProductionDetailedDesign
Ssdm	provideSsdm
Sddm	provideSddm
Sbdm	provideSbdm
Define Software detail design method	performDescribeSwDetailDesignMethod
Sdm	provideSdm
Detailed design of real-time software	performDetailedRealTimeSw
R-tddm	provideR-tddm
Utilization of description techniques for the software	performDescribeTechniquesSwBehavior
Sbdmt	provideSbdmt
Determination of design method consistency for realtime	performDeterminationDesignMethodConsistencyRT
CRtdm	provideCRtdm
Development and documentation of the software user manual	performDocumentationSwUserManual
Sum	provideInitialSum
Definition and documentation of the software unit test requirements and plan	performDefinitionSwUnitTestReq
Sutp	provideSutp
Conducting a detailed design review	performConductingDetailedDesignReview
Req-5.5.2.1 Detailed design of each software component Software components design	guidedByReq5-5-2-1
Req-5.5.2.5 for the Detailed design of real–time software	guidedByReq5-5-2-5
Req.-5.5.2.9 Definition and documentation of the software unit test requirements and plan	guidedByReq-5-5-2-9

5.1.1 Effort required to create the models (RQ1.1)

When using ACCEPT for creating a CaEPP for software projects, three models (the *ComplianceCheckingCustomization* is provided in the method), which are concretized in EPF-C as plugins, are required (see Fig 13). To populate the *ECSS-E-ST-40C-Requirements* plugin, we needed to formalize requirements in FCL. Performing a formalization process is, in general, a difficult task that requires skills, which cannot be taken for granted. Moreover, due to the sheer size of the standards, this work requires time and focus. An advantage of FCL is that it provides a valid set of understandable concepts to the users of the standards, i.e., obligations, prohibitions, and permission (see Sect. 2.3). Moreover, ACCEPT provides process compliance hints and patterns (recalled in Sect. 2.5), which facilitate the identification and modeling of compliance artifacts, i.e., requirements and their corresponding rules as well as compliance effects (see Sect. 4.1). However, the instantiation of hints and patterns is done manually in a process that is repetitive and prone-to-error. Defining the *ProcessElements plugin* required in a process is a task that is not difficult to perform



Design of the Software Items
- Detailed design of each software compone
- Development and documentation of the s
- Production of the detailed design model
- Define Software detail design method
- Detailed design of realtime software
- Utilization of description techniques for th
- Determination of design method consister
- Development and documentation of the s
- Definition and documentation of the softw
- Conducting a detailed design review

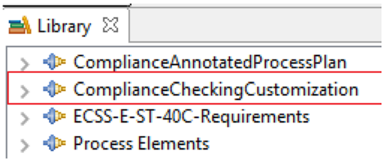
Scdd
Eid
Iid
Ssdm
Detailed design of each software component
Development software interfaces detailed design
Production of the detailed design model
Sddm
Sbdm

(a) Breakdown structure.

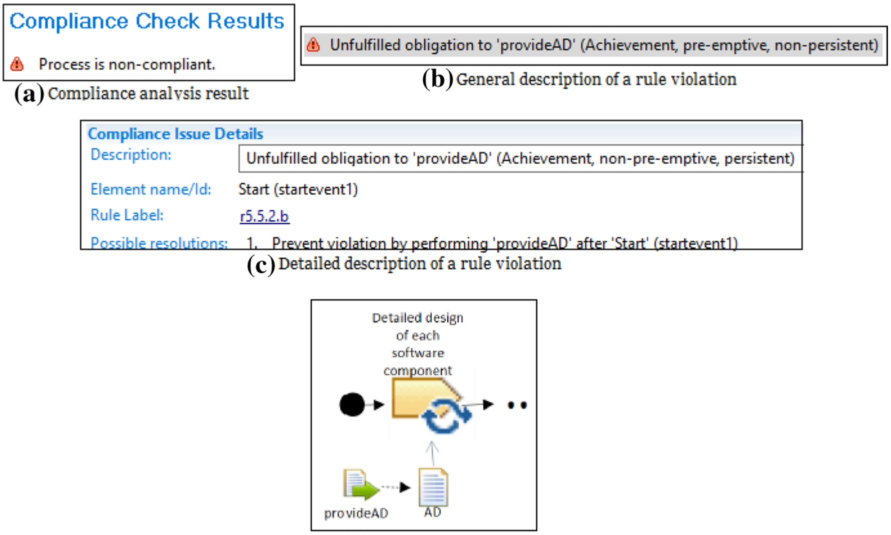
(b) Activity diagram.

Fig. 12 Delivery process

Fig. 13 EPF-C plugins



Library
- ComplianceAnnotatedProcessPlan
- ComplianceCheckingCustomization
- ECSS-E-ST-40C-Requirements
- Process Elements



Compliance Check Results

Process is non-compliant.

(a) Compliance analysis result

Unfulfilled obligation to 'provideAD' (Achievement, pre-emptive, non-persistent)

(b) General description of a rule violation

Compliance Issue Details

Description:	Unfulfilled obligation to 'provideAD' (Achievement, non-pre-emptive, persistent)
Element name/Id:	Start (startevent1)
Rule Label:	r5.5.2.b
Possible resolutions:	1. Prevent violation by performing 'provideAD' after 'Start' (startevent1)

(c) Detailed description of a rule violation

(d) Improved task

Fig. 14 Compliance checking results

since EPF-C has graphical representations of the elements that can be modeled in a well-defined structure (see Sect. 4.2). However, a specific effort in terms of time is also required. ACCEPT states that the elements required in the *ComplianceAnnotatedProcessPlan* plugin are linked to the elements in other models in two ways (see

Sect. 4.3). (1) By performing extensions between elements in the method content, to inherit the characteristics of the process elements. (2) By performing the compliance annotation process, which is the method that guarantees that the model is checkable for compliance. Currently, compliance annotations are performed manually, based on the domain expert's knowledge about the engineering process.

In summary, there is a need to manually (and iteratively) formalize requirements, graphically model compliance and process artifacts, and extend and annotate compliance effects. Thus, the effort required to create models is significant and may increase with the continued attempting to do the same tasks repeatedly. However, the effort required to model the *ECSS-E-ST-40C-Requirements* plugin is only significant during the first time. The reason is that such a model can be used several times in different CaEPPs that are modeled in compliance with the same standard (until new versions of the standard are released). Similar situations could occur with the other plugins, but they need to be evaluated in project-specific circumstances.

5.1.2 Effort required to comprehend processes and standards models (RQ1.2)

The method uses artifacts that are systematically organized in a hierarchical and visual structure that permits the identification of compliance information. In particular, standard requirements and their elements are arranged in a nested list of compliance artifacts (see Fig. 9a). Moreover, process elements are created in particular structures that differentiate, e.g., work products from tasks (see Fig. 10). The abstract association of elements within process tasks (depicted in Fig. 2b) permits the comprehension of the required compliance information provided by the compliance effects. This abstraction provides an approach for direct requirements allocation into process models. Thus, once the models are created, there is a required low effort to comprehend the information they contain. In summary, the models created in EPF-C have a specific structure that facilitates the visualization of their artifacts and their use, proving models that have an advantage over, e.g., text-based approaches.

5.1.3 Effort required to document compliance (RQ1.3)

The ability to provide means to document method content and processes in SPEM 2.0-like elements was exploited in ACCEPT. Having an structural, hierarchical representation of the standards (see Fig. 9a) with descriptive information (see (see Fig. 9b), as well as content elements organized according to their function (see Fig. 10) helps to have a written record of the artifacts required for compliance. This structural representation, originally provided by SPEM 2.0, may facilitate the work of a third party (independent) assessor in case the parties decide to include additional certainty to their assessment schema. Thus, the required high modeling effort results in lower compliance documentation effort.

5.1.4 Effort required to manage evolution (RQ1.4)

The compliance information in ACCEPT created in Sect. 4.1 could be seen as an initial frozen specification of the standard. However, such specification does not need to be bypassed altogether, when obsolescence no longer stands the strain of being frozen, i.e., a new version of the standard is released. In normal conditions, only some requirements change, and some get deprecated, but the majority remain. For example, the ECSS-E-ST-40-C has a log, which explicitly describes few adjustments regarding previous versions (ECSS-E-40 Part 1B, released on 28 November 2003, and the ECSS-E-40 Part 2B, released on 31 March 2005). In ACCEPT, such changes can be embraced. First, as specifications are reusable, a copy of the requirements can be performed and saved with the new version name. Second, as the requirements model is hierarchically designed, the changes can be absorbed in an orderly way. Once a new version of the standards is defined, changes in the rules may impact the compliance status. Thus, it becomes necessary to re-check the process plans. However, as the rulesets are executable, the checking is easier, and process plans can be improved according to the new version of the ruleset (as described in Sect. 4.5). However, standards evolution would require some human intervention. In particular, there is a need for monitoring the changes in the standards and maintain accuracy in the rulesets. EFP-C provides textual descriptions regarding, i.e., versioning or revisions, which can be used to maintain a log of information between users facilitating further revision work.

5.2 Coverage level of a CaEPP for software development (RQ2)

We judge the models' coverage level, taking into account how the information provided by the CaEPP models fit in the information required by the ECSS-E-ST-40C framework. We refer to the coverage level of the models, the compliance documentation, and the evolution management.

5.2.1 Level of coverage of the models (RQ2.1)

The models used in ACCEPT cover several aspects required in process compliance. First, standard artifacts (see Fig. 9a) are represented by a structure that covers the textual description of the requirements, their respective FCL rules, and compliance effects. Second, the method content provided (see Fig. 10) covers the elements required to describe detailed process plans. Third, the compliance effects annotation (see Fig. 11) covers the requirements allocating into process plans, which permits us to understand the explicit relationships between artifacts. Finally, the compliance analysis provided by Regorous (see Fig. 14) covers the compliance status of the process plan, the compliance violations, and possible resolutions that facilitate the compliance analysis.

5.2.2 Level of coverage of the compliance documentation (RQ2.2)

As previously described, the models provide all the required information for documenting compliance. Moreover, the compliance analysis is detailed enough to describe compliance status (full or non-compliance), the compliance violations, and the inactive rules. For our particular case study, this approach is sufficient. On the one side, as recalled in Sect. 2.1.2, a customer of a space software project needs to provide an ECSS Applicability Requirements Matrix (EARM), which can be extracted from the model that contain the requirements, i.e., the ECSS-E-ST-40C-Requirements plugin (see Fig. 9a). As the figure depicts, the plugin contains the requirements identifier and the text. Moreover, the applicability status can be obtained from the description of the identifier in the rules. For example, tailored out requirements are identified with the particle *Ommited* (see rule r5.5.2.10.Ommited). In contrast, modified ones are identified with the particle *ChangedRule* (see rule r5.5.2.10.ChangedRule). On the other side, the supplier needs to respond with the ECSS Compliance Matrix (ECM), which should be done at the level of each requirement (as opposed to a global statement of compliance) in order to allow the customer to detect early enough in the project the non or partial compliance. The information required in the ECM can be extracted from the compliance checking results (see Fig. 14b and c). Such results will identify compliance violations to the rules that belong to the requirements explicitly defined by the customer in the EARM. An analysis of the violations may lead to modification of the requirements upon agreements between the parties when compliance has become excessively demanding or unreachable (due to unpredictable or changing conditions in the project).

5.2.3 Level of coverage of the evolution management (RQ2.3)

ACCEPT is defined in an authoring platform that permits the organization the compliance information as standards evolve. Thus, successive models that represent the evolution of the standards can be defined and stored in EPF-C plugins as a library of reusable knowledge (see Fig. 13). Administrative directives from the organization that apply the standards could also be defined and included as FCL rules (as presented in Sect. 4.1). Consequently, the process engineer, whose expertise may be limited by specific knowledge, could find the hints that facilitate applying the specific standard version. Moreover, the process engineer could also include his/her knowledge (or lessons learned) after performing compliance practices, as part of the documentation that is permitted by EPF-C.

6 Discussion

As presented in Sect. 2.1.1, a software process engineer is responsible for the composition and documentation of compliant software engineering processes plans. In general, the planning of engineering processes, which criteria could be initially abstracted from ad-hoc practices, needs to be concretized to support manufacturers in achieving goals. Specifically, in the space context, baseline criteria for software process planning are defined by the de-facto standard ECSS-E-ST-40C (recalled in Sect. 2.1.2). ECSS-E-ST-40C proposes reference models that prescribe artifacts related to planning activities, i.e., a set of units of work necessary to engineer systems. ECSS-E-ST-40C also contains process-related requirements, which prescribe properties for the activities, e.g., the prerequisites for performing activities, the work products to be produced, and specific guidance (see Table 4). Guidance elements may not be required for compliance auditing. However, internal policies in a company may impose the need to have guidance that facilitates the process's execution. All those requirements need to be specified in the project-related documents, e.g., the EARM, after careful selection by the customer. The requirements specification should contain the definition of one party's obligations towards the other and the authorization from the customer to the supplier to deviate from the standard requirements. The specification of the customer's requirements is an input for software project-specific contractual agreements with the supplier, who use them to define a CaEPP to perform the job. Thus, for defining contractual obligations regarding software projects, the discussions about the technical specifications based on the requirements baseline provided by ECSS-E-ST-40C must be carried out early in the lifecycle process. Selected requirements must be correctly adopted in the software engineering process plan. Otherwise, they may constitute a legal cause of action for breaching the contractual agreements.

Manually checking software process plans compliance with the EARM requirements is a common practice in this context. Indeed, the ECSS secretariat provides the EARM matrix with all requirements[2] for that purpose. Filled checklists highlight specific defects in the process (e.g., missed tasks) respect the defined requirements, which could be the source of compliance risks (as well as legal risks) during process execution. Besides, these checklists provide hints to improve processes performance and re-negotiate requirements if full compliance has become too demanding or unnecessary for the specific project. However, performing manual checks could be overwhelming. In particular, the knowledge included in ECSS-E-ST-40C is abundant (656 requirements), and their complexity (there are connections between different requirements and standards) have a direct implication in the correctness of the resulting process plans, i.e., the sequencing of process tasks and the definition of the properties of such tasks. Moreover, standards evolve (new versions are frequently released). Extensive process plans, which typically have a high number of states and transitions, are difficult to verify against industry standards' changing nature. Thus, the lack of methodological support for dealing with compliance management could involve unstructured practices, uncertain outcomes, compliance, and legal risks. Due to the fact that we are performing a single-case study, no firm conclusions should be done for the results. However, the data collected (see Sect. 4) and its analysis results (see Sect. 5) can be used as indications to guide the shaping of future designs and prototypes. In the remaining part of this section, we present a specific discussion

[2] https://ecss.nl/standards/downloads/doors-download/

regarding the case study insights as well as the challenges and potential improvements that could be done to enhance ACCEPT.

6.1 Case study insights

When planning a software engineering process plan, the challenge is to understand how many process elements should be specified and their order. In the case study conducted, we defined a model of a software process plan by the book, i.e., we extract the process elements suggested by the selected portion of ECSS-E-ST-40C standard without any tailoring. It is called CaEPP since such process elements are enriched with compliance information. In case of deviation (e.g., tailoring), we can also know if the requirements are tailored out or modified (as done for rules r5.5.2.10.Ommited and r5.5.2.10.ChangedRule in Sect. 4.1). ACCEPT states that creating a CaEPP requires several models, which design is a process that is not free of effort, as presented in Sect. 5.1.1. However, initial observations have shown that the effort required to comprehend processes and standards models (see Sect. 5.2.2) and document models (see Sect. 5.1.3) is significantly less. The reason is that formal specifications are accompanied by informal explanations that clarify their meaning and place them in context. Moreover, the visual approach adopted allows for more focused reviews. It is clear that organizations may depart from normative practices (not creation process plans by the book) for project-inherent reasons that can be justified. In such cases, logic-based requirements representations can be effortlessly superseded (as analyzed in Sect. 5.1.4). In addition, the level of coverage of the models is higher (as analyzed in Sect. 5.2). Thus, we can take good advantage of such an initial effort in the long term. Specifically, the models are created in an authoring environment that permits a well-defined organization of compliance-related artifacts in a hierarchical, visual, and enriched structure, which can be reused. This modeling strategy minimizes the distance between the specification of the requirements' normative intention and the process elements that should respond to such requirements. We could also include the possible exceptions that are derived from the deviations. Once the models are created, the process plans' validity can be established by doing automatic reasoning about the standard conditions. In particular, compliance violations could be drafted better since failure to requirements is connected to textual sources. Therefore, the comprehension of processes, standards, and their relationships is more natural, and the documentation of compliance and the management of evolution get better support than in manual checklists. These features are a valuable gain since once industry standards and process plans are formalized, process engineers do not need to expend valuable hours on reading regulatory documentation to infer the actions that must be taken to maintain compliance.

6.2 Challenges and potential improvements

A key challenge in the use of ACCEPT is that standards are currently written in natural language, and formalizing them is an intimidating and fairly sophisticated task. The reason is that the number of requirements in a standard is significant and context-specific. Thus, their interpretation requires expertise. However, FCL has a limited set of constructs, which provide the expressivity required for formalizing requirements. Such constructs also provide a framework for thinking about the requirements in terms of deontic notions and exceptions, which could simplify their interpretation. Therefore, showing process engineers the FCL potential and its easy to use aspect may strive the interest for its exploitation.

The work to be done when creating a CaEPP for software projects in the space context has the tendency to be repetitive. Repetition could cause a drop in a subject's capacity to perform the modeling task (i.e., disinterest, boredom, fatigue), making relative task demands greater than necessary. Further automation of such tasks might reduce the absolute demands, and thus the actual effort. For example, the manual creation of the compliance effects is a repetitive task that has to be done for each effect. In this case, we repeated this task 48 times (see Fig. 8). It was also prone to error since the effects' names have similarities (e.g., almost all the tasks' effects have the word design). Thus, we needed to review our design several times and manually track the information we were writing in EPF-C. However, this task is systematic and supported by templates. As such, it could be automatized by using a domain-specific language that permits an adequate characterization of the specific compliance effects and their production.

In general, different mechanisms can be defined to determine the meaning of context-dependent situations that could affect rules' formalization. In particular, patterns facilitate the recognition of relevant requirements, improving efficiency and consistency when producing rules. Our current selection of compliance patterns is limited to general situations, and they are also manually instantiated. Still, they can provide some assistance. Moreover, the process compliance hints could be used to establish conceptual relationships between the elements composing process plans and their compliance effects in the general compliance status. Thus, automated formalization of requirements could also be provided by performing an intermediate translation step into controlled English. For the compliance annotation of processes, programming scripts that examine the semantic similarity between process elements and compliance effects can be created. The modeling part could also be facilitated by providing general-purpose process model repositories to process engineers. Indeed, EPF-C offers such kind of repositories with libraries that can be downloaded and assemble in specific projects[3].

[3] https://www.eclipse.org/epf/downloads/praclib/praclib_downloads.php

7 Related work

ECSS standards are difficult to manage since they involve hundreds of pages containing around 25.000 requirements for the development and operations of European Space Systems. These standards are available in the form of documents (Word and PDF). In an effort for helping organizations, The European Space Agency (ESA)[4] provides an Excel document that contains the ECSS Applicability Requirement Matrix (EARM), which is useful for selecting requirements and document tailoring procedures. However, process engineers need to check the applicable requirements one by one. In addition, the data model requirements are specified in the ECSS digital Requirements Management System (E-RMS) conceptual data model (ESA 2018), which guarantees the persistence of the ECSS requirements, but it does not have facilities for process modeling processes. A more sophisticated approach is presented in Armbrust et al. (2005), which proposes a persistent connection via relational databases to word processor documents that contain the work products required in the standard. With this approach, compliance checking results may be incomplete, as not all activities produce work products, leaving mandatory activities out of the checking scope. In contrast, ACCEPT provides compliance checking of the process workflow, which not only takes into account the results of the tasks (e.g., work products). For this reason, it is more useful at planning stages. Moreover, ACCEPT is not only domain-specific as the ESA Excel document. In principle, any process-based requirements catalog can be formalized, uploaded and applied to a variety of safety-critical related software process plans.

Compliance-related artifacts modeling has also been the target of some research efforts. For example, in Panesar-Walawege et al. (2010), the authors provide a model of the process concepts, via the UML (OMG 2017) class diagrams. Automated rule checking with OCL (Object Constraint Language) constructs[5] is also suggested (but not implemented). In de la Vara and Panesar-Walawege (2013), the authors introduce SafetyMet. SafetyMet is a generic metamodel that includes the concepts and relationships common to different safety standards and project practices. With SafetyMet, mapping standards models and project information is also possible. In Eito-Brun and Amescua (2017), the authors describe a SPEM 2.0 extension, which incorporates process requirements, guidelines, and their properties. The extension is used to generate an ontological representation that can be visualized with the Semantic Media Wiki (SMW)[6]. In contrast to the work presented in Panesar-Walawege et al. (2010), de la Vara and Panesar-Walawege (2013), Eito-Brun and Amescua (2017), we consider SPEM 2.0 (without performing any extension), an Object Management Group specification[7] that is well-documented, mature, and open, and permits to model not only the processes and their related library but also the artifacts required for performing compliance checking.

[4] https://ecss.nl/

[5] https://wiki.eclipse.org/OCL

[6] https://www.semantic-mediawiki.org/wiki/Semantic_MediaWiki

[7] https://www.omg.org/spec/

Logic-based approaches have also provided a suitable framework to represent and reasoning upon normative knowledge. In Emmerich et al. (1999), the authors propose a document schema specification in UML. The properties of the documents prescribed by the standards are formalized in first-order logic (FOL). Checks are performed when there is an attempt to read/write documents during process enactment. FOL can express property specifications, but it is insufficient to express the sequence of tasks in a process plan. In Martinelli et al. (2019), the authors propose an approach for compliance checking based on temporal logic. However, it does not build an operational model of the process from the beginning. Instead, it needs the extraction of knowledge from event logs provided by the systems to create traces. Traces only contain tasks, which means that other process elements are not explicitly defined. As in Emmerich et al. (1999), this approach only detects uncompliant states in the process execution. Moreover, the explicit definition of process elements beyond tasks is not possible. In Bartolini et al. (2016), the authors present a framework based on Natural Language Semantics and Natural Language Processing techniques for recognizing correlations between provisions in a standard and requirements in a given law. However, it does not provide logic constructs to represent process plans, which is also part or our work. Our approach, as in Emmerich et al. (1999), Martinelli et al. (2019), Bartolini et al. (2016), uses logical-based approaches for the formulation of requirements constraints. However, it is process-centered, planning-time, which means that a process and its elements are essential inputs.

Compliance checking by design is approached in the safety-critical context. In Chung et al. (2008), the authors propose the comparison between an initial model of the process lifecycle prescribed by the standards and the plan provided by the users. The compliance checking is the result of the matching between the two process-based models. In contrast to the approach provided by Chung et al. (2008), other approaches provide a comparison between the process and the requirements specification. For example, the authors in Golra et al. (2017) propose a framework that uses Linear Temporal Logic to model a specification of the reference model provided by a standard. This specification is used to check the model of a SPEM 2.0 process. The work presented in Rodriguez et al. (2010), Valiente et al. (2012) aims at facilitating the checking of constraints by using SWRL (Semantic Web Rule Language) (Horrocks et al. 2004) on a process defined in SPEM 2.0. In Wang et al. (2006), the authors present an approach for representing SPEM 2.0 process models in Description Logics, to provide process analysis such as reasoning and consistency checks. ACCEPT has similarities with the approaches provided in the previous works (Golra et al. 2017; Rodriguez et al. 2010; Valiente et al. 2012). First, it considers that the model of the norms and the model of the process are necessary inputs for compliance checking. Second, processes are modeled with SPEM 2.0-like artifacts. However, ACCEPT uses customized SPEM 2.0-like artifacts to provide visual relationships between compliance artifacts. Moreover, ACCEPT uses FCL, which is a deontic language able to directly provide normative notions, i.e., obligation, prohibition, and permission, without the need for combining expressions. Moreover, FCL, which is also a defeasible logic, is capable of providing the management of the tailoring rules.

8 Conclusions and future work

ACCEPT is a framework based on a proactive strategy called compliance-by-design that permits process engineers to create compliance-aware engineering process plans (CaEPP). A CaEPP can show the planning-time allocation of standard demands, i.e., if the elements set down by the standard requirements are present at given points in the engineering process plan. A CaEPP avoids that process engineers experience the tasks related to process compliance management as reactive, i.e., it provides a risk control mechanism at planning-time that facilitates the decision-making process. Thus, a CaEPP could increase confidence in process compliance, which at the same time could reduce liability in case of an adverse event occurs. This situation is essential in the safety-critical context since the duty of care and standards compliance are typically linked together. In this paper, we performed a case study to understand if the ACCEPT produced models could support the planning of space software engineering processes. Space software is safety-critical since a software failure could cause a space mission disaster leading to financial losses, environmental pollution, and people's endangerment. Space software production is frequently the result of industrial cooperation. Such cooperation is coordinated through compliance with relevant standards. In the European space context, in which projects are share between companies that act as supplies with others that act as customers, the de-facto standard that regulated software development is the ECSS-E-ST-40C. Such a standard provides requirements that help customers formulate their project-specific requirements (ECSS Applicability Requirements Matrix or EARM) and suppliers to prepare their responses and implement the work (ECSS Compliance Matrix or ECM). For this reason, the planning of software engineering processes in compliance with project-specific ECSS-E-ST-40C applicable requirements is mandatory during contractual agreements. The sheer volume of the requirements in this specific standard, which requires tailoring and documentation, make compliance duties challenging. The case study's goal was to qualitatively analyze the current effort required to model a CaEPP in ACCEPT for software development processes in compliance with ECSS-E-ST-40C and the coverage level of such models. In particular, we analyzed actual effort, which is determined by task demands. Initial observations show that the effort required to model compliance and processes artifacts is significant. However, the effort is reduced in the long term since models are, to some extend, reusable and flexible. Thus, process engineers in the space context do not need to start from scratch in every project. Reusing artifacts in the compliance checking process may simplify the work that process engineers need to perform in every process planning. Such gain could be interpreted as a benefit in terms of resource savings since professionals' time is costly. We also analyzed the coverage level of the models based on design decisions. In our opinion, such a coverage level is adequate since it responds to the information needs required by the ECSS-E-ST-40C framework, i.e., information requested by EARM and ECM matrices, the process they regulate, and their required alignment (compliance annotations, analysis, and results).

The outcome of this case study only applies to compliance-aware software engineering processes with the characteristics demanded by ECSS-E-ST-40C. Other safety-critical engineering processes, such as safety-critical processes in chemical plants, may exhibit additional challenges. Thus, to generalize our framework capabilities, we have to perform more case studies with a broader range of standards applicable to the safety-related context. Additional case studies may also help us further improve our framework and provide more cases that facilitate its introduction in different contexts. However, the analysis performed in this case study gave us insights that could lead to additional refinements and improvements. In general, ACCEPT methodology is systematic and can be further automated. Thus, we consider adding mechanisms that facilitate the edition of rules and the use of templates for safety compliance hints and patterns. We also aim to design algorithms that facilitate the automation of the formalization of requirements and examine the semantic similarity between process elements and compliance effects to facilitate the compliance annotation of process elements. In terms of analysis, we have a further job to do. The experience of effort (or perception of effort) is a factor that is also important to analyze since it can provide feedback on task difficulty. Therefore, we plan to conduct experiments that include users perceiving effort in the modeling tasks required to create CaEPPs. Moreover, we aim to evaluate user acceptance by using frameworks, such as the technology acceptance model (TAM) (Davis 1985)). We also need to specify well-defined metrics to demonstrate our approach's value-add in terms of efficiency. Finally, we could generate fitness functions that facilitate calculations regarding the adequacy of the information coverage level provided by the models, the compliance documentation, and the evolution management.

Acknowledgements This work is supported by the EU and VINNOVA via the ECSEL JU project AMASS (No. 692474)

Funding Open access funding provided by Mälardalen University.

References

Ahmad E, Raza B, Feldt R, Nordebäck T (2010) Ecss standard compliant agile software development: an industrial case study. In: Proceedings of the 2010 national software engineering conference, pp 1–6

Armbrust O, Ocampo A, Soto M (2005) Tracing process model evolution: a semi-formal process modeling approach. In: ECMDA Traceability Workshop, pp 57–66

Bartolini C, Giurgiu A, Lenzini G, Robaldo L (2016) Towards legal compliance by correlating standards and laws with a semi-automated methodology. In: Benelux conference on artificial intelligence. Springer, pp 47–62

Baumgarten G, Rosinger M, Todino A, de Juan Marín R (2015) Spem 2.0 as process baseline meta-model for the development and optimization of complex embedded systems. In: 2015 IEEE international symposium on systems engineering (ISSE). IEEE, pp 155–162

Blackwelder B, Coleman K, Colunga-Santoyo S, Harrison JS, Wozniak D (2016) The volkswagen scandal

Boutros T, Purdie T (2014) The process improvement handbook: a blueprint for managing change and increasing organizational performance. McGraw-Hill Education, New York

Buglione L, April A, Rejas-Muslera RJ (2010) The need for a legal perspective in software engineering maturity models. Intell Property 4(9):10

Castellanos Ardila J (2019a) Facilitating Compliance Checking of Processes against Safety Standards. Licentiate thesis, Mälardalen University, Sweden

Castellanos Ardila JP (2019b) Facilitating automated compliance checking in the safety-critical context. Electronic Communications of the EASST, p 78

Castellanos Ardila JP, Gallina B (2017) Formal Contract Logic Based Patterns for Facilitating Compliance Checking against ISO 26262. In: 1st workshop on technologies for regulatory compliance, pp 65–72

Castellanos Ardila JP, Gallina B (2020) Separation of concerns in process compliance checking: Divide-and-conquer. In: 27th European & Asian System, Software & Service Process Improvement & Innovation. Springer, pp 135–147

Castellanos Ardila JP, Gallina B, Muram FU (2018a) Transforming spem 2.0-compatible process models into models checkable for compliance. In: International conference on software process improvement and capability determination. Springer, pp 233–247

Castellanos Ardila JP, Gallina B, Ul Muram F (2018b) Enabling Compliance Checking against Safety Standards from SPEM 2.0 Process Models. In: Euromicro conference on software engineering and advanced applications. pp 45 – 49

Chung PW, Cheung LY, Machin CH (2008) Compliance flow-managing the compliance of dynamic and complex processes. Knowl Based Syst 21(4):332–354

Cosgrove J (2001) Software engineering and the law. IEEE Software 18(3):14–16

Cruz BS, de Oliveira Dias M (2020) Crashed boeing 737-max: Fatalities or malpractice? GSJ 8(1):2615–2624

Cusumano MA (2004) Who is liable for bugs and security flaws in software? Commun ACM 47(3):25–27

Davis FD (1985) A technology acceptance model for empirically testing new end-user information systems: Theory and results. PhD thesis, Massachusetts Institute of Technology

de la Vara JL, Panesar-Walawege RK (2013) Safetymet: A metamodel for safety standards. In: International conference on model driven engineering languages and systems. Springer, pp 69–86

de la Vara JL, Parra E, Ruiz A, Gallina B (2019) Amass: a large-scale european project to improve the assurance and certification of cyber-physical systems. In: International conference on product-focused software process improvement. Springer, pp 626–632

de la Vara JL, Parra E, Ruiz A, Gallina B (2020) The amass tool platform: An innovative solution for assurance and certication of cyber-physical systems. In: Joint Proceedings of REFSQ-2020 Workshops, Doctoral Symposium, Live Studies Track, and Poster Track co-located with the 26th International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2020) CEUR Workshop Proceedings, Vol-2584, urn:nbn:de:0074-2584-1

Denning PJ, Tedre M (2019) Computational thinking. MIT Press, Cambridge

Dowson M (1997) The ariane 5 software failure. ACM SIGSOFT Softw Eng Notes 22(2):84

Dwyer MB, Avrunin GS, Corbett JC (1999) Patterns in property specifications for finite-state verification. In: 21st international conference on software engineering, pp 411–420

Eastaughffe K, Cant A, Ozols M (1999) A framework for assessing standards for safety critical computer-based systems. In: 4th IEEE International software engineering standards symposium and forum.'best software practices for the internet age. IEEE, pp 33–44

Eclipse Foundation (2018) Eclipse Process Framework (EPF) Composer

Eito-Brun R, Amescua A (2017) Dealing with software process requirements complexity: an information access proposal based on semantic technologies. Requir Eng 22(4):527–542

Emmerich W, Finkelstein A, Montangero C, Antonelli S, Armitage S, Stevens R (1999) Managing standards compliance. IEEE Trans Softw Eng 25(6):836–851

ESA (2009a) ECSS-E-ST-40C—Space Engineering Software

ESA (2009b) ECSS-S-ST-00C—System, Description, implementation and general requirements

ESA (2013) ECSS-E-HB-40C—Space engineering—Software engineering handbook

ESA (2017) ECSS-Q-ST-40C—Space product assurance—Safety

ESA (2018) ECSS MasterDB—User requirements Document

Gallina B, Gómez-Martínez E, Earle CB (2016) Deriving safety case fragments for assessing mbasafe's compliance with en 50128. In: International conference on software process improvement and capability determination. Springer, pp 3–16

Gallina B, Muram FU, Castellanos Ardila JP (2018) Compliance of agilized (software) development processes with safety standards: a vision. In: 19th international conference on agile software development, pp 1–6

Generowicz M (2013) The easy path to functional safety compliance

Ghanavati S, Amyot D, Peyton L (2008) Comparative analysis between document-based and model-based compliance management approaches. In: Requirements engineering and law, pp 35–39

Golra F, Dagnat F, Bendraou R, Beugnard A (2017) Continuous process compliance using model driven engineering. in: international conference on model and data engineering. Springer, pp 42–56

Governatori G (2005) Representing business contracts in RuleML. Int J Cooper Inf Syst, pp 181–216

Governatori G (2015) The Regorous approach to process compliance. In: IEEE 19th international enterprise distributed object computing workshop, pp 33–40

Harkiolakis N (2013) Assurance. Springer, Berlin, pp 122–127

Hashmi M, Governatori G, Wynn M (2013) Normative requirements for business process compliance. Lecture Notes Bus Inf Process 177:100–116

Horrocks I, Patel-schneider P, Boley H, Tabet S, Grosof B, Dean M (2004) SWRL: A semantic web rule language combining OWL and RuleML. W3C Member Sub 21(79): 1–31

Icheku V (2011) Understanding ethics and ethical decision-making. Xlibris Corporation

IEC 61508 (2010) Functional safety of electrical/electronic/programmable electronic safety-related systems

Ingolfo S, Siena A, Mylopoulos J (2011) Establishing regulatory compliance for software requirements. In: International conference on conceptual modeling. Springer, pp 47–61

Javed MA, Gallina B (2018a) Get EPF Composer back to the future: a trip from Galileo to Photon after 11 years. In: EclipseCon

Javed MA, Gallina B (2018b) Safety-oriented Process Line Engineering via Seamless Integration between EPF Composer and BVR Tool. In: 22nd international systems and software product line conference, pp 23–28

Kalus G, Kuhrmann M (2013) Criteria for software process tailoring: a systematic review. In: International conference on software and system process, pp 171–180

Kienle HM, Sundmark D, Lundqvist K, Johnsen A (2012) Liability for software in safety-critical mechatronic systems: An industrial questionnaire. In: 2nd international workshop on software engineering for embedded systems. IEEE, pp 44–50

Ladkin PB (2019) Duty of care and engineering functional-safety standards. Digital Evidence & Elec. Signature L. Rev. 16: 51

Leveson N (2020) Are you sure your software will not kill anyone? Commun ACM 63(2):25–28

Leveson N, et al (1995) Medical devices: The therac-25. Appendix of: Safeware: System Safety and Computers

Leveson NG (2016) Engineering a safer world: systems thinking applied to safety. The MIT Press, Cambridge

Lill A (2018) Definition of an Agile Software Development Process for the European Space Industry. Master thesis, Technische Univesität München

Lu R, Sadiq S, Governatori G (2007) Compliance aware business process design. In: International conference on business process management. Springer, pp 120–131

Martinelli F, Mercaldo F, Nardone V, Orlando A, Santone A, Vaglini G (2019) Model checking based approach for compliance checking. Inf Technol Control 48(2):278–298

Moyón F, Méndez D, Beckers K, Klepper S (2020) How to integrate security compliance requirements with agile software engineering at scale? In: International conference on product-focused software process improvement. Springer, pp 69–87

OMG (2008) Software & Systems Process Engineering Meta-Model Specification. V. 2.0

OMG (2017) Unified Modeling Language Specification V. 2.5.1

O'Regan G (2018) Overview of software engineering. World of Computing. Springer, Cham, pp 179–202

Panesar-Walawege RK, Sabetzadeh M, Briand L, Coq T (2010) Characterizing the chain of evidence for software safety cases: A conceptual model based on the iec 61508 standard. In: 3rd International Conference on Software Testing, Verification and Validation. IEEE, pp 335–344

Rantala V, Könnölä K, Suomi S, Isomäki M, Lehtonen T (2017) Agile embedded system development versus european space standards. Int J Inf Syst Social Change 8(1):1–23

Rodriguez D, Garcia E, Sanchez S, Nuzzi CR-S (2010) Defining software process model constraints with rules using owl and swrl. Int J Softw Eng Knowl Eng 20(4):533–548

Ruiz A, Gallina B, de la Vara JL, Mazzini S, Espinoza H (2016) Amass: Architecturedriven, multi-concern, seamless, reuse-oriented assurance and certification of cpss. In: 5th International Workshop on Next Generation of System Assurance Approaches for Safety-Critical Systems (SASSUR), Trondheim, Norway, September, Computer Safety, Reliability, and Security (SAFECOMP), Lecture Notes in Computer Science, vol 9923, pp 311–321

Ruiz-Rube I, Dodero JM, Palomo-Duarte M, Ruiz M, Gawn D (2012) Uses and applications of spem process models a systematic mapping study. J Softw Maintenance Evolut Res Practice 1(32):999–1025

Runeson P, Höst M, Rainer A, Regnell B (2012) Case study research in software engineering: guidelines and examples. Wiley, Hoboken

Sadiq S, Governatori G, Namiri K (2007) Modeling Control Objectives for Business Process Compliance. In: International conference on business process management, pp 149–164

Schwartz A (2000) Statutory interpretation, capture, and tort law: the regulatory compliance defense. Am Law Econ Rev 2(1):1–57

SEI (2011) CMMI for Development V. 1.3– Capability Maturity Model Integration

Siena A, Mylopoulos J, Perini A, Susi A (2008) From laws to requirements. In: 2008 Requirements engineering and law. IEEE, pp 6–10

SINTEF (2016) BVR Tool, https://github.com/SINTEF-9012/bvr

Smith D (1985) The design of divide and conquer algorithms. Sci Comput Program 5:37–58

Steele J (2020) What is (perception of) effort? objective and subjective effort during task performance. PsyArXiv, https://. https://doi.org/10.31234/osf.io/kbyhm

Tuft B (2010) Eclipse Process Framework (EPF) Composer: Installation. Introduction, Tutorial and Manual

U.S. FDA (1906) U.S. Food and Drug-Medical Devices

Valiente M, García-Barriocanal E, Sicilia M (2012) Applying ontology-based models for supporting integrated software development and IT service. IEEE Trans Syst Man Cybern 42(1):61–74

Walkinshaw N (2017) Software quality assurance. Springer Int Publ 10:978–983

Wang S, Jin L, Jin C (2006) Represent software process engineering metamodel in description logic. World Acad Sci Eng Technol 11:109–113

Authors and Affiliations

Julieth Patricia Castellanos-Ardila[1] · Barbara Gallina[1] · Guido Governatori[2]

Barbara Gallina
barbara.gallina@mdh.se

Guido Governatori
guido.governatori@data61.csiro.au

[1] IDT, Mälardalen University, Västerås, Sweden

[2] CSIRO, Brisbane, Australia