Towards Automated Evidence Generation for Rapid and Continuous Software Certification

Joanna C. S. Santos
Department of Software Engineering
Rochester Institute of Technology, USA.
jds5109@rit.edu

Ali Shokri
Department of Software Engineering
Rochester Institute of Technology, USA.
as8308@rit.edu

Mehdi Mirakhorli
Department of Software Engineering
Rochester Institute of Technology, USA.
mxmvse@rit.edu

***Abstract*—Software system traceability plays a crucial role in the development and assurance of any dependable software-intensive system. Federal agencies developing or regulating mission-centric or safety-critical software systems require traceability as a core component of the approval and certification process. Manually generating and managing traceability links is a tedious and error-prone task that requires great effort. The state-of-the-art automated traceability techniques rely on information retrieval and machine learning techniques. While these techniques create semantic links between artifacts, they are inadequate for analyzing the satisfiability of the underling traceability criteria. Therefore, in this paper, we describe SHERLOCK, an integrated environment to facilitate the rapid and continuous certification of software systems. At the core of SHERLOCK's capabilities is the novel concept of Traceability Certification Models (TCM) and a Catalog of Certification Assurance Case Patterns (CACP) which guides the efforts of establishing the required trace links for fulfilling the criteria for a given certification. By relying on TCMs, CACPs as well as traceability and change impact analysis techniques, SHERLOCK can report the system's traceability coverage in terms of missing traceability links required for certification, and deprecated/unnecessary links. We showcase SHERLOCK's capabilities using a case study of certifying an Attitude and Orbit Control System (AOCS).**

***Index Terms*—Software certification, Software traceability, Traceability Certification Models**

I. INTRODUCTION

Software traceability concerns the capability of establishing trace links (relationships) between multiple software artifacts (e.g., requirements, code, test cases, etc) [28]. Software traceability is a key element of the software certification process [32]. It enables variety of software engineering activities required for continuous software certification such as impact analysis [41], compliance verification [20], as well as requirements and design validation [21], [23], [40], [42]. However, obtaining traceability information can be expensive and therefore must be performed strategically based on the certification criteria to prevent excessive costs of the analysis due to trace-link proliferation [13], [15], [19], [38], [40], [42], [46].

Current guidelines for certifying safety- and mission-critical systems prescribe traceability for two main reasons: (i) product certification (as a more direct measure to show that specific cyber-incidents, potential failures, and hazards have been explored, potential failure modes identified, mitigation techniques are in-place and the system is designed and implemented in a "demonstrably rational way"), and (ii) process certification (to indirectly measure that good practices have been followed during the software development). Unfortunately, there is a significant gap between *prescribed* and *actual* traceability due to the lack of automated tools adopted by the industry to facilitate this labor-intensive process [33], [47], [48].

Existing certifications for safety-critical systems have their own sets of objectives and criteria that a software product has to fulfill to be certified at a certain level. Reasoning over these heterogeneous aspects is challenging as each of them requires their own set of artifact analysis to collect evidence for certification.

Therefore, in this paper, we describe our early efforts in developing SHERLOCK, ***an integrated environment with a set of automated capabilities to extract rich and diverse software analytics to enable the rapid and continuous certification of software***. Our goal is to facilitate the process of establishing required trace links for a software system to be certified at the desired level.

The main contributions of this work are:

- we discuss the novel concept of *Traceability Certification Models* (TCM) that introduce a uniform language for communicating a certification's requisites. A TCM indicates the *evidence* that needs to be collected alongside the *required trace links* between *artifacts*.
- we present the concept of *Certification Assurance Case Patterns* (ACPC), which aims to help the construction of assurance cases to provide evidence that required trace links were established;
- we present customized traceability techniques to establish the required trace links between the artifacts as described in the TCM.
- we report our early results in developing SHERLOCK that integrates the above capabilities and report traceability coverage in terms of missed trace links and deprecated links for a given certification.

The rest of this paper is organized as follows: Section II describes SHERLOCK and the techniques developed to provide the aforementioned capabilities; Section III presents a case study of using SHERLOCK in Attitude and Orbit Control

System (AOCS). Section IV discusses past works in the area; Section V concludes the paper and elucidates on future work.

II. Sherlock Overview

Sherlock's design and how it integrates with a software development lifecycle is depicted in Figure 1. It is important to clarify that this design does not assume that the Verification and Validation (V&V) activities are conducted in a waterfall fashion, but instead, it highlights how we use the artifacts produced in each phase to provide Sherlock's capabilities. As shown in this figure, Sherlock is delivered as an integrated environment to enable automated certification and continuous monitoring. It has three components to provide this integrated environment: *Certification Assurance Case Patterns* (CACP), *Traceability Information Models* (TCM), and *Automatd Traceability Analysis*. In the next subsections, we elaborate on each of these elements.

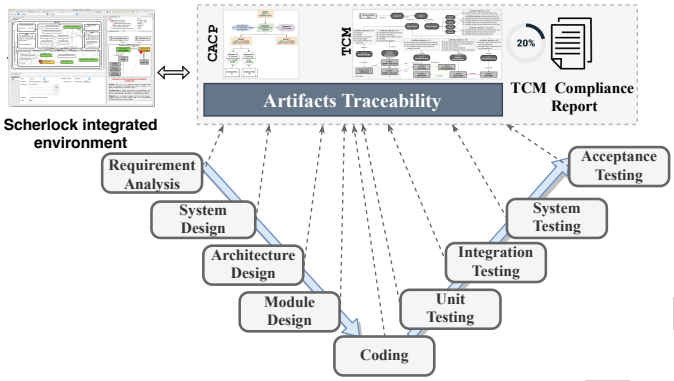

Fig. 1. Overview of Sherlock's Capabilities

A. A Catalog of Certification Assurance Case Patterns (CACP)

Assurance cases are structured arguments over a system's intended properties [51]. They provide a mechanism that allows engineers and external certifiers to assess the system's overall quality. Assurance cases are often needed for safety-critical systems due to regulations for certification purposes.

Creating assurance cases is a costly task that demands expertise and effort. Moreover, maintaining an assurance case in order to enforce that it accurately reflects the system that has been built is challenging as the system's properties that were once true, may become invalid due to bugs or new requirements that were added to the system. Furthermore, when software vendors are working towards certifying their software products, they face the challenge of navigating a series of technical documents for uncovering what has to be done in order to comply with the objectives from the certification [30]. These certification documents usually have hundreds of pages and appendixes which can be difficult for engineers to verify to which extent they are compliant with the certification of interest as well as to verify what are the certification's objectives that still has to be fulfilled. This problem also affects certifiers, that need to ensure that all criteria has been successfully met by inspecting any evidence provided by software vendors.

To overcome the aforementioned challenges, Sherlock contains **a catalog of Certification Assurance Case Patterns**. Each pattern has a set of structured arguments about the software's compliance with a given certification criteria. They are used to ensure that the software meets the criteria for certification.

One of the underlying premises of our work is that developing common assurance case patterns can facilitate the ***creation*** of accurate assurance cases, the ***identification*** (manually or automated) of evidence and aid the analysis of the software for the purpose of ***certification***. Each **Certification Assurance Case Pattern** (henceforth CACP) in Sherlock 's catalog is composed of the following elements:

— **Goals**: assertions (claims) about a system's property. This can either be a quality aspect (such as security, safety, etc) or concerning a certification goal.
— **Strategies**: describe how any sub-claim will support higher-level claims.
— **Solution**: proof (evidence) that the corresponding connected (sub-)claim is correct.
— **Context**: the scenario in which the claim is applicable.
— **Assumption**: a premise that is assumed as true such that the arguments and strategies are valid.
— **Evidence Rules**: rules for collecting evidence for the claim. They indicate the required trace link(s) that need to be in place for the claim to be valid.

Each of these CACP's elements has **instantiation points**, which are "holes" that are filled out according to the system under analysis. This way, each CACP acts as a template that will guide a software engineer to create a solid argument that can correctly represent the desired properties of the system. These assurance cases are intended to minimize the costs and efforts of writing assurance cases. As a result, CACPs start with a very high-level claim that the *"system complies with certification* X *level* Z*"*.

To create certification ACPs we (i) leverage existing technical and training documents for a certification of interest, and (ii) perform a systematic literature review on previous efforts on constructing assurance cases for certification criteria [29], [30]. As we scrutinize certification-related papers/documents, we construct claims. We then integrate and structure these claims in order to make valid a top argument in the form of *"system complies with certification* X *level* Z*"*.

B. Traceability Certification Models (TCM)

At the core of Sherlock's approach is the novel concept of ***Traceability Certification Models (TCM)***, designed as a uniform language for communicating a certification's requisites with all stakeholders. The TCM concept emerged as a result of an earlier study of which we conducted across a wide range of software-intensive systems including the Airbus A320/330/340 family, Boeing 777, Boeing 7J7, and NASA robots [40], [42].

A TCM delivers the following advantages to the current traceability techniques [38]:

(i) it provides a reusable infrastructure for cost-effective and strategic traceability analysis, that guides developers on establishing only the trace links that are necessary for certification. Therefore it enables a proactive approach to software certification and guides the efforts towards establishing the most important trace links;
(ii) it helps engineers establish strategic traceability links to be used during the certification phase as well as continuous assessment of software certification status while the products are being built;
(iii) it can be used to keep engineers fully informed of the *evidence* that needs to be collected alongside the *required trace links* between *artifacts*;
(iv) it can be used to visualize underlying certification criteria addressed and missing evidence that must be generated;

In our earlier work [37], [39], [40] we described the idea of using Traceability Information Models to support software certification and compliance for safety-critical and mission-centric applications. The TCMs are built from our prior works in Traceability Information Models [40] and we present an augmented set of TCMs which explicitly differentiate between ***reusable traceability links*** (i.e., those links internal to the certification criteria, which can be used across multiple projects) versus ***project-specific traceability links*** established as mappings from concrete software artifacts to components of the TCM.

For each certification guideline, we develop a TCM. Each TCM is centered around *traceability links* required to satisfy certification criteria or be in compliance with existing regulatory codes for the software system. Therefore, a TCM explicitly models the following elements:

• **Certification Criteria:** defining a criteria from the standard used for audit and certification of the software.

• **Software Artifact:** refers to any artifact types that can serve as source or target of traceability links and the certification requires the trace evidence to be collected for.

• **Proxy Elements:** each artifact type in a TCM has a proxy element which can be used to map an artifact type in TCM onto one or more concrete artifact in software repositories (e.g. models). Once these mappings are accomplished, all of the traceability information embedded in the TCM along with the certification criteria are automatically inherited by the project. As a result, TCM have been shown enable actionable analytics supporting software certification while reducing the cost and effort of traceability [40]. For instance, an engineer can review an instantiated TCM (a TCM with all the potential traceability links prescribed by it) to reason to what extent certification criteria are satisfied, what other evidences must be generated and more importantly what are the risks in software that must be addressed. While each TCM in our catalog will be a meta-model of traceability certification criteria, it can be operationalized through automated traceability capabilities delivered as part of SHERLOCK.

C. Automated Traceability Analysis

Performance-centric and mission-critical software systems are extremely large and include a multitude of artifacts, besides source code. They can include design models (e.g. AADL, UML, SysML), a variety of documents, requirement specifications (written in formal or natural language formats), test cases, bug reports, communications between stakeholders, etc [38]. These are created and maintained over long periods by different individuals. Establishing and maintaining explicit traceability between software artifacts is difficult yet a very important problem [10]. Furthermore, the analysis of the traceability link regarding certification criteria is a challenge not adequately addressed [14].

Therefore, SHERLOCK presents a set of automated techniques to support the on-demand creation of traceability links. The approach relies on information retrieval and data mining techniques from our previous works [19], [25], [42] to enable the creation and maintenance of traceability links. SHERLOCK contains a novel traceability platform that uses first-of-its-kind approaches for generating trace recommendations, and pushing them to knowledgeable project stakeholders within the context of their daily work. The goal is to minimize reliance on end-of-project big-bang trace activities by eliciting traceability decisions from informed stakeholders throughout the project. Our prior evaluations provide strong evidence that using this technique, the cost of establishing and maintaining traceability links and performing audit and certification of the software can significantly be reduced [22].

We deliver automated approaches for generating traceability reports required for generating reliable evidence to support Certification Assurance Case Patterns (CACP) described in Section II-A. A feasibility study for such capability was performed in an earlier work [23].

The traceability approach [23], [25], [36], [52] uses custom machine learning algorithms tailored to deal with software engineering artifacts. We have previously used this algorithm to trace quality goals (such as security, performance, resiliency, and usability) to project-level requirements, design models, and source code across a variety of projects, and technical safeguards to product level requirements in healthcare-related products. The approach involves three phases (*preparation*, *training*, and *detection*) which are described below [36]:
— *Preparation*: In this first phase the approach preprocesses the artifacts by (1) removing non-alphanumeric characters, (2) stemming words to their morphological roots, and (3) removing "stop words", which are commonly used terms that are not helpful for classification purposes (e.g., "the", "shall", etc). The remaining terms are then transformed into a *vector of terms*.
— *Training*: This phase takes as input a set of pre-classified software artifacts. From this input set, it computes a set of *indicator terms*, which are treated as representative for each artifact type. For example, terms such as *credentials*, *password*, and *login* are more recurrent in artifacts related to the *authentication of actors* than in other kinds of artifacts.

Thus, these terms receive a higher weighting with respect to that mitigation technique.

The trace by classification is defined more formally as follows. Let q be a specific artifact type such as *security requirements*. Indicator terms of type q are extracted considering the set S_q of all datasets that are related to type q. The cardinality of S_q is defined as N_q. Each term t is assigned a weight score $Pr_q(t)$ that corresponds to the probability that a particular term t identifies an artifact associated with type q. The frequency $freq(c_q, t)$ of term t in an artifact description c related with type q, is computed for each artifact description in S_q. $Pr_q(t)$ is then computed as:

$$Pr_q(t) = \frac{1}{N_q} \sum_{c_q \in S_q} \frac{freq(c_q, t)}{|c_q|} * \frac{N_q(t)}{N(t)} * \frac{NP_q(t)}{NP_q} \quad (1)$$

— *Detection*: During this phase, the indicator terms previously computed using Equation 1 are used to evaluate the likelihood ($Pr_q(c)$) that a given artifact c is associated with the type q. Consider I_q be the set of indicator terms for type q identified during the training phase. The classification score that artifact c is associated with type q is defined as:

$$Pr_q(c) = \frac{\sum_{t \in c \cap I_q} Pr_q(t)}{\sum_{t \in I_q} Pr_q(t)} \quad (2)$$

where the numerator is computed as the sum of the term weights of all type q indicator terms that are contained in c, and the denominator is the sum of the term weights for all type q indicator terms. The probabilistic classifier for a given type q will assign a higher score $Pr_q(c)$ to an artifact c that contains several strong indicator terms for q. Artifacts are considered to be related to a given type q if the classification score is higher than a selected threshold.

Furthermore, we design and evaluate algorithms based on the Vector Space Model (VSM) [49] because this technique is suitable for tracing artifacts/concepts that do not frequently recur in similar forms across multiple projects. When using VSM each software artifact is represented as a *vector of weighted terms*. In a trace link between a source artifact f and a target artifact q, the target artifact f is represented as a vector $\vec{f} = (w_{1,f}, w_{2,f}, ..., w_{n,f})$ and the source artifact q is represented as $\vec{q} = (w_{1,q}, w_{2,q}, ..., w_{n,q})$, where $w_{i,f}$ represents the weight of the term i for source file f. We assign weights to individual terms based on TF-IDF [49]. In this weighting scheme, the tf represents the term frequency, and the idf corresponds to the inverse document frequency. The term frequency is computed for target artifact f as $tf(t_i, f) = (freq(t_i, f))/(|f|)$, where $freq(t_i, f)$ is the frequency of the term in the artifact, and $|f|$ is the length of the artifact. The inverse document frequency idf, is typically computed as:

$$idf_{ti} = log_2 \frac{n}{n_i} \quad (3)$$

where n is the total number of artifacts in the corpus and n_i is the number of source files in which term t_i occurs. Thus, the individual term weight for term i in artifact f is then computed as $w_{id} = tf(t_i, f) \times idf_{ti}$. Given these definitions, the similarity score $Sim(f, q)$ between a target artifact f and source artifact q is computed as the cosine of the angle between the two vectors as:

$$Sim(f, q) = \frac{(\sum_{i=1}^{n} w_{i,f} w_{i,q})}{\left(\sqrt{\sum_{i=1}^{n} w_{i,f}} \cdot \sqrt{\sum_{i=1}^{n} w_{i,q}}\right)} \quad (4)$$

Our algorithm uses the *cosine similarity* score between all desired software artifacts defined in TCMs to establish the traceability links.

D. Effective Software (Re-)certification through Change Impact Analysis

SHERLOCK leverages an automated change impact analysis platform we previously examined [40]. This platform enables the developers and certifiers to perform a change impact analysis to identify the impact of software changes on certification criteria and assess whether new evidence must be collected or not. Early evaluation of this change impact analysis platform in our previous work [40] indicated the practicality of it and indicated that it can be a cost-effective solution to maintain software qualities in mission-critical software.

The change impact analysis works by continuously monitoring the software's artifacts (such as code, test cases, requirement documents, etc) and detecting changes. Based on the trace links previously established, SHERLOCK further identifies the change type (addition, removal or modification). By relying on the TCMs and the trace links automatically established SHERLOCK provides the following notifications for software vendors:

- a list of claims and/or sub-claims that must be revisited and which evidence may not hold anymore;
- trace links that might be affected by a removal/modification change type;
- trace links that are missing due to an addition change type;

III. Case Study: Certifying an AOCS

To illustrate how SHERLOCK can help the certification of a software product, consider as an example an ***Attitude and Orbit Control System*** (AOCS) [17], which is a system for controlling satellites. We collected multiple publicly available software artifacts such as architectural models [11], source code, test cases, and documentation [17], [45]. In this section, we demonstrate how SHERLOCK can facilitate the process of certifying the AOCS following the DO-178C certification. In the next subsections, first we describe the AOCS' architecture (Section III-A) and the DO-178C standard (Section III-B). Subsequently, we described the use of SHERLOCK in aiding the certification process.

A. AOCS Architecture

The satellite receives *telecommands* from the ground station and replies with *telemetry data*. The *telemetry data* indicates the status of the satellite along with data collected by the satellite's sensors. The *telecommands* sent by the ground station controls a variety of aspects of the satellite such

as its *nominal attitude*, *nominal orbit*, *power up/down* the unit, *reconfigure* unit, mark unit as "unhealthy" or *change its operational mode*.

Figure 2 contains the architecture of this AOCS modelled using the Architecture Analysis & Design Language (AADL) [11]. For the sake of clarity, this architecture diagram depicts only the AOCS' major components but the full architectural model is publicly available at [11]. As shown in this diagram, *telecommands* are received by the *central computer* and forwarded to the *main software* whereas the *telemetry data* generated by the sensors are received by the *central computer* and sent to the *ground station*. This AOCS has the following software elements [17]:

- *Telecommand Processing (TCP)*: a thread that processes incoming telecommands from the ground station and then computes the nominal attitude, nominal orbit, and maneuver command.
- *Attitude Control Function (ACF)*: a software thread that operates in a closed-loop to maintain the AOCS' nominal attitude (as provided by the TCP thread). This thread relies on sensor data to compute the necessary torque, which is a command sent to the attitude actuators (actuators are hidden).
- *Orbit Control Function (OCF)*: a thread that computes the forces that need to be applied to the satellite for it to maintain its nominal orbit. The computed forces are sent to delta-V actuators (not shown in the diagram).
- *Manoeuvre Execution (ME)*: a thread that computes a sequence of actions that have to be performed for the AOCS to fulfill a goal.
- *Failure Detection and Isolation (FDR)*: a thread that detects failures in the system and identifies their root causes.
- *Telemetry Processing (TMP)*: a thread that continuously collects telemetry data to be sent to the satellite's central computer. The actual contents of the telemetry packets are determined by the telecommands.

B. DO-178C: Software Considerations in Airborne Systems and Equipment Certification

When it comes to airborne systems and related equipment, multiple regulation authorities (e.g., FAA) require these systems to follow the DO-178C [27] certification to be approved for use. This certification requires trace links between multiple software artifacts for compliance.

The required trace links depend on the *software level*. This is determined by performing a safety assessment and an analysis of the potential hazards incurred by the software [30]. The DO-178C defines five certification levels: A (Catastrophic), B (Hazardous), C (Major), D (Minor), and E (No Safety Effect). Except for level E, all the other software levels require trace links between different artifacts [27].

As part of this case study, we have created a TCM for **DO-178C**, depicted in Figure 3. This TCM starts from a top-level argument, that is instantiated from the catalog of Certification Assurance Case Patterns (CACP). The TCM

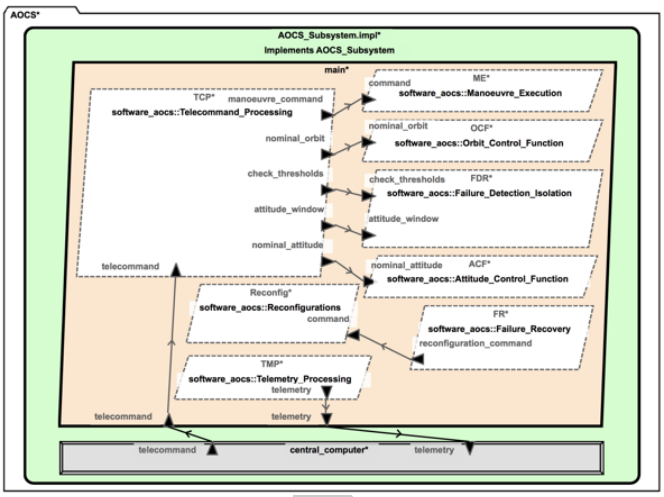

Fig. 2. AOCS' Architecture in AADL

models the Verification & Validation Evidence needed for certification and how they correlate with the objectives from the **DO-178C** standard [27].

As depicted in this figure, the TCM indicates the required trace links per certification level A-D [1]. For instance, in case the software is at any level A, B, C, or D, for every high-level system requirements there must be a test case or unit test developed to check the satisfaction of that requirement. Therefore the TCM in Figure 3 models artifact types *SysRS-High Level* and *Test Cases* and explicitly models trace links in between them. Such traceability links will be used during the software certification process to demonstrate software satisfies the certification criteria and is dependable.

C. Using SHERLOCK *for Certification*

To aid software certification, SHERLOCK can be used in four steps:

— **Step 1:** ***Certification Criteria Selection***

When assessing a software a certifier first selects the desired certification *standard* and *certification criteria*. In this case study, the specified standard is **DO-178C** and the criteria are **Level B** (Hazardous) since a failure in the AOCS can negatively impact on safety and/or performance of the satellite.

— **Step 2:** ***Assurance Cases for Required Traceability***

Based on the selected standard and criteria, SHERLOCK determines that the following traceability links are required for certifying any software at level B according to the DO-178C standard:

- Traceability between *low-level* and *high-level* requirements;
- Traceability between *high-level requirements* and *software architecture*;
- Traceability between *low-level requirements* and *source code*;
- Traceability between *low-level requirements* and *test cases*;

[1]The level E is not shown as it does not require trace links.

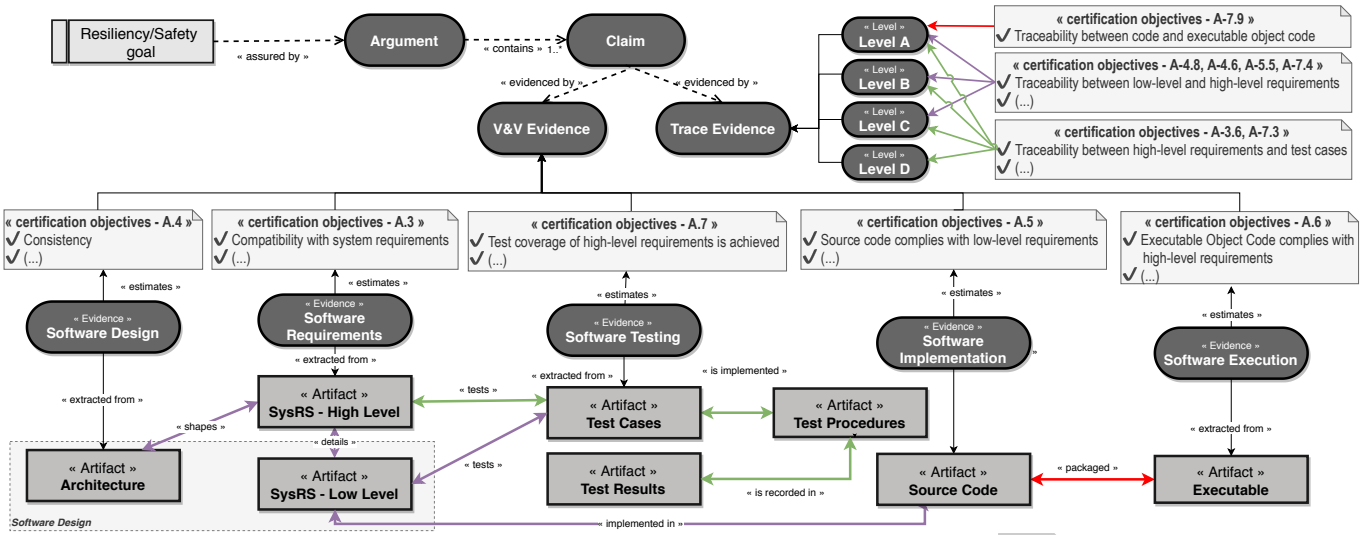

Fig. 3. A Traceability Certification Model (TCM) for DO-178C

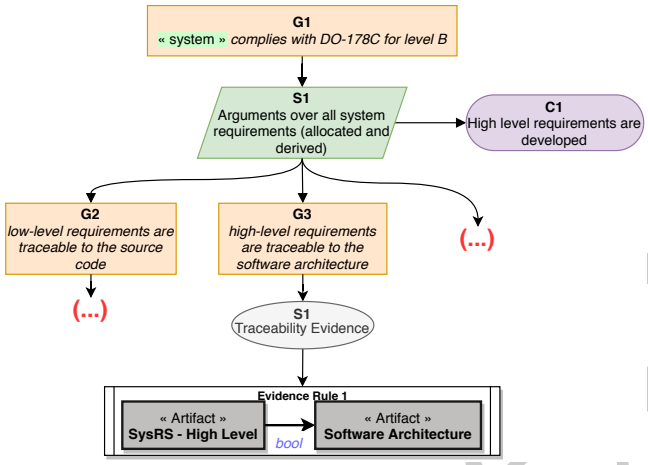

Fig. 4. An excerpt from a certification assurance case pattern for DO-178C

- Traceability between *high-level requirements* and *test cases*;
- Traceability between *test cases* and *test procedures*;
- Traceability between *test cases* and *test results*;

Based on this determination, SHERLOCK selects the CACP from its catalog associated with the DO-178C level B. Figure 4 shows an excerpt of a CACP claiming that the system is compliant with the Software Considerations in Airborne Systems and Equipment Certification DO-178C level B [29], [30]. For such a claim to be true, the ACP has sub-claims (goals) about aspects that are true to make the software compliant. The leaf evidence nodes in a certification ACP are connected to evidence rules nodes. These are used to collect concrete evidence from software artifacts that the claim is valid for the system under analysis. In this example, we demonstrated that for the claim **G#3** to be satisfied, there should be a trace link between *high-level requirements* and *software architecture*.

— **Step 3:** ***Traceability Analysis***

In this third step, *Sherlock* establishes the required trace links by leveraging the techniques described in Section II-C. Since we are currently in the process of collecting a curated dataset of trace links for this automated analysis, for this case study the trace links are established manually.

Consider the sample low-level requirements listed in Table I. Per DO-178C, these requirements must have trace links to test cases. According to the AOCS documentation [44], [45], the requirements R1 and R2 are tested at the "`TestCaseFailureDetection_1`" test case, therefore, there is a trace link between R1 to this test case as well as from R2 to the test case. However, the test cases do not provide an implementation for verifying that messages are exchanged via encryption (R3). As a result, SHERLOCK indicates a failure to comply with the DO-178C standard due to a missing trace link.

TABLE I
SAMPLE LOW LEVEL REQUIREMENTS FOR AOCS

ID	Requirement
R1	The x-axis shall not deviate by more that 20 degrees from the sun line.
R2	The actual attitude shall not deviate from the nominal attitude by more than 25 degrees.
R3	The messages exchanged between the ground station and the satellite (and vice versa) should be encrypted.

— **Step 4:** ***Continuous (Re)-Certification***

Once trace links are established and certification assurance case patterns are instantiated, SHERLOCK actively monitors the AOCS' artifacts to uncover changes. For instance, considering a change scenario in which the requirement R1 is changed to "the deviation threshold is 10 degrees instead of 20". This is detected by SHERLOCK as a *modification* that can impact the artifacts linked to the modified requirement. In this specific example, the software vendors are notified that the test case "`TestCaseFailureDetection_1`" needs to be updated to reflect the aforementioned change.

IV. Related Work

One of the ways to question and quantify compliance of a system with a certification is constructing *assurance cases* which can be represented textually or graphically. GSN [4] and CAE [16] provide well established graphical notations to express arguments easily. Besides, SACM [8] favors model-based approach which can be complemented with GSN and CAE for graphical support. There are some of the open and closed source tools used for constructing assurance cases [1], [3], [9], [24], [34]. However, these notations do not include an element that enables the automated extraction of evidence for a claim.

How to build an argument is of importance as the compliance relies on reasoning and evidence. Two fundamental approaches that guides argument structure development are top-down and bottom-up [4], [31]. Arguments can also be constructed from scratch or by using predefined patterns/templates. Kelly [31] took the lead to reuse and create a catalogue of safety patterns. Since then online and offline pattern catalogues have been published [2], [5]–[7], [12], [31], [50]. While most of the assurance case catalogues are dedicated to one or two quality attributes, SHERLOCK goes beyond asserting the system's quality attributes, but it has assurance case patterns to guide certification with respect to required trace links.

Customization of patterns by selecting relevant elements is a way to reuse assurance cases such as in [26] for security, However, it is not effective and possible to predefine all argument elements for all prospective certification requirements. Therefore, SHERLOCK uses parameterized pattern instantiation complying with given certification where a set of holes located in the argument elements are filled with artifact information.

Chen et al [18] proposed a certification process where regulator develops argument pattern templates and manufacturers instantiate accordingly and exemplified usage of patterns for insulin pumps. SHERLOCK's catalog of CACPs will include certification patterns that also helps to the certifier to assess the claims and show which claims are not true (e.g., a case is when executable code is not compatible with target computer).

Besides past works on assurance cases, some other research focused on modeling existing certification standards, such as for the DO-178C described in this paper. The DO-178C provides some guidance on how to produce a software for airborne systems that performs its functionalities with a level of confidence in safety that complies with airworthiness requirements. However, the relationship between the guidance and the purpose of DO-178C is remained implicit. To elucidate how exactly provided guidance in DO-178C would incorporate to its purpose, Holloway first proposed an explicit assurance case for DO-178C in [29] and then, applying some revisions, introduced the final case in [30]. This case is created based on four fundamental concepts of the case: (i) transforming safety into correctness, (ii) allowing life cycle flexibility, (iii) using confidence arguments, and (iv) explicating before evaluation. It also provides some salient characteristics about and some experts from the case.

There also have been some efforts on building software design infrastructures to enforce certification information on the software design level. Metayer et al [35] introduced a Doman Specific Modeling Language (DSML) which provides documentation infrastructure that enforces certification information required by DO-178C and its supplements to meet software's design assurance level. In another line of work, to overcome the complexity of understanding, interpreting and complying with a standard to create an evidence for a certification, an explicit conceptual model of that standard is created [43]. Then, this model is being used to build a UML profile that software suppliers could use to relate safety standard concepts with their application domain and provide evidence of compliance of their software with the standard. In the third step, a domain model of the application would be elaborated. Finally, an instance for a specific certification would be created.

V. Conclusion and Future Work

We described our current development of SHERLOCK, an integrated environment to provide capabilities for helping developers in identifying artifacts and traceability links that are necessary for certification. While it guides the traceability creation and maintenance, it also creates several reports including, missing traceability links, deprecated traceability links, certification criteria with/without trace evidence, and certification criteria adequately satisfied. Furthermore, traceability creation models (TCM) introduced in this paper can be used by a certifier to obtain necessary evidences about to what extent the software development process has followed the prescribed best practices. Therefore, it will not only support the certification of the software, it will also enable the certification of the development process used to build the software. For instance, it can help certifiers obtain an evidence that software testing was adequately performed, all requirements had test cases, or design decisions were thoroughly followed into downstream software artifacts and code.

References

[1] Assurance and safety case environment (asce). https://www.adelard.com/asce/choosing-asce/index/. Accessed: 2019-07-01.

[2] Assurance case patterns online catalogue, by gdańsk university of technology. http://www.nor-sta.eu/en/en/news/assurance_case_pattern_cataloguehttps://tct.nor-sta.eu/?username=norstaviewerasce_en&password=norstaviewer. Accessed: 2019-07-01.

[3] Certware. http://nasa.github.io/CertWare/. Accessed: 2019-07-01.

[4] Goal structuring notation community standard (version 2). https://scsc.uk/r141B:1. Accessed: 2019-07-01.

[5] The gsn community standard working group website under resources. http://www.goalstructuringnotation.info/archives/category/resources. Accessed: 2019-07-01.

[6] A repository of safety-related resources for medical devices by generic infusion pump research project. https://rtg.cis.upenn.edu/gip/. Accessed: 2019-07-01.

[7] Seam (systems engineering and assurance modeling), web-based collaborative modeling platform for modeling assurance cases integrated with the models of the system. https://modelbasedassurance.org/. Accessed: 2019-07-01.

[8] Structured assurance case metamodel (version 2). https://www.omg.org/spec/SACM/About-SACM/. Accessed: 2019-07-01.

[9] Support for achieving and assessing conformance to norms and standards (nor-sta). https://www.nor-sta.eu/en. Accessed: 2019-07-01.

[10] Grand Challenges, Benchmarks, and TraceLab: Developing Infrastructure for the Software Traceability Research Community. *International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, 6, 2011.

[11] AADLib. Aadlib examples. https://github.com/OpenAADL/AADLib/tree/master/examples/aocs, Sep 2015. (Accessed on 06/12/2018).

[12] R. Alexander, T. Kelly, Z. Kurd, and J. McDermid. Safety cases for advanced control software: Safety case patterns, 2007. Technical report, Department of Computer Science, University of York.

[13] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, Oct 2002.

[14] G. Antoniol, J. Cleland-Huang, J. Hayes, and M. Vierhauser. Grand challenges of traceability: The next ten years. 10 2017.

[15] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor. Software traceability with topic modeling. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 95–104, New York, NY, USA, 2010. ACM.

[16] P. Bishop and R. Bloomfield. A methodology for safety case development. In *Industrial Perspectives of Safety-critical Systems*, pages 194–203, Cham, 2000. Springer London.

[17] V. Cechticky, G. Montalto, A. Pasetti, and N. Salerno. The aocs framework. *European Space Agency-Publications-ESA SP*, 516:535–540, 2003.

[18] Y. Chen, M. Lawford, H. Wang, and A. Wassyng. Insulin pump software certification. In J. Gibbons and W. MacCaull, editors, *Foundations of Health Information Engineering and Systems*, pages 87–106. Springer Berlin Heidelberg, 2014.

[19] L. Chung, B. Nixon, E. Yu, and J. Mylopoulos. Non-functional requirements in software engineering., 2000.

[20] J. Cleland-Huang, O. C. Z. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman. Software traceability: Trends and future directions. In *Proceedings of the on Future of Software Engineering*, FOSE 2014, pages 55–69, New York, NY, USA, 2014. ACM.

[21] J. Cleland-Huang, M. Heimdahl, J. Huffman Hayes, R. Lutz, and P. Maeder. Trace queries for safety requirements in high assurance systems. In B. Regnell and D. Damian, editors, *Requirements Engineering: Foundation for Software Quality*, pages 179–193, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[22] J. Cleland-Huang, P. Mäder, M. Mirakhorli, and S. Amornborvornwong. Breaking the big-bang practice of traceability: Pushing timely trace recommendations to project stakeholders. In *2012 20th IEEE International Requirements Engineering Conference (RE), Chicago, IL, USA, September 24-28, 2012*, pages 231–240, 2012.

[23] J. Cleland-Huang, P. Mäder, M. Mirakhorli, and S. Amornborvornwong. Breaking the big-bang practice of traceability: Pushing timely trace recommendations to project stakeholders. In *2012 20th IEEE International Requirements Engineering Conference (RE)*, pages 231–240, Sep. 2012.

[24] E. Denney, G. Pai, and J. Pohl. Advocate: An assurance case automation toolset. In F. Ortmeier and P. Daniel, editors, *Computer Safety, Reliability, and Security*, pages 8–21, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[25] H. Dumitru, M. Gibiec, N. Hariri, J. Cleland-Huang, B. Mobasher, C. Castro-Herrera, and M. Mirakhorli. On-demand feature recommendations derived from mining public product descriptions. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, pages 181–190, 2011.

[26] D. A. W. E. Kenneth Hong Fong. A sample security assurance case pattern. *INSTITUTE FOR DEFENSE ANALYSES*, (P-9278):891–921, 2018.

[27] T. K. Ferrell and U. D. Ferrell. Rtca do-178c/eurocae ed-12c. *Digital Avionics Handbook*, 2017.

[28] O. Gotel and C. Finkelstein. An analysis of the requirements traceability problem. In *Requirements Engineering, 1994., Proceedings of the First International Conference on*, pages 94 –101, apr 1994.

[29] C. M. Holloway. Making the implicit explicit: Towards an assurance case for do-178c. 2013.

[30] C. M. Holloway. Explicate'78: Uncovering the implicit assurance case in do-178c. 2015.

[31] T. P. Kelly. Arguing safety - a systematic approach to managing safety cases, 1998.

[32] D. L. Lempia and S. P. Miller. Requirements engineering management findings report. http://www.tc.faa.gov/its/worldpac/techrpt/ar0834.pdf. Accessed: 2019-04-28.

[33] P. Mäder, P. L. Jones, Y. Zhang, and J. Cleland-Huang. Strategic traceability for safety-critical projects. *IEEE Softw.*, 30(3):58–66, 2013.

[34] Y. Matsuno and S. Yamamoto. An implementation of gsn community standard. In *2013 1st International Workshop on Assurance Cases for Software-Intensive Systems (ASSURE)*, pages 24–28, May 2013.

[35] N. Metayer, A. Paz, and G. El Boussaidi. Modelling do-178c assurance needs: A design assurance level-sensitive dsl. In *2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, pages 338–345. IEEE, 2019.

[36] M. Mirakhorli. Preserving the quality of architectural decisions in source code, PhD Dissertation, DePaul University Library, 2014.

[37] M. Mirakhorli and J. Cleland Huang. A decision-centric approach for tracing reliability concerns in embedded software systems. In *Proceedings of the Workshop on Embedded Software Reliability (ESR), held at ISSRE10*, November 2010.

[38] M. Mirakhorli and J. Cleland-Huang. *Tracing Non-Functional Requirements.* In: Andrea Zisman, Jane Cleland-Huang and Olly Gotel. Software and Systems Traceability., Springer-Verlag., 2011.

[39] M. Mirakhorli and J. Cleland-Huang. Transforming trace information in architectural documents into re-usable and effective traceability links. In *Proceedings of the 6th workshop on Sharing Architectural Knowledge*, May 2011.

[40] M. Mirakhorli and J. Cleland-Huang. Using tactic traceability information models to reduce the risk of architectural degradation during system maintenance. In *Proceedings of the 2011 27th IEEE International Conference on Software Maintenance*, ICSM '11, pages 123–132, Washington, DC, USA, 2011. IEEE Computer Society.

[41] M. Mirakhorli and Jane Cleland-Huang. Detecting, tracing, and monitoring architectural tactics in code. *IEEE Trans. Software Eng.*, 2015.

[42] M. Mirakhorli, Y. Shin, J. Cleland-Huang, and M. Cinar. A tactic centric approach for automating traceability of quality concerns. In *International Conference on Software Engineering, ICSE (1)*, 2012.

[43] R. K. Panesar-Walawege, M. Sabetzadeh, and L. Briand. Supporting the verification of compliance to safety standards via model-driven engineering: Approach, tool-support and empirical validation. *Information and Software Technology*, 55(5):836–864, 2013.

[44] A. Pasetti. AOCS framework - test report. Software & Web Engineering Group. Technical Report, 2002.

[45] A. Pasetti. AOCS framework project. https://www.pnp-software.com/AocsFramework/index.html, June 2002. (Accessed on 07/25/2020).

[46] M. Rath, J. Rendall, J. L. C. Guo, J. Cleland-Huang, and P. Mäder. Traceability in the wild: Automatically augmenting incomplete trace links. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 834–845, May 2018.

[47] P. Rempel, P. Mäder, T. Kuschke, and J. Cleland-Huang. Mind the gap: assessing the conformance of software traceability to relevant guidelines. In P. Jalote, L. C. Briand, and A. van der Hoek, editors, *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 943–954. ACM, 2014.

[48] P. Rempel, P. Mäder, T. Kuschke, and J. Cleland-Huang. Traceability gap analysis for assessing the conformance of software traceability to relevant guidelines. In U. Aßmann, B. Demuth, T. Spitta, G. Püschel, and R. Kaiser, editors, *Software Engineering & Management 2015, Multikonferenz der GI-Fachbereiche Softwaretechnik (SWT) und Wirtschaftsinformatik (WI), FA WI-MAW, 17. März - 20. März 2015, Dresden, Germany*, volume P-239 of *LNI*, pages 120–121. GI, 2015.

[49] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[50] M. Szczygielska and A. Jarzębowicz. Assurance case patterns on-line catalogue. In W. Zamojski, J. Mazurkiewicz, J. Sugier, T. Walkowiak, and J. Kacprzyk, editors, *Advances in Dependability Engineering of Complex Systems*, pages 407–417, Cham, 2018. Springer International Publishing.

[51] The Assurance Case Working Group (ACWG). GSN community standard. https://scsc.uk/r141B:1?t=1, January 2018. (Accessed on 06/27/2019).

[52] W. Zogaan, I. Mujhid, J. C. S. Santos, D. Gonzalez, and M. Mirakhorli. Automated training-set creation for software architecture traceability problem. *Empirical Software Engineering*, 22(3):1028–1062, 2017.