Leveraging Artifact Trees to Evolve and Reuse Safety Cases

Ankit Agrawal
Dept of Computer Science and Eng.
University of Notre Dame
South Bend, IN USA
aagrawa2@nd.edu

Seyedehzahra Khoshmanesh
Dept. of Computer Science
Iowa State University
Ames, IA USA
zkh@iastate.edu

Michael Vierhauser
Dept. of Computer Science and Eng.
University of Notre Dame
South Bend, IN USA
mvierhau@nd.edu

Mona Rahimi
Dept. of Computer Science
Northern Illinois University
Chicago, IL USA
m.rahimi@acm.org

Jane Cleland-Huang
Dept. of Computer Science and Eng.
University of Notre Dame
South Bend, IN USA
janeclelandhuang@nd.edu

Robyn Lutz
Dept. of Computer Science
Iowa State University
Ames, IA USA
rlutz@iastate.edu

***Abstract*—Safety Assurance Cases (SACs) are increasingly used to guide and evaluate the safety of software-intensive systems. They are used to construct a hierarchically organized set of claims, arguments, and evidence in order to provide a structured argument that a system is safe for use. However, as the system evolves and grows in size, a SAC can be difficult to maintain. In this paper we utilize design science to develop a novel solution for identifying areas of a SAC that are affected by changes to the system. Moreover, we generate actionable recommendations for updating the SAC, including its underlying artifacts and trace links, in order to evolve an existing safety case for use in a new version of the system. Our approach, *Safety Artifact Forest Analysis* (SAFA), leverages traceability to automatically compare software artifacts from a previously approved or certified version with a new version of the system. We identify, visualize, and explain changes in a Delta Tree. We evaluate our approach using the Dronology system for monitoring and coordinating the actions of cooperating, small Unmanned Aerial Vehicles. Results from a user study show that SAFA helped users to identify changes that potentially impacted system safety and provided information that could be used to help maintain and evolve a SAC[1].**

***Index Terms*—Change Impact, Safety Assurance Cases, Evolution, Traceability**

I. Introduction

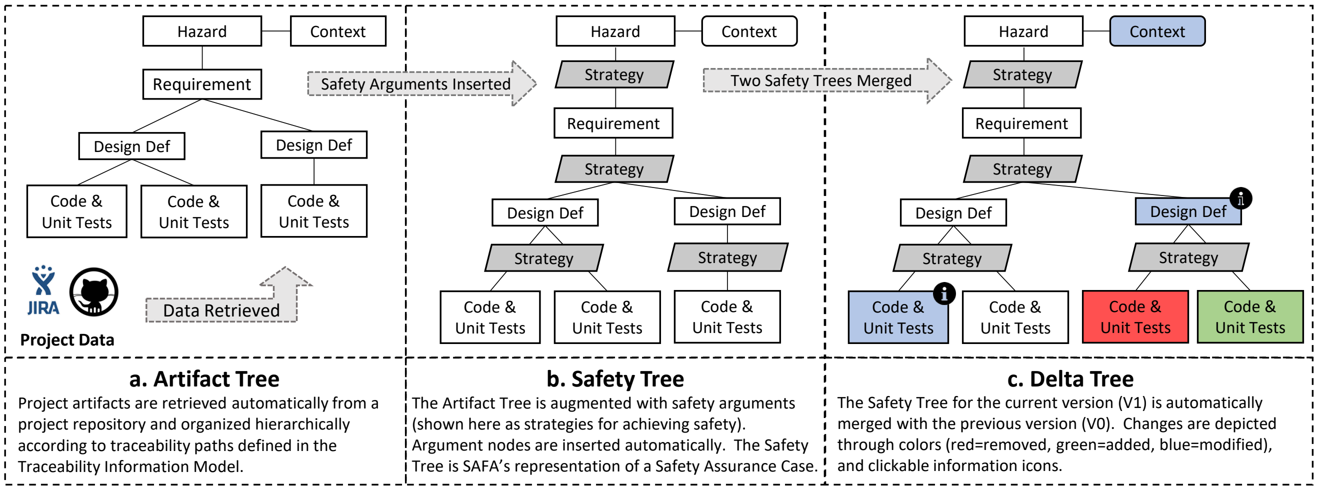
Safety-critical software systems represent a class of systems whose failure or malfunction could result in casualties or serious financial loss [45]. Building such systems requires rigorous safety analysis and the construction of a systematically argued safety case. A Safety Assurance Case (SAC) [9], [37] organizes goal-oriented or claim-based safety arguments [33] into a tree structure by decomposing a top-level safety goal or claim into several layers of arguments. Additionally, these arguments are supported by evidence such as test results, formal reviews, or simulations [7]. SACs are therefore a useful and well-established technique for supporting developers, architects, safety analysts, and other project stakeholders as they proactively build, evaluate, and provide evidence for the safety of a system [28], [61]. Currently, many certification and approval bodies for software-intensive, safety-critical domains recommend the use of SACs. For example, the US Food and Drug Administration (FDA) has issued formal guidelines requiring infusion pump manufacturers to submit SACs as part of the safety approval process [24]. Similarly, the Ministry of Defense in the UK requires all defense system contractors to provide SACs for their products and services [61]. SACs are also recommended by standards such as ISO 26262 for road vehicles, IEC 62425 for railway electronic systems, and IAEA SSG-23 for radioactive waste management systems [6], [17]. SACs can be modeled in various ways; however, the most common approaches are Goal Structuring Notation (GSN) [41] and Claims-Arguments-Evidence notation [2].

Despite their increasing adoption, creating and maintaining SACs is rather challenging for several reasons. Challenges include a lack of guidance for constructing effective safety arguments [60], a tendency to suffer from confirmation bias [47], an over reliance on the regulation culture [59], and lack of focus on confidence and uncertainty issues [21], [28]. Furthermore, SACs need to co-evolve with the system they represent, requiring analysts to identify the impact of a system change and to recognize when seemingly innocuous changes in the operating environment impact the SAC's validity [34].

Despite a number of papers discussing the use of SACs, problems associated with their evolution and maintenance have not yet been effectively addressed [11]. In this paper, we propose the new approach of *Safety Artifact Forest Analysis* (SAFA) to directly address this challenge. SAFA is designed to automatically identify the impact of system-wide changes on a previously approved or certified SAC in order to facilitate evolution and ultimately safe reuse of a SAC's elements within the context of a new version of the system.

Our SAFA approach adopts standard guidelines to create an initial SAC that includes goals, claims, evidence, and

[1]Software artifacts used in this study are available at https://github.com/SAREC-Lab/SAFA-Artifacts



Project artifacts are retrieved automatically from a project repository and organized hierarchically according to traceability paths defined in the Traceability Information Model.

The Artifact Tree is augmented with safety arguments (shown here as strategies for achieving safety). Argument nodes are inserted automatically. The Safety Tree is SAFA's representation of a Safety Assurance Case.

The Safety Tree for the current version (V1) is automatically merged with the previous version (V0). Changes are depicted through colors (red=removed, green=added, blue=modified), and clickable information icons.

Fig. 1: The SAFA process retrieves and visualizes artifacts, augments them with safety information, and then visualizes differences across versions. Artifact types can be customized to meet the needs of individual projects.

solution nodes in which the mitigation of specific hazards are represented as goals to be achieved. Hazards, and hence their associated mitigation goals, are often organized into a hierarchy, in which each leaf hazard is mitigated in the system through a set of artifacts that include safety requirements, design solutions, analytical models, source code, and assumptions, and are validated through diverse tests, reviews, and simulations. These artifacts are connected to each other, and to the leaf hazard, through a set of trace links. We refer to this collection of artifacts for a single hazard as an *Artifact Tree* (see Fig. 1.a), and to the collection of such trees as an *Artifact Forest*. SAFA retrieves Artifact Trees from project repositories (e.g., Jira and Github), and automatically augments them with argumentation structures that are designed to aid developers and analysts in reasoning about system safety. For proof of concept purposes we adopted the GSN approach, but SAFA could be adapted to work with other common SAC notations. We refer to this set of augmented Artifact Trees as *Safety Trees* (see Fig. 1.b). The entire set of Safety Trees forms a system-wide *Safety Forest*, composed of the top-level sets of goals, claims, and strategies, connected to a lower layer of Safety Trees through the leaf hazards.

When a new version of the system is developed, the previous version of the Safety Forest is retrieved and compared against the new Safety Forest. The differences between the two safety trees are then depicted in a Delta view (see Fig. 1.c), and SAFA then generates warnings within each Safety Tree to highlight changes that potentially undermine system safety [43]. Further, SAFA generates actionable recommendations to aid a safety analyst through the process of making changes that are needed to achieve and demonstrate safety. Recommendations include advice, such as to increase the diversity of evidence that a hazard has been mitigated through integrating test case results or simulations into the safety case; to investigate the impact of changed contexts or environmental assumptions on lower level nodes; or to assess or create trace links.

The contribution of our work is three-fold. First, we present an approach for *automatically generating a Safety Tree and highlighting safety-related deficiencies*. Second, we *compare the Safety Trees of a new version with those of a previously approved or certified version* to detect and visualize changes which could impact the validity of the SAC. Finally, we *generate warnings and recommendations to aid a user in evaluating and updating evidence of hazard mitigations* in the current Safety Trees. The ultimate goal of our work is to reduce the cost and effort of evolving and reusing safety case elements for a new version of the system, while maintaining or even increasing levels of safety.

The remainder of this paper is structured as follows. Sections II and III describe the SAFA approach for generating artifact trees, safety trees, and delta trees. Section IV introduces the SAFA toolkit. Section V describes SAFA's application to the Dronology system [15] and Section VI presents a controlled user study. Finally in Sections VII to IX we discuss threats to validity, related work, and conclusions.

II. Overview of the Process

We adopted a modified version of Wieringa's design science approach [63] to design, develop, and refine the SAFA solution. This included the five steps of information gathering and analysis, design, initial validation and refinement, user evaluation, and feedback based refinement. During the *information gathering and analysis* phase, we reviewed literature related to the use of Safety Assurance Cases (e.g., [8], [9], [41]) and traceability in safety-critical projects (e.g., [32], [33], [50]) including case studies and examples of completed SACs. This allowed us to reason about the ways in which SACs are impacted by changes made to the system, and to identify remediation steps that could update the SAC to reflect the current state of the system.

During the *design* phase, we leveraged our observations to iteratively design SAFA's process and algorithms. In the *initial validation and refinement* phase we developed features to generate Safety Trees. We then used those trees to build a SAC for each leaf hazard in order to evaluate the safety of our own system (Dronology) under development [14], [15], using our observations from this experience to improve the design of SAFA. After each increment we critically evaluated the effectiveness of the existing features, identified missing features and improvements, and then repeated the process. In the fourth phase of *external user evaluation* we conducted a formal study with developers as reported in Section VI. Finally, as a result of feedback from that study we entered the final phase of *feedback based refinement*. In the remainder of this section we discuss SAFA in more detail, using an example drawn from the Isolette case study, a safety-critical infant incubator used in hospitals [62]. We focus on the system's mitigation of the hazard, 'undetected thermostat failure'. We adopt the Goal Structuring Notation [41].

A. *Define a Traceability Information Model*

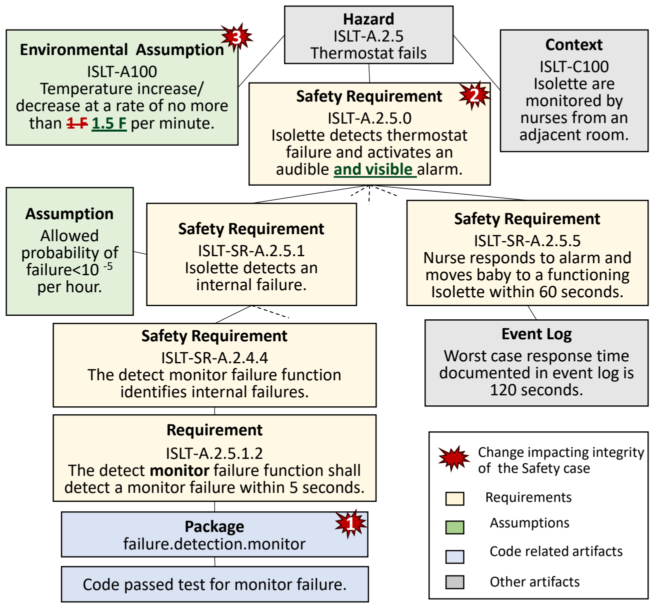
SAFA depends upon the presence of an underlying Traceability Information Model (TIM) that defines artifacts and their traceability paths [50]. We used the TIM depicted in Table I throughout our process, which depicts artifacts and their associations as commonly adopted across many safety critical development processes [12], [47], [53]. The artifacts are easily transformed into a hierarchical structure (i.e., Hazards→Requirements→Design Definitions→Code) with supporting evidence (i.e., context, environmental assumptions, acceptance tests, and simulations).

Artifact Type	H	SA	SR	R	DD	CX	EA
Hazard (H)	Ref					As	
Safety Analysis (SA)	An						As
Safety Req. (SR)	Mit		Ref				As
Requirement (R)			Ref	Ref			As
Design Def. (DD)				Rl	Ref		As
Src. Code & Tests (SC)					Imp		As
Context (CX)	Ctx	Ctx	Ctx	Ctx			
Acceptance Test (AT)				Test			
Simulation (SIM)				Sim			

TABLE I: Artifacts and directional trace paths: Ref=Refines, An=Analyzes, Mit=Mitigates, Rl=Realizes, Imp=Implements, Test=Tests, Sim=Simulates, As=Assumes, Ctx=Contextualizes

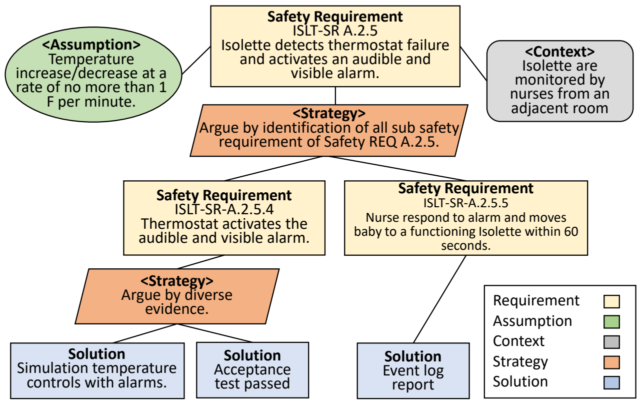
B. *Retrieving an Artifact Tree*

An *Artifact Tree* (Fig 1.a) is generated by recursively following trace links defined in the TIM. Contextual information, such as environmental assumptions, safety analysis artifacts, acceptance tests, and simulations, is also retrieved via trace links to provide evidence that the hazard has been mitigated. The artifact tree shown in Figure 2 represents some of the software artifacts associated with the Isolette hazard of "undetected thermostat failure". The generation of the Artifact Tree depends upon the existence of semantically-typed trace links defined in the TIM [50], [55]. Such trace links can be created manually, generated using an automated approach [48],



Hazard
ISLT-A.2.5
Thermostat fails
Environmental Assumption
3
ISLT-A100
Temperature increase/ decrease at a rate of no more than ~~1 F~~ 1.5 F per minute.
Context
ISLT-C100
Isolette are monitored by nurses from an adjacent room.
Safety Requirement
2
ISLT-A.2.5.0
Isolette detects thermostat failure and activates an audible and visible alarm.
Assumption
Allowed probability of failure<10 -5 per hour.
Safety Requirement
ISLT-SR-A.2.5.1
Isolette detects an internal failure.
Safety Requirement
ISLT-SR-A.2.5.5
Nurse responds to alarm and moves baby to a functioning Isolette within 60 seconds.
Safety Requirement
ISLT-SR-A.2.4.4
The detect monitor failure function identifies internal failures.
Event Log
Worst case response time documented in event log is 120 seconds.
Requirement
ISLT-A.2.5.1.2
The detect monitor failure function shall detect a monitor failure within 5 seconds.
Package
1
failure.detection.monitor
Code passed test for monitor failure.
Change impacting integrity of the Safety case
Requirements
Assumptions
Code related artifacts
Other artifacts

Fig. 2: A section of an Artifact Tree for the Isolette System, depicting mitigation of the hazard: "Undetected Thermostat Failure". The change events are discussed in Section III-A.



Safety Requirement
ISLT-SR A.2.5
Isolette detects thermostat failure and activates an audible and visible alarm.
<Assumption>
Temperature increase/decrease at a rate of no more than 1 F per minute.
<Context>
Isolette are monitored by nurses from an adjacent room
<Strategy>
Argue by identification of all sub safety requirement of Safety REQ A.2.5.
Safety Requirement
ISLT-SR-A.2.5.4
Thermostat activates the audible and visible alarm.
Safety Requirement
ISLT-SR-A.2.5.5
Nurse respond to alarm and moves baby to a functioning Isolette within 60 seconds.
<Strategy>
Argue by diverse evidence.
Solution
Simulation temperature controls with alarms.
Solution
Acceptance test passed
Solution
Event log report
Requirement
Assumption
Context
Strategy
Solution

Fig. 3: A section of the Safety Tree for the Isolette system showing claims, strategy, and solution nodes.

or mined from commit messages [56]. Many requirements management tools, such as Jira and DOORS, provide support for semantically typing links between different types of artifacts, and facilitate the creation of links as an integral part of the analysis and specification process. Furthermore, commits to version control systems, such as Github, can be tagged with requirement IDs to create links to source code. The trace links used by SAFA are required in most safety-critical domains [57], therefore our approach adds minimal overhead to the traceability effort [13].

C. *Transformation to a Safety Tree*

SAFA provides a set of heuristics listed in Table II for transforming an Artifact Tree into a Safety Tree (Fig 1.b). The heuristics define how claims, strategies, assumptions, and solution nodes should be inserted into the tree between

TABLE II: Rules for inserting SAC Nodes to transform an Artifact Tree to a Safety Tree

Id	Source	Target	Condition	Argument or Claim	Ref.
Strategy Nodes					
1	Hazard	Safety Anal.	Safety analysis exists	Argue that all critical faults for hazard ID have been identified.	[42]
2	Hazard	Safety Req.	Safety requirements exist	Argue that hazard ID is fully mitigated by its safety requirements.	[32]
3	Safety Req.	Env. Assump.	At least one environmental assumption is specified in the subtree	Assume that all environmental assumptions are identified for safety requirement ID.	[43] [29]
4	Hazard	Evidence	More than one independent evidence (FTA, Simulation, Test) is linked	Argument by diverse evidence	[35], [43]
5	Safety Req.	Design Def.	At least one design definition exists	Argument by claiming design definition meets safety requirement	[42]
6	Design Def.	Code with passed tests	At least one package is linked	Argument by claiming to have followed specific guideline in design definition (Argument by correct implementation of design)	[42]
7	Env. Assump.	Code with passed tests	At least one environmental assumption linked directly to code	Argument by operational assumptions satisfied ("test like you fly")	[44]
8	Safety Req.	Hazard	At least one hazard exists	Argument by claiming addressed all identified plausible hazards	[35], [42]
Assumption Node					
9	Prob.of hazard	Assumption	A FMEA assumption node exists	Claim that FMEA supports assumption of hazard's likelihood	[42]
Context Nodes					
10	Safety Req.	Context	System level hazard analysis context node exists	Claim that system level hazard analysis is a context node for top safety requirement	[42]
11	Safety Req. or Hazard	Context	A context node exists	Claim that defining and explaining ambiguous term in statement such as "correct","low","sufficient","negligible" is a context node for safety requirement or hazard	[35], [42]
Solution Node					
12	Prob. of hazard	FTA	FTA gives probability	Claim that FTA evidence is a solution for probability of hazard	[42]

existing artifacts. An example showing a partial Safety Tree is depicted in Figure 3. The heuristics for node insertion were derived from several literature sources describing guidelines for constructing safety arguments [32], [35], [42], [43]. For example, Row 2 of Table II implies that if a link exists between a hazard and one or more safety requirements, that an as-yet invalidated argument that the "hazard [ID] is fully mitigated by its safety requirements" should be inserted between them.

They include rules for inserting specific types of SAC nodes (e.g., strategies and claims), and for adding warnings when expected links and/or artifacts are missing. All inserted nodes are designed to help an analyst reason about the extent to which the hazard has been mitigated.

D. Engaging the User in Safety Tree Completion

There are three specific tasks that a user should perform on a Safety Tree. First, users should *respond to warnings and recommendations* that highlight problems such as missing design definitions or that make recommendations for inspecting various artifacts, rerunning an out of date acceptance test, or evaluating a set of candidate trace links. Second, *claims or strategy nodes that SAFA injects into a Safety Tree must be critically evaluated* by an analyst. For example, given a claim that a hazard mitigation has been sufficiently decomposed into safety requirements, an analyst should evaluate the claim to determine if it is correct. If a claim is found to be incorrect, then additional work is needed to remediate the problem. Finally, *safety arguments need sufficient contextual information* to enable future maintainers to understand the claims and to modify the SAC [19], [42]. In the context of SAFA this means that the analyst must carefully evaluate contextual information in the Safety Tree and incrementally refine it by adding additional arguments and contextual information so that the rationale for each claim is sufficiently clear [30].

III. Safety Tree Comparison

While SAFA is designed to help a user assess the safety of a single version of the system, its primary benefit is its support for reusing the safety analysis in subsequent releases. Kelly [42] provides examples of change scenarios that could impact system safety and trigger the need to review and potentially update the SAC. Examples include changes in regulatory requirements, system design, environmental assumptions, and operator behavior, all of which can cause the system to be deployed in ways that differ from the original intent. During the incremental design process for SAFA we observed concrete instances of each of these cases and also of cases in which change was initiated directly from the source code. SAFA is designed to recognize all these types of changes.

A. Types of Change Impact

Kelly identified three ways in which changes to a SAC argument structure could challenge its integrity [42]. These include changes to (1) solution nodes, i.e., low level nodes contributing to the achievement of higher level goals, (2) goals, which are direct or indirect ancestors of a set of solution nodes, and (3) contexts, which provide contextual information for individual nodes or entire subtrees.

We illustrate each type of change with examples drawn from the Isolette system (labeled as 1-3 in Figure 2). The first example introduces a change at the solution level within the *failure.detection.monitor* package. Source code has been modified and this triggers the need to evaluate whether associated higher level safety goals are still achieved. The second example reflects a high level change in which the safety requirement ISLT-A.2.5.0 is modified to add an additional hazard warning in the form of a visible light. This has a trickle-down effect on all its descendants including requirements and implementation. As a result, the entire subtree needs to be

TABLE III: Example rules for adding actionable advice to Delta SAC Tree

Change	**Warning**	**Secondary Condition**	**Recommendation**
W1: Feature is added or deleted	A new/deleted feature [FEATURE] potentially impacts the following Safety Trees: [LIST OF SAFETY TREES].	FMEA or FTA exists for feature.	Check that hazard analysis is complete for [FEATURE NAME]. If necessary create new Safety Trees associated with new hazards.
		At least one feature exists.	Perform a feature interaction analysis against existing features.
		At least one Safety Tree added or removed for the feature.	Perform a complete check of each Safety Tree associated with hazards for [FEATURE NAME] including all strategies, claims, and contexts.
W2: Requirement is added, deleted or modified	Requirement [REQUIREMENT ID] has been [Add., Del., Mod.]	Requirement has been added or modified.	Check the claim that requirement [REQUIREMENT ID] mitigates hazard [SAFETY TREE HAZARD ROOT] and is fully realized through the design, implemented in the code, with diverse and sufficient evidence provided.
		Requirement has been deleted or modified.	Check the claim that [REQUIREMENT PARENT] remains satisfied even though [REQUIREMENT ID] has been [DELETED—MODIFIED].
W3: Source code is changed	Source code in [PACKAGE] has been modified by [MODS]	All	Check that all code is covered by passed unit tests.
		SAFA recommends trace link maintenance	Check the [LIST OF RECOMMENDED TLE ACTIONS] to confirm additions and deletions of trace links.
W4: Context and/or assumptions have changed	Context|Assumption associated with [CONTEXT NODE] has changed.	Environmental assumption has changed	Check all goals, claims, strategies, and solutions that are influenced by [ASSUMPTION].
		Probability has changed	Check that all goals, claims, strategies, and solutions that are influenced by [PROBABILITY] are supported within the new probability context.
W5: Evidence has been modified or deleted	Evidence has been modified or removed.	Solution has been deleted or modified	Solution that [GOAL] has been satisfied is challenged by elimination or change of [EVIDENCE TYPE]. Check that [GOAL] is sufficiently satisfied.
		Diversity of evidence has been reduced.	Diversity of evidence that [GOAL] has been satisfied is reduced by elimination of [EVIDENCE TYPE]. Check that [GOAL] is sufficiently satisfied.
		Context node using this evidence has changed.	Check all goals, claims, strategies, and solutions that are influenced by [CONTEXT].

re-assessed. Finally, the third example illustrates a change to an environmental assumption to reflect new claims by the manufacturer of the heating units. Temperatures can now rise at a rate of 1.5°F/hr. This high-level contextual change potentially impacts the entire SAC requiring all safety requirements and their subtrees to be re-evaluated. SAFA recognizes these three types of change scenarios and generates appropriate recommendations. For example, in the case of a context change SAFA recommends: "The [CONTEXTUALIZED NODE] has been [MODIFIED|REMOVED] potentially invalidating the entire subtree. Inspect the subtree to determine if any elements depended upon the previous context."

B. Detecting Changes between two Safety Trees

To detect changes between two versions, V1 and V2, of a system, we compare the pair of Artifact Trees associated with each leaf hazard, checking for additions, deletions, and modifications of all artifacts defined in the TIM. Once the changes are identified, a *Delta View* is generated representing the superset of all artifacts found in a hazard's Safety Tree for both versions V1 and V2. Differences between the two versions are annotated to clearly depict which artifacts and/or trace links were *added*, *deleted*, and *modified* in the new version.

Furthermore, to gain a deeper understanding of any code modifications and their potential impact upon the safe mitigation of the hazards, we utilize Trace Link Evolution (TLE) [54], to reverse engineer source code refactorings, identify previously unlinked code that could impact the safety case, and recommend potential links between design definitions and code. TLE is a publicly available tool designed to detect a set of change scenarios based primarily on Fowler's dictionary of source code refactorings [25].

It employs a combination of information retrieval techniques (namely the Vector Space Model) [36], [48] and a refactoring detection tool [20] to detect 24 change scenarios organized into the six high-level change categories of *add class*, *delete class*, *add method*, *delete method*, *modify method*, and *basic*. TLE provides two useful functions – first, it offers explanations for why a specific class or set of classes has changed (e.g., due to a specific refactoring), enabling an analyst to determine whether the change is likely to impact safety or not. Second, when applied to design definitions (our lowest level of specification) and source code classes, TLE identifies and recommends potentially missing trace links, thereby introducing the possibility of adding new links, and therefore new artifacts, to the tree.

C. Generating the Delta View

Kelly previously defined a SAC change process as five steps: (1) recognizing challenges to the validity of the safety case, (2) expressing those challenges within the context of the SAC, (3) determining the impact of the challenge on the SAC, (4) determining actions needed to recover from the challenge, and (5) actual recovery [42]. Our SAFA visualization provides support for steps 1 to 3. Moreover, to address steps 4 and 5 SAFA augments the Delta View with recommendations designed to help the analyst address potential safety violations that were introduced through the changes. We incorporated a preliminary set of recommendations into SAFA based on the change scenarios discussed by Kelly and McDermid [43] and also our own observations from developing the Dronology systems of how change events impact SACs. However, our implementation of SAFA also includes an API for customizing the set of recommendations for a specific project to accommodate factors such as domain, safety integrity level, project-specific artifacts, supported traceability paths, adopted repositories and tools, and development processes.

An initial set of recommendations is depicted in Table III. Each recommendation includes a brief description, a primary condition under which it is triggered, an associated warning, and one or more secondary conditions which trigger specific recommendations. Kelly and McDermid [43] reported that when the integrity of a safety case is challenged, responsible parties should decide on an appropriate response ranging from a complete revision of the safety case to doing "nothing at all". For example, if a new feature has been added and a new hazard identified and documented, then the recommendation could be to perform an extensive safety analysis starting with hazard analysis for the feature by checking "that hazard analysis is complete for [FEATURE NAME]" and "if necessary create new Safety Trees associated with new hazards".

IV. Tool Support with the SAFA Toolkit

The SAFA Toolkit is designed to support the entire process of retrieving artifacts and links from project repositories, and generating and visualizing Artifact, Safety, and Delta Trees. In our project, all hazard descriptions, requirements, design definitions, environmental assumptions, acceptance test descriptions, and context information are stored in Jira, while code, unit tests, and acceptance test results logs are stored on Github. Our prototype implements the entire SAFA process described in this paper. It is implemented in Java with interfaces to Jira [5] and GitHub [27] repositories; however, it could easily be extended to interface with other repositories such as DOORS or Bitbucket. A TIM defining the trace links depicted in Table I was implemented using Jira's semantically typed trace link features. Trace links to source code are created by tagging GitHub commits with the respective Jira issue ID, while other trace links are created and maintained inside Jira.

The artifact parser first retrieves the hierarchy of hazards from Jira, identifies leaf hazards, and then retrieves all artifacts that are directly or indirectly linked to the leaf hazards in ways that comply with the TIM. This means that links are traversed in directions designated by the TIM, preventing the problem of a cyclic graph of artifacts. The SAFA toolkit displays the system-level hazard hierarchy, and then generates an Artifact Tree, Safety Tree, and Delta View for each leaf hazard. To generate a Delta View, the user specifies one version as the baseline, one as the current version, and requests all trees and views to be generated accordingly. SAFA uses Graphviz [1] to dynamically lay out nodes into a tree structure. Our SAFA toolkit currently shades *added* nodes green, *deleted* nodes red, and *modified* nodes blue. Warning nodes are shaded orange, and information is marked by an ! icon. Fig. 4 depicts an example of a Delta View for one of the Dronology leaf hazards rendered using the SAFA toolkit. An acceptance test and design definition have been added; another design definition and source code class have been modified; and one process requirement has been removed. In addition five recommendations are generated.

V. SAFA Evaluation on the Dronology System

During the *initial validation and refinement* phase of our research we used SAFA to support the safety analysis of two versions of the Dronology system. In this section we describe the process and key observations which led to the design

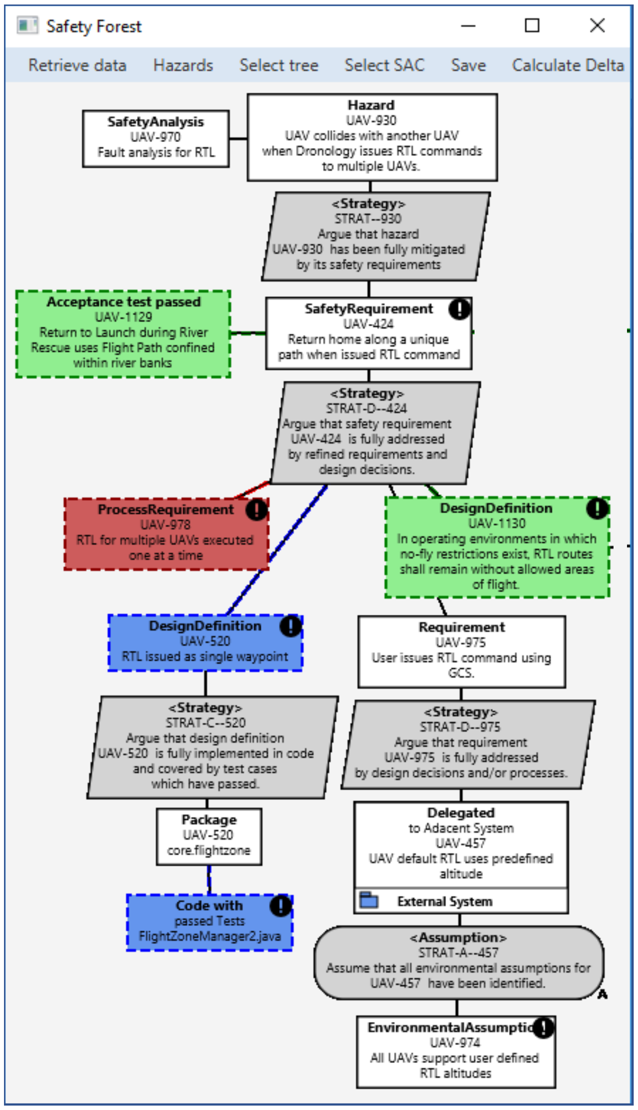

Safety Forest
Retrieve data
Hazards
Select tree
Select SAC
Save
Calculate Delta
SafetyAnalysis UAV-970 Fault analysis for RTL
Hazard UAV-930 UAV collides with another UAV when Dronology issues RTL commands to multiple UAVs.
<Strategy> STRAT--930 Argue that hazard UAV-930 has been fully mitigated by its safety requirements
Acceptance test passed UAV-1129 Return to Launch during River Rescue uses Flight Path confined within river banks
SafetyRequirement UAV-424 Return home along a unique path when issued RTL command
<Strategy> STRAT-D--424 Argue that safety requirement UAV-424 is fully addressed by refined requirements and design decisions.
ProcessRequirement UAV-978 RTL for multiple UAVs executed one at a time
DesignDefinition UAV-1130 In operating environments in which no-fly restrictions exist, RTL routes shall remain without allowed areas of flight.
DesignDefinition UAV-520 RTL issued as single waypoint
Requirement UAV-975 User issues RTL command using GCS.
<Strategy> STRAT-C--520 Argue that design definition UAV-520 is fully implemented in code and covered by test cases which have passed.
<Strategy> STRAT-D--975 Argue that requirement UAV-975 is fully addressed by design decisions and/or processes.
Package UAV-520 core.flightzone
Delegated to Adacent System UAV-457 UAV default RTL uses predefined altitude
External System
Code with passed Tests FlightZoneManager2.java
<Assumption> STRAT-A--457 Assume that all environmental assumptions for UAV-457 have been identified.
A
EnvironmentalAssumptio UAV-974 All UAVs support user defined RTL altitudes

Fig. 4: A Screenshot of a Delta Tree generated by SAFA for the leaf hazard "UAV collides with another UAV when Dronology issues a RTL (return to launch) command to multiple UAVs.

used in our formal user evaluation. We also address our first research question:

RQ1: What types of changes impacted hazard mitigations between versions V1 and V2, and is SAFA able to detect and visualize them?

A. The Dronology Project

We evaluated SAFA within the context of the Dronology system, which provides features for managing, monitoring, coordinating, and controlling small unmanned aerial vehicles (UAVs) [15]. It includes a flight manager responsible for scheduling flight routes for multiple UAVs, basic collision avoidance, a flight activity logger, several UIs for planning routes, monitoring UAVs in real time, registering UAVs, and a Ground Control Station (GCS) middleware, as well as a concrete implementation for communicating and controlling both simulated and physical UAVs. The Dronology GCS interfaces with ArduPilot-based UAVs [4] and has been flight-tested with five different physical UAV models. It also interfaces with the ArduPilot Software-in-the-loop (SITL) simulator [3] to enable high-fidelity simulations. The Dronology system exhibits non-trivial safety concerns due to its application to real-world scenarios including search-and-rescue and medical supply delivery. UAVs flying into traffic, crashing into the terrain during delivery operations, or failing to provide adequate area coverage during a life-critical search-and-rescue operation create numerous safety concerns and associated hazards.

Dronology's top level hazard tree includes 23 hazards with 15 leaf hazards. For all experiments described in this paper we utilize and compare two releases: V1 which was completed in the Spring of 2018 immediately prior to a public search-and-rescue demo, and V2 which introduced several new features. V1 included 49,400 LOC, 418 Java Classes, and a total of 146 Requirements, and 224 Design definitions. V2 involved over 500 hours of design and coding by a professional developer on a *collision avoidance* branch (branched August 2017 and merged June 2018), 1,500 hours of development by a team of 6 graduate and undergraduate students, and over 500 hours by senior members of our team with prior industrial experience on a *mission planning* branch (branched April 2018 and merged July 2018). V2 included 73,591 LOC, 646 Java classes, 185 Requirements, and 283 Design definitions. Both versions also included hazard analyses, acceptance and unit tests, simulations, environmental assumptions, and other contextual information.

B. Change Scenarios in Dronology

SAFA needs to handle realistic types of change scenarios. We therefore analyzed the types of change that occurred during the development of Dronology. Here we describe three changes,their potential impact on hazard mitigation, and SAFA's ability to detect and visualize them.

Context Changed: The Dronology system is subject to government regulations which provide strict guidelines for flights in various airspaces. Normally UAVs may fly up to 400 feet above ground level; however, a new deployment within close proximity to an airport, required us to fly in restricted airspace and to remain below 100 feet. This caused a change in a high level *context* node that previously stated that the airspace was uncontrolled, thus requiring all goals, strategies, and claims in lower levels of the SAC to be checked and modified. As a result, a new safety requirement to prohibit UAVs from flying above the maximum altitude was introduced, which in turn introduced a series of process requirements and software-related design decisions to prevent the hazard from occurring. SAFA visually depicted the change in context in the Delta View and also added warning (W4) as specified in Table III associated with a changed assumption.

Requirement Changed: In V1, a safety requirement associated with the "midair collision" hazard stated that "When two UAV's violate the MINIMUM_SEPARATION_DISTANCE they will both stop and hover in place". However, field tests showed that physical UAVs were unable to consistently stop in time unless the separation distance was unrealistically large. A new requirement was introduced in V2 requiring each UAV to progressively decrease its speed as it approached another UAV. SAFA adds warning (W2).

Evidence Changed: In V1 we made the assumption based on our prior operational experience that the Iris 3DR simulator was sufficiently accurate for testing flight plans prior to their execution. During development of V2 we started sending collision avoidance directives as velocity vectors instead of as waypoints and discovered that the simulator did not correctly simulate this behavior. We therefore removed simulations as a form of evidence that the safety requirements were met. As a result SAFA added warning (W5) associated with changed evidence causing us to address the lack of evidence by adding more diverse acceptance tests for use with the physical UAVs.

C. Observations from SAFA's Design Science Process

As a result of over 100 hours of effort dedicated to design, development, and use of SAFA we made several key observations which heavily influenced subsequent SAFA design decisions. Here we report four of them:

O1: During the refinement of the Safety Trees we heavily relied on the warning nodes to highlight errors of omission in the Safety Trees. Many of these issues were caused by inconsistencies between artifacts and missing trace links [50]. We modified SAFA's design by creating *direct hyperlinks* from each node to its associated entry in Jira so that links could be added and artifact text modified interactively.

O2: We noticed that the automatically inserted claims and strategies encouraged a false sense of security that an issue was fully addressed and that the tool should mark each claim as non-validated until it is actively checked and signed off by the analyst. This was addressed in the design, by setting the initial state of each claim and strategy node to non-validated, and then requiring the analyst to check and validate it. Any nodes potentially impacted by a new change are *reset to non-validated* so that the safety analyst is required to re-validate.

O3: Based on early evaluation with non-experts, we observed that users with limited safety experience, needed an explicit explanation for why a change might impact system safety and clear advice on actions to take. This observation led to the addition of *actionable recommendations* in the Delta Tree.

O4: Initially generated trees tended to contain sparse contextual information. Missing context was not apparent in Jira, but became evident once the trees were visualized. The hyperlinks to Jira enabled us to *incrementally add context and rationales* to each Safety Tree, to better support future maintenance and reuse through the Delta Views. This confirmed previous observations by Kelly [42].

These, and other, observations played a key role in the incremental design of the SAFA views.

TABLE IV: Dronology hazard mitigations impacted by changes between versions V1 and V2. The remaining five hazards (not shown) were not impacted.

	Hazard	Node CNT	Changes in v2							
			SR	PR	DD	AJ	CX	AS	TS	SC
H1*	Incorrect GOTO command causes terrain crash.	49					●	●		●
H2*	Midair collision during flight execution	28	●			●				
H3*	UAV flies above the altitude allowed by FAA.	7		●			●			
H4*	Physical failure of battery during flight	29			●				●	●
H5*	Collision occurs between UAVs at takeoff	58			●			●		●
H6*	Compass failure leading to localization error	8	●			●				
H7	Two UAVs collide when executing RTL commands	20							●	
H8$^+$	Insufficient coverage during search and rescue	259	●		●		●	●	●	●

Legend: Change in: SR (Safety Req), PR(Process), DD (Design Def.), AJ (Adjacent system), CX (Context), AS (Assumption), TS (Tests), SC (Code), Vis (Visualized in Delta Tree), Rec (Recommendation Provided). *Hazard included in the user study. $^+$ H8 has a high number of nodes due to numerous code classes.

D. Evaluation of SAFA's support for Change Scenarios

To answer RQ1 concerning the types of changes that impacted Dronology hazards, two researchers who had been part of the Dronology development process and had deep understanding of its safety issues, analyzed changes that potentially impacted the 13 leaf hazards. These were categorized as changes to requirements, design, process, environmental assumptions, context, tests, and source code, and formed the "answer set" for the first part of the study. A further analysis of these changes found that safety mitigations were impacted in 8 of the 13 cases. These hazards and the changed artifacts are shown in Table IV. For each changed artifact we checked whether SAFA had correctly detected and visualized the change, and whether an appropriate warning and recommendation had been issued by SAFA.

Results of our analysis, summarized in Table IV, indicated that source code changes impacted five hazards' mitigations. In one case (H8) this was associated with new functionality that introduced an entirely new sub-tree of requirements, design definitions and code. In two cases, the effect on hazard mitigations was the result of changes in the design (H4, H5). In two other cases the effects were associated with context and/or assumption changes, (H1, H7). In H1, TLE identified underlying code refactorings. In two cases (H2 and H6) a new safety requirement was delegated entirely to an adjacent system (ArduPilot and Ground Control Station, respectively), dramatically changing the system's response to those hazards. Finally, in one case a contextual change led to a modification in a process requirement in order to assume responsibility for handling that hazard (H3). Overall, we observed that Dronology's individual hazards were impacted by changes to multiple artifact types. SAFA correctly detected and visualized all these changes, and also provided correct warnings and recommendations (as designed) in all cases. As a future improvement, we also noted that SAFA displayed each change on the Delta View as an independent entity and was not yet capable of grouping related changes – for example, to indicate that a change in code was associated with a concurrent change in design. Grouping changes in this way could further aid analysts in understanding the impact of the identified changes.

TABLE V: Summary of participants' professional experience showing years in industry (Yrs) and whether they have experience in safety-critical projects (SC).

ID	Role	Domain	Yrs	SC
P1	Software Engineer	Aviation & Defense	8	Y
P2	Developer	Operating Systems	1	N
P3	Developer	Development	2+	Y
P4	Developer	Embedded Systems	1	N
P5	Developer	Embedded Systems	2	Y
P6	Software Engineer	Software Development	7	Y
P7	Developer	Information Systems	2	N
P8	Software Engineer	Unmanned Aerial Systems	1	Y
P9	Systems Engineer	Embedded Systems	35	Y
P10	Requirements Engineer	Defense Systems	23	Y

VI. User Evaluation

The next step of the design science approach involves "feedback based refinement" in which we conducted a formal user study to address the following research question:

RQ2: To what extent does SAFA's Delta View support an analyst in identifying changes which potentially impact the safety of a new version of the system?

A. Controlled User Study

Our study involved 10 participants with industry experience as listed in Table V. Seven of them had experience in developing safety-critical systems. At the start of the study, each participant viewed a 10 minute pre-recorded tutorial which provided background on the study, the Dronology system, and information about safety arguments and SACs. Based on a preliminary pilot study we decided to present 6 hazards to each participant. This kept the study duration to one hour (i.e., 15 minutes of initiation, and 45 minutes for analysis of the safety hazards). We selected 6 of our 11 leaf hazards based on the following inclusion criteria: (IC-1) there should be a non-trivial change that affected the hazard between versions V1 and V2, (IC-2) there should be diversity of change types in the Safety Trees, and (IC-3) the Safety Tree should be of a size that is analyzable within 10 minutes. Five hazards were discarded due to IC-1 and one was discarded due to IC-3. The final set, labeled H1-H6 in Table IV, were selected in a way that maximized diversity (IC-2).

Hazards were presented to the participants in two different ways, referred to as treatments T1 and T2:

T1: **Artifact Tree Pairs:** The user was given simultaneous access (in two windows) to V1 and V2 Artifact Trees for each hazard, and could click on any non-source code artifact to access its record in Jira. This treatment was representative of the current state of practice in which an analyst might assess change by retrieving two versions of the system in order to perform a comparison.

T2: **Delta View:** The user was given access to the Delta Tree for each hazard. As with the previous treatment, the user could also click on nodes to view records in Jira.

We divided participants as evenly as possible according to background and skill-set into two groups (A and B). Hazards were assigned in the same order to both groups (i.e., H1-H6); however, group A participants used Treatment T1 for hazards H1-H3, and Treatment T2 for hazards H4-H6, while group B used treatment T2 for H1-H3, and treatment T1 for H4-H6.

Participants assessed the six hazards using a think-aloud protocol. They were then asked the following question which was designed to evaluate the extent to which SAFA supported key steps in Kelly's safety case change process for recognizing challenges to the validity of the safety case [42]: (Q1) "With respect to this hazard, has the system changed in a way that potentially affects its safety? If so please explain your answer." After the participant had evaluated all six hazards we asked three additional questions. (Q2) "Was the information provided to you for each of the two methods sufficient for assessing the impact of change upon system safety?"; (Q3) "Which of the two methods did you prefer using? Why?"; and finally (Q4) "Can you suggest any improvements for the Delta Views?". One scribe documented the think-aloud statements while a second scribe observed the participant performing the tasks, and took additional field notes. Audio recordings were collected and used to confirm the transcripts.

B. Results

We first evaluated participants' responses to Q1. Based on the clear use of color encoding to visualize changes in the system, coupled with think-aloud protocol and the participants' use of the SAFA features to open up recommendations by clicking on specific nodes, we concluded that participants were aware of all color encoded changes in the Delta View. In contrast, participants inspecting changes using the paired artifact view only identified 80.6% of the potential problems.

On the other hand, once changes were identified, the participants were able to discuss their impact regardless of the technique used. For example, using the paired artifact method, participant P2 commented on hazard H6 that the design has "changed in a way that it now has to interface with an adjacent system therefore I would need to look at integration code". Similarly, using the Delta View for H6, on observing that two assumptions had been removed and one added, he commented that the "change in assumptions could create gaps" even though the new one "would improve safety".

However, one of the participants (P6) stated that visibility into modifications within artifacts (e.g., source code changes or rewording of requirements) was not available in the paired Artifact Trees, and that using the paired Artifact Trees he would have missed the fact that several source code classes had been modified which should trigger a safety review.

TABLE VI: Themes that emerged from user responses.

Q	Theme	Description	Cnt
Q3	Visualization	The extent to which color coding and other visualizations highlight issues	6
Q3	Speed	The extent to which problems can be identified quickly	6
Q3	Informative	The extent to which the provided information supports safety analysis	5
Q3	Process	The ease of the analysis process	5
Q3	Trust	The trust that a user places in SAFA to identify problems	1
Q4	Code insight	The ability of SAFA to identify and display impactful code changes	3
Q4	Rationale	Rationales explaining additions, deletions, or modifications of artifacts.	2

In response to Q2, all participants except one (P7), stated that they favored the Delta View over the artifact view. In fact one participant (P1), when asked to use treatment T1 after T2, specifically asked if he could have his Delta View back! Another (P9) said that he would "kill to have this (Delta View) in my work." Due to our adoption of the think-aloud protocol, we did not record the time taken to analyze each hazard; however, we observed that participants took significantly less time using the Delta View. Some participants – perhaps those with better spatial skills – were very good at finding differences in the paired Artifact Trees, even in quite large trees, while others found it particularly challenging. We observed that users frequently clicked on the informational (advice) icons and carefully read the recommendations generated by SAFA to describe the type of changes in the code. In one case, this allowed a participant to state that the change was "unlikely to impact safety" as it was a simple refactoring.

Two researchers (co-authors on this paper) then inductively coded the responses to questions Q3 and Q4 to identify prominent themes [51]. In total they identified six themes for Q3 and two for Q4. Following discussion, one of the (Q3) themes was removed, resulting in the themes reported in Table VI. As depicted in Table VI, six participants mentioned the benefits of visualization. For example, P1 stated that "color coding reduced confusion about what was new and not..." and P9 stated that it was "easier to understand change". Six participants also mentioned the speed with which they were able to identify impactful changes in the Delta View, using adjectives such as "quickly" and "at a glance". Five participants mentioned the informativeness of the Delta View. For example, P4 stated that you "can see code change, and as soon as code changes, everything up becomes suspect" while P9 stated that it would "help us [...] figure out risks." Conversely, P3 stated that "it is too easy to miss changes" using the paired Artifact Trees (Treatment T1). Five participants mentioned the simplification of the process using Delta Views. For example, P5 appreciated that it was "all in one place", while P1 noted that there was "no need to hunt between trees." Finally, participant P2 raised a particularly important issue of trust. He stated that he noticed himself putting "implicit trust" in the Delta View which could be problematic if it failed to highlight important safety concerns.

Responses to Q4 also provided valuable insights. Three participants requested additional information regarding the code changes, such as "whether I/O changed when code changes" (P2), displays depicting before and after views of committed code (P7), or the "complete history of commits between the two versions" (P9). Two participants requested rationales for all additions and deletions such as "reasoning behind the change" (P5). Throughout the study, most users discussed the actions they would take to resolve the safety concerns, for example questioning the validity of requirements (P10), identifying areas in which additional mitigations were needed (P9), investigating APIs (P6), and proposing additional acceptance tests (P1).

In summary, we conclude that SAFA's Delta View provided our users with affordances to quickly identify changes that potentially impacted the safety of the system. Furthermore, it helped them to identify actionable steps that were needed to ensure system safety and to update the SAC. In addition, once participants understood the potential of the Delta View they requested additional features that would provide insights into code changes and rationales for describing the addition, modification, and deletion of other artifacts. We plan to explore such requests in our future work.

VII. Threats to Validity

The user study that we conducted evaluated SAFA's support for safety-related change impact analysis. Participants all had industry experience, with seven having experience in safety critical development. However, only one participant had worked on an industrial UAV project. The use of only one project environment, (Dronology), for evaluation purposes is a key constraint caused first and foremost by the non-availability of test environments consisting of multiple versions of a system that contains rich samples of artifact types with associated trace links, and secondly by the cost and effort of building systems such as Dronology. To partially mitigate this limitation, we we illustrated our approach on an additional system (Isolette) from a different domain. Because the Dronology system used natural language requirements and not formal models, we make no claims about SAFA's generalizability to projects driven by formal methods. Furthermore, we defined an extensible TIM consisting of software artifacts and traceability paths that are commonly required in safety-certification guidelines, and the heuristics we defined could be extended or modified to reflect specific project contexts.

VIII. Related work

In the area of SAC maintenance, Kelly and Weaver [41] presented a set of patterns and recommended the use of modularity to support SAC evolution. Kelly and McDermid [43] investigated changes in evidence, context, assumption and requirements GSN nodes to determine how changes impact the safety assurance case. From this they proposed a safety case change management process but did not integrate traceability to automate the process. Kokaly *et al.* [46] use a KAOS goal tree to describe an assurance case and annotated evidence nodes when a change occurred. Greenwell *et al.* [29] introduced the safety case life cycle which feeds failure history of a system back into the SAC to find faulty assumptions or evidence and to recommend revisions. Several authors focused on specific types of artifacts and tools. For example, Jardat et al. [38], [39] used uncertainty analysis to extract contracts from Fault Tree Analyses (FTA) and to trace changes to the safety argument, while Felici [23] detected structural changes and content changes in nodes between two safety cases. Denney and Pai [18] automatically assembled auto-generated formal method sections of a SAC with manual parts and showed that integration of heterogeneous safety aspects can improve the correctness of safety assurance cases. Formal methods are just one type of node in a Safety Tree.

Several approaches have been proposed that automate traceability across diverse artifacts [16], [26], [52]. Other closely related work focuses on generating artifact slices or using formal verification techniques to support safety analysis [40], [58]. Falessi *et al.* [22] present SafeSlice that uses a TIM to guide the extraction of design slices with respect to functional safety requirements. Briand *et al.* [10] explored the use of SafeSlice to support the safety inspection process. More precisely they devised a technique for establishing traceability between safety requirements and SysML design models. Similar to our results, their study also showed a decrease in effort for assessing the safety of a system when providing traceability support. In our work we include diverse artifacts and provide additional information on how and why safety requirements might be affected by changes in linked artifacts. Several researchers have also investigated impact analysis from the perspective of system evolution – for example identifying the impact of operational anomalies [49] or source code changes on design [31].

IX. Conclusion

In this paper we have presented the SAFA approach for automatically generating Delta Trees designed to aid safety analysts and developers to identify changes that potentially impact system safety. We have described the application of approach across two versions of the Dronology system and have shown that SAFA is able to generate diverse warnings and recommendations commensurate with the types of changes we observed. Finally, we have reported results from a user study which shows the usefulness of SAFA for detecting and analyzing the impact of change upon an existing Safety Tree. The final Safety Tree produced for a new version can be integrated into the final safety assurance analysis process. In future work we plan to explore the integration of SAFA into diverse safety assurance contexts, and enrich the type of information and recommendations we provide to users.

Acknowledgments

This project has been funded by the US National Science Foundation Grants (CCF-1741781, CCF-1649448, and CCF-1513717) and Austrian Science Fund (FWF) (J3998-N31).

References

[1] Graphviz-graph visualization software, http://www.graphviz.org/.

[2] Adelard. Claims, Arguments and Evidence (CAE). https://www.adelard.com/asce, 2017. [Online; accessed 2019-02-10].

[3] ArduPilot. SITL Simulator. http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html, 2017. [Online; accessed 2019-02-10].

[4] Ardupilot Flight Controller. http://ardupilot.org, 2017. [Online; accessed 2019-02-10].

[5] Atlassian. Jira issue tracking system. https://www.atlassian.com/software/jira. [Online; accessed 2019-02-10].

[6] J. Birch, R. Rivett, I. Habli, B. Bradshaw, J. Botham, D. Higham, P. Jesty, H. Monkhouse, and R. Palin. Safety cases and their role in ISO 26262 functional safety assessment. In *Proc. of the Int'l Conf. on Computer Safety, Reliability, and Security*, pages 154–165. Springer, 2013.

[7] P. Bishop and R. Bloomfield. A Methodology for Safety Case Development. In *Ind. Perspect. Safety-critical Syst.*, volume 19, pages 194–203. Springer London, 1998.

[8] P. Bishop, R. Bloomfield, and S. Guerra. The future of goal-based assurance cases. In *Proc. of Workshop on Assurance Cases*, pages 390–395. Citeseer, 2004.

[9] R. Bloomfield and P. Bishop. Safety and assurance cases: Past, present and possible future–an Adelard perspective. In *Making Systems Safer*, pages 51–67. Springer, 2010.

[10] L. Briand, D. Falessi, S. Nejati, M. Sabetzadeh, and T. Yue. Traceability and SysML design slices to support safety inspections: A controlled experiment. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 23(1):9, 2014.

[11] J. Chen, M. Goodrum, R. A. Metoyer, and J. Cleland-Huang. How do practitioners perceive assurance cases in safety-critical software systems? In *Proc. of the 11th Int'l Workshop on Cooperative and Human Aspects of Software Engineering*, pages 57–60, 2018.

[12] J. Cleland-Huang, M. Heimdahl, J. H. Hayes, R. Lutz, and P. Maeder. Trace queries for safety requirements in high assurance systems. In *Proc. of the Int'l Working Conf. on Requirements Engineering: Foundation for software quality*, pages 179–193. Springer, 2012.

[13] J. Cleland-Huang, M. Rahimi, and P. Mäder. Achieving lightweight trustworthy traceability. In *Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering*, pages 849–852, 2014.

[14] J. Cleland-Huang and M. Vierhauser. Discovering, analyzing, and managing safety stories in agile projects. In *Proc. of the 26th IEEE Int'l Requirements Engineering Conf.*, pages 262–273, 2018.

[15] J. Cleland-Huang, M. Vierhauser, and S. Bayley. Dronology: an incubator for cyber-physical systems research. In *Proc. of the 40th Int'l Conf. on Software Engineering: New Ideas and Emerging Results*, pages 109–112, 2018.

[16] B. Dagenais, S. Breu, F. W. Warr, and M. P. Robillard. Inferring structural patterns for concern traceability in evolving software. In *Proc. of the 22nd IEEE/ACM Int'l Conf. on Automated Software Engineering*, pages 254–263. ACM, 2007.

[17] R. Dardar, B. Gallina, A. Johnsen, K. Lundqvist, and M. Nyberg. Industrial experiences of building a safety case in compliance with iso 26262. In *Proc. of the 23rd Int'l Symp. on Software Reliability Engineering Workshops*, pages 349–354. IEEE, 2012.

[18] E. Denney and G. Pai. Automating the assembly of aviation safety cases. *IEEE Transactions on Reliability*, 63(4):830–849, 2014.

[19] J. Dick, M. E. C. Hull, and K. Jackson. *Requirements Engineering, 4th Edition*. Springer, 2017.

[20] D. Dig, C. Comertoglu, D. Marinov, and R. E. Johnson. Automated detection of refactorings in evolving components. In *Proc. of the 20th European Conf. on Object-Oriented Programming*, pages 404–428, 2006.

[21] L. Duan, S. Rayadurgam, M. P. E. Heimdahl, O. Sokolsky, and I. Lee. Representing Confidence in Assurance Case Evidence. In *Proc. of the Int'l Conf. Comput. Safety, Reliab. Secur.*, pages 15–26, 2014.

[22] D. Falessi, S. Nejati, M. Sabetzadeh, L. Briand, and A. Messina. SafeSlice: a model slicing and design safety inspection tool for sysml. In *Proc. of the 19th ACM SIGSOFT Symp. and the 13th European Conf. on Foundations of Software Engineering*, pages 460–463. ACM, 2011.

[23] M. Felici. Modeling safety case evolution - examples from the air traffic management domain. In *Proc. of the Int'l Workshop on Rapid Integration of Software Engineering Techniques, Revised Selected Papers*, pages 81–96, 2005.

[24] Food and Drug Administration. *General Principles of Software Validation; Final Guidance for Industry and FDA Staff*, 2002.

[25] M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[26] A. Ghabi and A. Egyed. Exploiting traceability uncertainty among artifacts and code. *Journal of Systems and Software*, 108:178–192, 2015.

[27] Github. https://www.github.com. [Online; accessed 2019-02-10].

[28] P. J. Graydon and C. M. Holloway. An investigation of proposed techniques for quantifying confidence in assurance arguments. *Saf. Sci.*, 92:53–65, feb 2017.

[29] W. S. Greenwell, E. A. Strunk, and J. C. Knight. Failure analysis and the safety-case lifecycle. In *Proc. of the IFIP 18th World Computer Congress, TC13 / WG13.5 7th Working Conf. on Human Error, Safety and Systems Development*, volume 152 of *IFIP*, pages 163–176. Kluwer/Springer, 2004.

[30] J. Guo, N. Monaikul, and J. Cleland-Huang. Trace links explained: An automated approach for generating rationales. In *Proc. of the 23rd IEEE Int'l Requirements Engineering Conf.*, pages 202–207, 2015.

[31] M. Hammad, M. L. Collard, and J. I. Maletic. Automatically identifying changes that impact code-to-design traceability during evolution. *Software Quality Journal*, 19(1):35–64, 2011.

[32] R. Hawkins, K. Clegg, R. Alexander, and T. Kelly. Using a software safety argument pattern catalogue: Two case studies. In *Proc. of the Int'l Conf. on Computer Safety, Reliability, and Security*, pages 185–198. Springer, 2011.

[33] R. Hawkins, I. Habli, T. Kelly, and J. McDermid. Assurance cases and prescriptive software safety certification: A comparative study. *Saf. Sci.*, 59:55–71, 2013.

[34] R. Hawkins and T. Kelly. A Systematic Approach for Developing Software Safety Arguments. In *27th Int. Syst. Saf. Conf.*, pages 25–33, 2009.

[35] R. Hawkins, T. Kelly, J. Knight, and P. Graydon. A new approach to creating clear safety arguments. *Advances in Systems Safety*, pages 3–23, 2011.

[36] J. H. Hayes, A. Dekhtyar, and S. K. Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Trans. Softw. Eng.*, 32(1):4–19, 2006.

[37] C. M. Holloway. Safety case notations: Alternatives for the non-graphically inclined? In *Proc. of the 3rd IET Int'l Conf. on System Safety*, pages 1–6. IET, 2008.

[38] O. Jaradat, I. Bate, and S. Punnekkat. Using sensitivity analysis to facilitate the maintenance of safety cases. In *Proc. of the Ada-Europe Int'l Conf. on Reliable Software Technologies*, pages 162–176. Springer, 2015.

[39] O. Jaradat, I. Bate, and S. Punnekkat. Facilitating the maintenance of safety cases. In *Current Trends in Reliability, Availability, Maintainability and Safety*, pages 349–371. Springer, 2016.

[40] S. Kan. Traceability and model checking to support safety requirement verification. In *Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering*, pages 783–786. ACM, 2014.

[41] T. Kelly and R. Weaver. The Goal Structuring Notation–a safety argument notation. In *Proc. of the Dependable Systems and Networks 2004 WS on Assurance Cases*, page 6. Citeseer, 2004.

[42] T. P. Kelly. *Arguing safety: a systematic approach to managing safety cases*. PhD thesis, University of York York, 1999.

[43] T. P. Kelly and J. A. McDermid. A systematic approach to safety case maintenance. *Reliability Engineering & System Safety*, 71(3):271–284, 2001.

[44] J. Knight. *Fundamentals of Dependable Computing for Software Engineers*. Chapman Hall/CRC, 2011.

[45] J. C. Knight. Safety critical systems: challenges and directions. In *Proc. of the 24th Int'l Conf. on Software Engineering*, pages 547–550. ACM, 2002.

[46] S. Kokaly, R. Salay, V. Cassano, T. Maibaum, and M. Chechik. A model management approach for assurance case reuse due to system evolution. In *Proc. of the ACM/IEEE 19th Int'l Conf. on Model Driven Engineering Languages and Systems*, pages 196–206, 2016.

[47] N. G. Leveson. The Use of Safety Cases in Certification and Regulation. Technical report, MIT, 2011.

[48] A. D. Lucia, A. Marcus, R. Oliveto, and D. Poshyvanyk. Information retrieval methods for automated traceability recovery. In *Software and Systems Traceability.*, pages 71–98. 2012.

[49] R. R. Lutz, A. Patterson-Hine, S. Nelson, C. R. Frost, D. Tal, and R. Harris. Using obstacle analysis to identify contingency requirements on an unpiloted aerial vehicle. *Requir. Eng.*, 12(1):41–54, 2007.

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

[51] M. B. Miles, A. M. Huberman, and J. Saldana. *Qualitative data analysis: a methods sourcebook.* Sage, 2014.

[52] L. G. Murta, A. van der Hoek, and C. M. Werner. Continuous and automated evolution of architecture-to-implementation traceability links. *Automated Software Engineering*, 15(1):75–107, 2008.

[53] S. C. of RTCA. DO-178C, software considerations in airborne systems and equipment certification, 2011.

[54] M. Rahimi and J. Cleland-Huang. Evolving software trace links between requirements and source code. *Empirical Software Engineering*, 23(4):2198–2231, 2018.

[55] B. Ramesh and M. Jarke. Toward reference models of requirements traceability. *IEEE Trans. on Software Engineering*, 27(1):58–93, 2001.

[56] M. Rath, J. Rendall, J. L. C. Guo, J. Cleland-Huang, and P. Mäder. Traceability in the wild: automatically augmenting incomplete trace links. In *Proc. of the 40th Int'l Conf. on Software Engineering*, pages 834–845, 2018.

[57] P. Rempel, P. Mäder, T. Kuschke, and J. Cleland-Huang. Mind the gap: assessing the conformance of software traceability to relevant guidelines. In *Proc. 36th Int'l Conf. on Software Engineering*, pages 943–954, 2014.

[58] M. Sabetzadeh, S. Nejati, L. Briand, and A.-H. E. Mills. Using SysML for modeling of safety-critical software-hardware interfaces: Guidelines and industry experience. In *Proc. of the IEEE 13th Int'l Symp. on High-Assurance Systems Engineering*, pages 193–201. IEEE, 2011.

[59] R. Steinzor. Lessons from the North Sea: Should "safety cases" come to America? *Bost. Coll. Environ. Aff. Law Rev.*, 38(2):417–444, 2011.

[60] M. Sujan, I. Habli, T. Kelly, S. Pozzi, and C. Johnson. Should healthcare providers do safety cases? Lessons from a cross-industry review of safety case practices. *Saf. Sci.*, 84:181–189, Apr 2016.

[61] U.K. Ministry of Defence. Defence Standard 00-56, Issue 7: Safety Management Requirements for Defence Systems. Part 1: Requirements, 2017.

[62] US Department of Transportation. Requirements Engineering Management Handbook, Appendix A - Isolette Thermostat Example. *DOD/FAA/AR-08/32*, 2009.

[63] R. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Springer, 2014.