SafeTrace: A Safety-Driven Requirement Traceability Framework on Device Interaction Hazards for MD PnP

Andrew Yi-Zong Ou
Maryam Rahmaniheris
University of Illinois at Urbana-Champaign, USA

Yu Jiang
Tsinghua University, China

Lui Sha
University of Illinois at Urbana-Champaign, USA

Zhicheng Fu and Shangping Ren
Illinois Institute of Technology, USA

ABSTRACT

Requirements management and safety analysis have been the key foundations of the successful development of life-critical systems, and the traceability of safety-related artifacts across such systems is becoming ever more important. Unless safety analysts can trace when and how requirements and design change, their analysis will become inconsistent, and eventually fail as proof that a given system can mitigate certain faults during certification processes. However, most prior research on traceability has focused on requirements, design and source code changes, rather than the integration of safety analysis by considering device interactions such as the Medical Device plug-and-play (MD PnP) into traceability and change-impact analysis. To help fill this gap, this paper proposes a safety-driven requirement traceability framework, SafeTrace, that traces the relations between safety requirements, design, and safety analysis, and the impact of requirement and design changes on safety analysis for life-critical systems with a focus on medical device interaction hazards.

CCS CONCEPTS

•Software and its engineering → Risk management; Fault tree analysis; *Requirements analysis;*

KEYWORDS

Traceability, Requirement Engineering, Fault-Tree Analysis

ACM Reference format:
Andrew Yi-Zong Ou, Maryam Rahmaniheris, Yu Jiang, Lui Sha, and Zhicheng Fu and Shangping Ren. 2018. SafeTrace: A Safety-Driven Requirement Traceability Framework on Device Interaction Hazards for MD PnP. In *Proceedings of SAC 2018: Symposium on Applied Computing , Pau, France, April 9–13, 2018 (SAC 2018),* 10 pages.
DOI: 10.1145/3167132.3167270

1 INTRODUCTION

Unlike commercial or business software development that does not require safety analysis during planning and development by certification agencies, developing medical systems requires safety analysis and it is mandated by agencies such as the FDA or standards such as the Advancement of Medical Instrumentation (AAMI)/IEC 62304 [1]. These safety analyses are performed to prove that the system meets the *safety requirements* collected by requirement analysts from medical professionals. Moreover, each configuration of the medical systems requires performing a safety analysis on the artifacts by following FDA requirements. A commonly used approach for safety is Fault-Tree Analysis and its associated fault mitigation procedures. With software design changes, the safety analysis might be out-synchronized with system design soon; hence, it might no longer reflect whether the current software design still meets the safety requirements.

One important type of safety hazards in medical systems are interaction hazards that can create safety hazards with no changes to existing medical devices' hardware or software. For example, in a medical environment that requires device interoperations such as the Medical Device Plug-and-Play (MD PnP) Interoperability program [2], a surgical fire could be the result of a laser scalpel emission while a ventilator is supplying oxygen. Neither of the devices causes the harm to the patient. Instead, it is the interaction of both devices that brings the harm to the involved patient. Furthermore, the failure of non-safety critical components in an integrated clinical environment such as the OpenICE [3] of the MD PnP can lead to critical mishaps. For example, a network router providing communication is not considered as a safety-critical device. But router failures can cause the ventilator to not receive commands to resume oxygen supply and cause brain hypoxia to the patient. As such, a new change impact analysis that considers the interactions between components for medical life-critical system development is needed.

Many studies have examined how to integrate safety analysis into traceability research. One such approach [4–7] leverages model-based development, so that system models can generate safety analysis automatically; examples of this include Fault-Tree Analysis (FTA) [8] and Failure Mode and Effect Analysis (FMEA) [9]. However, the imperfect process of translation from system models to safety analysis can cause blind spots in impact analysis. A significant amount of research has analyzed the impact of requirement changes on source code [10], but unfortunately, the impacts of requirement and design changes on safety analysis have yet to be addressed. Moreover, most existing impact-analysis work provides information about the affected areas in artifacts but does not indicate whether or not an upstream artifact change (*e.g.,* a design change) causes negative impacts to downstream ones (*e.g.,* safety analysis), despite such information being crucial for safety analysis. Furthermore, an intuitive approach to avoid the safety analysis from becoming outdated is to flag critical components when a change is made to the component in safety critical system development.

However, this is only necessary but insufficient for medical systems that require device interactions to provide correct services. Because configurations of medical devices will be changed due to device interactions and the configurations themselves can create safety hazards with no change to safety-critical medical devices (Class III medical devices).

Commercial tools such as IBM Rational DOORS [11], Yakindu Traceability [12], and Intland codeBeamer [13], are effective tools to support traceability for general use, but, they do not and do not need to meet the FDA requirements of safety analysis when used to develop non-medical systems. Even though some tools (*e.g.*, codeBeamer) support safety analysis such as Failure Mode Effect Analysis (FMEA) [9], the trace links are at a relatively higher level and lack a more fine-grained control of trace links. Other tools, such as the TraceLab [14], are a test-bed for instrumenting trace links and not designed for project management. Our tool is a complement to the existing tools. We address this research gap from a safety aspect on traceability for medical device plug-and-play systems.

Specifically, the focus is how to track down the software components that need to be changed when a new configuration creates a new fault tree. The framework ensures the modified or newly created fault trees, due to the components changes, still meet the safety requirements based on the minimum cut set generated by FTA for safety. Then, with traceability between safety requirements, design, and safety analysis in place, our tool for modified system design can decide whether the change can cause potential faults based on its change-impact-analysis algorithm.

The **main contributions** of this paper are **(1)** the design of the SafeTrace framework capable of managing traceability among safety requirements, design, and safety analysis in MD PnP systems, and **(2)** a change-impact analysis, focused on safety analysis, that can provide the information needed to answer questions such as whether a change of requirements or design may cause safety violations.

The rest of this paper is organized as follows. A network-connected airway laser-surgery system we use as a case study is introduced in Section 2, along with background information about requirements; system design with collaboration diagrams; and FTA. In Section 3, we present the proposed SafeTrace framework. Two change-impact-analysis algorithms for safety analysis, one for requirement changes and the other for design changes, are presented in Section 4. In Section 5, we test how SafeTrace applied to an MD PnP system which can counter communication failures, help trace the root cause of a failure, and add a safety requirement, as well as how a life-critical system design change can affect safety analysis. Section 6 discusses related work, and Section 7 presents our conclusions.

2 MEDICAL LIFE-CRITICAL SYSTEM EXAMPLE

The present work was motivated by our desire to improve the safety of airway laser surgery (also known as laser tracheotomy). Using the design of an airway laser-surgery life-critical system as an example, this section provides an introduction to life-critical system safety requirements, system design, and safety analysis.

2.1 Tracheotomy Laser Surgery

In a laser tracheotomy [15], the life-critical system incorporates a ventilator and a laser scalpel. A physician uses the scalpel to unblock the patient's trachea. Because the patient is under anesthesia, s/he breathes through a mask that supplies a high concentration (usually 100% [16]) of oxygen from the ventilator. During surgery, the laser beam could accidentally ignite the tube; thus, fire in the operating room is the primary safety hazard. As such, the flow of oxygen supplied by the ventilator should be blocked while the laser is emitting. However, if the blocking of the oxygen flow from the ventilator exceeds a certain duration, it will cause hypoxia and potential brain damage to the patient.

With this intuition of airway laser surgery, there are a few observations of the fire hazard. First, the laser emission can ignite in a close vicinity with high concentration oxygen such as an airway. It means that a laser scalpel can cause safety hazards only in a certain condition, and, in this case, the condition is that an airway with high concentration oxygen. Second, even we stop the oxygen supply to the patient under anesthesia, we cannot enable the laser emission immediately. Because there is still high concentration oxygen in the airway and needs to be ventilated outside the airway. A safe threshold of oxygen concentration with potential surgical fire is less than 30%.

From a system perspective, for the laser scalpel, "no-operation, 0" represents it is not emitting the laser, and "in-operation, 1" represents it is emitting. For the ventilator, "in-operation, 1" indicates that it is supplying high concentration oxygen, and "no-operation, 0" represents that it is supplying plain air only.

In the following three sub-sections, we introduce a Medical Device Plug-and-Play (MD PnP) system [2, 3] to assist physicians to perform airway laser surgery, and set forth its safety requirements, design, and safety-analysis procedures. Following the introduction of SafeTrace in Sections 3 and 4, we will evaluate our methods using MD PnP for airway laser surgery in Section 5.

2.2 Safety Requirements

Requirements engineering guides the whole system development process. For purposes of this paper, we have limited the scope of safety requirements. An annotated requirement artifact is defined as a text description of a safety goal that the system that needs to achieve. An example of annotations is the universal identifier for a particular requirement. Hereafter, we use the term "requirement" to refer an annotated requirement artifact.

In airway laser surgery, as discussed above, there are two safety requirements derived from clinical needs that should be satisfied during the surgery operation:

- Safety Requirement 1 (SafeReq-1): To avoid fire, the ventilator and the laser scalpel should never be in their respective in-operation states at the same time.
- Safety Requirement 2 (SafeReq-2): To avoid patient brain damage due to hypoxia, the ventilator should remain in its no-operation state for no longer than a specified period.

The two requirements are related to two challenges during surgery, either of which can lead to severe surgical failures. In regard to SafeReq-1, current practices rely heavily on the surgery team to ensure that the laser scalpel and the ventilator are not in the in-operation state at the same time, and despite their best efforts, mistakes can still easily occur. Second, to meet SafeReq-2, surgical teams currently must simply remember what the safe period of the ventilator's no-operation state is, and when such a state began; so again, the possibility of human error causing a tragic accident is fairly strong [17].

2.3 System Design

There are multiple ways to represent a system's design. For example, standard UML diagrams include class-, component-, and sequence diagrams, among others. In this paper, we use a collaboration diagram to describe software components, hardware devices, or platform including both hardware and software as the main system-design artifacts. This allows a level of design detail sufficient for our traceability framework to remain visible on users' operation views of the system.

To mitigate both medical hazards, an MD PnP laser-surgery system [2] is proposed to solve the problem. Figure 1 presents its architecture, in which either the ventilator or the laser scalpel has an MD PnP device adapter to communicate with a supervisory computer through a wired network, enabling the coordination of a series of actions that can prevent failures due to violations of SafeReq-1 and SafeReq-2.

By coordinating actions through the supervisor computer, the system can effectively prevent operating-room fires, and thus complies with SafeReq-1. We use a scenario that a laser scalpel operated by a surgeon requests to laser emission to demonstrate event propagations. When the laser client adapter sends a laser-emission request to the MD PnP Application (Commands 1 to 4), the supervisor computer examines the current state of the ventilator (Commands 5 to 11). Then, the MD PnP Application sends a command to acknowledge the laser operations (Commands 12 to 15). And whenever the ventilator's oxygen supply is cut, the MD PnP Application sends out a command to resume it within the timeframe required by SafeReq-2.

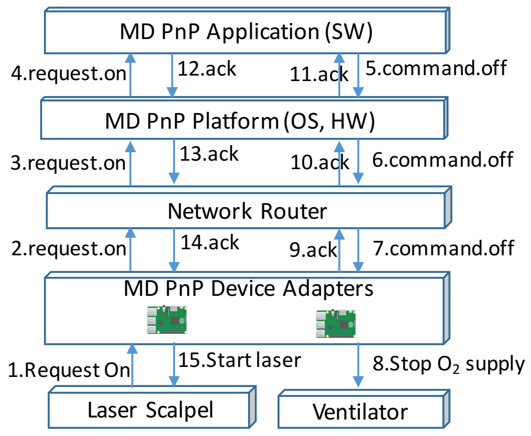


Figure 1: The collaboration diagram of MD PnP Tracheotomy Systems. A rectangle is a software component, hardware device, or a platform. Each link between blocks represents an information flow.

2.4 Safety Analysis

Safety analysis helps system engineers to identify the potential hazards associated with a particular scenario, and to analyze whether the proposed system can mitigate such hazards in a range of situations. Among various safety-analysis methods, Fault-Tree Analysis (FTA) is one of the most widely used. One approach to FTA evaluation involves quantitative evaluation of performance metrics, such as reliability or Mean Time To Failure (MTTF). Another involves qualitative analysis of whether or not an input event that occurs at a tree's leaf node – referred to as a *primary event* – propagates to the failure in the root, an event known as the undesired *top event* of the tree. Between primary events and the undesired top event in a fault tree, there are intermediate events and logic gates. However, provided that a tree's logic gates are preserved, all the intermediate events can be eliminated. A method of quality analysis for a fault tree is called a minimum cut set (MCS) [18], as further defined below:

Definition 2.1. A cut set in a fault tree is a set of primary events whose occurrence (at the same time) ensures that the TOP event occurs. A cut set is said to be *minimal* if the set cannot be reduced in size without loosing its status as a cut set.

For example, given the $MCS = \{\{A\}, \{B, C\}\}$, if either A, or B-and-C becomes true, the hazard of the tree becomes True. For purposes of this paper, the undesired top event is a medical safety hazard (*e.g.,* fire or hypoxia). For more detail on fault trees and their applications, see Hoyland and Rausand [18] and Stamatelatos *et al.* [19].

3 SAFETRACE TRACEABILITY FRAMEWORK

This section first presents SafeTrace aimed at facilitating life-critical system development. Second, it defines the traceable artifacts in requirements, design, and safety analysis. And third, it introduces trace links between artifacts and the evolutions of artifacts throughout development, with a focus on trace management for safety analysis.

3.1 Overview of SafeTrace

The goal of SafeTrace is to ensure that whenever design documents or system requirements are changed, the impact on safety analysis is evaluated. With this concept in mind, Figure 2 presents the SafeTrace's architecture. The left-hand side shows the three targeted types of artifact: requirements, design documents, and safety analysis. Each artifact has corresponding stakeholders, *i.e.,* its designers; and all artifacts are either co-located in a physical machine or in a distributed environment. The middle box circled with a dashed line represents the SafeTrace framework. The SafeTrace's main component is the Traceability Manager, which contains a change management unit that defines the various trace links for different artifact types. Once trace links have been created by stakeholders, these links are stored in a traceability repository.

To detect changes made to requirements and design artifacts, each artifact type is associated with an Artifact Monitor, whose monitoring rules are specified in Change Management. One way for a Monitor to detect an artifact change is by monitoring the artifact's repository. Specifically, once a change of requirements or safety analysis is made and checked into the associated repository, the corresponding Monitor can retrieve the artifact from the repository and run an artifacts-parsing program to examine how it differs from the previous version. Another method is to monitor artifacts during development, which allows the stakeholders to be alerted before an updated artifact is checked into the repository. If a change

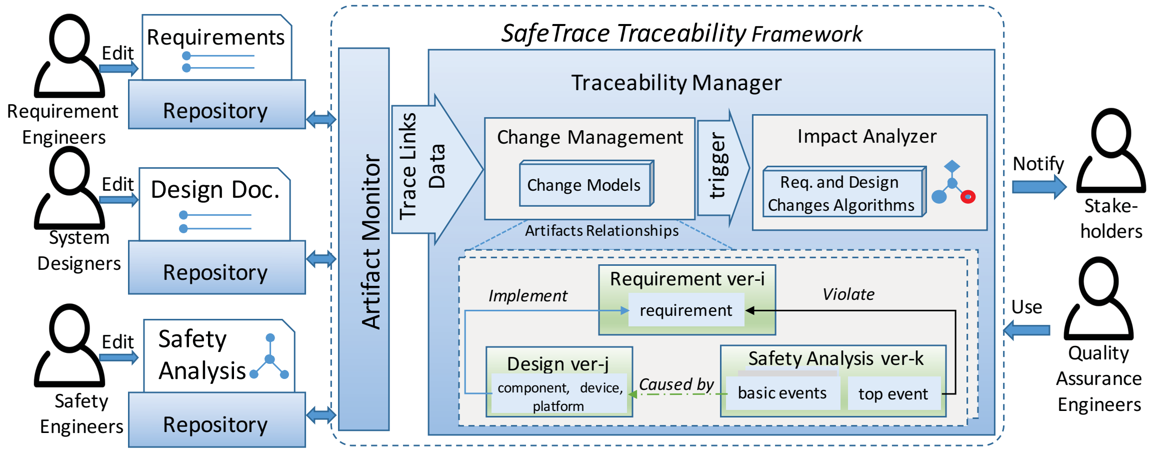


Figure 2: The Architecture of the SafeTrace Framework

to a requirements or design artifact is observed, and that change is configured to trigger an impact analysis, then SafeTrace will run the change-impact analysis (which will be discussed further in Section 4). If the results of change-impact analysis show that some part of the safety analysis might be affected (potentially causing hazards), the SafeTrace's notification module notifies the relevant safety engineers.

3.2 Artifacts of Requirements, Design and Safety Analysis

A traceable artifact a_i can be (1) an annotated requirement, (2) an annotated component or device, (3) a top event in a fault tree, or (4) a basic event in a fault tree (we will further discuss this particular event in short). All artifacts are annotated, but a *required annotation* is a universal identification number used for indexing/identification of all traceable artifacts.

We elaborate each traceable artifact a_i below and present the trace links in the next section.

3.2.1 Requirements. A traceable *requirement artifact* is defined as a text description of a requirement with annotations. The text description may specify safety aspects of the system's desired goal.

3.2.2 Design. A traceable design artifact is a software component, a hardware device, or a platform including both software components and hardware devices in design diagrams such as a component diagram (for software components) or a deployment diagram (for hardware devices). Depending on the need for granularity in tracing, an artifact might also be a platform that includes both software components and hardware devices in an implementation diagram. Although other system design levels such as finite state machines, system configurations, and *etc.* of the system are considered significant to a system design, it is beyond the scope of our work. Here, we focus on the abstract architecture of a system.

3.2.3 Fault-Tree Analysis. In FTA, we can use the MCS theorems mentioned in Section 2.4 to reduce a tree to its MCSs. For this purpose, a *primary event* could be a basic event, an external event, or some other type of event [8], but for the purposes hereof, we focus only on basic events and external events. Each event contains a proposition, which can be True or False. The True value represents the event being triggered, whereas the False value means the event

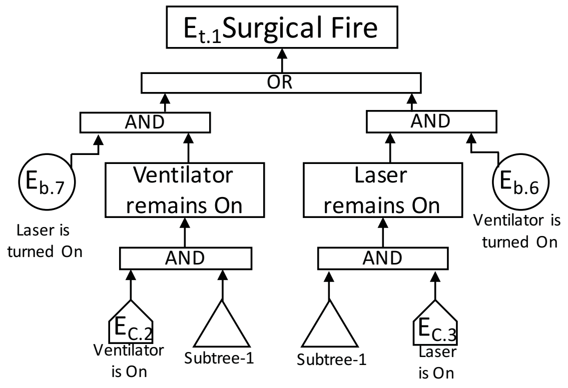


Figure 3: Fault-Tree Analysis, $\text{Fire}_{P_2.FT_1}$**, for Surgical Fire for MD PnP Tracheotomy Systems**

does not occur. A basic event is one that does not develop further, *i.e.,* is a leaf of a fault tree. For example, a basic event could be that a ventilator is switched to its no-operation state, stopping the supply of oxygen, and does not develop further. On the other hand, a primary event can be an external event, which is treated as having a constant value. The purpose of defining such events is to specify conditions when used with a basic event. For example, one type of failure in airway laser surgery is hypoxia caused by $Sp0_2$, an estimate of amount of oxygen in the blood, falling below a clinically required threshold, due to the ventilator not supplying oxygen for a certain period of time, and the supervisor not being able to send a command to the ventilator to re-supply it. In this example, the *external event* is the ventilator remaining at no-operation, which serves as a condition for the basic event: that $Sp0_2$ falls below the required threshold.

Figure 3 is the FTA safety analysis of the MD PnP system with regard to fire hazard, and its Subtree-1 is presented on the right-hand side of Figure 4. For brain hypoxia, the fault tree is shown on the left-hand side of Figure 4; and further information on both of these fault trees is presented in Table 1.

For safety purposes, we define a third type of event, a *safeguard event.* A safeguard event is also a basic event, but is *caused by* a

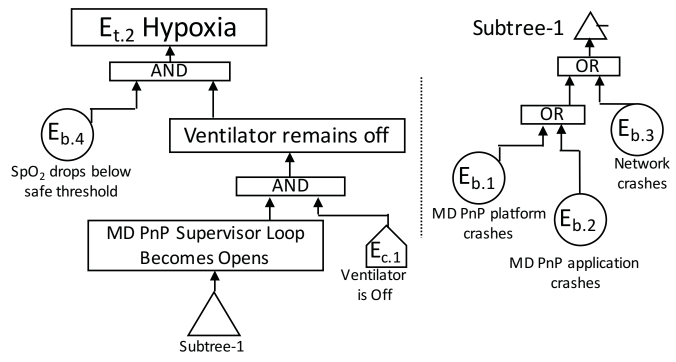


Figure 4: Fault-Tree Analysis, $\text{Hypoxia}_{\text{P}_2.\text{FT}_2}$**, for Hypoxia for MD PnP Tracheotomy Systems**

safety feature in the system design: for example, a watchdog timer for monitoring how long a ventilator remains at no-operation. An event for the watchdog timer could be that it fails to notify the physicians that the specified duration has elapsed. The proposition will have the value False, because the timer is designed to monitor the system at all times.

In an MCS, if all basic events become True simultaneously, the top event becomes True and causes the type of failure associated with that top event. A single system may have multiple fault trees, and each fault tree may have multiple MCSs. Based on the MCS theory, we specify that a traceable artifact for FTA includes a top event of the tree (i.e., a failure proposition), a basic event, and a safeguard event. We omit the traceability of intermediate events (*i.e.,* events between top events and basic events) and logic gates. An intermediate event, meanwhile, is caused by some combination of input events. That is, an intermediate event can always be reduced to a combination of basic events, so tracing only the basic events will be sufficient. And logic gates, which are used to set up the relations between events, are preserved in the MCSs of each tree, so we only need to trace the basic events belonging to each such MCS rather than the gates themselves. The fire-hazard tree's MCS is shown in Equation 1, and the brain-hypoxia tree's MCS in Equation 2.

$$\begin{aligned}\text{Fire}_{\text{P}_2.\text{FT}_1} = \{&\{E_{b.1}, E_{b.7}, E_{c.2}\}, \{E_{b.2}, E_{b.7}, E_{c.2}\},\\ &\{E_{b.3}, E_{b.7}, E_{c.2}\}, \{E_{b.1}, E_{b.6}, E_{c.3}\}, \{E_{b.2}, E_{b.6}, E_{c.3}\},\\ &\{E_{b.3}, E_{b.6}, E_{c.3}\}\}\end{aligned} \tag{1}$$

$$\begin{aligned}\text{Hypoxia}_{\text{P}_2.\text{FT}_2} = \{&\{E_{b.4}, E_{b.1}, E_{c.1}\}, \{E_{b.4}, E_{b.2}, E_{c.1}\},\\ &\{E_{b.4}, E_{b.3}, E_{c.1}\}\}\end{aligned} \tag{2}$$

With the minimum cut sets, we can then define an affected fault-tree in Definition 3.1. As mentioned before, a safeguard event in an MCS is used to guarantee that the proposition of the top event will not be True. For basic events, we assume at least one basic event has a trace link to a design artifact so that once the design artifact changes, the framework can trace the corresponding events.

Definition 3.1. An *affected fault-tree* is defined as:

$$\begin{aligned}&\{\text{AffectedFaultTree} \mid \exists \text{ Minimum Cut Set } mcs_k \text{ that has}\\ &\ 1) \text{ at least one traced basic event},\\ &\ 2) \text{ no safeguard events},\\ &\ \text{in AffectedFaultTree}, \forall i, k \in \mathbb{N}\}\end{aligned} \tag{3}$$

Table 1: Events Used In Fault-Tree Analysis

Events	Appeared Fault-Trees	Meaning
$E_{t.1}$	$P_2.FT_1$[1], $P_3.FT_1$	Surgical Fire
$E_{t.2}$[2]	$P_2.FT_2$[3], $P_3.FT_2$	Brain Hypoxia
$E_{b.1}$[4]	$P_2.FT_2$, $P_3.FT_2$, $P_2.FT_1$, $P_3.FT_1$	MD PnP platform crashes
$E_{b.2}$	$P_2.FT_2$, $P_3.FT_2$, $P_2.FT_1$, $P_3.FT_1$	MD PnP application crashes
$E_{b.3}$	$P_2.FT_2$, $P_3.FT_2$, $P_2.FT_1$, $P_3.FT_1$	Network crashes
$E_{b.4}$	$P_2.FT_2$, $P_3.FT_2$	SpO_2 drops below safe threshold
$E_{b.5}$	$P_3.FT_2$, $P_3.FT_1$	Open-loop safe software crashes
$E_{b.6}$	$P_2.FT_1$, $P_3.FT_1$	Ventilator is turned On
$E_{b.7}$	$P_2.FT_1$, $P_3.FT_1$	Laser is turned On
$E_{c.1}$[5]	$P_2.FT_2$, $P_3.FT_2$	Ventilator is Off
$E_{c.2}$	$P_2.FT_1$, $P_3.FT_1$	Ventilator is On
$E_{c.3}$	$P_2.FT_1$, $P_3.FT_1$	Laser is On
$E_{s.1}$[6]	$P_3.FT_2$, $P_3.FT_1$	Open-loop safe MD PnP device adapter crashes

[1]Fault-Tree 1 (*i.e.,* Fire) in Phase 2
[2]t represents the undesirable top event of a tree.
[3]Fault-Tree 2 (*i.e.,* Hypoxia) in Phase 2
[4]b represents a basic event of a tree.
[5]c represents an external event with default value True.
[6]s represents an event with value False caused by safe hardware or software.

3.3 Linking Traceability Relationships Among Domain Artifacts

Based on the artifact definitions presented in the previous subsection, we can proceed to defining and discussing a trace link between two artifacts, as follows:

Definition 3.2. A *trace link* $t_{(a_{src}, a_{dst})}$ is a directed edge from a source artifact a_{src} to a destination artifact a_{dst} in a traceability graph $G = (V, T)$, where $a_{src}, a_{dst} \in V$ and $t_{(a_{src}, a_{dst})} \in T$. A (a_{src}, a_{dst}) tuple can be either one of the following: $(a_{component}, a_{requirement})$, $(a_{device}, a_{requirement})$, $(a_{basicEvent}, a_{component})$, $(a_{basicEvent}, a_{device})$, and $\left(a_{topEvent}, a_{requirement}\right)$.

A trace link can be represented as a directed relation, with the arrow pointed to a destination artifact from a source artifact. For traceability within a group of artifacts of the same type, the link can be built natively inside the artifacts. Here, therefore, we focus on trace links between artifacts of different types. A link between a design artifact and a requirement artifact can be realized as the design implements the requirement. In the bottom of Traceability Manager, Figure 2 presents such a trace link. For FTA purposes, a trace link between a top event and a requirement artifact means that the top event (*i.e.,* a failure) *violates* the requirement. A trace link between a basic event at a leaf and a design artifact, meanwhile, indicates that the basic event is caused by the design artifact. Typically, such a link is built when developing a lower-level artifact that traces back to its origin, an upstream artifact. For example, links between requirement- and design artifacts are built during the design phase by system designers, and links from safety-analysis to requirement- or design artifacts are built by safety engineers while analyzing the system. Traceability between requirements and design objects can be built during the system-design phase, because a system designer can design the system to satisfy a given set of requirements.

4 CHANGE IMPACT ANALYSIS

Having extended traceability to safety analysis, we are now ready to discuss the SafeTrace's impact-analysis algorithms for safety-analysis artifacts. Again, SafeTrace employs FTA as its safety-analysis method. Since FTA shows the logical relations between different input events, we can leverage it to perform impact analysis, which can reveal whether a change in requirements or design will propagate to a failure at the root of the tree. In the first subsection below, we present our change models. Second, we present the necessary versions of different artifacts to update one. Then, we discuss the impact of each change in each model. For impact analysis, we first look at the impact of requirement changes on design, and then, at the impact of design changes on safety analysis. Finally, we develop an integrated view of requirement and design changes, and discuss the effects of impact analysis on safety analysis.

4.1 Change Models

For requirement and design artifacts, we define that a *change* as:

Definition 4.1. A *change* c_i made to a requirement or a design artifact a_j includes the actions Creating, Deleting, or Updating the artifact a_j, $\forall i, j, \in \mathbb{N}$

In the best case, we assume that there are no isolated artifacts. In other words, an artifact must be linked to a source artifact, a destination artifact, or both: for example, a design artifact might be linked to both a requirement and to an event in the fault tree. A trace link can be added at the time an artifact is created, though this is not required. When we add a requirement, for instance, we do not add a trace link at the time it is created, because (as previously mentioned) the building of trace links is associated with the design or development of downstream artifacts. Hence, creating a requirement may not add any trace links to the traceability graph. On the other hand, when adding a new software component to a design, we generally set up one or more trace links between that component and one or more requirements.

Given the assumption that there are no isolated artifacts, deleting a requirement artifact or a design artifact will affect the traceability graph. For example, if a software module is linked to a requirement and to safety analysis, then deleting this module from the design may lead to that requirement becoming unsupported, or the event in the fault tree not occurring. We will discuss the impact of deletion in Section 4.3 and 4.4.

A design or a requirement update might consist of editing the text description of an artifact or its attributes. Since SafeTrace only traces artifacts based on their identification numbers rather than their semantics, an update does not by itself change the relations in the traceability graph, nor does the framework detect whether the change has a positive or negative impact on its upstream or downstream artifacts. Thus, it is vital to notify potentially impacted artifacts about the update. Then, the corresponding stakeholders must check the affected artifacts. In Section 4.3 and 4.4, we will discuss change-impact analysis for requirements and design changes in terms of their relation to safety analysis.

4.2 Artifact Evolutions

After the trace links between artifacts have been established, they need to be updated as artifacts change. In the bottom of Traceability Manager in Figure 2 in Section 3 depicts evolutions and relations of different artifacts across multiple iterations. Here, an iteration is defined as one or more changes made in one of the artifacts at a time. For example, an iteration might only modify the requirements, and affect neither design nor safety analysis. Hence, each artifact might have multiple versions, as denoted in the solid boxes in Figure 2.

In the bottom of Traceability Manager in Figure 2, a directed link between two boxes in the same iteration represents the direction of a trace link from a downstream artifact to an upstream one. The subscript of each artifact represents a committed version. For instance, the top event of SafetyAnalysis$_{ver.k}$ is traced to a requirement in Requirements$_{ver.i}$, and basic events are traced to design artifacts in Design$_{ver.j}$. A dashed-line outer box, on the other hand, represents an iteration. For each new version of an artifact, we need the artifact from the previous version and the latest version upstream artifact, if available. For instance, if a new requirement is given and the design adjusted accordingly, performing SafetyAnalysis$_{ver.k+1}$ requires that we have the previous SafetyAnalysis$_{ver.k}$, the latest upstream artifacts Requirements$_{ver.i+1}$, and Design$_{ver.j+1}$. The following two subsections discusses requirement and deign change-impact analysis for artifact evolutions.

4.3 Requirement Change Impact Analysis

Assume a traceability graph with reverse links between artifacts, that is, $G' = (V, T')$. Reverse links can be built automatically by traceability-management tools when a stakeholder sets up the trace link. A requirement has a reverse direct link to the artifacts it is immediately related to: a design artifact and the root of a fault tree. In other words, a requirement has information about which design- and fault-tree artifacts trace to it. A requirement usually has direct links to at least one top event in a fault tree and at least one design artifact, so we need to examine the potential effects on both design and on safety analysis of each requirement change.

If a change creates a requirement, system designers and safety engineers need to check whether the current design- and safety analysis artifacts support the newly created requirement, and modify or create new design- or safety-analysis artifacts accordingly. If a change deletes a requirement, system designers and safety engineers need to check whether the original corresponding design- or safety-analysis artifacts have become isolated. If they have, stakeholders should consider removing them. Lastly, if a requirement change is an update to a safety-related requirement, then it is necessarily related to certain fault trees, and the stakeholders of the relevant FTA need to review it and decide whether the fault tree needs modifications due to the updated requirement. Then, if the stakeholders of a design artifact consider modifying their design because of the updated requirement, the original requirement change becomes a design change. We address design changes in the next subsection.

4.4 Design Change Impact Analysis

When creating a design artifact, system designers need to set up the links between the new artifact and certain requirements. Safety engineers must also examine whether the new artifact is covered by safety analysis.

When deleting a design artifact, if its corresponding requirements become unsupported by any design artifact, then the system designers need to check their design for whether or not other design artifacts can support the requirement. (There should be no unsupported requirements.) Likewise, safety engineers need to examine their safety analysis, since deletion of a design artifact might lead to basic events having no corresponding design artifact. In that case, safety engineers should consider removing such events from the trees and re-evaluating the relevant FTA.

The intuition of our approach to change-impact analysis of the effects of design updates on safety analysis is that, when all events in a fault-tree's MCS become True, the failure proposition of the tree becomes True. If one of the events in the MCS remains False – *e.g.,* an event caused by a safe design element with high reliability – then, state-changes of the rest of the elements in the MCS will not result in such failure. It should be borne in mind that the source of a trace link between design- and safety-analysis artifacts is a basic event of a fault tree. If a basic event is also an element of a tree's MCS, when a design artifact associated with the basic event changes, our impact-analysis algorithm simply checks whether or not the event in the associated MCSs caused the proposition of the failure to become True. For example, given the $MCS = \{\{A\}, \{B, C\}\}$, assume an upstream design artifact is linked to element A in the fault-tree. When the design artifact is updated, the algorithm will determine that event A on the fault-tree might cause the hazard to occur, because event A is the only element in the inner MCS, and if A's state becomes True, it will make the proposition of the failure become True. However, if there is more than one element in a cut set, such as $\{B, C\}$, we need to further discuss the impact of the event.

With this in mind, we present Algorithm 1, for analyzing design changes' impact on safety analysis. The goal of this algorithm, given a design-artifact change as the input, is to find all the fault trees whose failure proposition might become True as a result, along with the related requirements. This step is described in L:1 and L:2. The outputs of the algorithm are the affected fault trees and the related requirements. Affected fault tree is previously defined in Definition 3.1 in Section 3.2.

Essentially, Algorithm 1 finds fault trees whose MCSs include basic events associated with the given design artifact. If an MCS that does not have safeguard events, a change made to the design artifact can lead to the basic event becoming True, and hence trigger the failure of the tree.

First, the algorithm finds the relevant basic events in all FTAs associated with a given design artifact a_i in L:4. Second, for each basic event, the algorithm finds the MCSs that include this event from L:5 to L:7. Next, if a basic event e_j associated with the design artifact a_i in an MCS mcs_k is the only element, then the algorithm reports that the failure proposition of the fault tree might be True, and reports the requirements linked to the failure proposition of the tree to the stakeholders.

Algorithm 1: Impact Analysis Algorithm for Design Update Changes

```
1  Input: An update change on an design artifact $a_i$
2  Output:
     $\{FaultTree_\alpha \mid AffectedFaultTree_\alpha$ by $a_i, \forall\alpha, i \in \mathbb{N}\}$
3  begin
4      linkedEvents ← getLinkedEvents($a_i$)
5      for each $event_j$ in linkedEvents do
6          relatedMCSs ← getMcsBy($event_j$)
7          for each $mcs_k$ in relatedMCSs do
8              requirement $r_l$ ⟵ getRequirementInMcs($mcs_k$)
9              if $event_j$ is the only element in $mcs_k$ then
10                 report requirement $r_l$ may be violated
11                 report fault tree(s) linked by requirement $r_l$
12             else
13                 if no safe component in $mcs_k$ then
14                     report requirement $r_l$ may be violated
15                     report fault tree(s) linked by requirement $r_l$
16                 else
17                     report requirement $r_l$ may NOT be violated
```

Additionally, if there is more than one basic event in mcs_k and none of its elements is a safeguard event (*i.e.,* an event that will remain safe, causing the value to remain False in the tree), then the algorithm also reports that the failure of the fault tree might be triggered, and identifies the related requirements. On the other hand, if there is a safeguard event in mcs_k, then the change to design artifact a_i does not violate the failure proposition of the tree. This is presented in L:9 through L:17. Since the algorithm needs to examine all the MCSs related to artifact a_i, the complexity of the algorithm is then $O(n)$, where n is the number of MCSs associated with a_i.

5 CASE STUDIES

This section evaluates SafeTrace and the impact-analysis algorithms for the MD PnP laser-surgery system example via a case study, as described in Section 2. The original MD PnP system represents the second phase of system development as depicted in Figure 5, and therefore, the updated version of the system that has a communication fail-safe requirement is the third phase. We begin by introducing the case of a requirement change: adding a communication fail-safe requirement to Phase 2. Then, we show how the system design must be altered in Phase 3 to prevent communication failures that might be caused by the requirement change.

5.1 Requirement Changes

As previously mentioned, the original MD PnP system relies on a wired network. Bearing in mind SafeReq-1 and SafeReq-2, suppose that a surgical team wants a wireless-communication version, to improve the system's usability in an already crowded operating

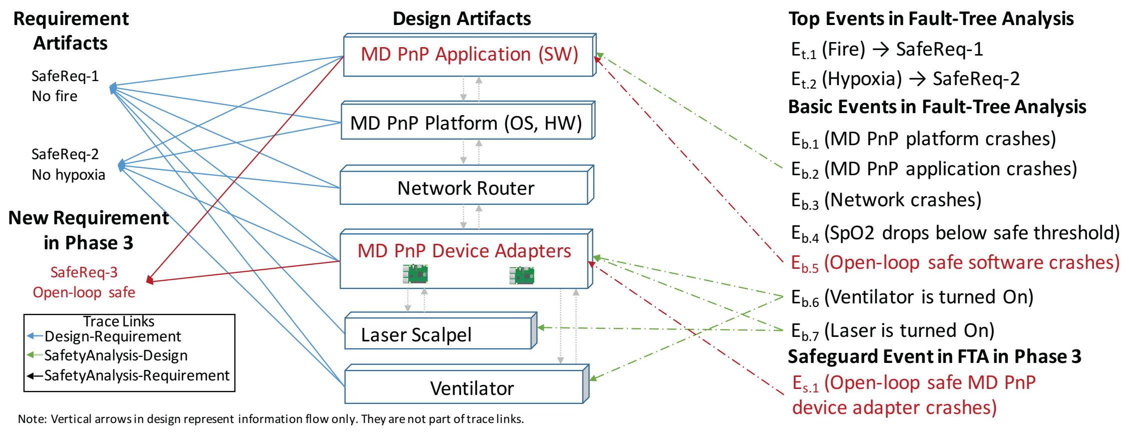

Requirement Artifacts
SafeReq-1 No fire
SafeReq-2 No hypoxia
New Requirement in Phase 3
SafeReq-3 Open-loop safe
Trace Links
Design-Requirement
SafetyAnalysis-Design
SafetyAnalysis-Requirement
Design Artifacts
MD PnP Application (SW)
MD PnP Platform (OS, HW)
Network Router
MD PnP Device Adapters
Laser Scalpel
Ventilator
Note: Vertical arrows in design represent information flow only. They are not part of trace links.
Top Events in Fault-Tree Analysis
E_{t.1} (Fire) → SafeReq-1
E_{t.2} (Hypoxia) → SafeReq-2
Basic Events in Fault-Tree Analysis
E_{b.1} (MD PnP platform crashes)
E_{b.2} (MD PnP application crashes)
E_{b.3} (Network crashes)
E_{b.4} (SpO2 drops below safe threshold)
E_{b.5} (Open-loop safe software crashes)
E_{b.6} (Ventilator is turned On)
E_{b.7} (Laser is turned On)
Safeguard Event in FTA in Phase 3
E_{s.1} (Open-loop safe MD PnP device adapter crashes)

Figure 5: Traceability Graph in Phase 2 and Phase 3. The areas in read color represent changes made in Phase 3. $E_{b.5}$ and $E_{s.1}$ are the newly added events in Phase 3. No trace links setup for uncontrollable basic events $E_{b.1}$, $E_{b.3}$, and $E_{b.4}$.

room. However, in such a scenario, the supervisor computer might lose contact with its medical client devices due to wireless signal interference, or suffer long delays in the execution of its instructions to them. In our case, this might mean the supervisor not being able to send further commands to the ventilator to re-supply oxygen, and such failure resulting in hypoxia. As pointed out by the FDA, the development process of wireless medical devices needs to guarantee their safety in the face of potential wireless-communication failures [17]. Communication failure can open the supervisor control loop, and is thus referred to as the *open-loop safe* problem. As such, a new requirement to guard against communication failure, known as SafeReq-3 or the open-loop safe requirement [20], is added.

- Safety Requirement 3 (SafeReq-3): The system shall bring the patient connected to the system to a safe state (*i.e.,* supply the patient with oxygen) without causing either fire or hypoxia if communications between the supervisor computer and medical devices fail.

In Phase 2, for both Eq. 1 and Eq. 2 as set forth in Section 2.2, we only need to trace basic events (*i.e.,* events with subscript b) and do not need to trace external ones that serve as conditions (*i.e.,* lasting events with subscript c). As such, to comply with SafeReq-3, the system designer first needs to know the system components in the second phase that may be related to communication failure, and then address the related design issues. From Figure 5, we can see that SafeReq-1, associated with hypoxia, is supported by the MD PnP Application, MD PnP Platform, Network Router, Adapter, and Ventilator. However, we only have control over the MD PnP Application, and Adapter, meaning that we need to trace events related to these artifacts.

At this point, the system designers know that they need to address the design in the MD PnP Application – *i.e.,* supervisory control computer – and the Adapter for the medical devices. (Because a device adapter only performs the commands it receives, the only difference between the adapters for the ventilator and for the laser scalpel in the real world is the interface; so, for simplicity's sake, we do not differentiate between the adapters for these two devices.) From Figure 5, we can see that $E_{b.2}$, to the MD PnP Application; and $E_{b,6}$ to the Adapter. $E_{b.2}$ appears among the minimum cuts in the first set in both $\text{Fire}_{\text{P}_2.\text{FT}_1}$ and $\text{Hypoxia}_{\text{P}_2.\text{FT}_2}$ in Eq. 1 and Eq. 2, respectively. However, because neither of these sets includes any safeguard events, if $E_{b.2}$ becomes True, then both hypoxia and fire may occur. Hence, according to L:13 in Algorithm 1, we report that both SafeReq-1 and SafeReq-2 may be violated due to the new requirement.

5.2 Design Changes

In Phase 3, the system designer modifies MD PnP Application software and Device Adapters, and sets up a trace link between it and SafeReq-3. The application software sends out the predefined timed commands to client adapters, and hence there are periods when no communication is needed, *i.e.,* after a predefined command has been received by a MD PnP Adapter.

With SafeReq-3 and the modified design artifacts in place, safety engineers can update safety analysis in Phase 3. By running Algorithm 1, they can identify the relevant events from the MCSs of each fault-tree. Given the updated requirements and the updated system design, the engineers are now able to update the safety-analysis in the previous version: specifically, to enable examination of whether the modified components satisfy the new requirement, and what its impact on the existing system is.

Figure 6 shows the updated safety analysis of fire hazard, with the fault-trees updated to take account of the open-loop safe MD PnP device adapters. The new events and trace links are colored in red and presented in Figure 5. The MCSs of the fire fault-tree in Phase 3 are shown in Eq. 4, and the events are listed in Table 1.

$$\begin{aligned}&\text{Fire}_{\text{P}_3.\text{FT}_1} = \{\{E_{s.1}, E_{b.1}, E_{b.6}, E_{C.3}\},\\ &\{E_{s.1}, E_{b.2}, E_{b.5}, E_{b.6}, E_{c.3}\}, \{E_{s.1}, E_{b.3}, E_{b.6}, E_{c.3}\},\\ &\{E_{s.1}, E_{b.1}, E_{b.7}, E_{c.2}\}, \{E_{s.1}, E_{b.2}, E_{b.5}, E_{b.7}, E_{c.2}\},\\ &\{E_{s.1}, E_{b.3}, E_{b.7}, E_{c.2}\}\}\end{aligned} \tag{4}$$

From Eq. 4, we can see that a safeguard event $E_{s.1}$ is now present in each MCS. $E_{s.1}$ is "Open-loop safe MD PnP device adapter crashes". Since event $E_{s.1}$ is caused by a open-loop safe MD PnP device adapter, the value remains False, meaning that no MCS in the $\text{Hypoxia}_{\text{P}_3.\text{FT}_2}$ will become True, even if every basic event within it becomes True.

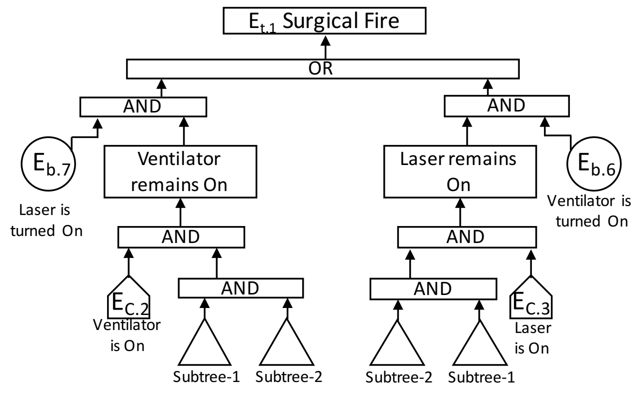
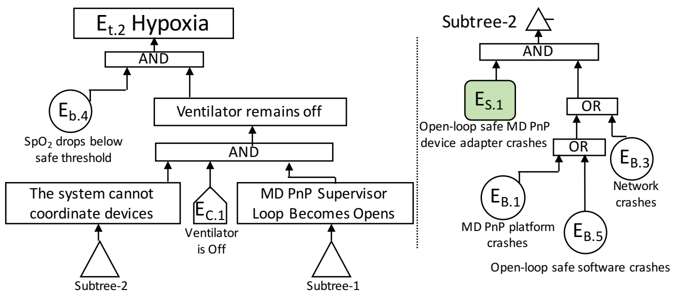


Figure 6: Fault-Tree Analysis, $\text{Fire}_{P_3.FT_1}$**, for Surgical Fire for MD PnP Open-Loop Safe Tracheotomy Systems**

Figure 7 depicts the updated fault-tree analysis for brain hypoxia. The MCSs of the Phase 3 hypoxia fault-tree are shown in Eq 5.



E_t.2 Hypoxia
AND
E_b.4
SpO₂ drops below safe threshold
Ventilator remains off
AND
The system cannot coordinate devices
E_C.1
Ventilator is Off
MD PnP Supervisor Loop Becomes Opens
Subtree-2
Subtree-1
Subtree-2
AND
E_S.1
Open-loop safe MD PnP device adapter crashes
OR
OR
E_B.3
Network crashes
E_B.1
MD PnP platform crashes
E_B.5
Open-loop safe software crashes

Figure 7: Fault-Tree Analysis, $\text{Hypoxia}_{P_3.FT_2}$**, for Hypoxia for MD PnP Open-Loop Safe Tracheotomy Systems**

$$\text{Hypoxia}_{P_3.FT_2} = \{\{E_{s.1}, E_{b.4}, E_{b.1}, E_{c.1}\},\\ \{E_{s.1}, E_{b.4}, E_{b.2}, E_{b.5}, E_{c.1}\}, \{E_{s.1}, E_{b.4}, E_{b.3}, E_{c.1}\}\} \tag{5}$$

The above medical example illustrates how SafeTrace can be used to set up trace links between different artifact types. With traceability between artifacts having been established, we demonstrated a scenario in which a new safety requirement was added; and having done so, we used the change-impact analysis algorithms described in Section 4 to demonstrate how to identify impacts, and update safety analysis and traceability, when a design is changed.

6 RELATED WORKS

Fault-tree analysis is a widely used safety analysis method to evaluate a system's capabilities against potential hazards. According to Matins and Gorschek [21], FTA is the most used safety-analysis method for handling safety requirements: accounting for 23% among the 21 methods they surveyed. Hussian and Eschbach [6] proposed a framework to automatically generate FTAs using system models. In that framework, the negation of safety constraints is treated as a fault in a fault-tree, and the combination of events then becomes the path of the fault tree to reach the root, which is a failure. However, this model-based approach requires having correct and detailed system models, which are hard to maintain over the long term; and out-of-date models will eventually make a generated fault-tree deviate from reality. Mason [7] proposed an integrated framework, MATra, which focuses on providing traceability between the records generated by CASE tools for safety-critical systems. However, it is not clear how to generalize the framework without using the CASE tools, which in practice are frequently updated and constantly evolving.

Traceability is an established tenet in the software-engineering community, and a large body of research confirms its importance in and positive impact on project development. Many regulatory agencies covering a range of industry sectors have also recognized its importance, and subsequently incorporated it into various standards and guidelines. For example, the FDA [22] mandates that traceability analysis be used to verify that the software design of a medical device implements the specified software requirements; that all aspects of the design are traceable to software requirements; and that all code is linked to established specifications and test procedures. The FAA standard DO-178C [23] specifies that at each stage of development, software developers need to demonstrate the capability of tracing designs against requirements. Research by Gries [24] also indicates that the most powerful aspect of traceability is that it provides the foundation for impact analysis, and hence provides the affected components once a change is made – even in a large-scale project like the Boeing 777's autopilot flight-director system. Rahimi *et al.* [25] proposed a framework that enables automatic updating of the links between requirements and source code across different software- and requirement versions. Hill and Victor [26] created a software safety risk taxonomy for safety-critical systems; and Kugele and Antkowiak [10] proposed a method for visualizing trace links based on component-based software development, along with an impact-analysis algorithm based on the traceability graph generated by the visualization. Guo *et al.*'s [27] automated approach to generating the rationales for each link improves upon the legacy approach, in that many links can share the same semantics and a richer expression per link. And based on an examination of numerous open-source projects, Rempel and Mader [28] found that software quality was positively affected by increases in the completeness of requirements traceability.

Although it is important to maintain requirement-to-code traceability, it is also necessary – from a safety-critical system-development perspective – to maintain trace links to safety analysis, since such analysis is mandated by certification organizations and development standards. Mader *et al.* [29] recommended that, instead of focusing on tracing all requirements, a more pragmatic approach would be to focus on creating trace links that specifically support safety-analysis feasibility. However, it is not always clear what safety-analysis methods are in use, or how to trace their relationships with requirements and design. Our work is a complement to the above-mentioned studies. Katta *et al.* [30] proposed a conceptual model of traceability for safety systems, in which the level of traceable artifacts is relative higher when specific safety-analysis methods are not referred to. We would take this line of thinking further, by elaborating safety analysis and pinning down a detailed traceability framework for FTA. Bishop and Bloomfield [31] used safety cases as traceable artifacts from different system levels, and although they mentioned FTA, the question of how to set up and manage trace links within FTA remained unclear.

Another approach to integration traceability for safety-critical systems is model-driven development. Briones *et al.*'s [4] methodology, for example, integrates safety analysis into software development – the key being to have software engineers and system analysts edit the same system models. This helps to avoid consistency issues by linking models with different versions of software. Sanchez *et al.* [32] also proposed a model-driven methodology to support safety requirements, by embedding such requirements into software development, thus rendering them traceable. Peraldi-Frati [33] proposed another method of model-driven development, focusing on timing-critical systems; but it did not include safety analysis, which is also crucial to the success of safety-critical systems development. The present research can therefore be regarded as a complement to model-driven development for traceability in life-critical systems.

7 CONCLUSION

This paper has proposed a safety-driven traceability framework, SafeTrace, for managing traceability in life-critical systems, including trace links between safety requirements, design objects, and events of safety-analysis. Specifically, our proposed method sets up trace links (1) between design artifacts and basic events in fault trees' MCSs, and (2) between requirements and the top event (*i.e.,* failure proposition) of each tree; and once such trace links have been established, the proposed impact-analysis algorithms can be used to identify the effects on safety analysis that are caused by requirement- and design changes. In the case study of an airway laser-surgery system with SafeTrace, we added a new safety requirement: that the system, once modified to a wireless version, can function fail-safe during communication failures. The results demonstrate that SafeTrace was capable of quickly and accurately locating the impacted safety-analysis areas, and of correctly updating and maintaining traceability within the system. Our approach on safety-driven traceability could be potentially apply to other application areas that safe interaction between devices is a key concern. We plan to adopt the SafeTrace framework in the development of a pediatric cardiac resuscitation system from an adult version [17, 34]. Future directions of our work include source-code-to-fault-tree traceability, and traceability on quantitative safety analysis.

ACKNOWLEDGMENT

We thank anonymous reviewers' efforts and comments that greatly improved the manuscript. This work is supported in part by NSF CNS 1329886, by NSF CNS 1545002 and in part by ONR N00014-14-1-0717.

REFERENCES

[1] Peter Jordan. Standard iec 62304-medical device software-software lifecycle processes. 2006.

[2] Julian M Goldman, Richard A Schrenker, Jennifer L Jackson, and Susan F Whitehead. Plug-and-play in the operating room of the future. *Biomedical Instrumentation & Technology*, 39(3):194–199, 2005.

[3] Jeffrey Plourde, David Arney, and Julian M Goldman. Openice: An open, interoperable platform for medical cyber-physical systems. In *Cyber-Physical Systems (ICCPS), 2014 ACM/IEEE International Conference on*, pages 221–221. IEEE, 2014.

[4] J. F. Briones, M. de Miguel, J. P. Silva, and A. Alonso. Integration of safety analysis and software development methods. In *The First Institution of Engineering and Technology International Conference on System Safety, 2006.*, pages 275–284, June 2006.

[5] I. Galvao and A. Goknil. Survey of traceability approaches in model-driven engineering. In *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, pages 313–313, Oct 2007.

[6] Tanvir Hussain and Robert Eschbach. Automated fault tree generation and risk-based testing of networked automation systems. In *Emerging Technologies and Factory Automation (ETFA), 2010 IEEE Conference on*, pages 1–8. IEEE, 2010.

[7] Paul Mason. On traceability for safety critical systems engineering. In *Software Engineering Conference, 2005. APSEC'05. 12th Asia-Pacific*, pages 8–pp. IEEE, 2005.

[8] Wen-Shing Lee, Doris L Grosh, Frank A Tillman, and Chang H Lie. Fault tree analysis, methods, and applications, a review. *IEEE transactions on reliability*, 34(3):194–203, 1985.

[9] Dean H Stamatis. *Failure mode and effect analysis: FMEA from theory to execution*. ASQ Quality Press, 2003.

[10] S. Kugele and D. Antkowiak. Visualization of trace links and change impact analysis. In *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*, pages 165–169, Sept 2016.

[11] IBM Rational DOORS, 2017.

[12] Yakindu Traceability, 2017.

[13] Intland codeBeamer, 2017.

[14] Ed Keenan, Adam Czauderna, Greg Leach, Jane Cleland-Huang, Yonghee Shin, Evan Moritz, Malcom Gethers, Denys Poshyvanyk, Jonathan Maletic, Jane Huffman Hayes, et al. Tracelab: An experimental workbench for equipping researchers to innovate, synthesize, and comparatively evaluate traceability solutions. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 1375–1378. IEEE, 2012.

[15] Jeanne M Hermens, Michael J Bennett, and Carol A Hirshman. Anesthesia for laser surgery. *Anesthesia & Analgesia*, 62(2):218–229, 1983.

[16] Lennart Edmark, Kamelia Kostova-Aherdan, Mats Enlund, and Göran Hedenstierna. Optimal oxygen concentration during induction of general anesthesia. *The Journal of the American Society of Anesthesiologists*, 98(1):28–33, 2003.

[17] A. Y. Z. Ou, Yu Jiang, P. L. Wu, L. Sha, and R. B. Berlin. Using human intellectual tasks as guidelines to systematically model medical cyber-physical systems. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 004394–004399, Oct 2016.

[18] A Hoyland and M Rausand. *System reliability theory: models, statistical methods, and applications*. NJ: Wiley-Interscience, 2004.

[19] Michael Stamatelatos, William Vesely, Joanne Dugan, Joseph Fragola, Joseph Minarick, and Jan Railsback. Fault tree handbook with aerospace applications. 2002.

[20] Andrew Y.-Z. Ou, Maryam Rahmaniheris, Yu Jiang, Po-Liang Wu, and Lui Sha. Toward safe interoperations in network connected medical cyber-physical systems using open-loop safe protocols. In *Computer-Aided Design (ICCAD), 2017 International Conference on*. IEEE, 2017.

[21] Luiz Eduardo G Martins and Tony Gorschek. Requirements engineering for safety-critical systems: A systematic literature review. *Information and Software Technology*, 75:71–89, 2016.

[22] US Food, Drug Administration, et al. Guidance for the content of premarket submissions for software contained in medical devices. *Center for Devices and Radiological Health*, 2005.

[23] RTCA (Firm). SC 167. *Software considerations in Airborne Systems and equipment certification*. RTCA, Incorporated, 1992.

[24] Michael J Gries. System engineering for the 777 autopilot system. *IEEE transactions on aerospace and electronic systems*, 33(2):649–655, 1997.

[25] Mona Rahimi, William Goss, and Jane Cleland-Huang. Evolving requirements-to-code trace links across versions of a software system. In *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on*, pages 99–109. IEEE, 2016.

[26] Janice Hill and Daniel Victor. The product engineering class in the software safety risk taxonomy for building safety-critical systems. In *Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on*, pages 617–626. IEEE, 2008.

[27] Jin Guo, Natawut Monaikul, and Jane Cleland-Huang. Trace links explained: An automated approach for generating rationales. In *Requirements Engineering Conference (RE), 2015 IEEE 23rd International*, pages 202–207. IEEE, 2015.

[28] P. Rempel and P. Mader. Preventing defects: The impact of requirements traceability completeness on software quality. *IEEE Transactions on Software Engineering*, PP(99):1–1, 2016.

[29] Patrick Mader, Paul L Jones, Yi Zhang, and Jane Cleland-Huang. Strategic traceability for safety-critical projects. *IEEE software*, 30(3):58–66, 2013.

[30] Vikash Katta and Tor Stålhane. A conceptual model of traceability for safety systems. In *Proceedings of the complex systems design & management conference*, 2011.

[31] Peter Bishop and Robin Bloomfield. A methodology for safety case development. In *Safety and Reliability*, volume 20, pages 34–42. Taylor & Francis, 2000.

[32] Pedro Sanchez, Diego Alonso, Francisca Rosique, Barbara Alvarez, and Juan A Pastor. Introducing safety requirements traceability support in model-driven development of robotic applications. *IEEE Transactions on Computers*, 60(8):1059–1071, 2011.

[33] Marie-Agnès Peraldi-Frati and Arnaud Albinet. Requirement traceability in safety critical systems. In *Proceedings of the 1st Workshop on Critical Automotive applications: Robustness & Safety*, pages 11–14. ACM, 2010.

[34] Po-Liang Wu, Min-Young Nam, Jeonghwan Choi, Alex Kirlik, Lui Sha, and Richard Berlin. Medical best practice guidance, navigation and control (GN&C) system for supporting situation awareness. Technical report, University of Illinois at Urbana Champaign, 2015.