

# Diagnosing Assumption Problems in Safety-Critical Products

Mona Rahimi  
School of Computing  
DePaul University  
Chicago IL, USA  
m.rahimi@acm.org

Wandi Xiong  
Computer Science  
Iowa State University  
Ames, IA, USA  
wdxiong@iastate.edu

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

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

**Abstract**—Problems with the correctness and completeness of environmental assumptions contribute to many accidents in safety-critical systems. The problem is exacerbated when products are modified in new releases or in new products of a product line. In such cases existing sets of environmental assumptions are often carried forward without sufficiently rigorous analysis. This paper describes a new technique that exploits the traceability required by many certifying bodies to reason about the likelihood that environmental assumptions are omitted or incorrectly retained in new products. An analysis of over 150 examples of environmental assumptions in historical systems informs the approach. In an evaluation on three safety-related product lines the approach caught all but one of the assumption-related problems. It also provided clearly defined steps for mitigating the identified issues. The contribution of the work is to arm the safety analyst with useful information for assessing the validity of environmental assumptions for a new product.

**Index Terms**—environmental assumptions, safety-critical systems, product lines, software traceability

## I. INTRODUCTION

Safety-critical systems pervade our society. In order to reduce time-to-market and development costs, families of safety-critical software systems increasingly are developed as *software product lines (SPL)*. Examples include the software in pacemaker devices, medical infusion pumps, caretaker robots, brake-assist systems, and flight-control systems.

However, SPLs also introduce new safety-related risks, such as the risk that an assumption about the operational environment of one product will be inaccurate for the operational environment of a subsequent product. A new feature, a new user, or a new adjacent system in a subsequent product may result in the need for new or modified environmental assumptions for safe operations. We define an environmental assumption to be a statement about the software systems operational context that is accepted as true by the developers [57]. This definition is consistent with common usage on projects and with dictionary definitions; however, environmental assumptions are also often referred to as contextual assumptions in both literature and practice. Finding inaccurate or missing environmental assumptions that may have safety-critical impact for a new product is currently very difficult. This difficulty obstructs the reuse of safety case elements across the products in a product line.

Teams building safety-critical software products must typically perform a rigorous hazard analysis to identify risks and

a set of mitigating, safety-related requirements. Often these requirements are associated with environmental assumptions that must hold in the planned operational context. Although it only makes sense to talk about the safety of an individual product, not of an entire product line, prior work has shown that certain safety analyses (including preliminary hazard analysis [39], and software failure modes, effects and criticality analysis (FMECA) [46]) can be performed during the domain engineering of a product line and efficiently pruned and/or extended during the application engineering of each product [17], [13]. Safety analysts often construct a formal or informal safety case providing claims, arguments, and evidence to demonstrate the safety of the product, typically to a government certification body. In practice, safety cases are produced from scratch for each product, with only the assistance of checklists [50], [20]. In this paper we focus on one aspect of reuse pertaining to the role of assumptions.

Undetected changes in the validity of environmental assumptions cause many failures of safety-critical systems [35], [25]. Detecting and avoiding the risks associated with changes to environmental assumptions poses a special problem in safety-critical product lines. New products often have new operational environments, but the documented environmental assumptions used in previous products may be inappropriately reused rather than updated in the new product. Perhaps the best known example is the Ariane 5 accident, in which an assumption made about the maximum horizontal velocity of the previous product, Ariane 4, did not hold for the subsequent Ariane 5. As a result, an overflow error occurred and both the primary and backup guidance systems failed [37], [12].

The 2007 U.S. National Research Council report on software for dependable systems highlighted the danger of inaccurate domain assumptions. It cited as an example the 1993 Airbus case in which the invalid assumption, “lack of compression always accompanies being airborne” [35] was a contributing cause to an accident. The report further stated that “construction of a dependability case might have revealed that this assumption was invalid,” noting that the dependability case “will involve reasoning about both the code and the environmental assumptions.” Dealing with the correctness and completeness of assumptions in any system, particularly an evolving or product-line one, is extremely challenging.

We therefore set a realistic research goal of flagging potential problems in a set of environmental assumptions associated with a new or modified product in a product line and of proposing practical mitigations. Our approach, which we refer to as the *Assumption Diagnostics and Rationale Process (ADRP)* leverages the trace links required by many certifying bodies for safety-critical systems [53], [47] augmented by information retrieval techniques to search for additional undocumented links [29], [5]. We exploit these links to reason about the likelihood that assumptions are missing or incorrectly retained in the current product under release. ADRP was developed through evaluating 150 documented assumptions from industrial systems, including cases which contributed to system failure. Examples are shown in Table I. We describe and evaluate ADRP using three safety-related product lines for drone deliveries, search-and-rescue missions, and environmental monitoring. Each system was developed over a six month period by a team of graduate level Software Engineering students. We address two primary research questions:

**RQ1:** To what extent can ADRP detect potential risks to the validity of environmental assumptions in a new product?

**RQ2:** To what extent can ADRP help a human analyst assess assumption-related risks and reason about mitigations?

The paper is structured as follows. Section II provides additional background about safety-critical product lines, environmental assumptions, and safety cases. Section III describes the ADRP diagnostic model and the approach taken to design it. Section IV describes the experiments and analysis performed to evaluate ADRP. Finally Sections V through VII describe threats to validity, related work, and conclusions.

## II. BACKGROUND

Many system failures arise from interactions between software and aspects of the environment in which it operates. In safety-critical systems, these failures can contribute to accidents. An accident is an unplanned event that results in death, injury, illness, loss of property or damage to surroundings or habitat [39], [55]. The environment is the broader context of the software to be developed, that is, the problem world, such as the hardware on which it runs, concurrently executing software components, physical devices and surroundings, regulatory dependencies, and user interactions [62], [58], [35].

The problem of flawed environmental assumptions is well documented. Van Lamsweerde describes its scope as, “Many reported problems originate in missing, inadequate, inaccurate or changing assumptions about the environment in which the software operates [58].” Inadequately handled changes in environmental assumptions cause many failures and, in safety-critical systems, have caused or contributed to many accidents.

A software product line (SPL) is a set of software-intensive systems that share a common set of features and are developed from a set of core assets in a prescribed way [14], [60], [51]. A product is typically generated by selecting a set of alternative and optional features (variabilities) and composing them with a set of common base features (commonalities).

TABLE I: Environmental Assumption Types with examples from historical sources. A complete list of the historical assumptions we identified from the literature is available online at <http://tinyurl.com/ASE2017AssumptionSamples>.

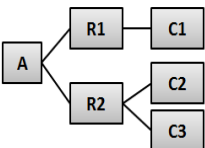
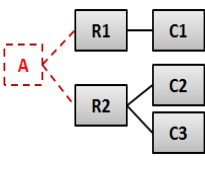
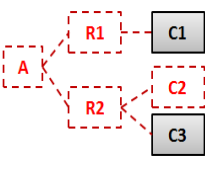
<b>Physical environment:</b> Expected to hold invariantly regardless of the system, e.g., “A train is moving iff its physical speed is non-null” [58].
<b>Operational environment:</b> Describes the operational environment surrounding the system, e.g., “There is no interference from other wireless devices in the vicinity” [45].
<b>Adjacent system:</b> Describes the behavior of adjacent systems that interact with the system being developed, e.g., “The Sensor will provide the current temperature to the Thermostat with an accuracy of $\pm 0.1^\circ F$ ” [19].
<b>User interface:</b> Describes users and their behavior, e.g., “The operator will not enter data faster than X words per minute” [39].
<b>Regulatory:</b> Describes how regulations affect the system or related components, e.g., “The device meets industrial standards for electrical safety” [7].
<b>Development process:</b> Describes policies or procedures impacting the development process and/or operation of the system, e.g., “The developer knows that transient signals should be ignored when the spacecraft lander’s legs unfold” [2].

Change in a product line occurs when new features are introduced for new products and also when individual products evolve across releases [18]. For product lines, where the operational environment and intended usage scenarios typically vary among products, changes to environmental assumptions pose an especially frequent problem [44]. In safety-critical product lines the problem of flawed environmental assumptions is complicated by the desire of developers to reuse environmental assumptions across products. Distinguishing when this reuse of assumptions is appropriate is essential to avoid hazards but is currently an open problem. Our work reported here aims to provide the safety analyst with an efficient way to check the reuse of safety-related environmental assumptions in a new product of a product line.

## III. DIAGNOSTIC MODEL

The aim of the ADRP diagnostic model is to identify potential problems with assumptions that are associated with new or modified products of a product line. For example, consider the assumptions, requirements, and classes depicted in Table II.a which show that assumption A, is associated with two Requirements *R1* and *R2* and indirectly with classes *C1*, *C2*, and *C3* in product *P1*. Table II.b shows the case in which assumption *A* is not present in product *P2*, even though all the originally associated requirements and classes from *P1* remain. The assumption could have been *correctly* removed because it is no longer needed, or *incorrectly* removed even though it is still relevant. It also could be the case that the assumption was replaced by another (possibly similar) assumption. Of these three scenarios, the second is clearly problematic as the assumption is still needed, while the third is also problematic because even though a replacement assumption is provided, its trace links to the impacted requirements are missing. In an additional example, shown in Table II.c, assumption *A*, and

TABLE II: Changes in trace dependencies across the baseline (P1) and a new/modified product (P2).

	Sample Trace Slice	Description
a		<b>P1:</b> trace slice showing dependencies of requirements R1 and R2 on assumption A, and indirect dependencies of classes C1-C3.
b		<b>P2 vs. P1:</b> Assumption A has been removed in P2 but all other artifacts dependent upon A in P1 have been retained. This may indicate that assumption A has been incorrectly removed. ADRP uses textual similarity analysis and structural analysis to search for an existing replacement assumption.
c		<b>P3 vs. P1:</b> Assumption A, requirements R1 and R2, and class C2 are missing from P2. Classes C1 and C3 are retained. ADRP checks whether C1 and C3 are service classes and/or implement other retained requirements. Depending upon this analysis ADRP flags A as a potentially missing assumption with a medium or high risk.

Artifacts and Links:   P1 only,   Both P1 and P2

much of its original downstream trace slice has been removed from product P3. Only two classes, C1 and C3 remain. It is most probable that assumption A is no longer relevant; however, further inspection is needed to validate this.

The goal of ADRP is two-fold, first to flag errors of inclusion and exclusion without raising false alarms; and secondly, to provide the analyst with the information they need in order to determine whether an assumption-related error exists or not. Two specific types of assumption-related error that are of interest are incorrectly **included** and incorrectly **excluded** assumptions. ADRP leverages knowledge of assumption use in previous product(s) as well as in the current product to identify possible problems. However, two specific sub-cases are outside the scope of ADRP’s current capabilities. First, ADRP is not capable of differentiating between correct and incorrect facts. For example, given a claim by a train manufacturer that its model X passenger train, traveling at 50 miles per hour, can stop in 200 feet under perfect conditions, we assume it to be true; however, we question the relevancy of the assumption if and when the usage context or the specific train model changes. Second, some missing assumptions may represent *unknown* or even *unknowable* unknowns [54]. These are the set of assumptions that neither appeared in a previous product nor in the domain assets of a product line and are therefore entirely outside ADRP’s knowledge base. While ADRP does not fully address these problems, it does draw attention to scenarios in which they may occur by identifying assumptions for inspection which have not been included in a previously certified product. ADRP also draws attention to new features, and to new combinations of features introduced into a new product – with the specific aim of encouraging a

systematic analysis that could identify and document relevant new assumptions for these features.

#### A. ADRP Design Methodology

We adapted Wieringa’s design science approach [61] to design ADRP. The process included (1) information gathering and analysis, (2) design, (3) initial validation and refinement, (4) user evaluation, and (5) feedback based refinement.

During the *information gathering and analysis* phase, we reviewed literature related to the use of assumptions in safety-critical systems (e.g., [58], [39], [57], [9], [7], [2], [42]) and case studies containing assumptions and lifecycle artifacts such as requirements, models, and safety cases. This allowed us to reason about the impact of assumptions across the software development lifecycle. We found evidence that different types of assumptions impacted multiple artifacts including source code and requirements. During the *design* phase, we leveraged our observations to design ADRP. First we identified properties of a software product that served as indicators of assumption-related problems and then defined metrics that enabled properties to be measured. We also identified *evidence* and *counter arguments* which might be used to support or refute the diagnosis of assumption related problems. Section III describes these metrics, evidence, and counter arguments.

In the *initial validation and refinement* phase we tested ADRP’s logic using examples from the literature (see Table I) in order to improve ADRP’s design. The design, validation, and refinement steps were repeated several times until we were satisfied that ADRP was able to identify the majority of targeted assumption errors. In the fourth phase of *external user evaluation* we conducted a more formal study with external developers as reported in Section IV. We will execute the final step of *feedback based refinement* in our ongoing work.

#### B. Product Artifacts

Certification guidelines for safety-critical systems prescribe traceability across a broad suite of artifact types [53], [28], [4], [3], [48] providing coverage for planning, analyzing, designing, implementing, verifying, validating, and assuring the quality of a system. For example, the DO-178C guidelines, adopted by the US Federal Aviation Authority, specify that trace links must be created between specific artifact pairs including software requirements, design, and source code [4]. Requirements traceability is defined as “the ability to describe and follow the life of a requirement in both a forwards and backwards direction through periods of ongoing refinement and iteration” [23].

Safety-critical projects include diverse artifacts and associated trace links; however, ADRP utilizes a common subset of artifacts referenced across certification guidelines [4], [3], [48], and described in literature discussing the role of assumptions in safety-critical systems. These artifacts, depicted in Figure 1, include product line specifications, safety assets, core implementation artifacts, and documentation. All artifacts used in this study are available at <http://tinyurl.com/ADRP-Data>.

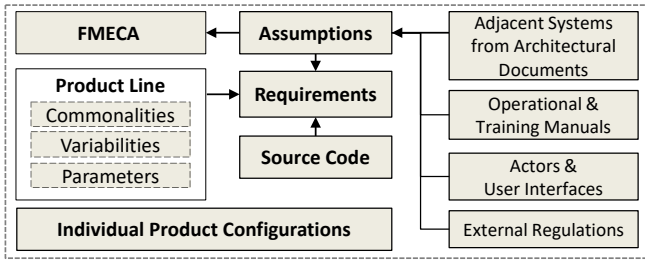


Fig. 1: Software and Safety Artifacts used by ADRP are often prescribed by certifying bodies and common across product line definitions. Arcs represent traceability paths.

**Assumptions** for a product are often documented in the product’s requirements specification document, consistent with the recommended requirements engineering practice in ISO/IEC/IEEE 29148 [34] or the older IEEE Std-830 [33].

**Requirements** describe the functionality and expected behaviour of the system while **source files** represent the implementation. A **trace slice** originates with a single artifact and shows all of its downstream dependencies. ADRP focuses on trace slices that originate with assumptions and include dependent requirements and source files.

**Product Lines** define commonalities, variabilities, dependencies, and constraints. For our study we adopted the Commonality and Variability Analysis (CVA) model [60] which documents commonalities, variabilities, parameters of variation, and a decision model to configure the individual products.

**Safety Assets** include a Failure Mode, Effects and Criticality Analysis (FMECA) generated for the entire product line and pruned/extended for each new product [17], and a safety case for each new product requiring it. Safety-related assumptions are linked to faults in the FMECA, while assumption-related problems are mapped onto safety cases for analysis purposes.

**Supplemental Documentation** includes use case specifications, architectural documents listing adjacent systems, external regulations, and procedural documents defining training and operational processes.

### C. Artifact Properties and Metrics

ADRP checks for properties in the artifacts of P1 and P2 in order to assess dependencies of artifacts upon assumption  $A$  in each product. It also identifies the delta between the two products with respect to assumption  $A$ . The properties and their associated instruments of measurement are used in ADRP’s diagnostic algorithm and summarized in Table III.

**Presence of Assumptions:** Different classes of assumption problems are possible when assumptions appear in specific combinations of P1 and P2 products. For example, an assumption that is present in P1 but not present in P2 could represent an incorrectly *excluded* assumption from P2 but not an incorrectly *included* one. Similar logic can be applied to an assumption that is present in P2. ADRP therefore defines attributes  $inP1$  and  $inP2$  to represent whether assumption  $A$  is

present in P1 and in P2 respectively.

We provide a concrete example for each of the four generic *Assumption Diagnostics (AD)* targeted by ADRP. These examples come from our catalog of assumptions that were assembled from historical failures and the literature.

**AD1 Incorrectly removed assumption:** Operational experience with Unmanned Aerial Vehicles has shown that developers need to assume that: “The UAV’s camera lens cap will, at times, be accidentally left on during flight.” Therefore, the assumption must be retained across all products, and software must accommodate the error, for example by detecting it and switching to a backup camera [45].

**AD2 Assumption missing for new feature:** In the Isolette (a hospital incubator for infants), the software was developed under the assumption that “All temperatures will be entered and displayed in degrees Fahrenheit.” [19]. However, if a new version supports Celsius, then the original assumption becomes invalidated and should be replaced by a new one [19].

**AD3 Assumption is incorrectly retained.** There was an operational environment assumption that was originally correct regarding the stopping distance of New York subway trains which was incorrectly retained when heavier trains with longer stopping distances were introduced. The false assumption contributed to several accidents [32]. This type of assumption was also the cause of the Ariane 5 rocket failure [43].

**AD4 Incorrect new assumption.** Developers of the Therac 25 radiation therapy machine introduced a new assumption that the independent protective circuits and mechanical interlocks used in the previous Therac 20 system were no longer needed as the software’s monitoring and control of the hardware was sufficient. This adjacent system assumption was false and resulted in several fatal accidents [39].

**Assumption Properties:** ADRP measures two properties directly from the assumption. First, as our interest is in safety-critical assumptions we define the boolean attribute  $SC$  (safety critical) to indicate whether an assumption has a trace link to a failure mode in its FMECA or not. Only assumptions linked to the FMECA are considered safety-related. Second, each attribute is assigned a type. As previously shown in Table I there are six commonly recognized types of assumptions. ADRP performs custom analysis for assumptions related to adjacent system assumptions, user interface assumptions, regulatory assumptions, and development process assumptions. For example, if an assumption  $A$  is related to an adjacent (external) system in P1, but that adjacent system is not used in P2, then it is unsurprising if assumption  $A$  is not included in P2. Similar arguments can be made for actors and user interfaces, applicable regulations, and training and procedural policies. Therefore, ADRP is concerned with *AssumptionType*.

**Trace Slice Metrics:** As previously depicted in Table II, the downstream artifacts which depend on an assumption, can be modeled as a trace slice. In our current version of ADRP we define a boolean metric  $REM$  (i.e. remnant) which is set to *true* if even one element of the original trace slice from P1 is retained in the new product. This overly stringent metric is

TABLE III: Properties used by the Assumption Diagnostic and Rationale Process (ADRP)

Metric	Description
SC	Assumption $A$ or any of its directly linked requirements are traced to a high criticality fault in the FMECA
inP1	Assumption $A$ exists in product $P1$ , $A \in P1$
inP2	An assumption with identical ID and identical text to assumption $A$ in $P1$ exists in product $P2$
T1	$A$ is associated with an Adjacent System
T2	$A$ is associated with a user interface
T3	$A$ is associated with an external regulatory code
T4	$A$ is associated with a development or training process
M1	Adjacent system associated with assumption $A$ is retained (as-is or modified)
M2	All features linked to actor $AC$ are retained.
M3	All features linked to regulation $R$ are retained
REM	At least one requirement or source file from the artifacts linked to $A$ in $P1$ has been retained in $P2$

TABLE IV: Properties for Diagnosing Assumption Problems.

AD1: <b>Incorrectly removed assumption</b>	
$\text{inP1} \wedge \neg \text{inP2} \wedge \dots$	
Adj	$T1 \wedge SC \wedge (M1 \vee \text{REM})$
UI	$T2 \wedge SC \wedge (M2 \vee \text{REM})$
Reg	$T3 \wedge SC \wedge (M3 \vee \text{REM})$
Dev	$T4 \wedge SC$
Other	$\neg(T1 \vee T2 \vee T3 \vee T4) \wedge SC \wedge (\text{REM})$

AD2: <b>Missing assumption for new feature</b>	
$\neg \text{inP1} \wedge \neg \text{inP2} \wedge \dots$	
All	Warning issued to analyst when new features are introduced.

AD3: <b>Incorrectly retained assumption</b>	
$\text{inP1} \wedge \text{inP2}$	
Adj	$T1 \wedge SC \wedge (\neg M1 \vee \neg \text{REM})$
UI	$T2 \wedge SC \wedge (\neg M2 \vee \neg \text{REM})$
Reg	$T3 \wedge SC \wedge (\neg M3 \vee \neg \text{REM})$
Dev	$T4 \wedge SC$
Other	$\neg(T1 \vee T2 \vee T3 \vee T4) \wedge SC \wedge (\neg \text{REM})$

AD4: <b>Incorrectly added assumption</b>	
$\neg \text{inP1} \wedge \text{inP2}$	
Adj	$T1 \wedge SC \wedge (\neg M1)$
UI	$T2 \wedge SC \wedge (\neg M2)$
Reg	$T3 \wedge SC \wedge (\neg M3)$
Dev	$T4 \wedge SC$
Other	$\neg(T1 \vee T2 \vee T3 \vee T4) \wedge SC$

used to flag potential problems, while additional information about the number and properties of remaining artifacts is used to populate the diagnostic report with information that enables the analyst to refute or confirm the diagnoses.

**Artifact Similarity:** ADRP needs to compute the similarity between assumptions, requirements, and other artifacts, for example to check whether a new assumption in  $P2$  is textually similar to one that existed in  $P1$  and therefore could potentially be serving as its substitute. Such information later will be used in the generated reports. We utilize the **Vector Space Model (VSM)** to compute the similarity between two artifacts such

as a pair of assumptions, or an assumption and a requirement. VSM is chosen because it computes quickly and has been shown to perform consistently, to handle both large and small datasets, and to treat each artifact as an unstructured bag of words. This makes it appropriate for use across various artifact types. The VSM algorithm is described in introductory information retrieval text books [10] and has been broadly used for trace retrieval purposes [29], [5].

**Source Code Analysis** ADRP performs static analysis to identify low-level service classes. For source code written in Java, C#, etc. we use standard metric analysis tools (e.g. JHAWK) to compute *afferent coupling* metrics (fan-in) of the class. Service classes are not considered as the remaining artifacts in REM metric.

#### D. The Diagnostic Algorithm

The properties used to diagnose each of the assumption errors are shown in Table IV. The table was built systematically as part of the research design process. We include subcases to differentiate between assumptions associated with adjacent systems, user interfaces, regulatory codes, development processes, as well as assumptions that do not fit any of these categories (i.e., other). Each row specifies the properties that must be true in order for that specific diagnosis to be made. For example, the properties shown in the top content row of the table specify that if an assumption is associated with an adjacent system (T1), is safety critical (SC), was included in  $P1$  but not  $P2$ , and either the adjacent system referenced by  $A$  in  $P1$  is still present in  $P2$  or remnants of the  $A$ 's previous trace slice still exist in  $P2$ , then we diagnose assumption  $A$  to be incorrectly excluded from  $P2$ . Similar logic is applied for each diagnosis. The entire process for detecting these properties, given two different versions or products, is fully automated.

#### E. Implementation

The assumption of the ADRP diagnostic algorithm is that the set of artifacts shown in Figure 1, and a set of validated trace links, exist for both products  $P1$  and  $P2$ . Trace links include links between assumptions and requirements, requirements and code, and requirements and faults. For experimental purposes we extracted all artifacts from their project environments (i.e., requirements and assumptions from Jira, code from Github, and faults from Excel), and stored them as csv formatted text files which are readable by our ADRP tool. ADRP then imported and parsed all artifacts for products  $P1$  and  $P2$ , checked them for properties defined in Table III, and then automatically generated a list of assumption diagnoses including a listing of specific artifacts that led to the underlying properties being detected. The entire process was automated except for the final step of constructing the assumption reports (e.g., Figure 2). We created these manually based upon the listing of assumption violations and their causes produced by ADRP. We plan to automate this final step in the future.

#### F. Rationales and Mitigating Steps

Once potential assumption-related problems are detected, ADRP produces a report designed to aid the safety analyst in

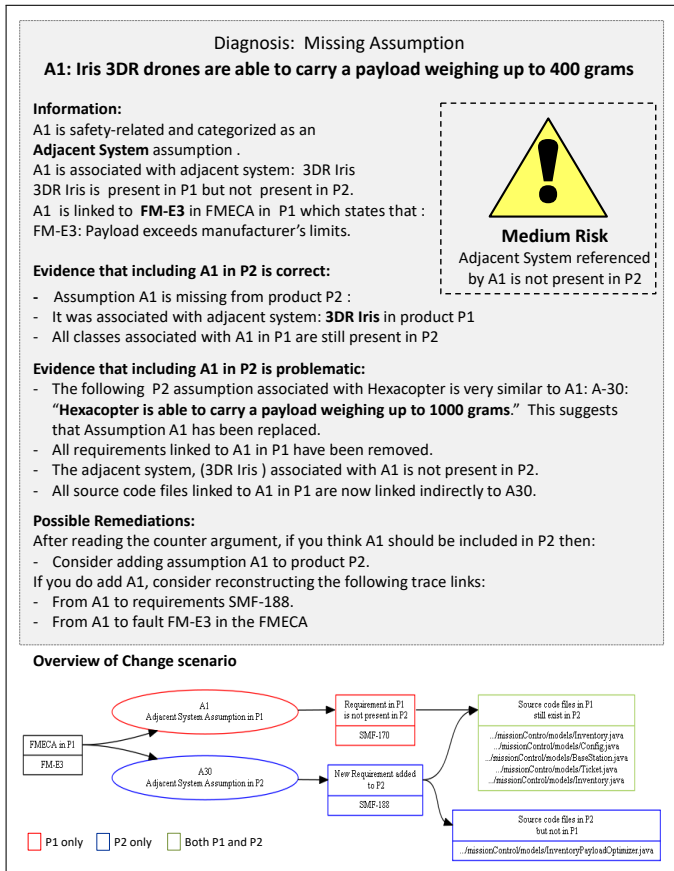


Fig. 2: Diagnosis Rationale Report generated for a low risk missing assumption. (Note: Additional data not shown here.)

determining whether an assumption-related problem actually exists. Making this determination is a non-trivial task which requires human analysis of the assumptions and all of their dependent artifacts (i.e. requirements, design, and code) in the original product (P1) and the subsequent one (P2). A report is produced for all diagnosed assumption problems. These reports are currently created manually by using the values of metrics ADRP generates. Such reports are inherently complex, as analysts must inspect diverse information sources in order to decide whether a problem diagnosis indicates a real assumption problem [22].

As depicted in Figure 2, the report currently includes four main sections: a *diagnostic summary*, *evidence* explaining the problem diagnosis, a *counter-argument* providing evidence that the observations are not indicative of a problem, and finally *suggested remediations* that could be taken to remedy the problem if the analyst concurs with the diagnosis. The report is generated using predefined templates which prescribe text to be output when diagnostic and supporting properties are found to be true. Sample reports are at <http://tinyurl.com/ADRP-Data>.

**Diagnostic Summary** The diagnostic summary is presented in a header section. It uses the properties defined in Table IV to describe the diagnosis as illustrated in Figure 2. In

this example, ADRP diagnoses a missing assumption and provides additional contextual information, for example, that the assumption is Safety-Related and associated with the 3DR Iris adjacent system.

**Evidence** The evidence sections of the report convey information associated with all detected properties. In our example, ADRP reports that all classes associated with A1 have been retained in P2 even though A1 is missing. This type of evidence supports the “Missing Assumption” diagnosis. Similarly, ADRP reports that it has found a new assumption that is textually similar to A1. The new assumption (A-30) states that Hexacopter is able to carry a payload weighing up to 1000 grams. This provides possible evidence that Assumption A1 was correctly removed and has been replaced.” The templates for generating each argument were produced as a result of our research design process.

**Warning level** Finally, each problem diagnosis is labeled as *Medium* or *High* risk. Medium-risk diagnoses are made when assumptions’ references (e.g., adjacent systems or regulations) have been removed entirely even though dependent requirements and/or source code are retained. All other cases are marked as *High Risk*.

**Visual Overview** The report displays a visual summary of the assumption’s role in P1 and P2 generated automatically by ADRP using GraphVis [21].

**Appendix** Finally, ADRP includes supplemental data to augment each of the evidence snippets. This may include relevant requirements, source code, and/or design artifacts referenced by the ADRP reports. This data appears as additional pages in the Diagnostic report and is not shown in Fig. 2.

#### IV. EVALUATION

We evaluated our two research questions against products for three different cyber physical systems (CPS) as depicted in Table V. We established the criterion that each of the projects in our evaluation would include a hazard analysis, requirements, assumptions, source code and/or detailed class-level design artifacts [53] as these are common across safety-critical products. Further, each CPS needed multiple products with artifacts for each product. Because such datasets are not currently available in the public domain, we recruited Professional Masters Students enrolled in a six-month Graduate Software Engineering Capstone course at DePaul University to build such systems. While the data sets are not from industrial projects, 90% of the graduate students were currently working full-time in the IT industry. Each CPS was developed using diverse sensors, actuators, programming languages, and frameworks, and the resulting products were non-trivial and fully executable. Metadata describing the artifacts for each product’s baseline are provided in Table VI. For example, the baseline (i.e. P1) of MedFleet, our largest data set, included 436 unique source files, 78 requirements, 24 assumptions, and over 1,725 trace links. These data sets are quite large in comparison to the 15 data sets publicly shared by the The Center of Excellence for Software Traceability (CoEST) [1]–all of which have been used extensively on numerous research

TABLE V: Products used in ADRP’s evaluation. P1 serves as a baseline, while P2-P4 represent subsequent products.

	MedFleet (MF)	Search & Rescue (SR)	Environment (ENV)
P1 Base-line	Fleet of Iris 3DR drones delivers medical supplies. Aid requests received via a mobile app. Mission control processes tickets and plans routes. Ground station manages drones in flight. Interactive map shows drone location	Manages search-and-rescue missions. Rescuers monitored via health sensors. Current location shown on real-time map. Directives transmitted to rescuers via a wrist display.	Monitors environmental pollutants using crowd-sourced mobile sensors. Data is streamed to a central server. Local pollution levels displayed on a map in a mobile app.
P2	Hexacopters replace Iris 3DR for <b>longer flights</b> and <b>heavier payloads</b> , Deliveries exceed max payload distributed across tickets.	History Logging features added. Voice commands added (for firefighters working in low visibility conditions)	Network of fixed position health sensors integrated into system
P3	Improved accuracy for identifying requester location through use of interactive map. Drone base stations elevated above tree line.	New speed monitor feature. All terrain vehicles (ATV) replace human-on-foot. Impacts range and velocity of search.	Network of fixed position health sensors integrated into system.
P4	Supplies ordered from medical clinic. Bio-hazardous materials. No mobile app.	Use different types of alerts corresponding to different situations.	A new version is released for blind users.

TABLE VI: Products used in the evaluation

Software Artifacts		Files per Product		
		MF	SR	ENV
Requirements		78	35	20
Assumptions		24	19	18
FMECA	Failures	13	18	9
Product Line (PL)	Commonalities	13	15	10
	Variabilities	12	9	12
Source Code	Java/C#	78	198	17
	Java Script	80	18	33
	XML	57	87	25
	MongoDB/Waspnote	221	0	21
Trace Links	Assump-Reqs	77	38	41
	FMECA-Reqs	18	23	11
	PL Common/(Vars)-Reqs	101	30	26
	Source Code-Reqs	1,529	1299	113

papers (e.g., [26]) but lack the diversity of artifacts needed for our study. As an additional contribution of this paper, we release the data sets and ADRP source code for replication and reuse at <http://tinyurl.com/ADRP-Data>.

#### A. RQ1: Diagnosing Assumption-Related Problems

The first study addresses RQ1: “To what extent can ADRP detect potential risks to the validity of environmental assumptions in a new product?”

1) *Study Design*: Three products, each containing a unique combination of features, were created for each CPS. All ADRP properties used to diagnose assumption problems are applicable at both the source-code and the design level. As reasoning about environmental assumptions ideally takes place when design-level decisions are being made, for experimental purposes we applied ADRP at the design-level. Product P1 therefore represents a fully-functioning, executable product (i.e. the existing baseline), whereas Products P2, P3, and P4 include requirements, assumptions, FMECA, and design-level artifacts representing classes and their operations.

As a result of creating new products, several changes were made to assumptions including removing, replacing, adding, and modifying them. For experimental evaluation purposes, we injected several assumption-related errors representative of the change-related assumption problems that we had previously collected from real-world historical accidents and accounts in

the literature. These included (AD1) incorrectly removing assumptions that should have been retained, (AD2) failing to add new assumptions that should have been added for new features, (AD3) retaining assumptions that should have been removed, and (AD4) incorrectly adding unnecessary assumptions to new products. For example, in the Search and Rescue product, the assumption that “Android hardware will have location services and cellular network turned on” was *incorrectly removed* from P2 even though location services and cellular network were still used. MedFleet’s P2 had a *missing assumption* because it replaced the 3DR Iris with a HexaCopter and introduced a corresponding new requirement that “When the total weight of payload exceeds allowed maximum, the payload must be split into two or more bundles and dispatched on separate drones,” but introduced no environmental assumption describing the maximum weight that the HexaCopter could carry. In the Environment project, P2 replaced human users carrying mobile collection sensors with fixed-position sensors. However, the assumption “Collector users have an Android device with properly working Bluetooth” was *incorrectly retained*. Any future functionality dependent upon this assumption would be problematic. A new assumption stating that “The gas sensors are not physically damaged” was *incorrectly added* to the Environment’s P3 product, even though P3 used radiation sensors and not gas sensors. ADRP also provides support for *modification errors* by treating each one as a deletion followed by an addition. As an example from the Search & Rescue product, the assumption stating that “The speed of personnel with an all-terrain vehicle will not exceed 5 m/s” was *modified* to “The speed of personnel without an all-terrain vehicle will not exceed 5 m/s”. ADRP treats this as a deletion followed by an addition and merges both actions into a single *Incorrectly Modified* diagnosis.

An answer set was systematically created for each product as follows. All safety-critical assumptions that were added, removed, or modified correctly as part of the product development process were marked as “not a problem”. All safety-critical assumptions modified as a result of the error-injection process were marked according to the type of error introduced. ADRP was then run for products P2, P3, and P4 against their respective P1 baselines. The diagnostic results were then

TABLE VII: RQ1: Results showing Actual versus Diagnosed problems across nine products

	MedFleet				Search & Rescue				Environment			
	FN	TP	TN	FP	FN	TP	TN	FP	FN	TP	TN	FP
AD1: Incorrectly Removed Assumption	0	11	4	7	0	5	0	1	0	4	4	1
AD2: Missing Assumption for new Feature	0	3	0	0	0	0	0	0	0	1	0	0
AD3: Incorrectly Retained Assumption	0	6	29	1	0	2	26	0	0	3	28	2
AD4: Incorrectly Added Assumption	1	1	4	0	0	1	5	0	0	1	6	0
Totals	1	21	37	8	0	8	31	1	0	9	38	3

**Legend**

FN: False Negative (1)  
 TP: True Positive (38)  
 TN: True Negative (106)  
 FP: False Positive (12)

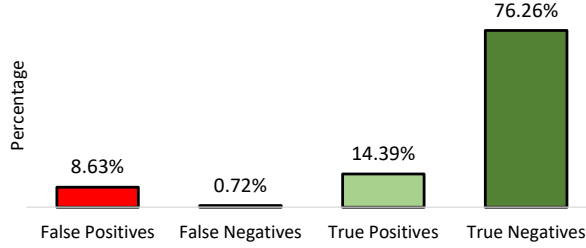


Fig. 3: RQ1: 90.65% of ADRP’s classifications were correct. There was one false negative (i.e. 0.72%). The remaining diagnoses (i.e., 8.63%) were false positives.

compared against the established answer set. Table VII shows the coverage of the four assumption diagnostics for all three products in each data set. As depicted in the table (i.e. the sum of the true positive and false negatives in each row), the data sets included 20 cases of *incorrectly removed* assumptions, 4 cases of *missing assumptions* for new functionality, 11 cases of *incorrectly retained* functions, and 4 cases of *incorrectly added* assumptions.

2) *Results*: We report results for each of the four problem diagnoses (i.e. AD1, AD2, AD3, and AD4) in Table VII. Results show that 38 out of 39 problem cases were correctly diagnosed; however, ADRP also diagnosed 13 additional cases (False Positives). There was only one case in which an actual problem was missed. This occurred for MedFleet where a new assumption was added and associated with an adjacent system. However, the assumption was not needed and was irrelevant. It was not connected via trace links to requirements or code in the product, and should have been diagnosed as incorrectly added. As a result of this missed diagnosis, and following the research-design methodology step of ‘feedback and refinement’ we have added this scenario to ADRP’s logic so that such cases will be classified as potential *incorrectly added* assumption errors in the future. We can now address RQ1 and state that in our experiment ADRP’s classification system successfully diagnosed 97% of the injected assumption errors at a precision of 75%.

**B. RQ2: User Evaluation of Diagnostic Reports**

The second study addresses RQ2: “To what extent can ADRP help a human analyst assess assumption-related risks and reason about mitigations?”

1) *Study Design*: We designed a qualitative study that approximately parallels the intended usage scenario of ADRP. The study design was inspired in part by Holzmann’s previous evaluation of a code review tool for safety-critical systems [31]. In our experiment, the user, standing in for the safety analyst, was presented with parts of the ADRP report showing

a diagnosis of an assumption-related problem, and evidence supporting and countering the diagnosis. We constructed eight individual reports for the MedFleet system. The reports contained five diagnoses of missing assumptions (3 correct and 2 false), and three diagnoses of incorrectly included assumptions (1 correct and 2 false). We established this ground truth by carefully examining each diagnosis and carefully inspecting the project artifacts and features.

Five study participants were recruited. One was a full-time software engineer with safety experience, and the other four were graduate computer science students. One of these students had played an integral role in developing the MedFleet project and assumed the role of *internal safety assessor*, while the others assumed the role of *external safety assessor*. Each session started with a 30 minute presentation on the MedFleet system, its interactions with adjacent systems, software artifacts used by ADRP, and the role of assumptions in building safety critical systems. Participants were then presented with eight ADRP reports, asked to confirm or refute the diagnoses, state their confidence in their decision, assess the quality and completeness of the information provided in the report for confirming or refuting the diagnosis, and explain their decisions. Users could also respond as *uncertain*, or claim that a *different* problem existed from the diagnosed one.

2) *Results*: Table VIII summarizes the diagnostic reports and decisions that users made with respect to the ground truth. For example, we see that three of the reports diagnosed AD1; however, only two were correct and the other was a ‘No Problem’. The columns shown under ‘Analysts’ decision’ depict the response provided by the user. As there are 5 users, the numbers in each row sum to a multiple of five (i.e.  $5 \times$  Correct Diagnosis Count). For AD1, five of the analysts agreed with the AD1 diagnosis; one said there was no problem; 2 claimed there was a different problem (which we discuss shortly); and 2 were uncertain. In the case of AD2 and AD4, all users successfully accepted the correct diagnoses and rejected the incorrect ones as not being AD2 or AD4 respectively. However, there was less agreement about the four ‘No Problem’ cases, all of which were presented as specific problems (i.e. as diagnoses of AD1-AD4). Of 20 decisions (i.e. 4 cases  $\times$  5 users), 3 were marked as AD1 errors, 1 as a AD3 error, 3 as AD4 errors, 3 as uncertain, and only half (i.e. 10) were correctly recognized as ‘No Problem’ scenarios. We now discuss two of the controversial diagnoses as they provide more insights into the strengths and weaknesses of ADRP.

In one case ADRP diagnosed a missing assumption of “A9: The drone will only operate in areas which allow for unobstructed vertical takeoff and landing.” The assumption



was in P1 but not present in a new product P2 in which drones only land on elevated base stations. The assumption had been removed because obstructions from trees and buildings were considered no longer relevant. One participant stated that there was no problem in removing the assumption, and three stated that the assumption should not have been removed. All three participants who disputed the removal of the assumption stated that even though the base station was now elevated to avoid trees, other obstacles might get in the drone’s way and that a replacement assumption (or even Assumption A9) was still needed. Their response highlights the kind of thought process that ADRP makes possible by drawing attention to changes in the system which potentially impact assumptions.

In general, we observed that raising assumption-related issues caused our analysts to not only think about whether the assumptions were correctly or incorrectly included in the product, but also to consider their correctness. For example, given a new feature that attached a locator beacon to each drone and the associated assumption “A40: The locator beacon can broadcast reliably over a distance of 400 feet” one of our participants looked up the manufacturer’s product information and discovered that the actual claim was that the beacon would broadcast over a “range of **up to 400ft (122m) in clear line of sight**”. As a result, he marked the assumption as inappropriate – even though other participants accepted it.

In 50% of the reviews, users were very confident in their decisions, while in 37% they were somewhat confident, and in 12.5% they had low confidence. In 68% of the reviews they claimed that no additional information was needed and in the remaining 32% they stated that some additional information was needed. In general, the cases where additional information was needed reflected the users’ desire to understand more about the context of the system and information about other potentially relevant assumptions. We also observed that our *internal safety assessor* took almost three times as long to complete the study as the external assessors (2 hours versus 45 minutes), inspected artifacts at a more detailed level, used external resources to check claims on adjacent systems, and was more likely to propose modifications to assumptions than the external assessors. This suggests that the supplemental information provided in the ADRP reports is most useful to safety analysts with more knowledge of the system.

We now address RQ2 based on the findings from this study. We conclude that while there was relatively strong consensus both the users and also between the users and ADRP’s diagnoses, the primary benefit from ADRP came from presenting diagnostic information to the users which they then used to reason about the assumptions in the new system.

3) *ADRP Usage Scenarios*: In Fig. 4 we illustrate a potential, practical usage scenario for ADRP, namely guiding a safety analyst to the parts of a new product or version’s safety case affected by a change to the assumptions. The safety analyst here works within an organization to incrementally construct and analyze the safety case for a new MedFleet product as the system is developed. Fig. 4 shows part of the safety case for P1, based on the Goal Structuring Notation

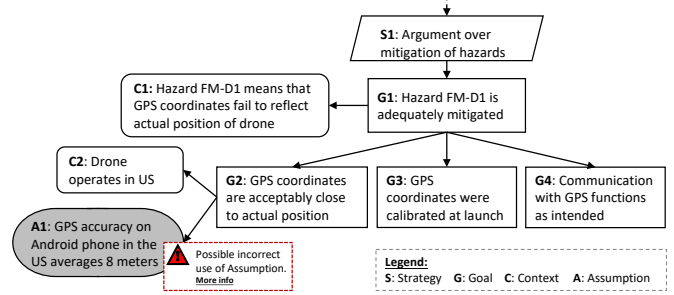


Fig. 4: Safety Case fragment showing ADRP usage scenario

(GSN) notation [27], [36], [16]. This piece of the safety case argues that the hazard FM-D1, “GPS coordinates fail to reflect actual position of drone,” is adequately mitigated when the three subgoals G2, G3, and G4 are met, given the assumption A1 and the operational context C2.

We now consider the use case in which the safety analyst is constructing the safety case for MedFleet’s new P3. P3 is similar to P1 and also will fly only in the US, but P3 has a new feature – an interactive map. If the safety case for P3 incorrectly omits the assumption A1 (shaded in gray in Fig. 4), ADRP will report that A1 *may be missing*, which guides the analyst to investigate whether A1 should be added to P3 and used in P3’s safety case. On the other hand, if we suppose that P3 is being developed for sale in a country with poorer GPS coverage than A1 asserts but that P3’s safety case mistakenly retains A1, ADRP will catch this. It will report that A1 *may be inappropriate*, which guides the analyst to consider whether A1 should be deleted from P3 and not used in P3’s safety case. In both cases ADRP could be used to map the problematic assumption to the safety-case element affected by it.

## V. THREATS TO VALIDITY

There are several potential threats to validity. First, to address the threat that ADRP would not provide sufficient coverage of assumption-related problems we conducted an in-depth study of problems that occurred in the real-world and which were reported in historical documents. ADRP was designed to detect these types of problems.

Second, the CPS products we used for our study were created by graduate students. However, the projects were non-trivially sized, used diverse programming languages, architectural frameworks and platforms, and the majority of team members were currently employed in the IT industry. Environmental assumptions were identified throughout the development process and included claims made by the manufacturers of the devices used in each application. The assumption-related problems that were injected into our products all trace back to categories of errors which we observed in the real world examples. Due to time constraints, the graduate teams were only able to deliver one viable executable project each, and so subsequent products were constructed by researchers. All of the ADRP techniques are extensible to larger systems. While we cannot claim generalizability across all types of safety-critical systems, our results suggest that ADRP can be effective

TABLE VIII: Analysts’ decisions versus actual assumption problems based on the ground truth behind each diagnosis.

Ground Truth		Number of Diagnostics		Analysts’ decision						
		Generated	Correct	AD1	AD2	AD3	AD4	NP	Diff. Prob.	Uncertain
Exclusion Error	AD1:Incorrectly Removed Assumption	3	2	5	0	0	0	1	2	2
	AD2:Missing Assumption for New Feature	2	1	0	5	0	0	0	0	0
Inclusion Error	AD3:Incorrectly Retained Assumption	0	0	0	0	0	0	0	0	0
	AD4:Incorrectly Added Assumption	3	1	0	0	0	5	0	0	0
NP: No Problem		0	4	3	0	1	3	10	0	3

for flagging assumption-related errors and can provide users with information they need to evaluate possible problems.

Third, in order to address RQ2 we engaged students as proxies for internal and external safety analysts. Only one was an original developer of the product and only one had external safety experience. The behavior of the other three is likely to differ from experts experienced in performing safety analysis. While the study provided invaluable insights into the use of ADRP to identify and describe assumption related problems, such controlled experiments cannot replace actual industrial usage. However, this type of study is a critical precursor to designing and evaluating an industrial strength solution.

Finally, the artifacts used by ADRP are likely to be present in most safety-critical systems; however, they may be modeled in different ways. For example the hazard analysis might be conducted using fault tree analysis instead of FMECA. While we do not envision any problems in integrating different styles or configurations of artifacts, we have not yet evaluated this.

## VI. RELATED WORK

While much prior work on validating environmental assumptions has been in the area of formal modeling of requirements [62], [58], it is essential to develop better techniques to support safety analysts’ work on projects without formal modeling. Recent work incorporates more information about environmental assumptions into safety cases but, as Graydon notes, the documentation of assumptions and context is still informal and has some ambiguity [24]. Handling assumptions in new or changed products is an on-going problem in practice. For example, Nair et al. found safety-related assumptions across the automotive, aviation, medical, and railway domains [49]. Weiss and Leveson presented the risks of reusing safety-critical software in a different environment [41], and Leveson described technical and organizational approaches to not violating assumptions as a safety-critical system and/or its environment change [39], [40]. Automatically flagging potential assumption problems, as we propose here, is a necessary next step. In closely related work De la Vara et al. surveyed safety analysts and found that most safety-critical systems had regular modifications requiring safety analysts to understand the impact on safety evidence [15]. This confirmed the *need for improved support in practice for updating safety artifacts when software changes*. Our work focuses on automatically detecting problematic assumptions, a historically troublesome type of safety artifacts, when change occurs.

Some recent results on the reuse of safety cases address assumptions, although that is not their focus. Most interestingly, Kokaly et al. described a model management framework for

reusing portions of an assurance case when a system’s design evolves [38]. For product line safety cases, de Oliveira et al. recorded information about the operational environments in the safety case to support reuse [16].

In regard to traceability for safety-critical systems, Briand et. al used trace slices between safety requirements and SysML design models to evaluate design conformance [8]. Hill and Tilley developed a database schema for tracing among safety artifacts [30]. Rahimi documented patterns of changes among artifacts in a safety-critical system and proposed work to automate the evolution of trace links [52]. Sanchez et al. described a traceability metamodel and techniques for model-driven development of safety-critical systems [56].

Several studies have sought to improve automated support for change management. Borg et al. conducted a series of interviews and identified the need for better tools to help maintain traceability information when software changes in complex systems [6]. Castro et al., identified the need for tools that can integrate information used by both requirements engineers and safety engineers in order to maintain traceability links among these artifacts [59]. Charrada et al. proposed tool support to highlight potentially impacted parts of a requirements specification when code changes [11]. In contrast, we focus on automating the diagnosis of potentially problematic environmental assumptions for a new or changed product.

## VII. CONCLUSION

Safety analysts have difficulty determining when an environmental assumption needed for one product or release is inappropriate or missing for a subsequent, related product or release. Such assumption problems have contributed to many accidents. The work reported here shows the benefit of automating the identification of four common risks to assumption validity in a new product. The ADRP technique described in this paper uses trace links required by certifying bodies for safety-critical systems, together with information retrieval searches for additional links, to provide the safety analyst with insights into assumptions that may be missing or incorrectly retained in the next product. Results from evaluation of ADRP show that it consistently diagnosed problematic assumptions across three safety-critical product lines and that ADRP’s results, evidence, and suggested remediations helped users correctly assess the assumptions.

## ACKNOWLEDGMENTS

The work in this paper was partially funded by the US National Science Foundation Grants CCF:1647342 and CCF:1513717.

## REFERENCES

- [1] Center of Excellence for Software and Systems Traceability. <http://coest.org>. Accessed: 2016-08-23.
- [2] *Report on the loss of the Mars Polar Lander and Deep Space 2 missions (Casani Report)*. Jet Propulsion Laboratory, California Institute of Technology, 2000.
- [3] *ECSS-E-40C: principles and requirements applicable to space software engineering*. 2009.
- [4] *RTCA/EUROCAE. DO-178C/ED-12C: Software considerations in airborne systems and equipment certification*. 2011.
- [5] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Trans. Softw. Eng.*, 28(10):970–983, 2002.
- [6] M. Borg, J. L. de la Vara, and K. Wnuk. Practitioners’ Perspectives on Change Impact Analysis for Safety-Critical Software – A Preliminary Analysis. In *SAFECOMP Workshops*, pages 346–358, 2016.
- [7] Boston Scientific. Pacemaker system specification. [http://sqlr.mcmaster.ca/\\_SQLRDocuments/PACEMAKER.pdf](http://sqlr.mcmaster.ca/_SQLRDocuments/PACEMAKER.pdf), 2007.
- [8] L. C. Briand, D. Falessi, S. Nejati, M. Sabetzadeh, and T. Yue. Traceability and SysML design slices to support safety inspections: A controlled experiment. *ACM Trans. Softw. Eng. Methodol.*, 23(1):9, 2014.
- [9] L. Brownsword and P. Clements. A case study in successful product line management. Technical report, CMU/SEI-96-TR-016. Pittsburgh, PA: SEI, Carnegie Mellon University, 1996.
- [10] C.D.Manning, P. Raghavan, and H. Shtz. *Introduction to Information Retrieval*. Cambridge University Press, 2008.
- [11] E. B. Charrada, A. Koziolok, and M. Glinz. Supporting requirements update during software evolution. *Journal of Software: Evolution and Process*, 27(3):166–194, 2015.
- [12] A. Classen. Problem-oriented feature interaction detection in software product lines. In *International Conference on Feature Interactions in Software and Communication Systems, ICFI*, pages 203–206, 2007.
- [13] J. Cleland-Huang, M. P. E. Heimdahl, J. H. Hayes, R. R. Lutz, and P. Maeder. Trace queries for safety requirements in high assurance systems. In *REFSQ*, pages 179–193, 2012.
- [14] P. C. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, 2001.
- [15] J. L. de la Vara, M. Borg, K. Wnuk, and L. Moonen. An industrial survey of safety evidence change impact analysis practice. *IEEE Trans. Software Eng.*, 42(12):1095–1117, 2016.
- [16] A. L. de Oliveira, R. T. V. Braga, P. C. Masiero, Y. Papadopoulos, I. Habli, and T. Kelly. Supporting the automated generation of modular product line safety cases. In W. Zamojski, J. Mazurkiewicz, J. Sugier, T. Walkowiak, and J. Kacprzyk, editors, *Theory and Engineering of Complex Systems and Dependability - Proceedings of the Tenth International Conference on Dependability and Complex Systems DepCoS-RELCOMEX*, volume 365 of *Advances in Intelligent Systems and Computing*, pages 319–330. Springer, 2015.
- [17] J. Dehlinger and R. R. Lutz. PLFaultCAT: A product-line software fault tree analysis tool. *Autom. Softw. Eng.*, 13(1):169–193, 2006.
- [18] T. Devine, K. Goseva-Popstojanova, S. Krishnan, and R. R. Lutz. Assessment and cross-product prediction of software product line quality: accounting for reuse across products, over multiple releases. *Automated Software Engineering*, 23(2):253–302, 2016.
- [19] DOT/FAA/AR-08/32. *Requirements Engineering Management Handbook*. 2009.
- [20] K. Fowler. *Mission-critical and safety-critical systems handbook: Design and development for embedded applications*. Newnes, 2009.
- [21] E. R. Gansner and S. C. North. An open graph visualization system and its applications to software engineering. *Software Practice and Experience*, 30(11):1203–1233, 2000.
- [22] M. Goodrum, J. Cheng, R. Metoyer, J. Cleland-Huang, and R. Lutz. What requirements knowledge do developers need to manage change in safety-critical systems? In *Requirements Engineering Conference (RE)*, To appear, 2017.
- [23] O. Gotel and C. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of the First International Conference on Requirements Engineering*, pages 94–101, April 1994.
- [24] P. J. Graydon. Towards a clearer understanding of context and its role in assurance argument confidence. In A. Bondavalli and F. D. Giandomenico, editors, *Computer Safety, Reliability, and Security - 33rd International Conference, SAFECOMP*, volume 8666 of *Lecture Notes in Computer Science*, pages 139–154. Springer, 2014.
- [25] W. S. Greenwell, E. A. Strunk, and J. C. Knight. Failure analysis and the safety-case lifecycle. In *Human Error, Safety and Systems Development, IFIP 18th World Computer Congress, TC13 / WG13.5 7th Working Conference on Human Error, Safety and Systems Development*, volume 152 of *IFIP*, pages 163–176. Kluwer/Springer, 2004.
- [26] J. Guo, M. Rahimi, J. Cleland-Huang, A. Rasin, J. H. Hayes, and M. Vierhauser. Cold-start software analytics. In *Proceedings of the 13th International Conference on Mining Software Repositories, MSR 2016, Austin, TX, USA, May 14-22, 2016*, pages 142–153, 2016.
- [27] I. Habli and T. Kelly. A safety case approach to assuring configurable architectures of safety-critical product lines. In *Architecting Critical Systems, First International Symposium, ISARCS*, pages 142–160, 2010.
- [28] J. Hatcliff, A. Wassyn, T. Kelly, C. Comar, and P. L. Jones. Certifiably safe software-dependent systems: challenges and directions. In J. D. Herbsleb and M. B. Dwyer, editors, *Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014*, pages 182–200. ACM, 2014.
- [29] 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.
- [30] J. Hill and S. Tilley. Creating safety requirements traceability for assuring and recertifying legacy safety-critical systems. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pages 297–302, 2010.
- [31] G. J. Holzmann. Mars code. *Communications of the ACM*, 57(2):64–73, 2014.
- [32] E. Hull, K. Jackson, and J. Dick. *Requirements engineering*. Springer Science & Business Media, 2010.
- [33] IEEE. IEEE Recommended Practice for Software Requirements Specifications. Institute of Electrical and Electronics Engineers, 1998.
- [34] ISO/IEC/IEEE. International standard - systems and software engineering – life cycle processes – requirements engineering. *ISO/IEC/IEEE 29148:2011(E)*, pages 1–94, 2011.
- [35] D. Jackson, M. Thomas, and L. I. Millet. In *Software for Dependable Systems: Sufficient Evidence?*, National Research Council, 2007.
- [36] J. Knight. *Fundamentals of Dependable Computing for Software Engineers*. Chapman Hall/CRC, 2011.
- [37] J. C. Knight. Safety critical systems: challenges and directions. In *Proceedings of the 24th International Conference on Software Engineering, ICSE*, pages 547–550, 2002.
- [38] S. Kokaly, R. Salay, V. Cassano, T. Maibaum, and M. Chechik. A model management approach for assurance case reuse due to system evolution. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, Saint-Malo, France, October 2-7, 2016*, pages 196–206, 2016.
- [39] N. G. Leveson. *Safeware, System Safety and Computers*. Addison Wesley, 1995.
- [40] N. G. Leveson. *Engineering a Safer World: Systems Thinking Applied to Safety*. MIT Press, 2012.
- [41] N. G. Leveson and K. A. Weiss. Making embedded software reuse practical and safe. In *SIGSOFT FSE*, pages 171–178, 2004.
- [42] G. Lewis, T. Mahatham, and L. Wrage. Assumptions management in software development. Technical Report CMU/SEI-2004-TN-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2004.
- [43] J.-L. Lions et al. Ariane 5 flight 501 failure, 1996.
- [44] R. Lutz, M. Lavin, J. Lux, K. Peters, and N. F. Rouquette. *Mining Requirements Knowledge from Operational Experience*, pages 49–73. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
- [45] 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. *Requirements Engineering*, 12(1):41–54, 2007.
- [46] R. R. Lutz and R. M. Woodhouse. Requirements analysis using forward and backward search. *Ann. Software Eng.*, 3:459–475, 1997.
- [47] P. Mäder, P. L. Jones, Y. Zhang, and J. Cleland-Huang. Strategic traceability for safety-critical projects. *IEEE Software*, 30(3):58–66, 2013.
- [48] T. members of the Task Force on Safety Critical Software. *2016:25, Licensing of safety critical software for nuclear reactors*. 2016.
- [49] S. Nair, J. L. de la Vara, M. Sabetzadeh, and L. C. Briand. An extended systematic literature review on provision of evidence for safety certification. *Information & Software Technology*, 56(7):689–717, 2014.
- [50] NASA. *Software Safety Standard*. 2013.

- [51] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.
- [52] M. Rahimi. Trace link evolution across multiple software versions in safety-critical systems. In *Proceedings of the 38th International Conference on Software Engineering, ICSE*, pages 871–874, 2016.
- [53] P. Rempel, P. Mäder, T. Kuschke, and J. Cleland-Huang. Mind the gap: assessing the conformance of software traceability to relevant guidelines. In *ICSE*, pages 943–954, 2014.
- [54] D. Rumsfeld. *Known and Unknown: A Memoir*. New York: Penguin Group, 2011.
- [55] J. Rushby. Critical system properties: Survey and taxonomy. *Reliability Engineering and System Safety*, 43(2):189–219, 1994.
- [56] P. Sánchez, D. Alonso, F. Rosique, B. Álvarez, and J. A. Pastor. Introducing safety requirements traceability support in model-driven development of robotic applications. *IEEE Trans. Computers*, 60(8):1059–1071, 2011.
- [57] T. T. Tun, R. R. Lutz, B. Nakayama, Y. Yu, D. Mathur, and B. Nuseibeh. The role of environmental assumptions in failures of DNA nanosystems. In *1st IEEE/ACM International Workshop on Complex Faults and Failures in Large Software Systems, COUFLESS*, pages 27–33, 2015.
- [58] A. van Lamsweerde. *Requirements Engineering: From System Goals to UML Models to Software Specifications*. Wiley, 2009.
- [59] J. Vilela, J. Castro, L. E. G. Martins, and T. Gorschek. Integration between requirements engineering and safety analysis: A systematic literature review. *Journal of Systems and Software*, 125:68–92, 2017.
- [60] D. Weiss and C. Lai. *Software product-line engineering: a family-based software development process*. Addison-Wesley, 1999.
- [61] R. Wieringa. *Design Science Methodology for Information Systems and Software Engineering*. Springer, 2014.
- [62] P. Zave and M. Jackson. Four dark corners of requirements engineering. *ACM Trans. Softw. Eng. Methodol.*, 6(1):1–30, 1997.