CRACKING THE BIG FREEZE!

TRACEABILITY SOLUTIONS FOR HIGH-DEPENDABILITY AGILE PROJECTS

Jane Cleland-Huang, PhD
JaneClelandHuang@nd.edu
Department of Computer Science and Engineering
University of Notre Dame

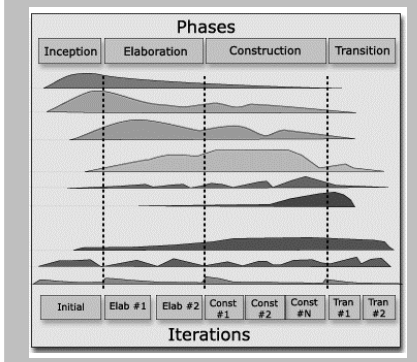
Some of the work described in this talk was funded by the US National Science Foundation under Grants CCF-0959924 and CCF-1265178.

Safety Critical Software

Software whose failure of malfunction may result in death or serious injury to people or in loss or severe damage to equipment or property.

Safety is paramount!
Innovation thwarted by the big freeze!

Natural Strengths



Traditional Planned (Waterfall/RUP)

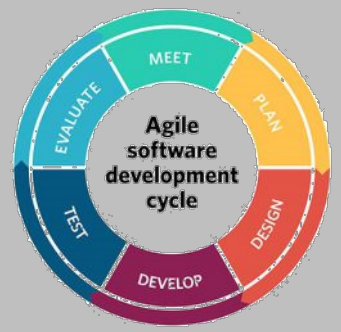
Fast paced, Incremental

Agility

Scale up

Distributed

Safety Critical



Agile

Fast paced, Incremental

Scale up

Distributed

Safety Critical

Safety Assurance

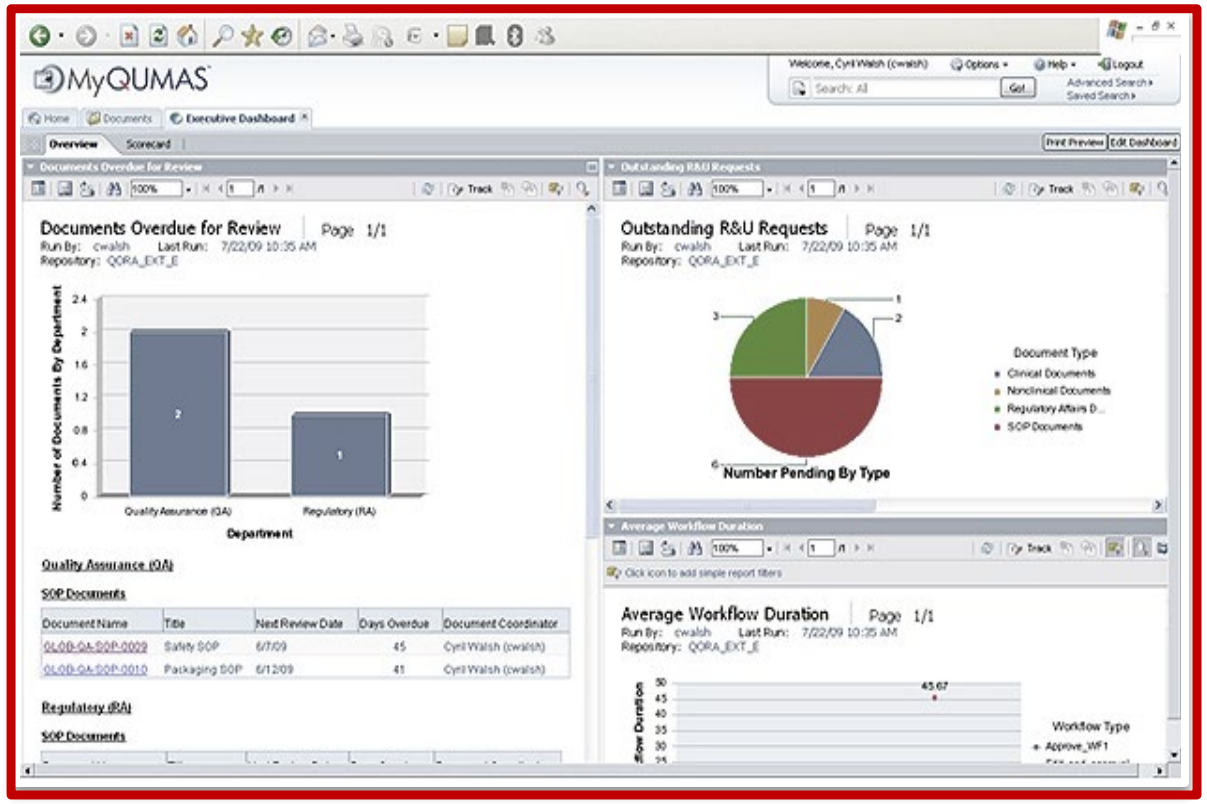
The Agile Manifesto

Individuals and **interactions** over **processes and tools**
Working software over **comprehensive documentation**
Customer collaboration over **contract negotiation**
Responding to change over **following a plan**

That is, while there is value in the items on the right, we value the items on the left more.

Processes/Tools	Documentation	Negotiate contract	Follow a plan
Ensure safety in a fast-paced environment	Auto-generate documentation to construct safety case	Demonstrate compliance to standards. Deliver safe product.	Manage the impact of change on safety.

Can we merge Agility and Safety Critical?



QUMAS

Agile process in regulated domain

Hardened to address safety issues

Regulated

Dronology

Agile development for Cyber-Physical System (CPS) with safety concerns in a regulated space (e.g., USA FAA)

Studies on Agility in Safety Critical Environments

Scaling Agile Methods to Regulated Environments: An Industry Case Study

Brian Fitzgerald*, Klaas-Jan Stol*, Ryan O'Sullivan†, and Donal O'Brien†
*Lero—The Irish Software Engineering Research Centre, University of Limerick, Ireland
†QUMAS, Cleve Business Park, Monahan Road, Cork, Ireland
bf@ul.ie, klaas-jan.stol@lero.ie, rosullivan@qumas.com, dobrien@qumas.com

Abstract*—Agile development methods are growing in popularity with a recent survey reporting that more than 80% of organizations now following an agile approach. Agile methods were seen initially as best suited to small, co-located teams developing non-critical systems. The first two constraining characteristics (small and co-located teams) have been addressed as research has emerged describing successful agile adoption involving large teams and distributed contexts. However, the applicability of agile methods for developing safety-critical systems in regulated environments has not yet been demonstrated unequivocally, and very little rigorous research exists in this area. Some of the essential characteristics of agile approaches appear to be incompatible with the constraints imposed by regulated environments. In this study we identify these tension points and illustrate through a detailed case study how an agile approach was implemented successfully in a regulated environment. Among the interesting concepts to emerge from the research are notions of *continuous compliance* and *living traceability

***Index Terms*—Agile methods, regulated environments, case study.**

Empirical Study

I. Introduction

The widespread penetration ... evidenced in a large-scale ... that 80% of respondents ... agile approach [1]. ... as best suited to (a) ... teams and (c) non-critical ... of these constraints ... have been addressed: ... published of agile adoption ... and in distributed environments ... the final constraining characteristic, ... *regulated environments*, has yet to ... In this area, there is very little rigorous evidence ... successful application of agile approaches—typically ... experience reports in workshops or practitioner reports (see [10] for a review). A lack of such evidence inhibits the adoption of agile methods in regulated environments. A number of key agile advocates have argued that agile software development methods are best suited to non-critical systems. For instance, Boehm [11] cited Scott Ambler [12] (originator of agile modeling) in stating that,"*I would be leery of applying agile modeling to life-critical systems.*"

Regulated environments, such as automotive, aviation, financial services, food, medical devices, nuclear, pharmaceutical and railway, pose particular challenges for software development as software has not been traditionally viewed as core in these sectors. Recent changes in the medical device sector, for instance, illustrate clearly the need for ... software to regulated environments, and the challenges this presents. Traditionally, medical devices ... ily hardware with perhaps some embedded software. In 2010, an amendment to the EU Medical Devices Directive classifies stand-alone software as a medical device [13]. This has significant implications ... which was traditionally ... to an end in the sector.

Agile ... are often seen as ... reason for this can ... [15] which identifies four ... for agile as:

... teractions **over** Processes and tools.
... ware **over** Comprehensive documentation.
... collaboration **over** Contract negotiation.
... ponding to change **over** Following a plan.

While the agile advocates acknowledged the statements on the right as having value, they valued the statements on the left more. However, in regulated environments the statements on the right represent values which are of particular importance. Thus, an initial assessment might conclude that agile approaches and regulated environments are incommensurable.

Agile software development methods are faced with some fundamental challenges in regulated environments. Agile processes follow an empirical logic in a plan-do-check-act (PDCA) cycle [3], whereby some development is planned and done, the results are inspected, and adaptations are made to improve the process to solve any problems that have arisen. However, in regulated environments, a *defined* logic rather than *empirical* logic is more desirable. Development processes in regulated environments are typically audited by external assessors. Thus, the granularity at which development processes are expressed and adapted requires careful tailoring in a regulated environment. Furthermore, regulated environments require rigorous traceability. In the case of requirements, for example, these need to be traced from initial requirement through to final implementation in the code-base.

This study presents an in-depth account of agile method implementation in a regulated environment at QUMAS, a leading supplier of regulatory compliance management so-

978-1-4673-3076-3/13/$31.00 © 2013 IEEE

ICSE 2013, San Francisco, CA, USA
Software Engineering in Practice

ESCAPE THE WATERFALL: AGILE FOR AEROSPACE

Steven H. VanderLeest, Calvin College and DornerWorks, Ltd., Grand Rapids, MI
Andrew Buter, DornerWorks, Ltd., Grand Rapids, MI

Abstract

Agile is an umbrella software methodology that incorporates many of the best practices of the last couple of decades. In this paper, we will examine some of those key techniques for possible application in the aerospace domain, starting with a brief literature review to identify the key Agile publications and the germane DO-178B work. Virtually all Agile practices can be mapped to a DO-178B software development process. We provide a detailed analysis of the key practices, with a preliminary assessment of the ease of implementation for each. An analysis of a number of the difficulties involving transitioning from a traditional waterfall software development process to Agile practices will show that, though difficult, a transition is possible. The transition to Agile development does not require sudden, sweeping change, but instead can be accomplished through incorporating Agile methods into an existing process. We will document successful integration of test-driven development, pair programming, refactoring, an iterative approach, and other Agile methods into a traditional DO-178B software development process. We conclude with a call for a collaborative effort to further explore Agile as an answer to the urgent need for new approaches to complex systems that have become increasingly difficult to verify and validate.

Introduction

Agile is an umbrella term that refers to a family of best-of-class iterative software development practices, methods, and techniques. Two of the better-known practices are Extreme Programming (XP) and Scrum. The iterative approach to development has been widely accepted as definitively better than traditional waterfall (staged development) approaches. Yet, the aerospace industry has been slow to adopt these proven methods.

In this paper, we will examine several of the Agile key practices and discuss their applicability for safety-critical aerospace software projects. We will also provide a survey of some pivotal projects that have implemented Agile techniques in safety-critical domains.

Literature Review

In this section, we provide a brief review of the relevant research, identifying some key Agile publications and the germane DO-178B work. We include an overview of safety-critical work in Agile, an overview of Agile software development for complex projects with multiple levels of contractors, and finally a comprehensive review of the published work on using Agile in the aerospace domain.

Key Agile Work

As context, this sub-section identifies some of the key research work on Agile development.

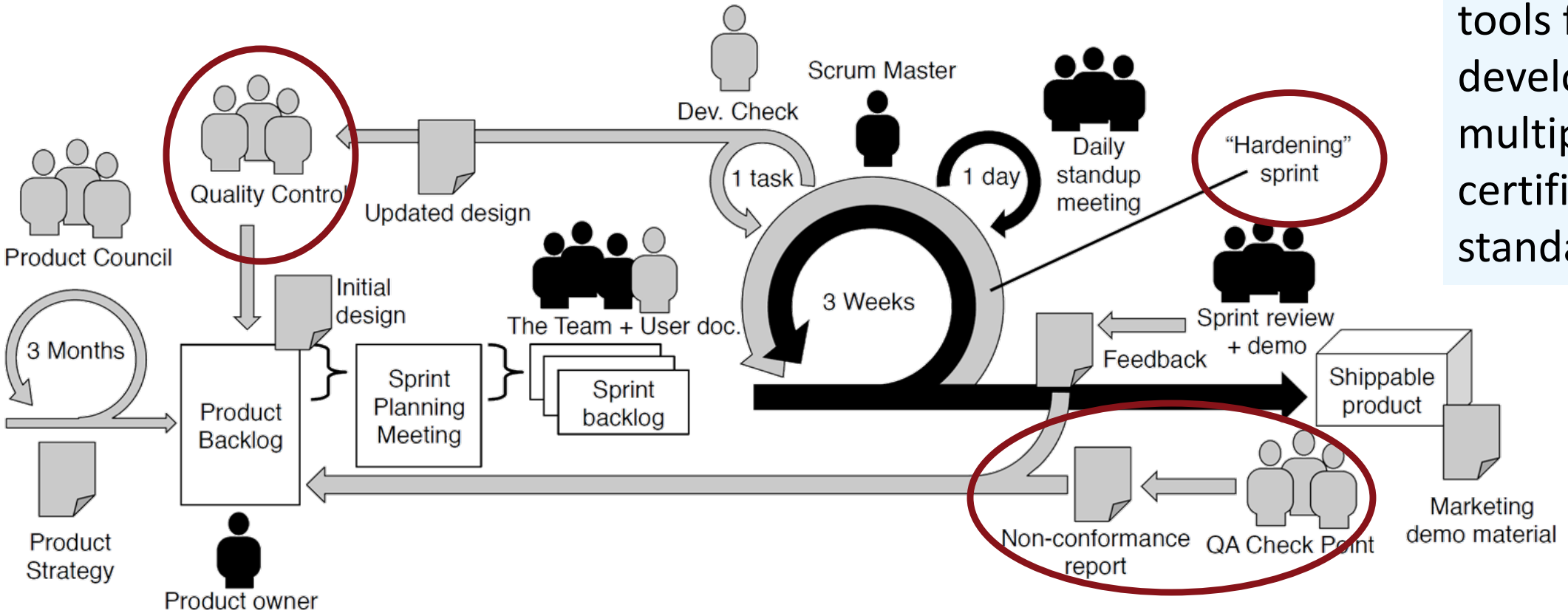
Alan MacCormack[1] argued that the stage-gate model (sequentially doing one phase of development at a time, i.e., the waterfall model) does not work well when the market or technology advances at a faster pace than the project can respond. In those cases where speed is important, an Agile approach provides a more flexible means of responding to change. MacCormack studied over 160 software projects at several major software companies. His major findings showed that success in using Agile depends on multiple, early releases to customers, rapid feedback to the development team, and a software architecture that allows independence of component teams. He also found that market uncertainty produced late changes (regardless of whether Agile or waterfall was used). Furthermore, the supposed benefits of higher productivity or lower defect rate due to a waterfall method are more myth than reality based on his data. In an earlier study [2], MacCormack found that getting low-functionality versions of the product to the customer (early release in an iterative development approach) had a dramatic impact on

28th Digital Avionics Systems Conference, October 25-29, 2009

Few empirical studies of agile processes applied in safety critical domains.

Many "thought exercises"….

QUMAS: RScrum



Builds conformance tools for systems developed under multiple certification standards.

Strong internal quality management & culture. All development sprints audited by QA.

User stories assigned risk factors, and risk managed proactively.

Living traceability has huge impact esp. on compliance.

Hardening sprint to prep system for release.

Traceability is important for Safety Critical Systems

The ability to **interrelate any uniquely identifiable** software engineering artifact to any other, **maintain** required links over time, and **use the resulting network** to answer questions of both the software product and its development process.
- CoEST Definition


FOCUS: SAFETY-CRITICAL SOFTWARE

Strategic Traceability for Safety-Critical Projects

Patrick Mäder, Ilmenau Technical University

Paul L. Jones and Yi Zhang, US Food and Drug Administration

Jane Cleland-Huang, DePaul University

An evaluation of traceability information for 10 submissions prepared by manufacturers for review at the US Food and Drug Administration identifies nine widespread traceability problems that affected regulators' ability to evaluate products safety in a timely manner.

FAILURE OF SAFETY-CRITICAL software systems to operate correctly can cause serious harm to the public—consider devices such as pacemakers, nuclear power systems, and train signals, all of which run on safety-critical software. Therefore, teams building safety-critical software products must perform rigorous risk analyses to identify potentially unsafe conditions and their contributing factors. Many projects conduct this process using techniques such as failure modes and effects analysis, fault tree analysis, and hazard and operability studies. The risk analysis produces a set of system-level requirements specifically designed to mitigate or eliminate faults and reduce the likelihood of accidents.[1] These requirements relate to a broad range of factors including training, testing, process improvements, hardware, human factors, and software design constraints.

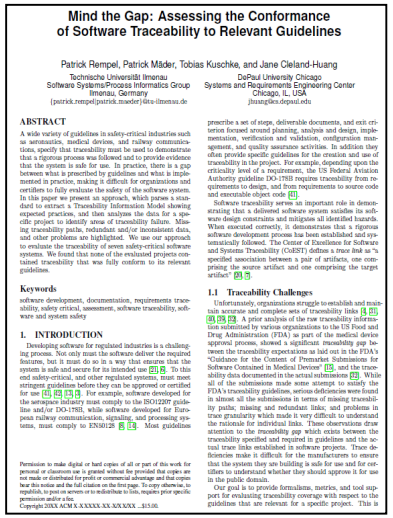
In this article, we focus on traceability's role in establishing evidence that device specifications and implementations address identified hazards and their risk control measures (see the "Traceability Standards in Safety-Critical Projects" sidebar).[2] Creating and maintaining trace links can be an arduous, error-prone, and costly process that can have a significant effect on the overall costs and time-to-market for a product.[3–5] Traceability practices, therefore, need to be strategically planned and carefully implemented to provide cost-effective support for evaluating and demonstrating a specific system's safety and security.[6] When traceability isn't implemented strategically, individual stakeholders might create traces that they personally consider to be important or attempt to provide complete trace coverage without considering how the resulting trace links will be used. A brute-force approach to traceability has been shown in practice to be difficult to implement, almost impossible to maintain, and not particularly helpful for providing evidence that a system or device is safe for its intended use.

We present six practices for strategic traceability, derived from our own observations of effective traceability in industrial projects and supported by current literature.[1–6] We also identify nine recurring problems, each of which reduces the effectiveness of traceability verification efforts and increases the difficulty experienced by regulators in evaluating product safety. All the observations in this article are based on actual observations, but the illustrative examples are either fictitious or built on obfuscated data.

Effective Practices for Tracing in Safety-Critical Projects

Although all the cases reported in this article are safety-critical in nature, many of the problems that we discuss are also applicable to software and

56 IEEE SOFTWARE | PUBLISHED BY THE IEEE COMPUTER SOCIETY


Mind the Gap: Assessing the Conformance of Software Traceability to Relevant Guidelines

Patrick Rempel, Patrick Mäder, Tobias Kuschke, and Jane Cleland-Huang

Technische Universität Ilmenau
Software Systems/Process Informatics Group
Ilmenau, Germany
{patrick.rempel|patrick.maeder|ilts-ilmenau.de

DePaul University Chicago
Systems and Requirements Engineering Center
Chicago, IL, USA
jhuanglas.depaul.edu

ABSTRACT

A wide variety of guidelines in safety-critical industries such as aerospace, medical devices, and railway communications, specify that traceability must be used to demonstrate that a rigorous process was followed and to provide evidence that the system is safe for use. In practice, there is a gap between what is prescribed by guidelines and what is implemented in practice, making it difficult for organizations and certifiers to fully evaluate the safety of the software system. In this paper we present an approach, which parses a standard to extract a Traceability Information Model showing expected practices, and then analyses the data for a specific project to identify areas of traceability failure. Missing traceability paths, redundant and/or inconsistent data, and other problems are highlighted. We use our approach to evaluate the traceability of seven safety-critical software systems. We found that none of the evaluated projects contained traceability that was fully conform to its relevant guidelines.

Keywords

software development, documentation, requirements traceability, safety critical, assessment, software traceability, software and system safety

1. INTRODUCTION

Developing software for regulated industries is a challenging process. Not only must the software deliver the required features, but it must do so in a way that ensures that the system is safe and secure for its intended use [2] [6]. To this end safety-critical, and other regulated systems, must meet stringent guidelines before they can be approved or certified for use [4] [6] [13] [7]. For example, software developed for the aerospace industry must comply to the ISO12207 guideline and/or DO-178B, while software developed for European railway communication, signaling, and processing systems, must comply to EN50128 [8] [13]. Most guidelines prescribe a set of steps, deliverable documents, and exit criterion focused around planning, analysis and design, implementation, verification and validation, configuration management, and quality assurance activities. In addition they often provide specific guidelines for the creation and use of traceability in the project. For example, depending upon the criticality level of a requirement, the US Federal Aviation Authority guideline DO-178B requires traceability from requirements to design, and from requirements to source code and executable object code [11].

Software traceability serves an important role in demonstrating that a delivered software system satisfies its software design constraints and mitigates all identified hazards. When executed correctly, it demonstrates that a rigorous software development process has been established and systematically followed. The Center of Excellence for Software and Systems Traceability (CoEST) defines a *trace link* as "a specified association between a pair of artifacts, one comprising the source artifact and one comprising the target artifact." [9] [7].

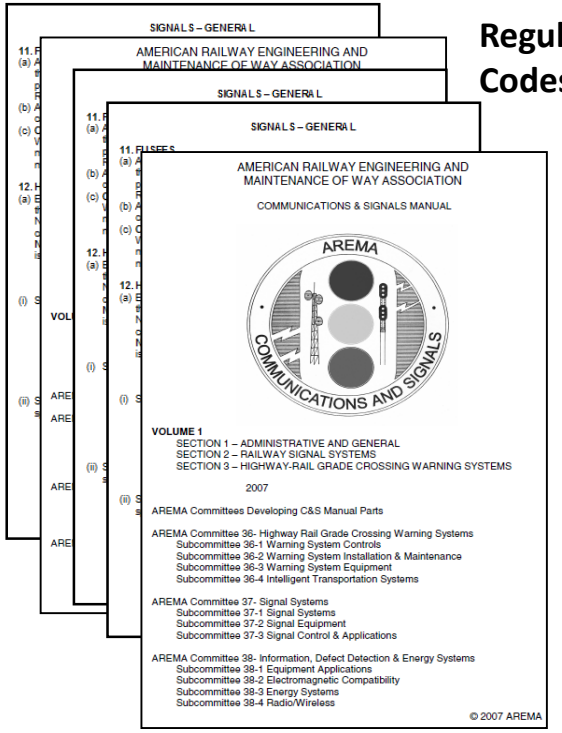
Software traceability serves an important role in demonstrating that a delivered software system satisfies its software design constraints and mitigates all identified hazards. When executed correctly, it demonstrates that a rigorous software development process has been established and systematically followed. The Center of Excellence for Software and Systems Traceability (CoEST) defines a *trace link* as "a specified association between a pair of artifacts, one comprising the source artifact and one comprising the target artifact." [9] [7].

1.1 Traceability Challenges

Unfortunately, organizations struggle to establish and maintain accurate and complete sets of traceability links [4] [5] [8] [9] [2]. A prior analysis of the raw traceability information submitted by various organizations to the US Food and Drug Administration (FDA) as part of the medical device approval process, showed a significant traceability gap between the traceability expectations as laid out in the FDA's "Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices" [15], and the traceability data documented in the actual submissions [2]. While all of the submissions made some attempt to satisfy the FDA's traceability guidelines, serious deficiencies were found in almost all the submissions in terms of missing traceability paths; missing and redundant links; and problems in trace granularity which made it very difficult to understand the rationale for individual links. These observations draw attention to the traceability gap which exists between the traceability specified and required in guidelines and the actual trace links established in software projects. Trace deficiencies make it difficult for the manufacturers to ensure that the system they are building is safe for use and for certifiers to understand whether they should approve it for use in the public domain.

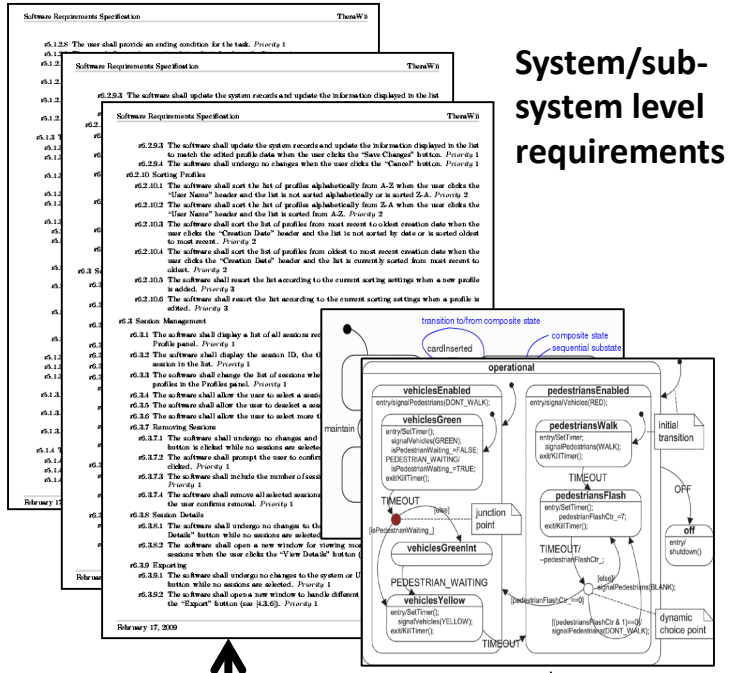
Our goal is to provide formalisms, metrics, and tool support for evaluating traceability coverage with respect to the guidelines that are relevant for a specific project. This is

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$15.00.



Regulatory Codes

Satisfies relevant codes



System/sub-system level requirements

Realizes

Validates

Code

Tests

Test cases (unit, integration, acceptance)

Required by many regulatory bodies and standards..

But hard to achieve in practice.

Accurate & Complete Traceability is challenging

Based on over a decade of traceability engagements in industrial projects we have observed a **traceability gap** between what is prescribed and what is delivered:

FOCUS: SAFETY-CRITICAL SOFTWARE

Strategic Traceability for Safety-Critical Projects

Patrick Mäder, Ilmenau Technical University

Paul L. Jones and Yi Zhang, US Food and Drug Administration

Jane Cleland-Huang, DePaul University

An evaluation of traceability information for 10 submissions prepared by manufacturers for review at the US Food and Drug Administration identifies nine widespread traceability problems that affected regulators' ability to evaluate products safety in a timely manner.

FAILURE OF SAFETY-CRITICAL software systems to operate correctly can cause serious harm to the public—consider devices such as pacemakers, nuclear power systems, and train signals, all of which run on safety-critical software. Therefore, teams building safety-critical software products must perform rigorous risk analyses to identify potentially unsafe conditions and their contributing factors. Many projects conduct this process using techniques such as failure modes and effects analysis, fault tree analysis, and hazard and operability studies. The risk analysis produces a set of system-level requirements specifically designed to mitigate or eliminate faults and reduce the likelihood of accidents.[1] These requirements relate to a broad range of factors including training, testing, process improvements, hardware, human factors, and software design constraints.

In this article, we focus on traceability's role in establishing evidence that device specifications and implementations address identified hazards and their risk control measures (see the "Traceability Standards in Safety-Critical Projects" sidebar).[2] Creating and maintaining trace links can be an arduous, error-prone, and costly process that can have a significant effect on the overall costs and time-to-market for a product.[3,4] Traceability practices, therefore, need to be strategically planned and carefully implemented to provide cost-effective support for evaluating and demonstrating a specific system's safety and security.[5] When traceability isn't implemented strategically, individual stakeholders might create traces that they personally consider to be important or attempt to provide complete trace coverage without considering how the resulting trace links will be used. A brute-force approach to traceability has been shown in practice to be difficult to implement, almost impossible to maintain, and not particularly helpful for providing evidence that a system or device is safe for its intended use.

We present six practices for strategic traceability, derived from our own observations of effective traceability in industrial projects and supported by current literature.[1–4] We also identify nine recurring problems, each of which reduces the effectiveness of traceability verification efforts and increases the difficulty experienced by regulators in evaluating product safety. All the observations in this article are based on actual observations, but the illustrative examples are either fictitious or built on obfuscated data.

Effective Practices for Tracing in Safety-Critical Projects

Although all the cases reported in this article are safety-critical in nature, many of the problems that we discuss are also applicable to software and

TRACEABILITY STANDARDS IN SAFETY-CRITICAL PROJECTS

Traceability is an established tenet in the software engineering community and is essential for assuring that software is safe for use. Many regulatory agencies of various industry sectors have recognized its importance and have subsequently incorporated it into various standards and guidelines. For example, the Federal Aviation Administration DO-178C standard specifies that at each stage of development, "software developers must be able to demonstrate traceability of designs against requirements."[1] The automotive safety standard ISO 26262:2011 dedicates an entire section to requirements management and states, for example, stating that "safety requirements shall be traceable … to each source of a safety requirement at the upper hierarchical level, each derived safety requirement at a lower hierarchical level, or to its realization in the design, and the specification of verification."[2]

The ANSI/AAMI/IEC 62304:2006 standard addresses the medical device software development life-cycle processes, requiring "traceability between system requirements, software requirements, software system test, and risk control measures implemented in the software" and that "the manufacturer shall verify and document that the software requirements are traceable to the system requirements or other source."[3] Similarly, the US Food and Drug Administration (FDA) states that traceability analysis must 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.[4] Fergal Mc Caffery and his colleagues provide a comprehensive discussion of traceability requirements for medical device software development.[5]

References

1. *DO-178C/ED-12C, Software Considerations in Airborne Systems and Equipment Certification*, RTCA, 2011.
2. *ISO 26262:2011, Road Vehicles — Functional Safety*, International Organization for Standardization, 2011.
3. *ANSI/AAMI/IEC 62304:2006, Medical Device Software — Software Life Cycle Processes*, Assoc. Advancement Medical Instrumentation, 2006.
4. *Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices*, US Food and Drug Administration, 2005.
5. F. Mc Caffery et al., "Medical Device Software Traceability," *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and Andrea Zisman, eds., Springer, 2011, pp. 321–338.

Practice 1: Plan Your Traceability

Project managers should strategically plan traceability in a project's early phases and document it using a traceability information model (TIM).[1] A TIM models the traceable artifact types (requirements, design, code, and so on) and their permitted trace links as a Unified Modeling Language (UML) class diagram. Figure 1 depicts a TIM for a safety-critical project. Artifact

FIGURE 1. A typical traceability information model (TIM) for a safety-critical system. The TIM depicts the planned trace paths among development life-cycle artifacts such as hazards, requirements, and design.

MAY/JUNE 2013 | IEEE SOFTWARE 59

Our study of Medical Device submissions to the FDA showed **incomplete and sometimes entirely missing**, inaccurate, redundant trace links – delivered as a big bang!

Mind the Gap: Assessing the Conformance of Software Traceability to Relevant Guidelines

Patrick Rempel, Patrick Mäder, Tobias Kuschke, and Jane Cleland-Huang

Technische Universität Ilmenau
Software Systems/Process Informatics Group
Ilmenau, Germany
{patrick.rempel|patrick.maeder}@tu-ilmenau.de

DePaul University Chicago
Systems and Requirements Engineering Center
Chicago, IL, USA
jhuang@cs.depaul.edu

ABSTRACT

A wide variety of guidelines in safety-critical industries such as aeronautics, medical devices, and railway communications, specify that traceability must be used to demonstrate that a rigorous process was followed and to provide evidence that the system is safe for use. In practice, there is a gap between what is prescribed by guidelines and what is implemented in practice, making it difficult for organizations and certifiers to fully evaluate the safety of the software system. In this paper we present an approach, which parses a standard to extract a 'Traceability Information Model showing expected practices, and then analyzes the data for a specific project to identify areas of traceability failure. Missing traceability paths, redundant and/or inconsistent data, and other problems are highlighted. We use our approach to evaluate the traceability of seven safety-critical software systems. We found that none of the evaluated projects contained traceability that was fully conform to its relevant guidelines.

Keywords

software development, documentation, requirements traceability, safety critical, assessment, software traceability, software and system safety

1. INTRODUCTION

Developing software for regulated industries is a challenging process. Not only must the software deliver the required features, but it must do so in a way that ensures that the system is safe and secure for its intended use [5] [6]. To this end safety-critical, and other regulated systems, must meet stringent guidelines before they can be approved or certified for use [4], [6], [7], [8]. For example, software developed for the aerospace industry must comply to the ISO/12207 guideline and/or DO-178B, while software developed for European railway communication, signaling, and processing systems, must comply to EN50128 [6] [8]. Most guidelines prescribe a set of steps, deliverable documents, and exit criterion focused around planning, analysis and design, implementation, verification and validation, configuration management, and quality assurance activities. In addition they often provide specific guidelines for the creation and use of traceability in the project. For example, depending upon the criticality level of a requirement, the US Federal Aviation Authority guideline DO-178B requires traceability from requirements to design, and from requirements to source code and executable object code [6].

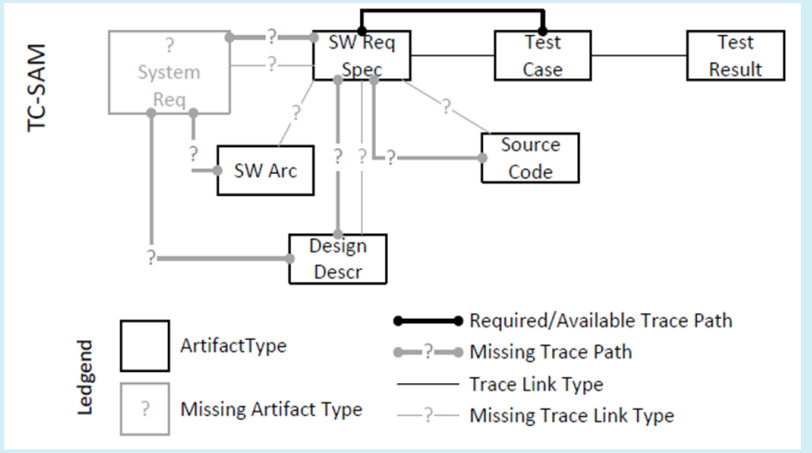
Software traceability serves an important role in demonstrating that a delivered software system satisfies its software design constraints and mitigates all identified hazards. When executed correctly, it demonstrates that a rigorous software development process has been established and systematically followed. The Center of Excellence for Software and Systems Traceability (CoEST) defines a *trace link* as "a specified association between a pair of artifacts, one comprising the source artifact and one comprising the target artifact" [9] [7].

1.1 Traceability Challenges

Unfortunately, organizations struggle to establish and maintain accurate and complete sets of traceability links [4, 5] [6], [9], [5]. A prior analysis of the raw traceability information submitted by various organizations to the US Food and Drug Administration (FDA) as part of the medical device approval process, showed a significant traceability gap between the traceability expectations as laid out in the FDA's "Guidance for the Content of Premarket Submissions for Software Contained in Medical Devices" [10], and the traceability data documented in the actual submissions [9]. While all of the submissions made some attempt to satisfy the FDA's traceability guidelines, serious deficiencies were found in almost all the submissions in terms of missing traceability paths; missing and redundant links; and problems in trace granularity which made it very difficult to understand the rationale for individual links. These observations draw attention to the traceability gap which exists between the traceability specified and required in guidelines and the actual trace links established in software projects. Trace deficiencies make it difficult for the manufacturers to ensure that the system they are building is safe for use and for certifiers to understand whether they should approve is for use in the public domain.

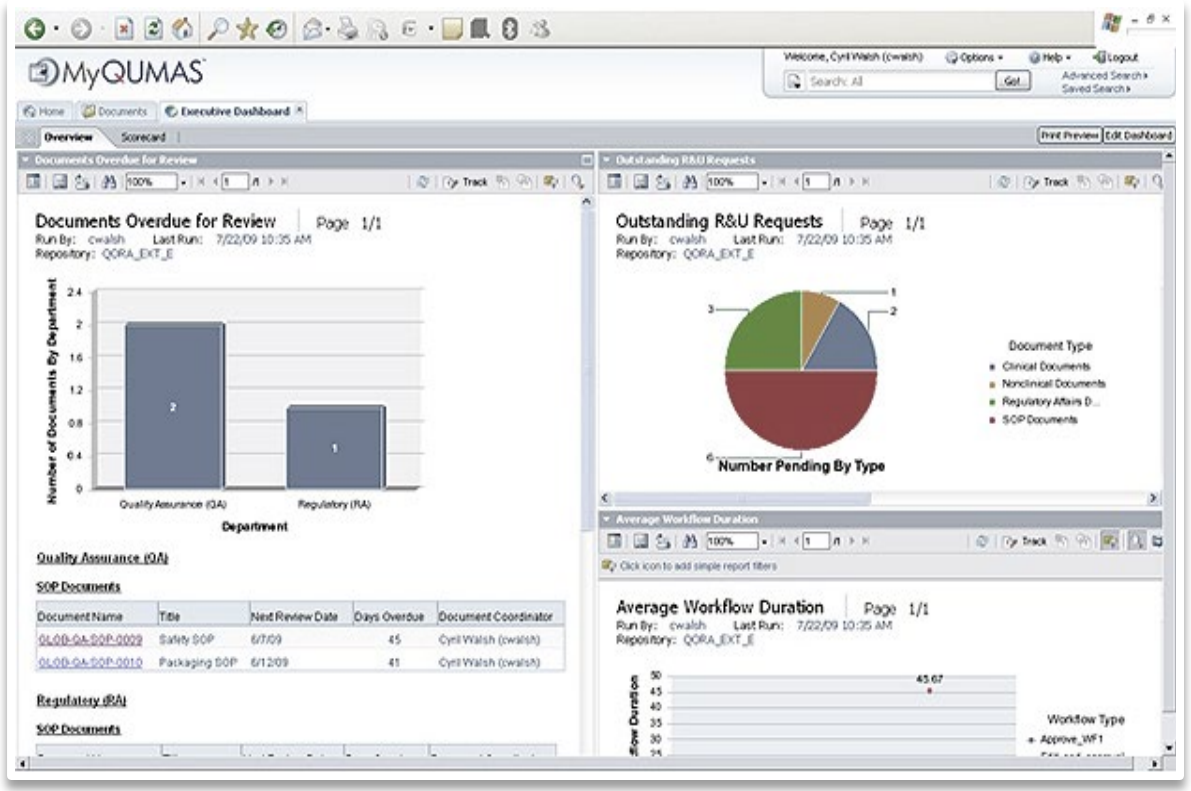
Our goal is to provide formalisms, metrics, and tool support for evaluating traceability coverage with respect to the guidelines that are relevant for a specific project. This is

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX …$15.00.



A **formal comparison of five safety-critical software systems** which claimed to follow various standards and guidelines showed **similar traceability problems**.

Example 2: Dronology



QUMAS

Agile process in regulated domain

Hardened to address safety issues

Regulated

Dronology

Agile development for Cyber-Physical System (CPS) with safety concerns in a regulated space (e.g., USA FAA)

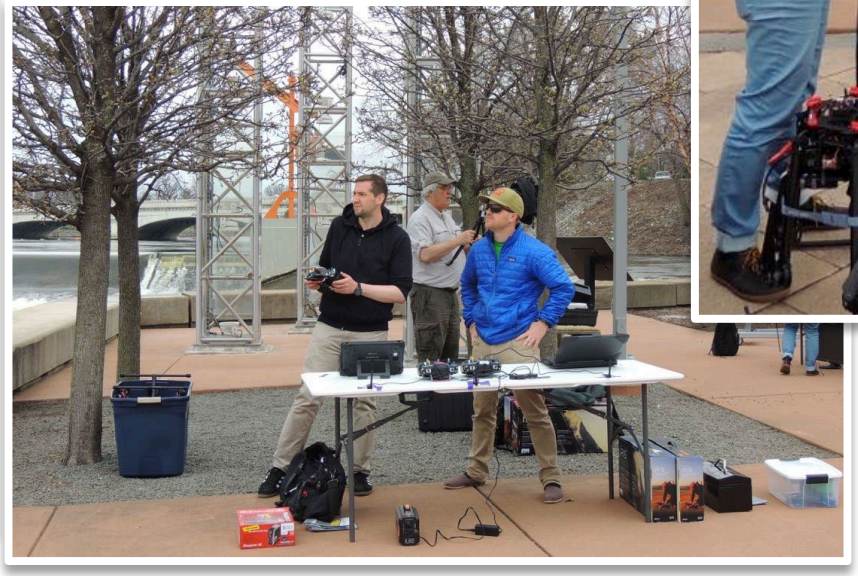
Dronology Project @ Notre Dame

http://sarec.nd.edu/pages/Dronology.html

A platform for coordinating the flight of UAVs. Supports research in safety assurance, runtime monitoring, & adaptation.

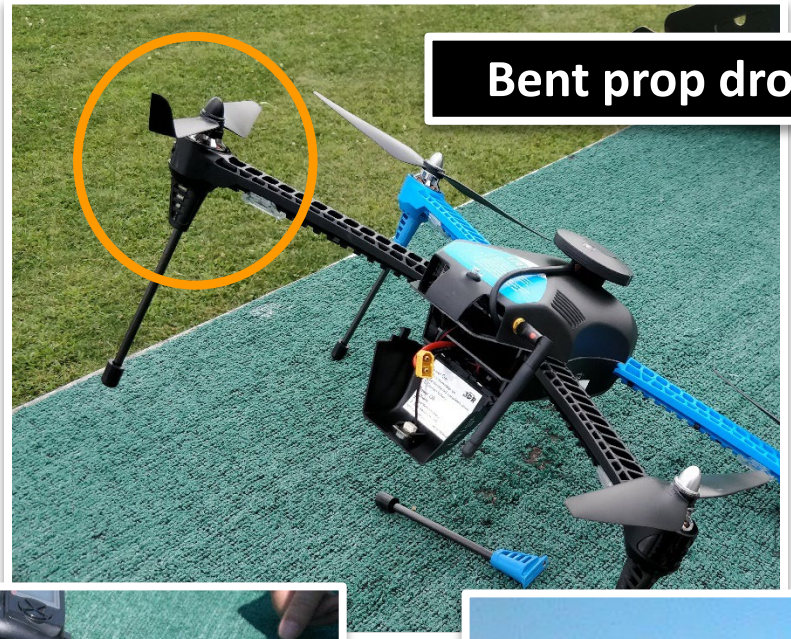
Developed by the Notre Dame Team: Michael Vierhauser, Jane Wyngaard, Jinghui Cheng, Sean Bayley, Greg Madey, Joshua Huseman, Jane Cleland-Huang, & more…

River Rescue Demo with Dronology

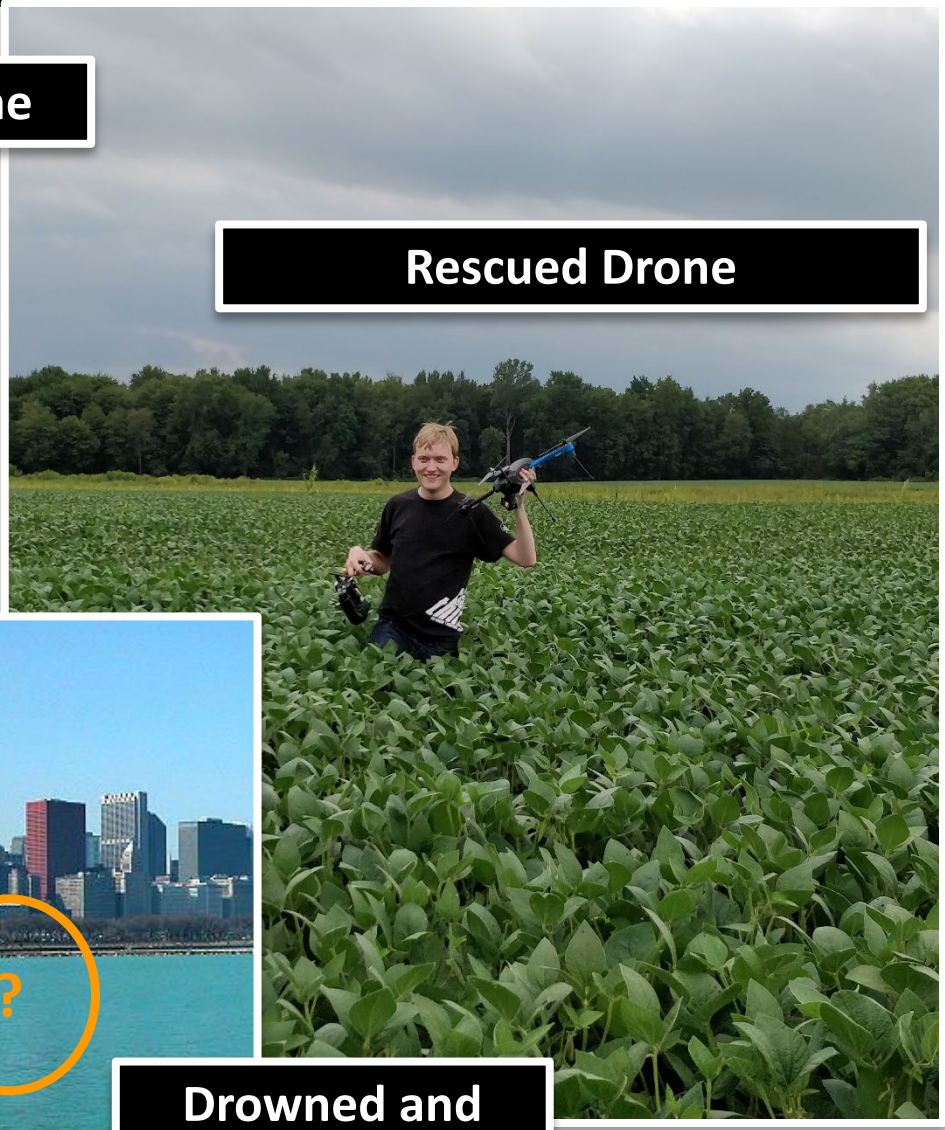


Drone technology is imperfect

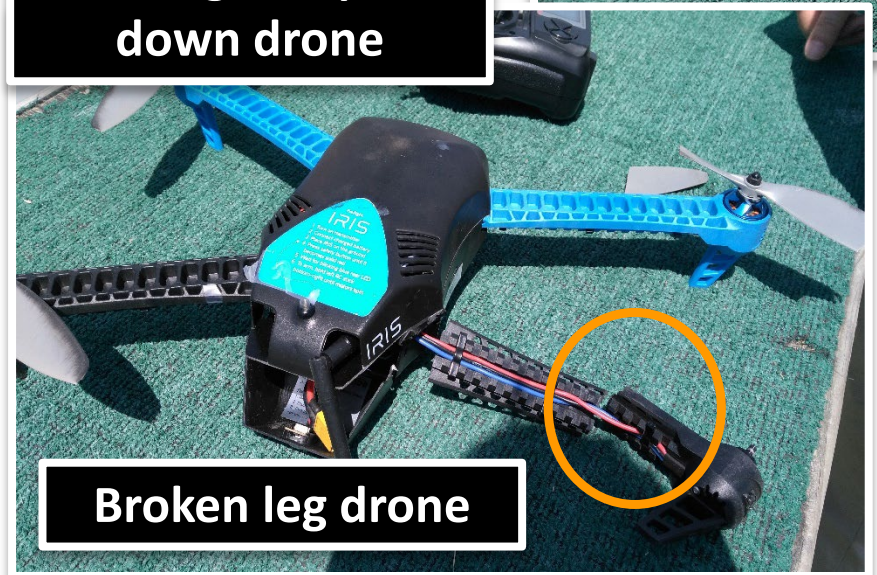
Trajectory challenged, upside down drone



Bent prop drone



Rescued Drone



Broken leg drone

Drowned and missing drone

Testing an AED drop

Drones delivering medical supplies must fly far beyond line of sight, circum-vent obstacles and changing terrain, fly over urban areas, & deliver heavy payloads in potentially populous regions.

Agility and Safety Critical Process Converged in Dronology

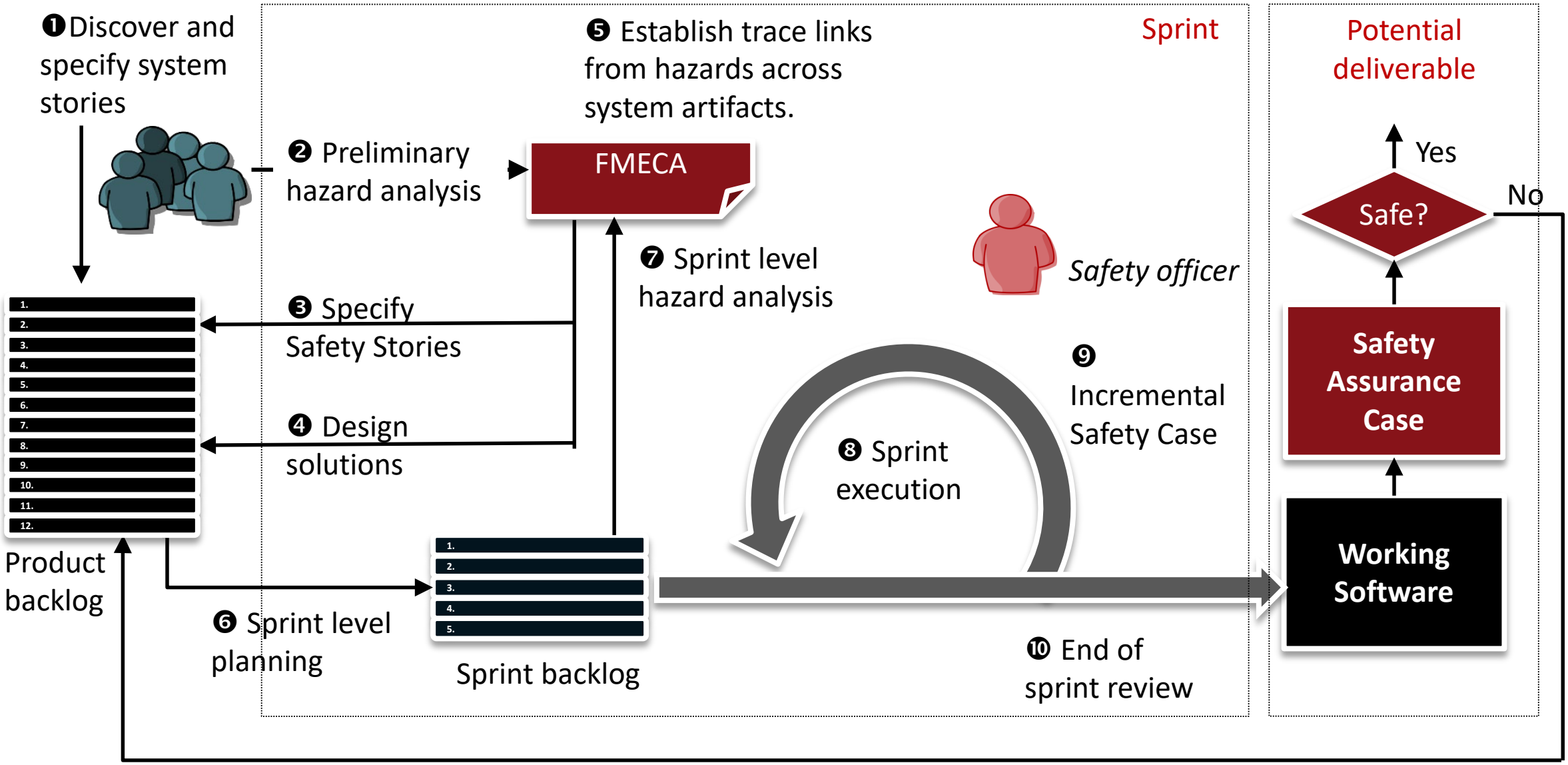
❶ SCRUM provided insufficient support for systematic safety.

❷ Many safety concerns emerged alongside the functionality. Surveys supported this.

❸ Building an arguably safe system (i.e., through a safety case), was incredibly time consuming.

❹ Instrumenting and tooling the agile environment was essential to deliver living, ubiquitous traceability.

Safety Scrum (SScrum)



A1: Discover and Specify System Stories

Work with project stakeholders to elicit and specify an initial set of safety stories, representing the functional requirements of the project.

Ubiquitous

The *<component name>* shall *<response>*

The drone shall maintain a *minimum-separation distance* at all times.

Event Driven

When *<trigger>* the *<system name>* shall *<system response>*

When the drone is within X centimeters of minimum separation distance from another drone, the collision avoidance system shall provide directives to all drones in the vicinity.

State Driven

While *<in a specific state>* the *<system name>* shall *<system response>*

While in landing mode the drone shall descend vertically until it reaches the ground.

Option

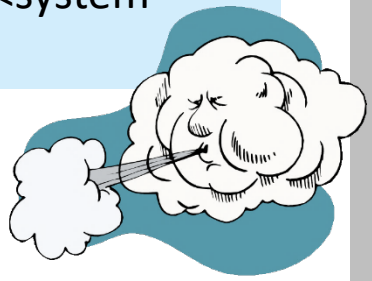
Where <feature is included> the <system name> shall <system response>

Where parachute mode is enabled and a drop is initiated the drone shall scan the dropzone for obstacles.

Unwanted Behavior

If <optional preconditions> <trigger>, then the <system name> shall <system response>

If wind gusts exceed desired wind velocity but are below the maximum wind velocity, the drone shall return to base.



A2: Perform Preliminary Hazard Analysis

Conduct a hazard analysis in early phases of the project to identify potential hazards and failure modes in order to drive activities such as architecture design, story specification, safety analysis, and rigorous testing throughout the remainder of the project.

Preliminary Hazard:
UAV's battery fails unexpectedly and the UAV falls to the ground.

❶ The battery level **detector** on the UAV fails to detect a low battery level.

❷ The battery level **indicator** on the UAV reports incorrect battery level.

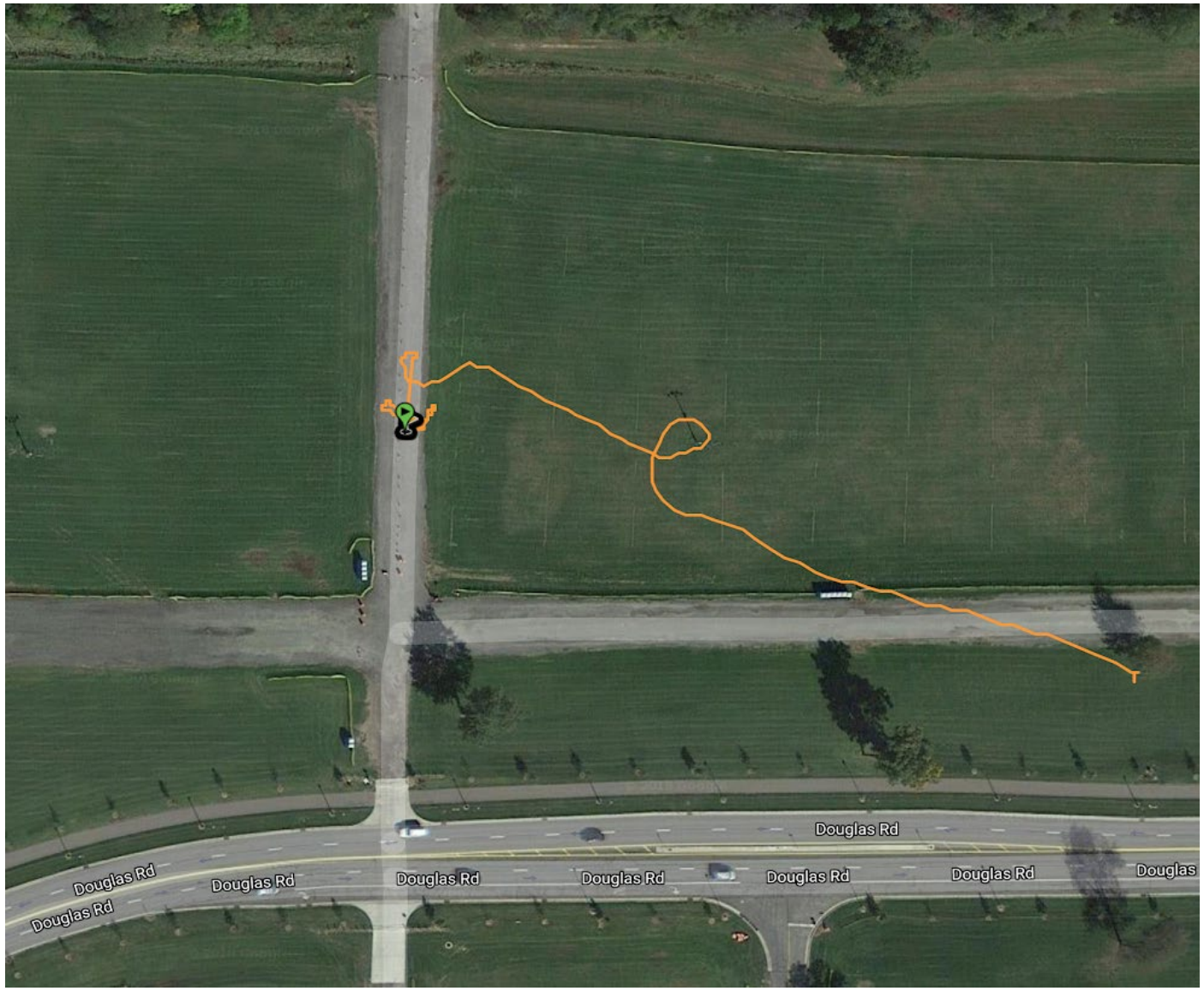
❸ Dronology software fails to check battery level in time to take responsive actions before the battery fails.

❹ Dronology software is aware of low battery warning but fails to take responsive actions.

A3: Specify Safety Stories

Identify, analyze, and specify safety stories that, if satisfied, will prevent the hazard from occurring or reduce the impact of its occurrence. Place safety stories into the product backlog.

Safety Story (SAF-1):
The GPS coordinates of each UAV must be accurate within one meter at all times.



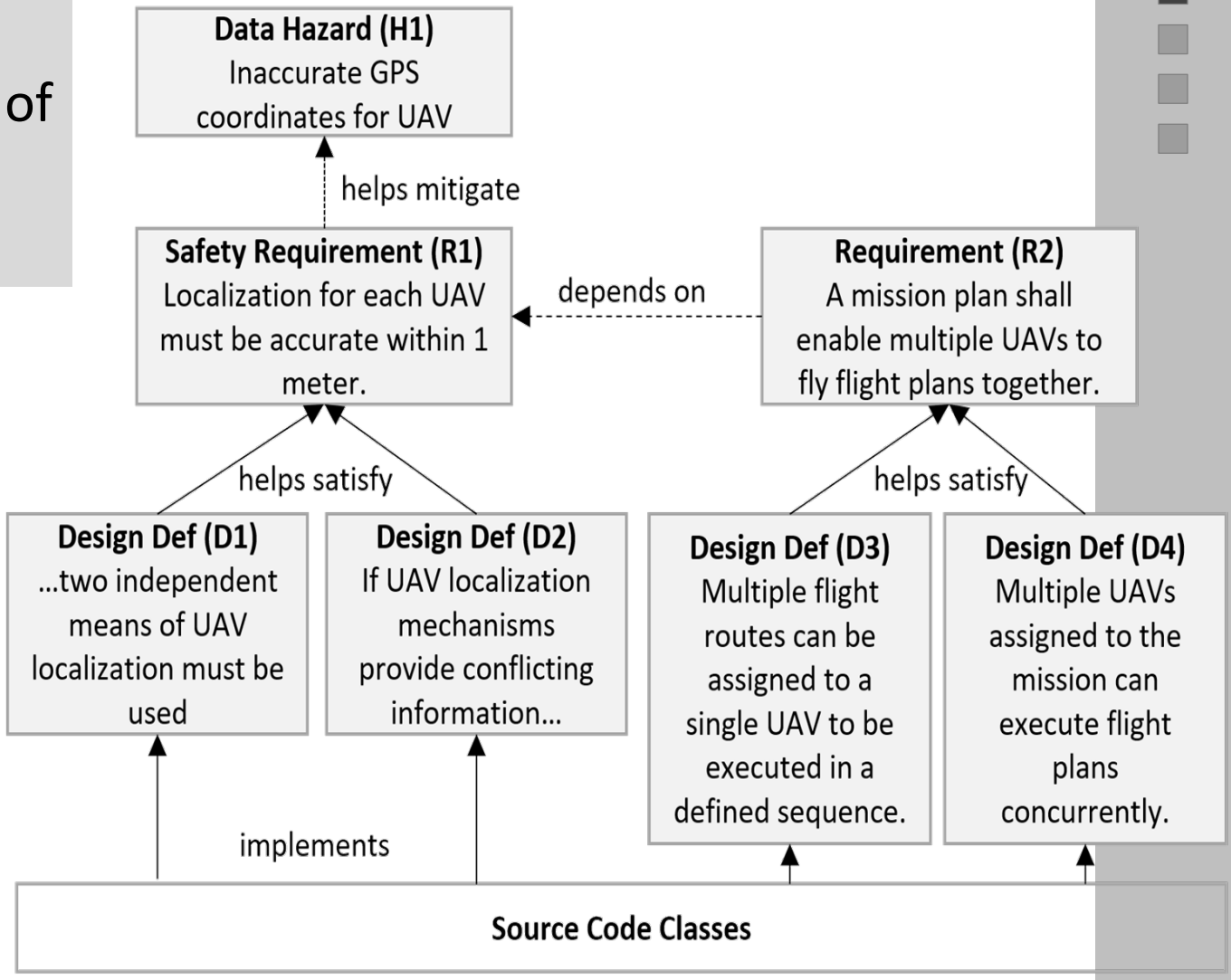
A4: Design Safety Solutions

Design a solution to address each of the safety stories. Specify the solution as a set of design definitions prior to scheduling its associated safety-stories into a sprint.

(DD-1): When Dronology is deployed in an urban environment at least two independent means of UAV localization must be used.

(DD-2): If UAV localization mechanisms provide conflicting information, the lowest reported distance between two UAVs shall serve as their current separation distance.

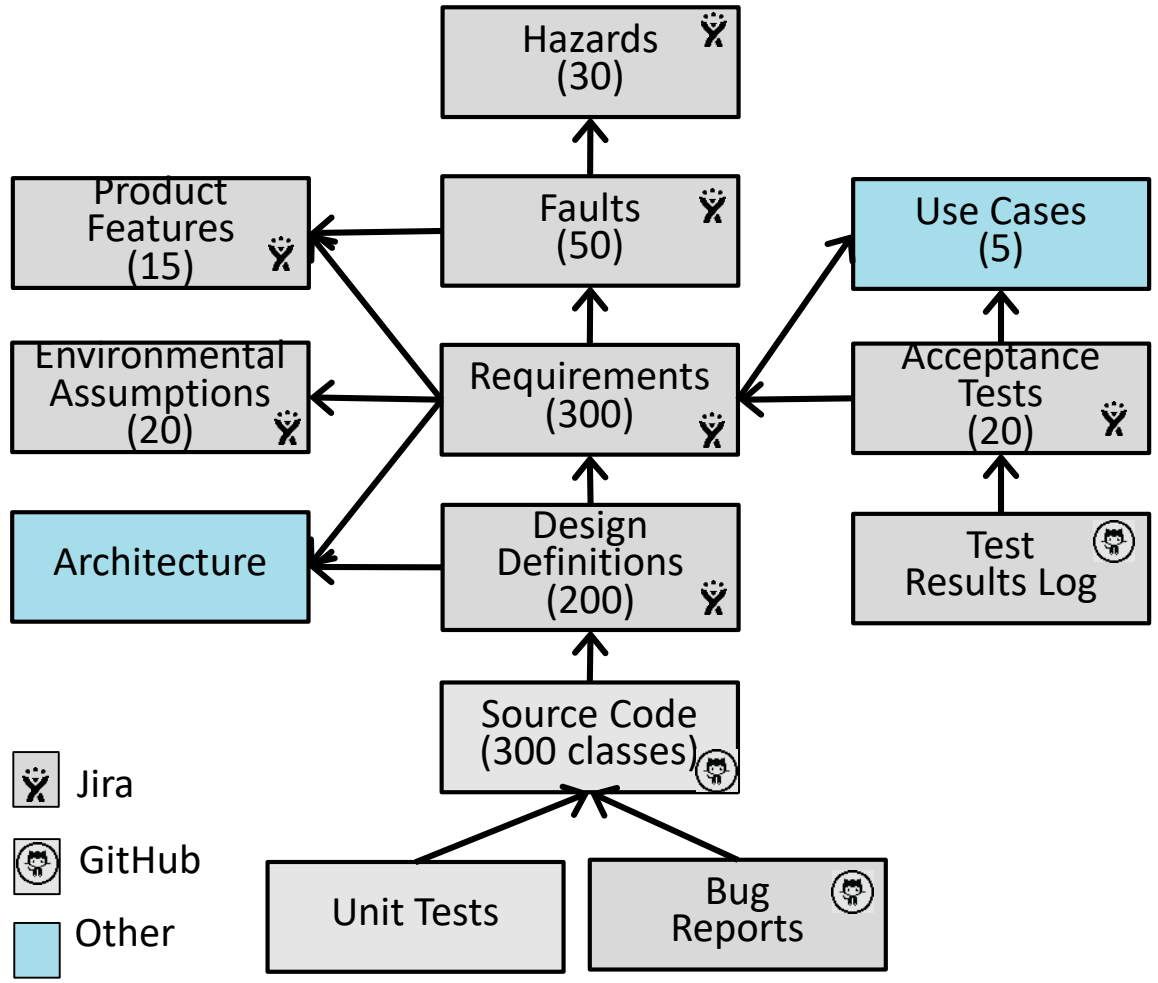
(DD-3): Real-Time Kinematic (RTK) shall be deployed. (Note: This increases the guaranteed accuracy of GPS to 2 centimeters.)



A5: Establish Hazard related Trace Links

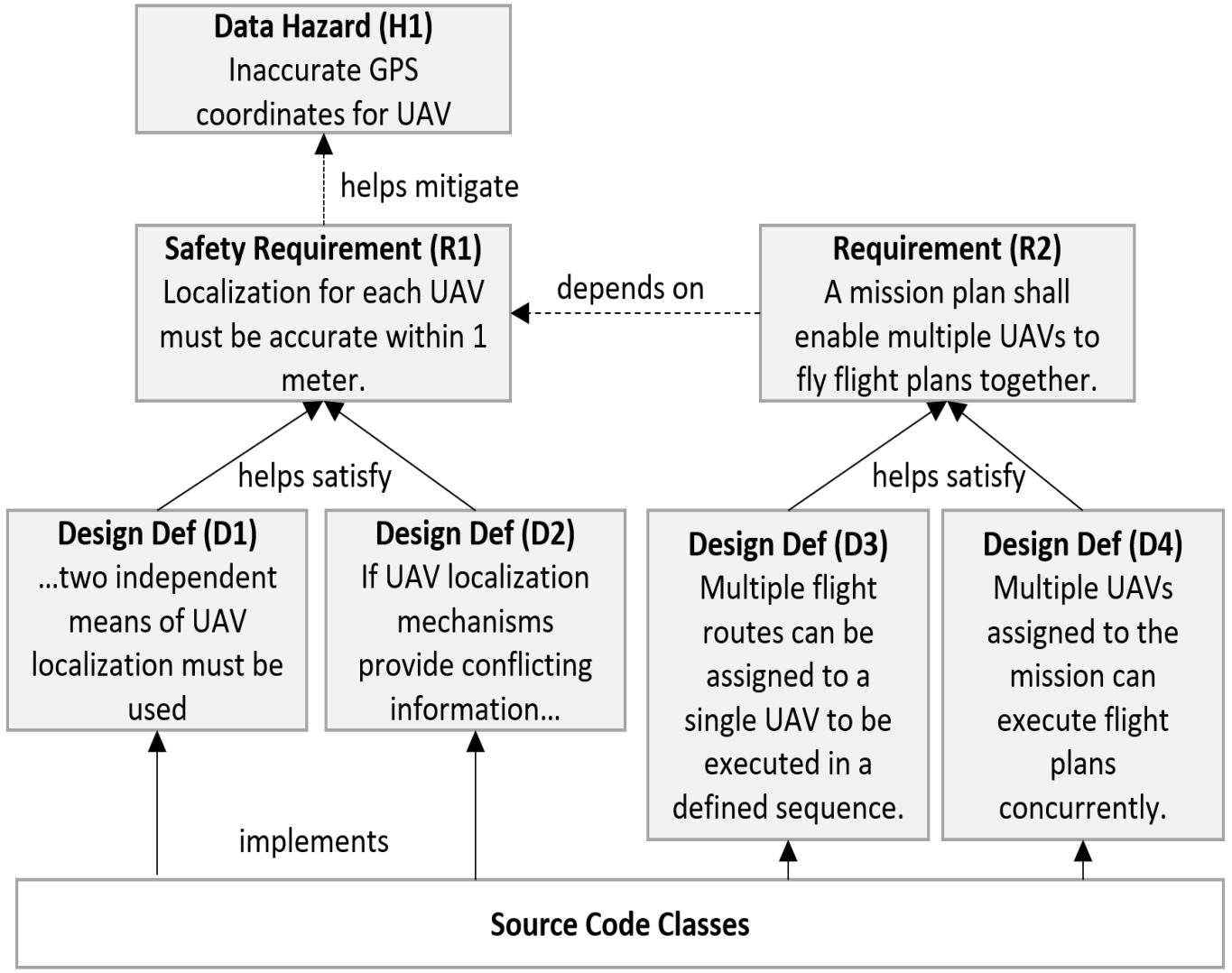
Leverage capabilities of tools commonly adopted in agile projects (e.g., Jira and Github) to incrementally construct trace links from safety stories to hazards, design definitions to safety stories, dependent system stories to safety stories, source code to design, and from acceptance tests to safety stories.

More details later in the talk!



A6: Sprint Planning

Track the design status of each safety requirement. Be aware of dependencies between system stories and safety stories and favor schedules in which safety stories and their associated design decisions are implemented as close as possible to their dependent system stories.



A7: Sprint Level Hazard Analysis

Perform an in-depth hazard analysis at the start of each sprint to identify new hazards, failure cases, and mitigations associated with new features and their interactions with other features.

System story (R5): UAVs shall be launched from a boat during river rescue.

Requirement (R6): When the return to launch (RTL) command is issued, the UAV shall return to its original launch coordinates.

Safety Story (R6'): When the return to launch (RTL) command is issued, the UAV shall return to its home coordinates.

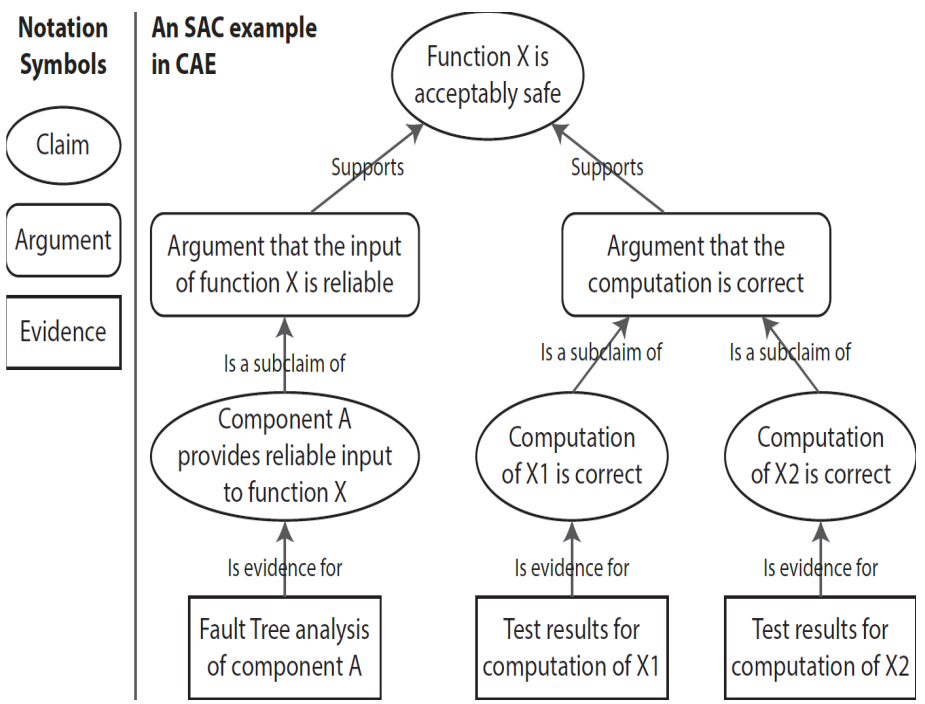
A8: Handle Emergent Faults

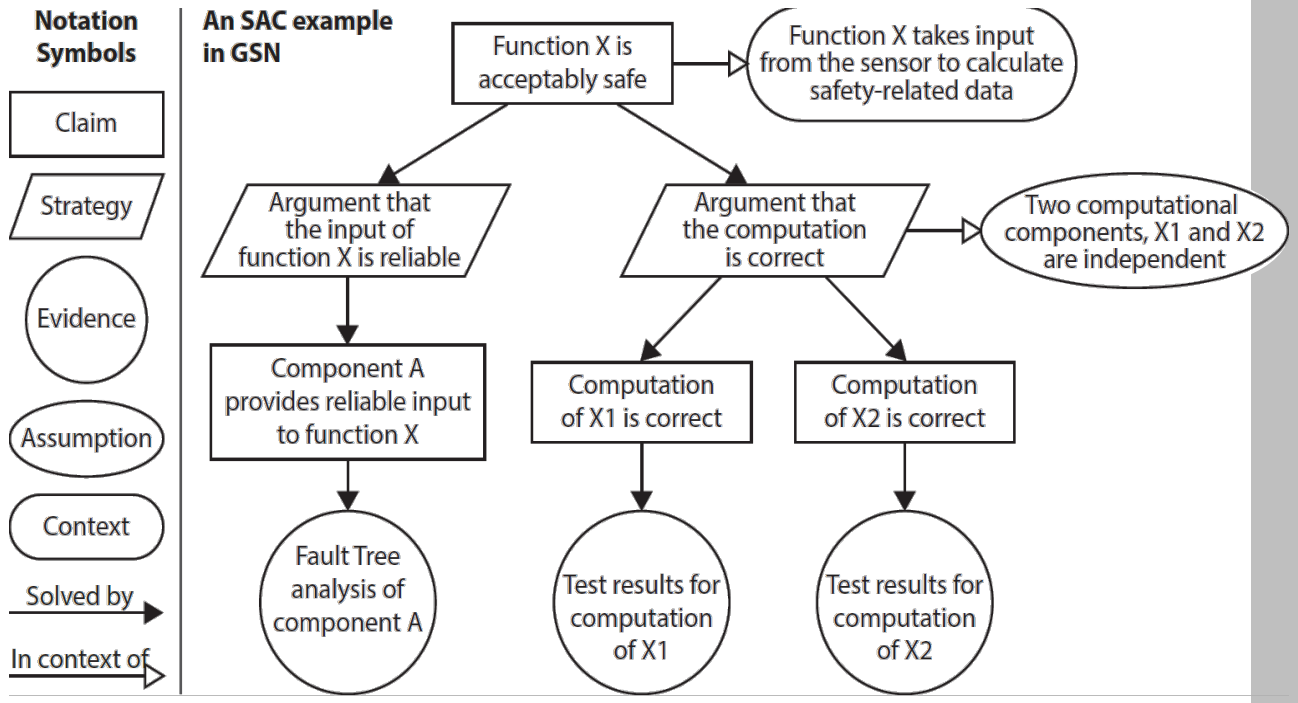
When new hazards, failure modes, and safety stories emerge as a result of observed faults during testing, the faults are documented, new safety stories specified and design solutions explored. The new stories and design definitions are added to the product backlog, and dependencies are documented as trace links..

A9: Incremental Safety Case

Throughout the sprint, the team must incrementally refine the safety case (SC) and construct an accurate, clear, comprehensive and defensible argument that a system is acceptably safe to operate in a particular context.

Safety Assurance Cases





Claim:
Assertion of compliance with key requirements and properties.
Must be within a specific context of use.

Arguments/Strategy:
Link evidence to claims via inference rules. Can be deterministic (true/false), probabilistic, or qualitative (i.e. link to regulations).

Evidence:
Process and people, testing, reviews, mathematical proofs.

Context:

Environmental Assumptions

Examples provided by Jinghui Cheng

Challenges of Safety Assurance Cases

Problems include:

- Building them
- Reviewing them
- **Maintaining** them
- **Reusing** them

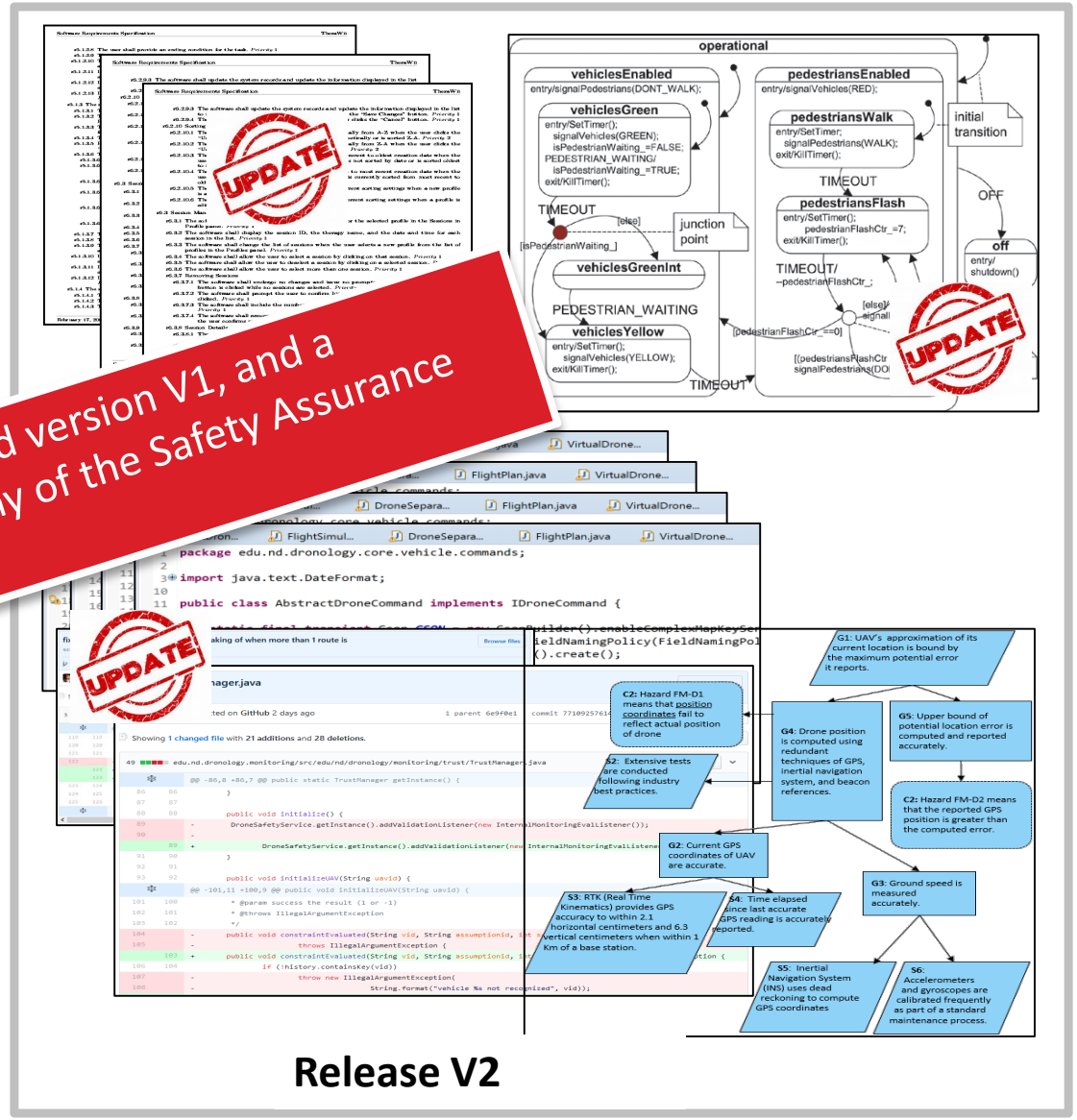
It is difficult to understand the impact of a change on the assurance case because:

- Huge volumes of data
- The impact of a change on the assurance structure is complex

T. Scott Ankrum, MITRE (2012)

Incrementally Evolving Safety Case

Release V1



Release V2

Given a fully approved/certified version V1, and a modified V2 – can we reuse any of the Safety Assurance Case, and if so, which parts??

A10: End of Sprint Safety Gateway

Analyze the safety of the system at the end of each sprint.

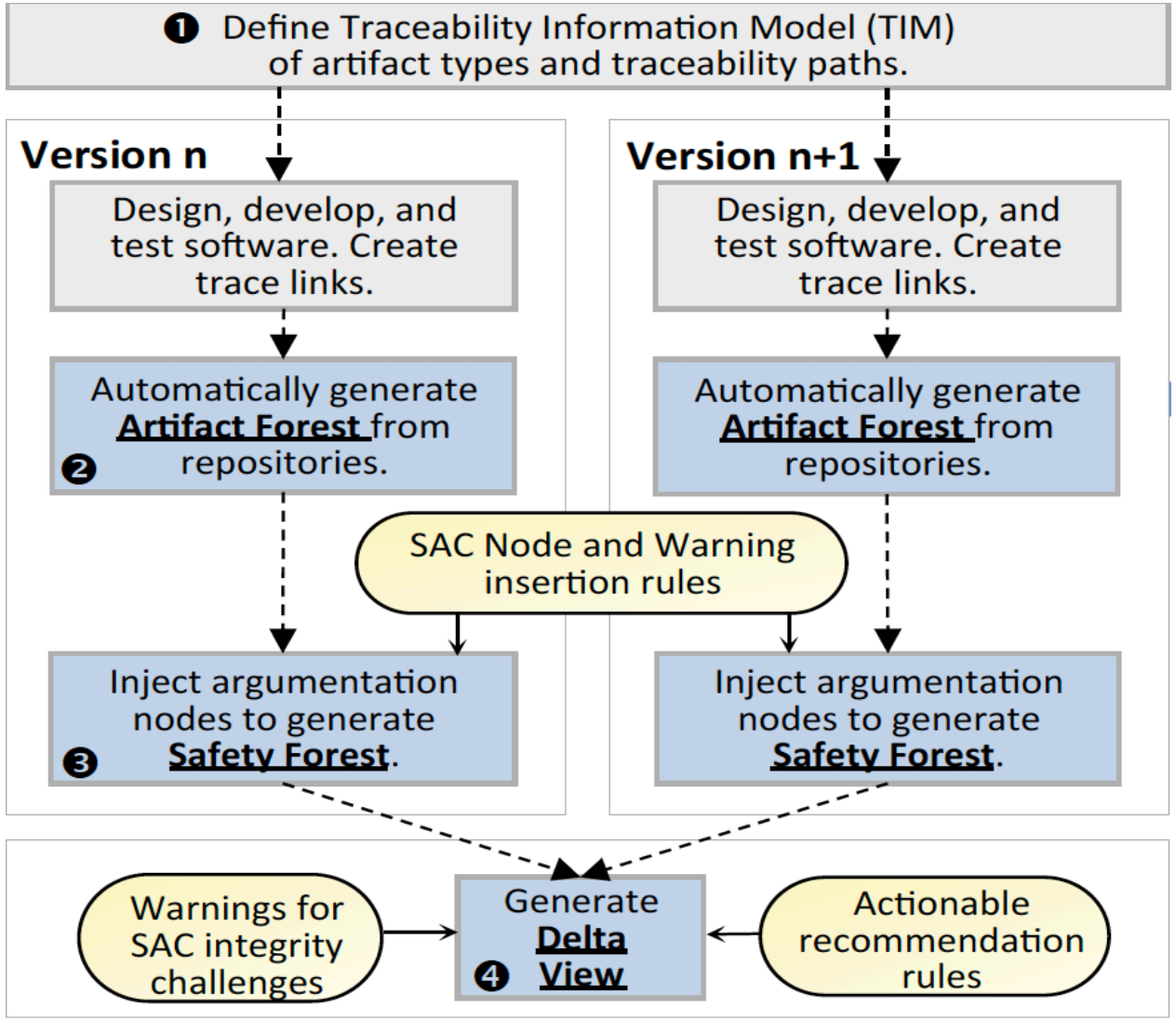
- If necessary conduct “hardening” sprints to address safety concerns and/or to prepare systems in regulated domains for certification.
- Understand the safety status of the release

System Story (RX):
Dronology shall directly control the flights of up to 50 concurrent UAVs.

Safety Story (SR-3):
UAVs must maintain a MINIMUM_ SEPARATION_DISTANCE of 3 meters at all times

Resolved through a combination of manual and encoded design decisions

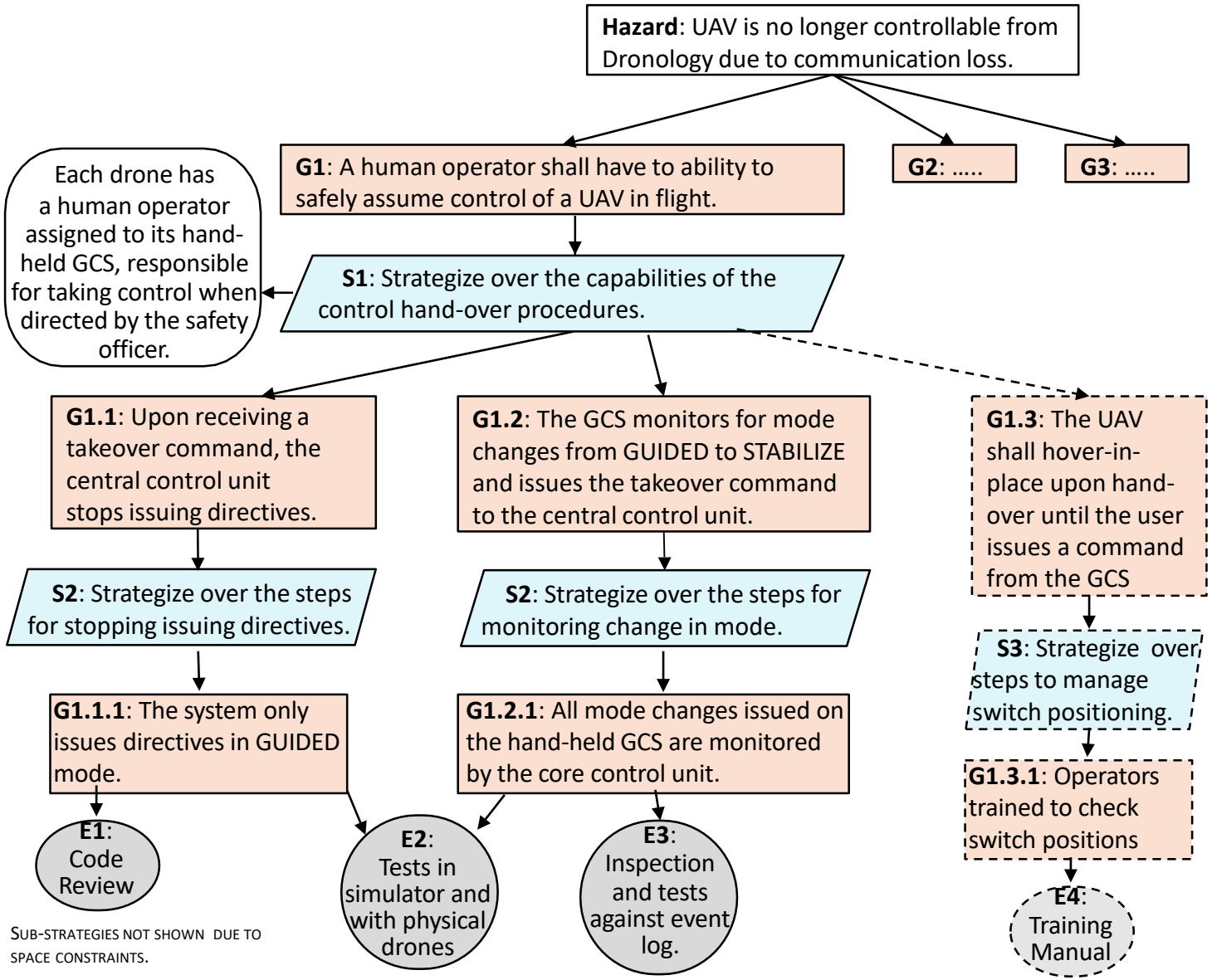
Addressing the Incremental Safety Case Challenge



Instrument the environment with "living traceability" so that trace links are current.

Establish traceability between hazards and artifacts so that you can understand exactly how changes in the system impact the safety case, and evolve it accordingly.

A Safety Assurance Case Argues that a Hazard is mitigated



SUB-STRATEGIES NOT SHOWN DUE TO SPACE CONSTRAINTS.

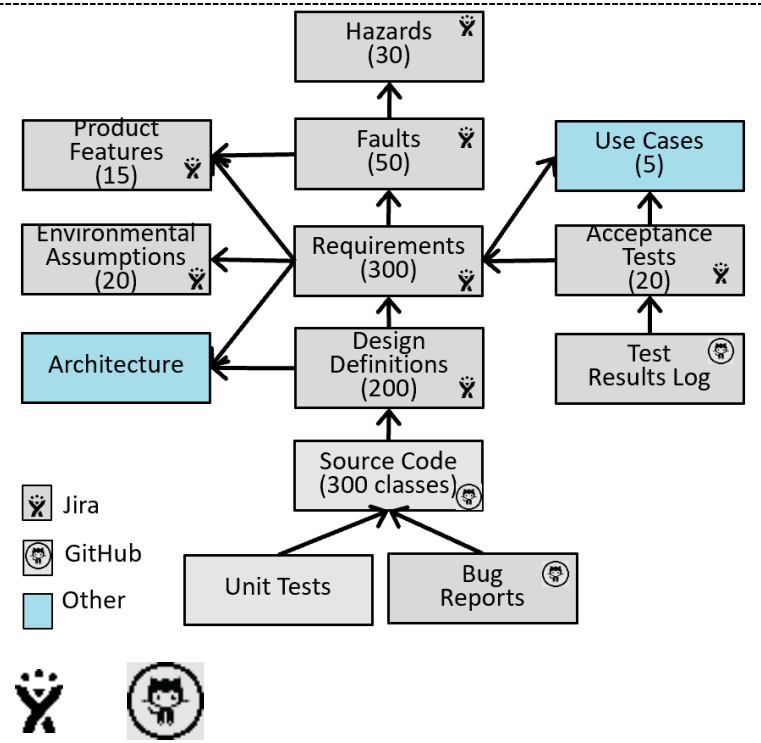
A goal in the safety assurance case is designed to mitigate a specific hazard.

GSN Notation

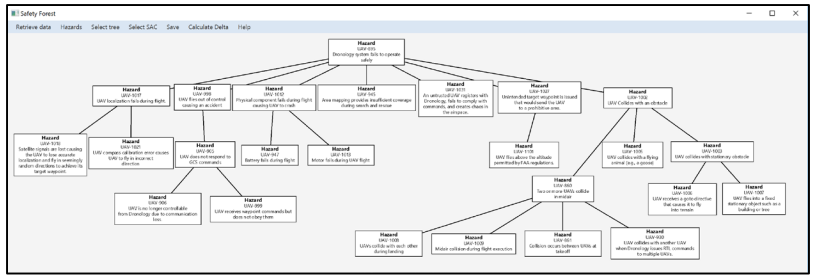
- Goal
- Strategy
- Evidence
- Context

We take a slightly different approach from a traditional safety argument.

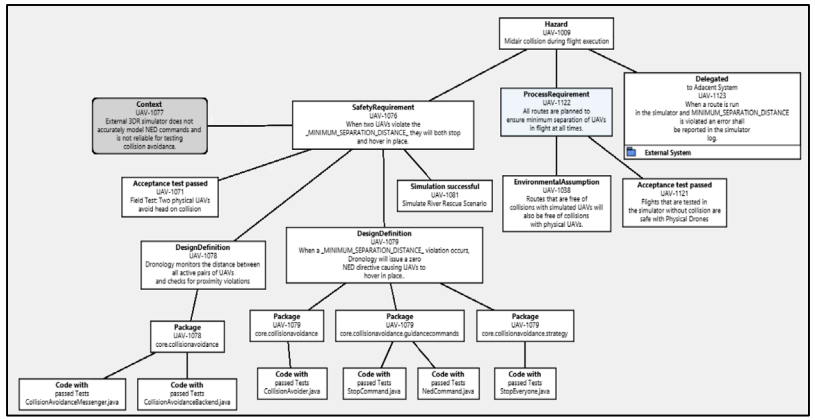
Overview of SAFA (Software Artifact Forest Analysis)



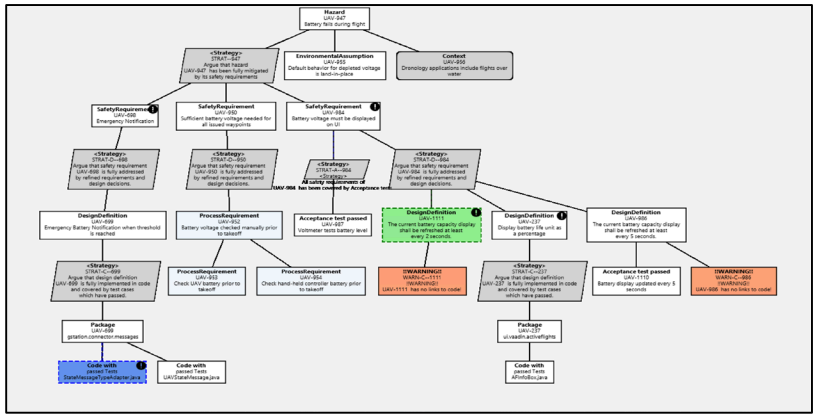
❶ Design the TIM (Traceability Information Model) and instrument the environment.



❷ Identify hazards and organize them hierarchically.



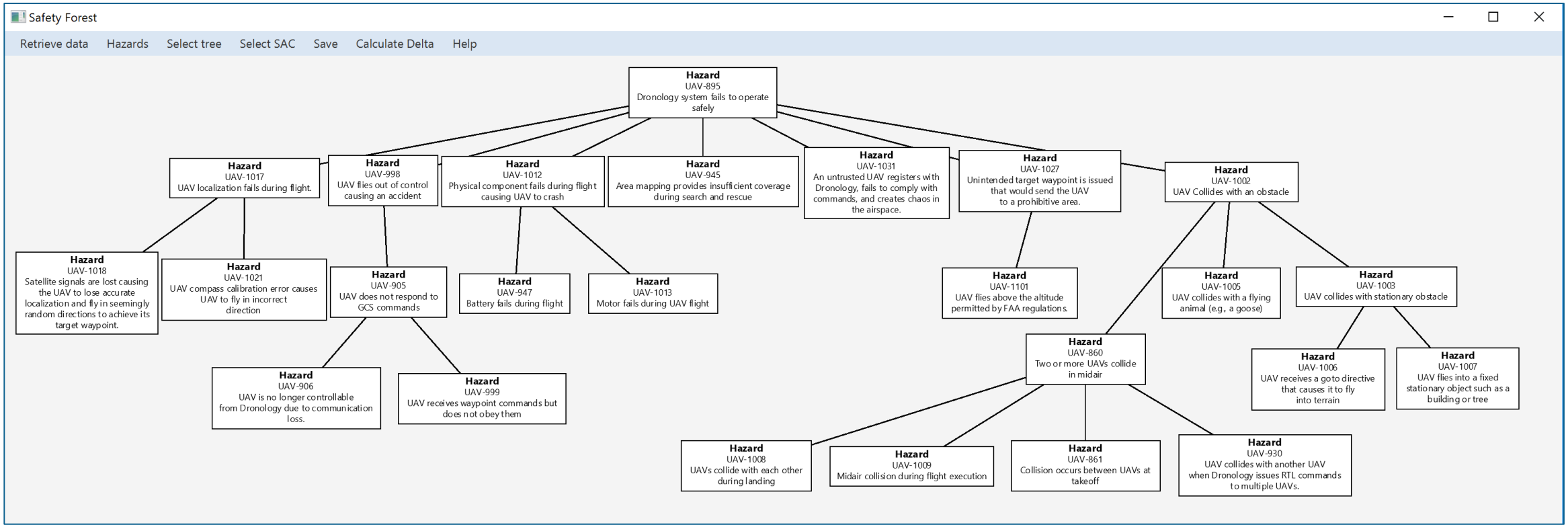
❸ Generate a safety tree and an artifact tree for each hazard. Inject warnings and recommendations.



❹ Generate a delta tree showing changes between versions and analysis of their impact.

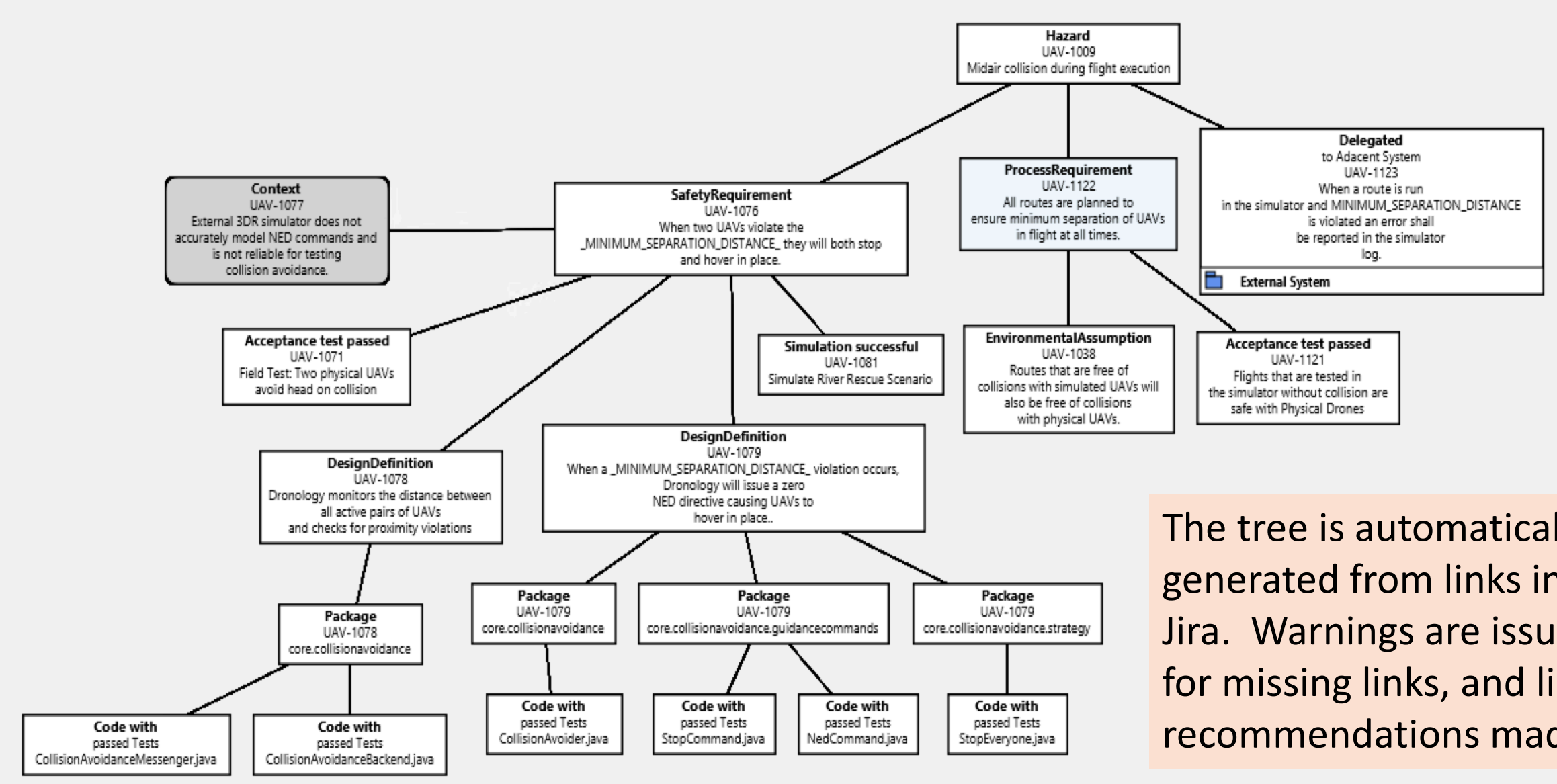
hazard_warning_safety_signs_p1254-38488_zoom.jpg

Identify hazards



Hazards can be identified through any hazard analysis process such as: brainstorming, checklists, Fault Mode Effect Analysis, or through Fault Tree Analysis.

Generate the Raw Artifact Tree

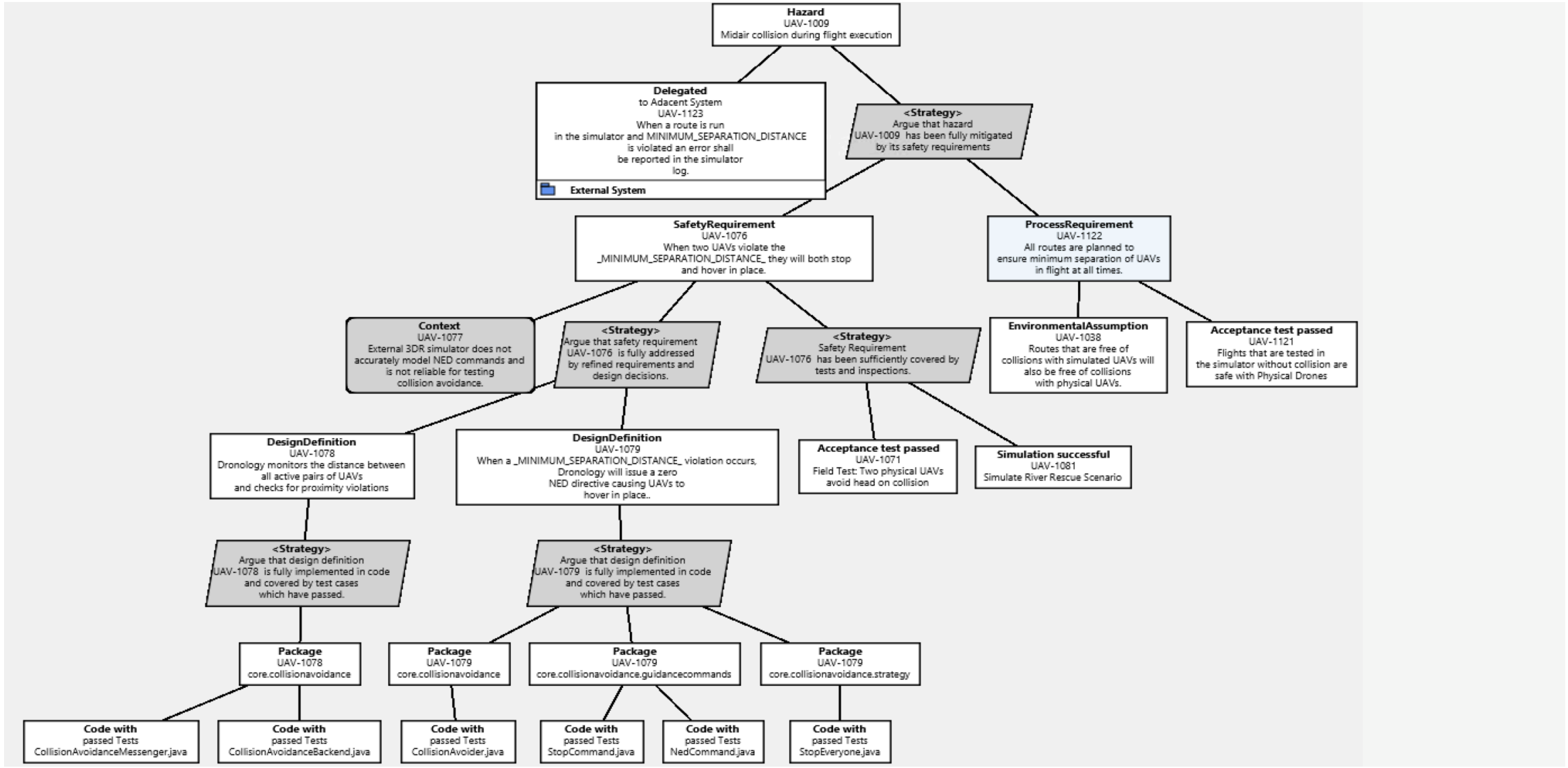


The tree is automatically generated from links in Jira. Warnings are issued for missing links, and link recommendations made.

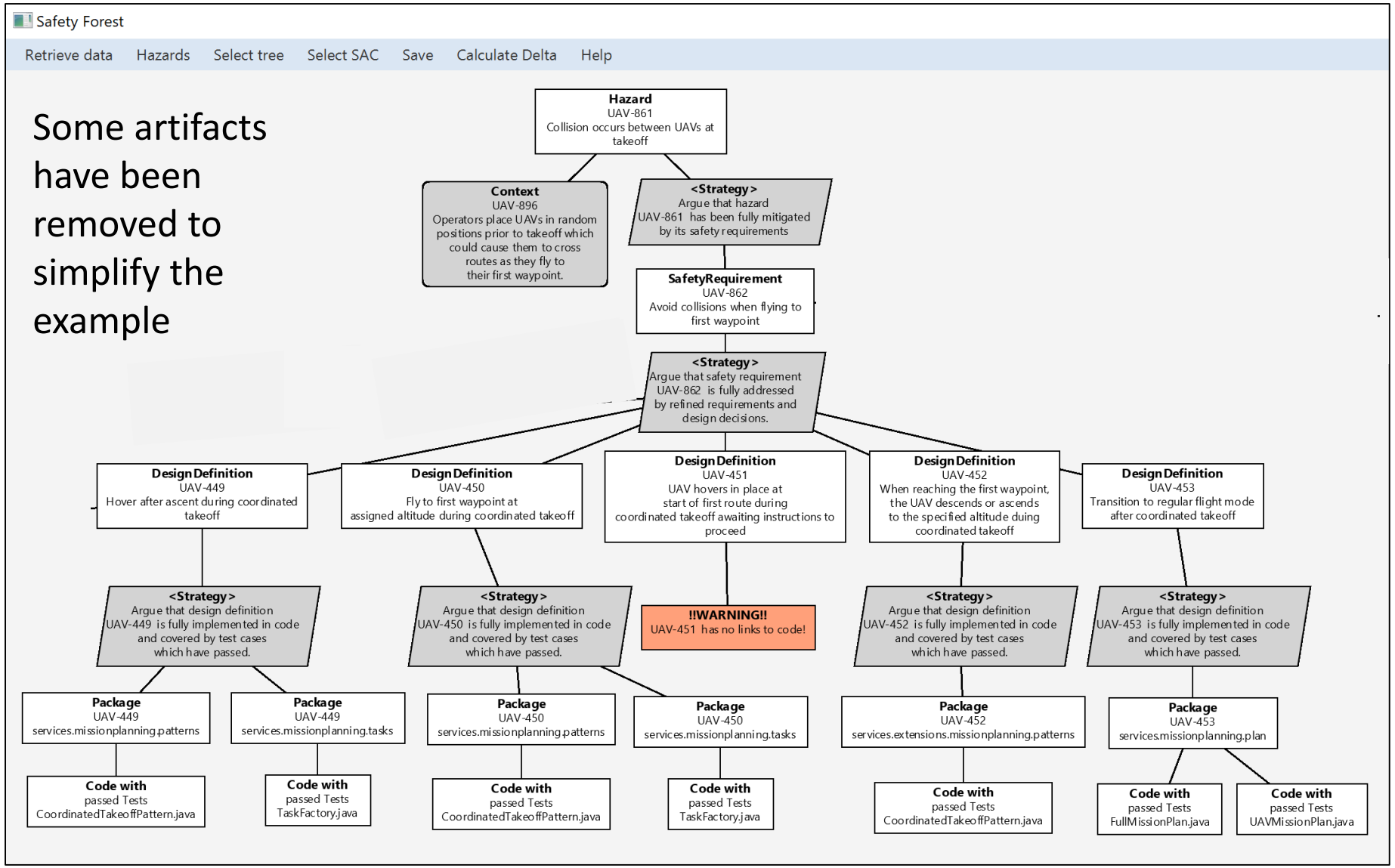
Transform the Tree into a Safety Case

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

Add SAC nodes to create a Safety Tree



Add Warnings



Warnings:

Egregious omissions
e.g., no source code links

Nuanced omissions
e.g., insufficient diversity of evidence

Recommendations:

Vet a set of candidate links

Add missing artifacts

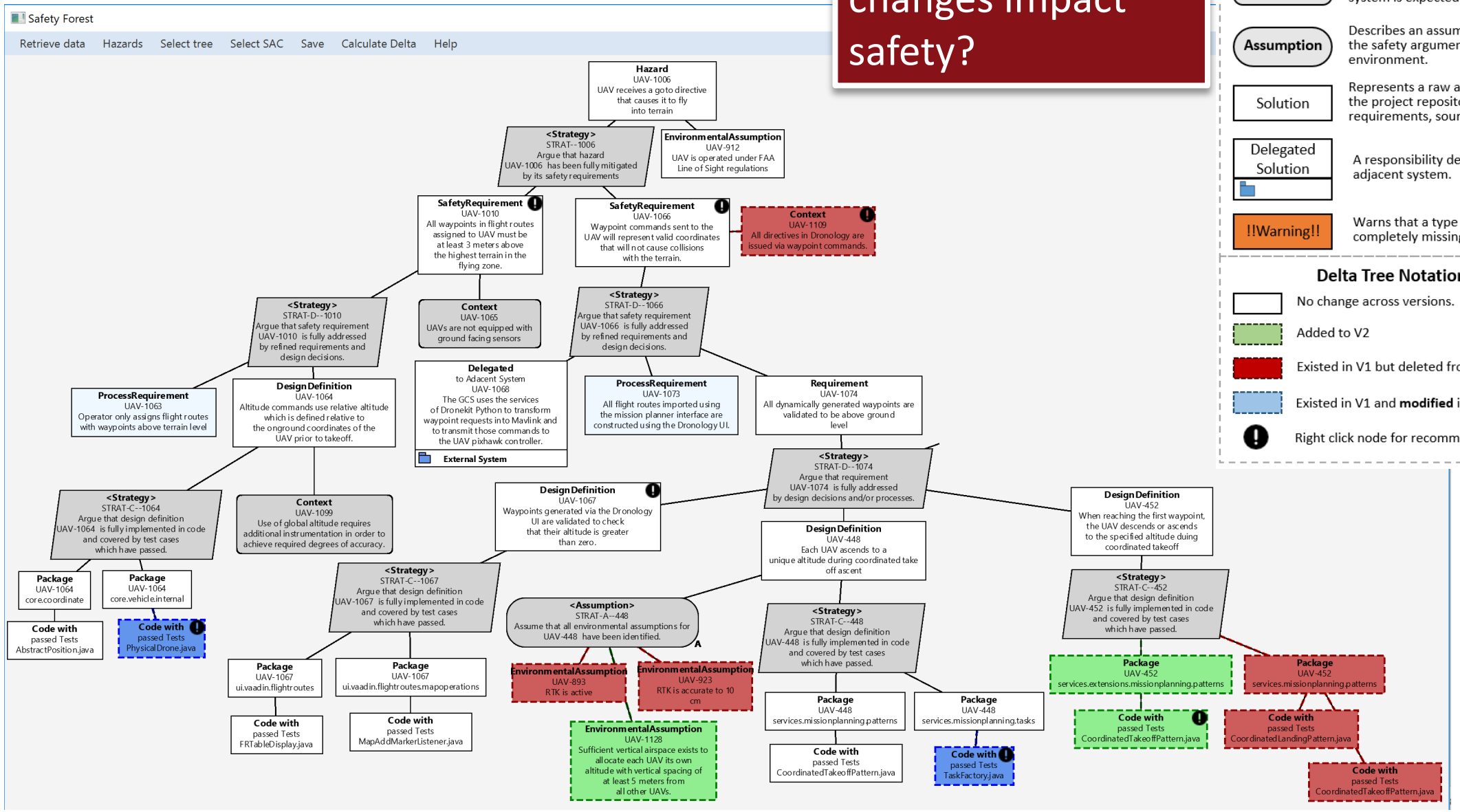
Explain a specific strategy.

Add Warnings

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

Compare Two Trees

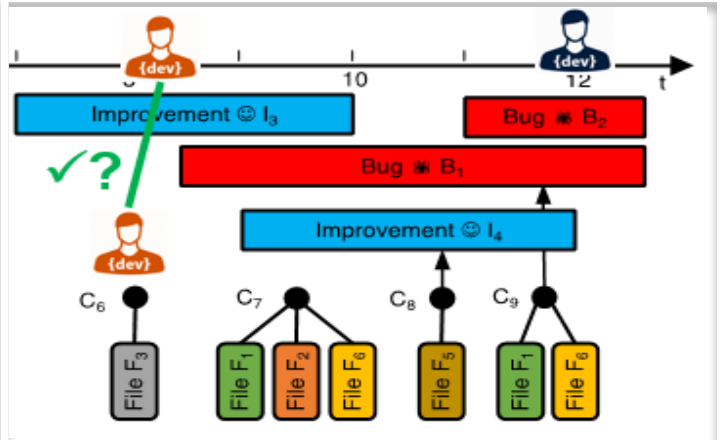
What has changed and how do those changes impact safety?



Automated Tracing Support

Semantic Solutions

Evolving Links



Mine Links

Current state of the art traceability techniques show great promise for supporting industrial projects. They can be used to recommend missing links and to support trace link evolution.

Empirical Study

RQ1: What types of changes impacted hazard mitigations between versions v1 and v2, and is SAFA able to detect and visualize them?

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

Version 1 (v1)
49,400 LOC
418 Java Classes
146 Requirements
224 Design definitions

Version 2 (v2)
73,591 LOC
646 Java classes
185 Requirements
283 Design definitions

Legend:

SR(Safety Req)
PR(Process)
DD(Design Def.)
AJ(Adjacent system)
CX(Context)
AS(Assumption)
TS(Tests)
SC(Code)

Results

All changes were identified by SAFA. Changes from our study covered 8 different artifact types.

User Study

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

Version 1 (v1)
49,400 LOC
418 Java Classes
146 Requirements
224 Design definitions

Version 2 (v2)
73,591 LOC
646 Java classes
185 Requirements
283 Design definitions

Each participant was given six hazards to evaluate using two treatments.

T1: View artifact trees for v1 and v2
T2: View delta tree

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

(Q1) With respect to this hazard, has the system changed in a way that potentially affects its safety? If so please explain your answer. **SAFA 100%, Paired view 80.6%**

(Q2) Was the information provided to you for each of the two methods sufficient for assessing the impact of change upon system safety? **Yes but SAFA provided more information about changed nodes.**

(Q3) Which of the two methods did you prefer using? Why? **SAFA 9, Paired view 1**

(Q4) Can you recommend any changes for SAFA?

User Study

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

I would kill to have SAFA in my workplace

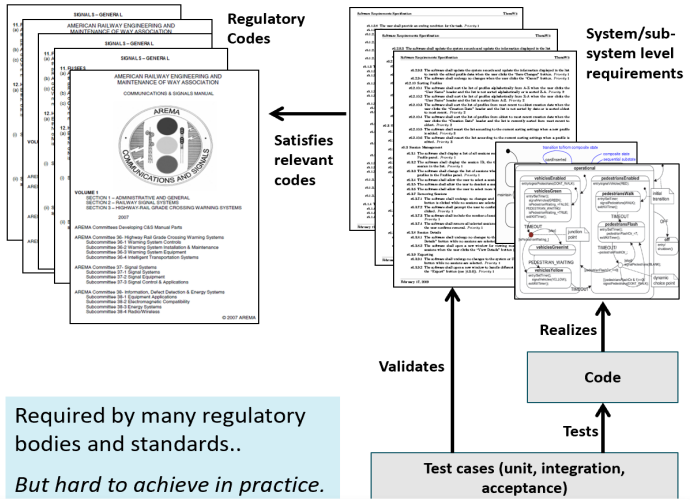
Give me back the delta view!

I find myself implicitly trusting the tool. Is the tool certified?

Many helpful suggestions for AI features that SAFA could support in the next iterations!

Conclusion

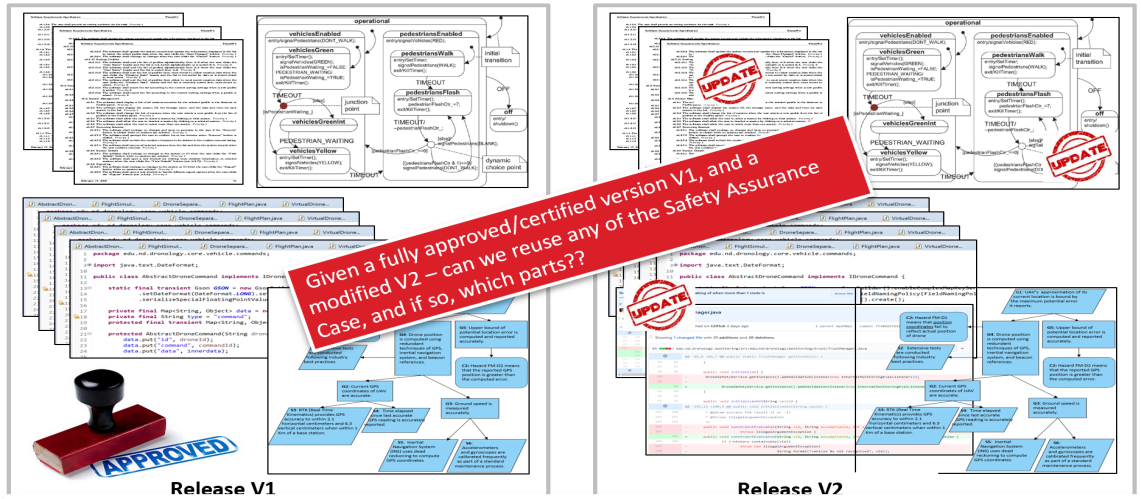
The ability to **interrelate any uniquely identifiable** software engineering artifact to any other, **maintain** required links over time, and **use the resulting network** to answer questions of both the software product and its development process.
- *CoEST Definition*



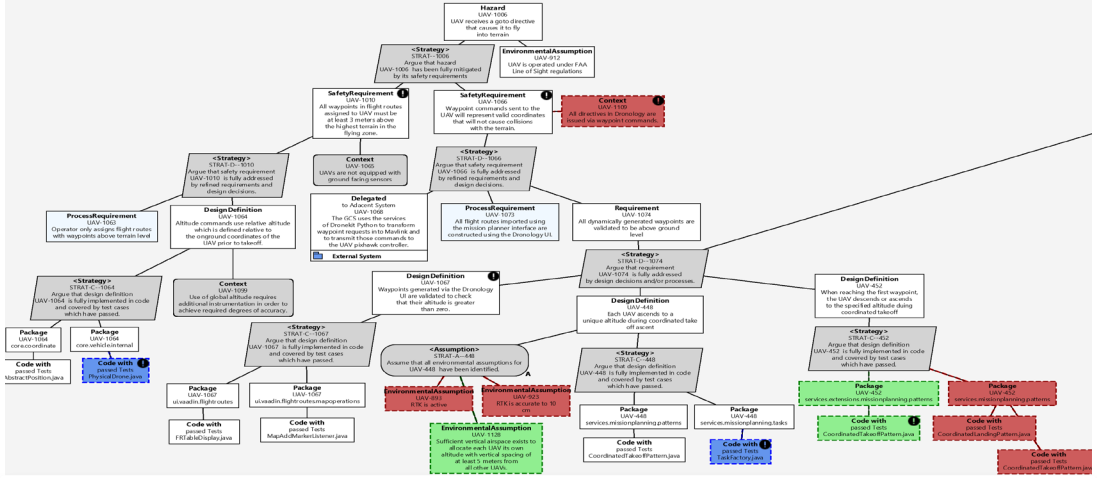
Required by many regulatory bodies and standards..

But hard to achieve in practice.

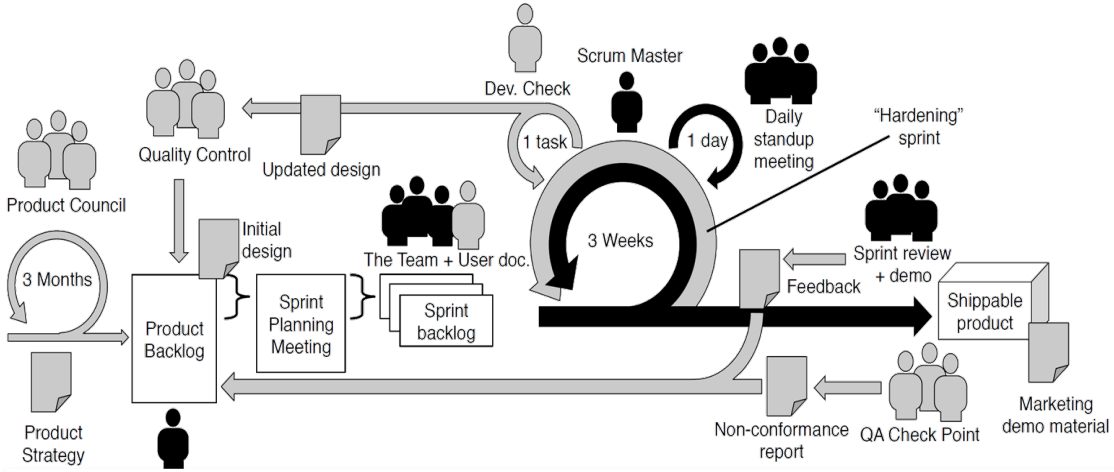
Traceability is important for safety critical projects



We need to build safety cases incrementally



We can achieve it with agile automation



Breaking the big freeze is highly feasible

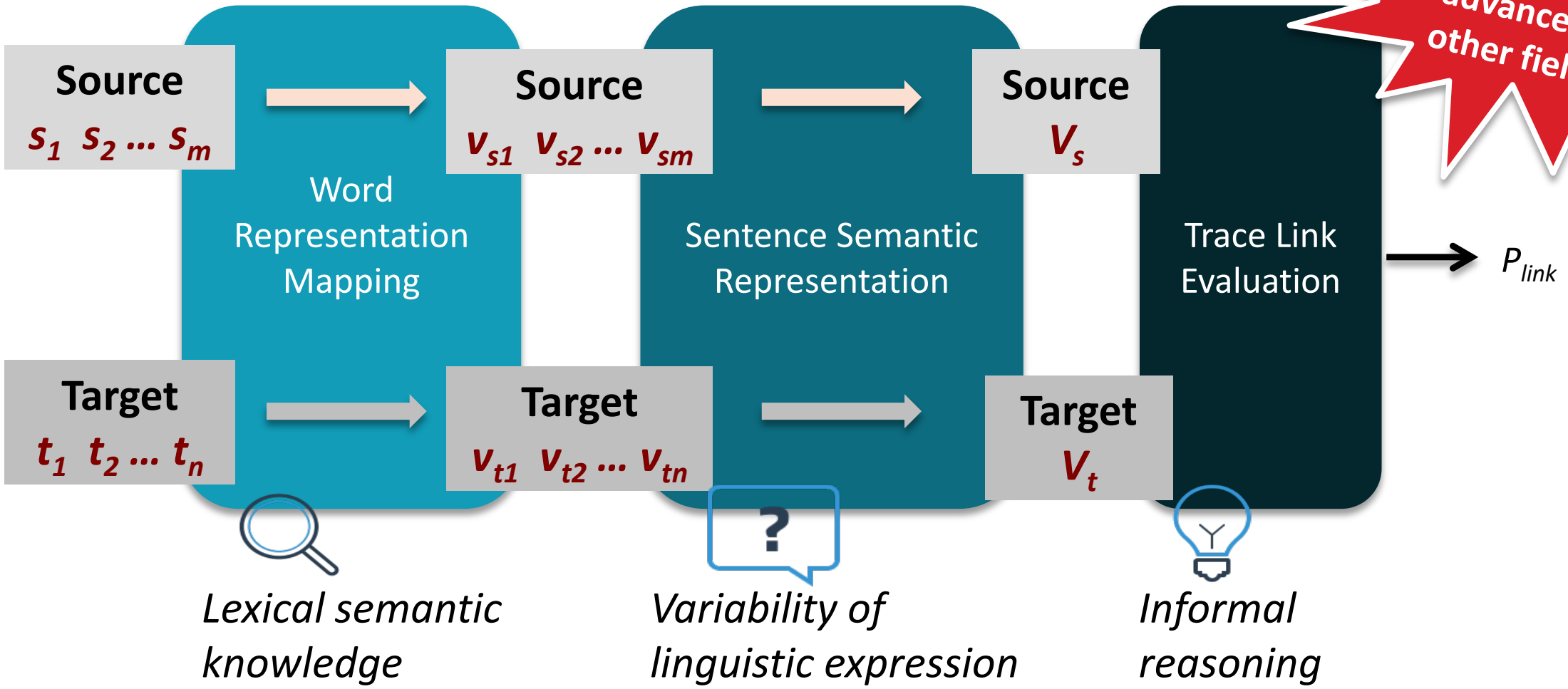
CRACKING THE BIG FREEZE!

TRACEABILITY SOLUTIONS FOR HIGH-DEPENDABILITY AGILE PROJECTS

Jane Cleland-Huang, PhD
JaneClelandHuang@nd.edu
Department of Computer Science and Engineering
University of Notre Dame

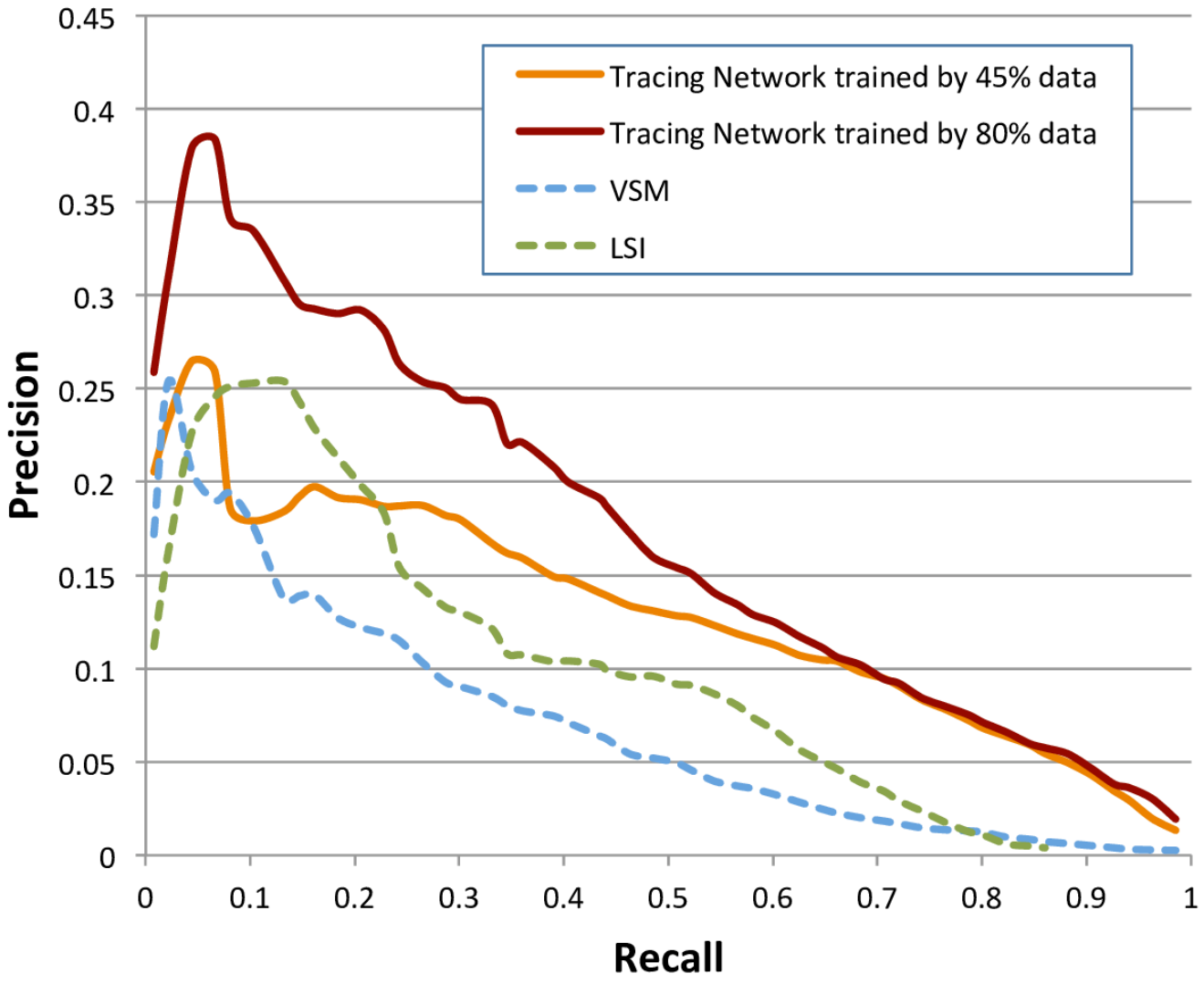
Some of the work described in this talk was funded by the US National Science Foundation under Grants CCF-0959924 and CCF-1265178.

Leverage Deep Learning Techniques



Semantically enhanced Software Traceability using Deep Learning techniques.
Jin Guo, Jinghui Cheng, Jane Cleland-Huang: ICSE 2017: 3-14

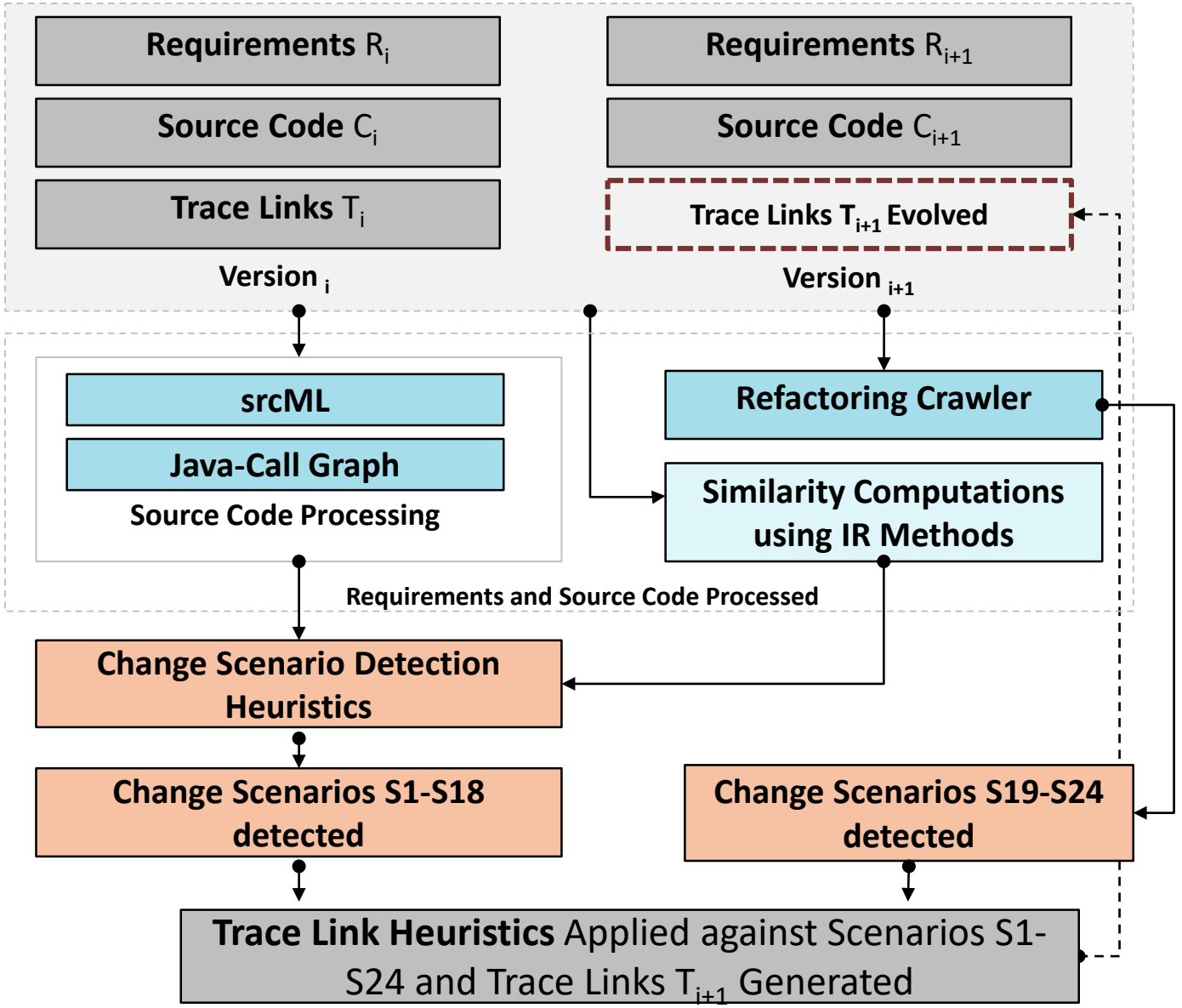
A Proof of Concept



Automated approaches that generate trace links from scratch, return imprecise results.

They are useful for **supporting** tasks such as Impact analysis, but not currently sufficiently reliable on their own.

Solution 2: Evolving and Discovering Links



❶ Software artifacts changed across versions

❷Tools for detecting changes in code and requirements.

❸ TLE tools and algorithms for recognizing change patterns and evolving trace links.

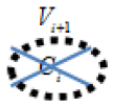
Evolving Links

P_1: Added Class
New Functionality

P_2: Added Method
New Functionality

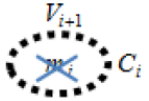
P_3: Modified Method
New Functionality

P_4: Deleted Class
Obsolete Functionality



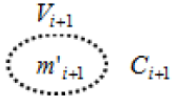
Pattern Description:
Obsolete functionality is getting deleted in form of a class.

P_5: Deleted Method
Obsolete Functionality



Pattern Description:
Obsolete functionality is getting deleted in form of a method.

P_6: Modified Method
Obsolete Functionality



Pattern Description:
Obsolete functionality is getting deleted in form of a modified method.

1. Identify types of changes that could invalidate existing trace links.

2. Define properties to detect when the change has occurred.

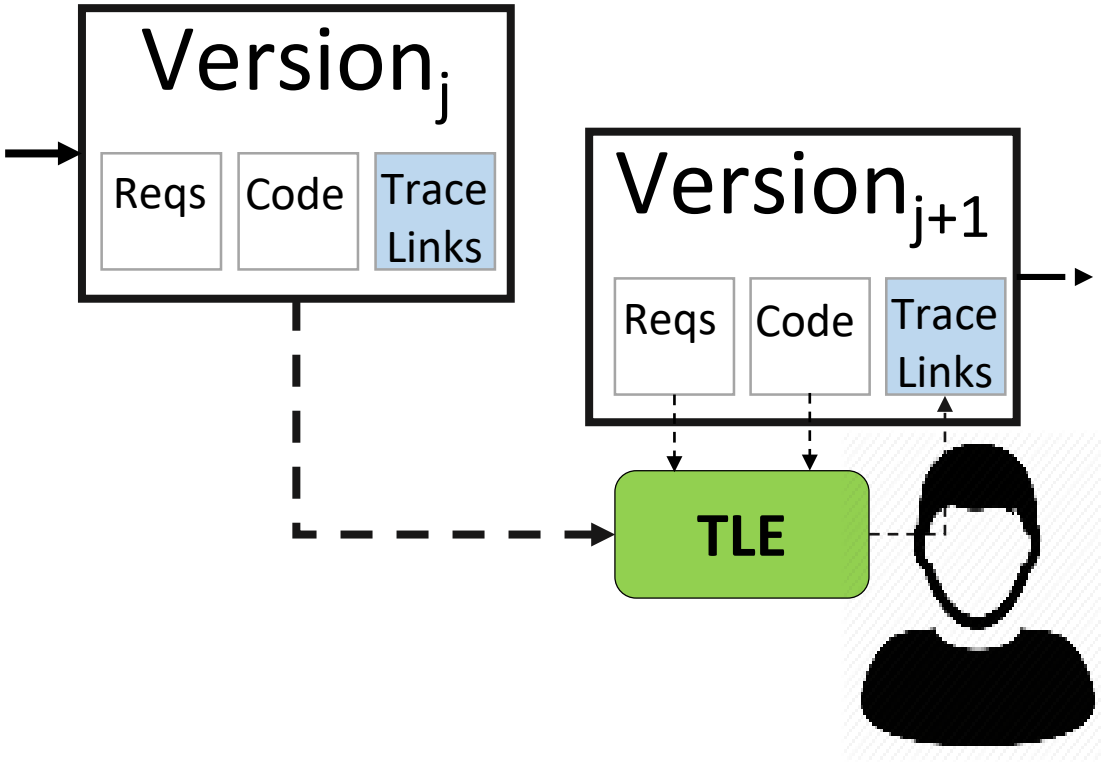
Rule Type		Artifact Properties	Rules for D… S_1	S_2	S_3	S_4	S_5	S_6	S_7	S
Add Class Delete Class Add Method Del Method Mod Method	1	$\exists C_{i+1} \mid \nexists C_i$	•	•	•	•	•	•		
	2	$\nexists C_{i+1} \mid \exists C_i$							•	•
	3	$\exists m_{i+1} \mid \nexists m_i$								
	4	$\nexists m_{i+1} \mid \exists m_i$								
	5	$\exists m_i \wedge \exists m_{i+1} \mid \nexists Sim(m_i, m_{i+1})$								
Checks for links between classes	6	$\exists Sim(C_{i+1}, C'_j)$	•		•	•	•			
	7	$\exists Sim(C_{i+1}, C'_i) \wedge \exists Sim(C_{i+1}, C''_i)$		•						
	8	$\exists Sim(C_i, C'_{i+1}) \wedge \exists Sim(C_i, C''_{i+1})$								
	9	$\exists Sim(C_i, C'_{i+1}) \wedge \exists Sim(C'_i, C''_{i+1})$								
Checks for links between classes and requirements	10	$\exists Sim(C_{i+1}, R_{i+1})$	•							
	11	$\forall R \mid Sim(r, C_{i+1}) \subset R_{C'_i}$		•						
	12	$\exists R \mid Sim(r, C_{i+1}) \subseteq R_{m'_i}$								
	13	$\forall R \mid Sim(r, C_{i+1}) \subseteq R_{m'_i}$								
	14	$\exists R \mid Sim(r, C_{i+1}) \wedge \nexists Sim(r, C'_{i+1})$				•	•	•		
	15	$\forall R \mid Sim(r, C_{i+1}) \subseteq R_{C'_i} \cup R_{C''_i}$		•						
	16	$\forall R \mid Sim(r, C_{i+1}) \subseteq R_{m'_i} \cup R_{m''_i}$								
	17	$R_{C_i} \subseteq Sim(r, C'_{i+1}) \cup Sim(r, C''_{i+1})$								
	18	$R_{m_i} \subseteq Sim(r, C'_{i+1}) \cup Sim(r, C''_{i+1})$								
	19	$R_{C_i} \cup R_{C'_i} \subseteq Sim(r, C''_{i+1})$								
	20	$R_{m_i} \cup R_{m'_i} \subseteq Sim(r, C''_{i+1})$								
	21	$\forall R \mid Sim(r, C_{i+1}) \subseteq R_{C'_i}$				•				
Checks for methods in classes	22	$\forall m \in C_{i+1} \subset m \in C'_j$	•			•	•	•		
	23	$\forall m \in C_{i+1} \subseteq m \in C'_i \wedge C''_i$		•						
	24	$\forall m \in C_i \wedge C'_i \subseteq m \in C''_{i+1}$								
	25	$\forall m \in C_i \subseteq m \in C'_{i+1} \wedge C''_{i+1}$								
Checks for associations between classes	26	$\exists A(D_{i+1}, C_{i+1}) \subset A(D_i, C'_i)$	•			•	•	•		
	27	$\exists A(M_{i+1}, m_{i+1}) \subset A(M_i, m'_i)$								
	28	$\forall A(M_{i+1}, m_{i+1}) \subseteq A(M_i, m'_i)$								
	29	$\forall A(D_{i+1}, C_{i+1}) \subseteq A(D_i, C'_i) \wedge A(D_i, C''_i)$		•						
	30	$\forall A(M_{i+1}, m_{i+1}) \subseteq A(M_i, m'_i) \wedge A(M_i, m''_i)$								
	31	$\forall A(D_i, C_i) \subseteq A(D_i, C'_{i+1}) \wedge A(D_i, C''_i)$								
	32	$\forall A(M_i, m_i) \subseteq A(M_i, m'_{i+1}) \wedge A(M_i, m''_i)$								
	33	$\forall A(D_i, C_i) \wedge A(D_i, C'_i) \subseteq A(D_i, C''_{i+1})$								
	34	$\forall A(M_i, m_i) \wedge A(M_i, m'_i) \subseteq A(M_i, m''_{i+1})$								
	35	$\exists A(C_{i+1}, C'_{i+1})$				•	•	•		
	36	$\exists A(C'_{i+1}, C''_{i+1})$								
	37	$\exists A(m'_{i+1}, m''_{i+1})$								
	38	$\exists A(C_i, C'_i)$								
	39	$\exists A(m_i, m'_i)$								
	40	$\nexists D_i \mid Sim(D_i, C_{i+1})$	•							
	41	$\nexists D_{i+1} \mid Sim(D_{i+1}, C_i)$							•	
	42	$\nexists D_{i+1} \mid Sim(D_{i+1}, R_{c_i})$							•	
	43	$\nexists D_{i+1} \mid Sim(D_{i+1}, R_{m_i})$								
	44	$\nexists C'_{i+1}$		•						
	45	$\nexists m'_{i+1}$								
	46	$\nexists C'_{i+1} \wedge C''_{i+1}$		•						
	47	$\nexists m'_{i+1} \wedge m''_{i+1}$								
	48	$\exists E(C_{i+1}, C'_i)$					•			
	49	$\exists E(C'_i, C_{i+1})$					•			

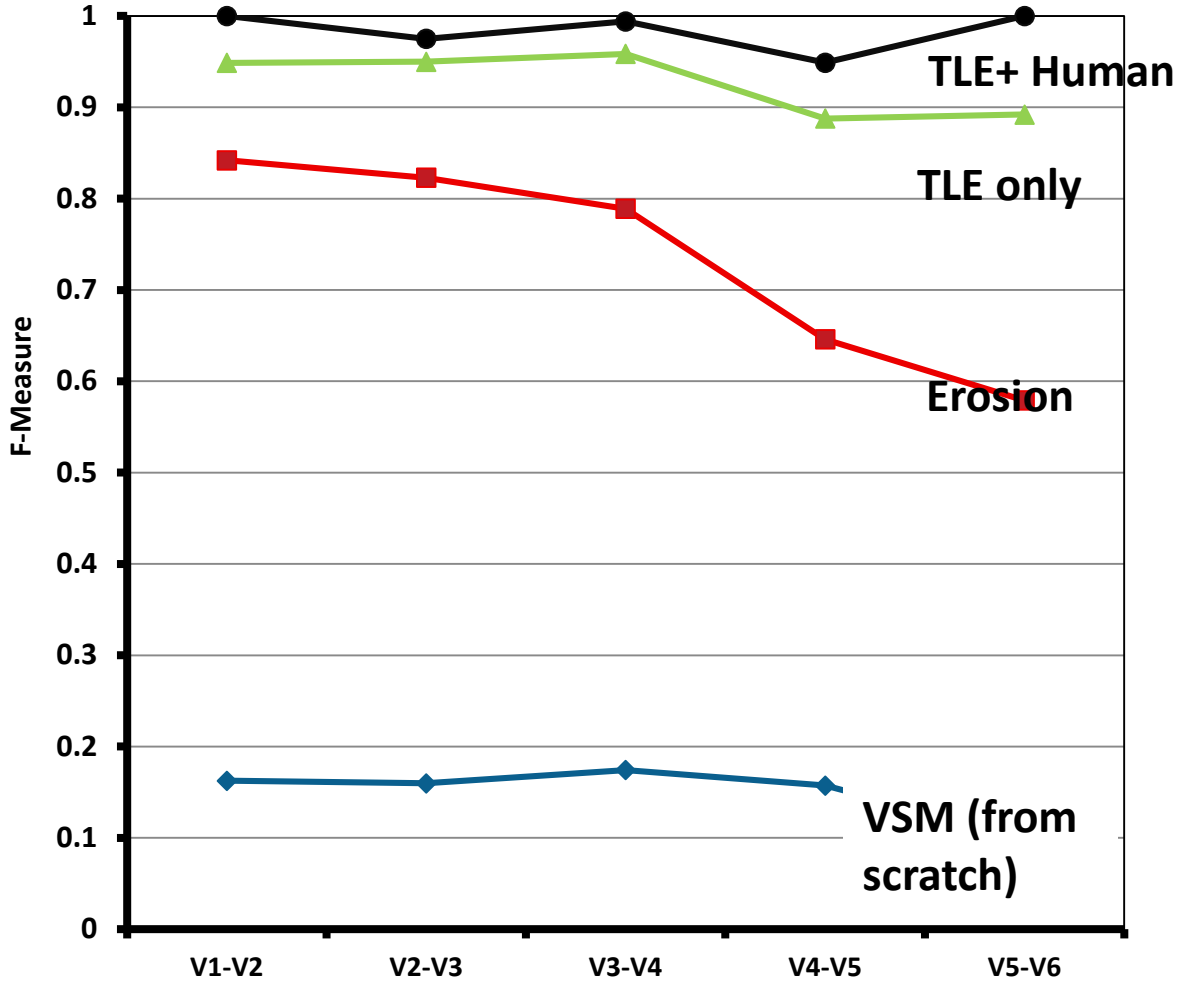
i; D_i: all other classes in version i; m_i: a method in Version i; M_i: all other methods in Version i
on of the system (applied to classes, methods, requirements, and links)
n version i; $Sim(x, y)$: similarity of x and y with similarity value of n
etween x and y and $E(x, y)$ means that x extends y
C_{i+1}: in *Add Class* Scenarios($S_1 - S_6$)is the added class; C_i: in *Delete Class* Scenarios($S_7 - S_9$)is the deleted class
m_{i+1}: in *Add Method* Scenarios($S_{10} - S_{13}$)is the added method; m_i: in *Delete Method* Scenarios($S_{14} - S_{16}$)is to the deleted method

	Link Evolution Actions
S1	$l(C_{i+1}, R_{i+1}) \mid Sim(C_{i+1}, R_{i+1})$
S2	$l(C_{i+1}, R_{C'_i})$
S3	$l(C_{i+1}, R_{C'_i} \cup R_{C''_i})$ and $!!(C'_i, R_{C'_i})$ and $!!(C''_i, R_{C''_i})$
S4	$l(C_{i+1}, R_{m_i})$ and $!!(C'_{i+1}, R_{m_i})$
S5	$l(C_{i+1}, R_{C'_i})$
S6	$l(C_{i+1}, R_{C'_i})$
S7	$!!(C_i, R_{C_i})$
S8	$l(C'_{i+1}, R_{C_i})$ and $l(C''_{i+1}, R_{C_i})$ and $!!(C_i, R_{C_i})$
S9	$l(C''_{i+1}, R_{C_i})$ and $l(C''_{i+1}, R_{C'_i})$ and $!!(C_i, R_{C_i})$ and $!!(C'_i, R_{C'_i})$
S10	$l(m_{i+1}, R_{i+1}) \mid Sim(m_{i+1}, R_{i+1})$
S11	$l(m_{i+1}, R_{m'_i})$
S12	$l(m_{i+1}, R_{m'_i} \cup R_{m''_i})$ and $!!(m'_{i+1}, R_{m'_i})$ and $!!(m''_{i+1}, R_{m''_i})$
S13	$l(m_{i+1}, R_{m'_i})$ and $l(m''_{i+1}, R_{m'_i})$ and $!!(m'_i, R_{m'_i})$
S14	$!!(m_i, R_{m_i})$
S15	$l(m'_{i+1}, R_{m_i})$ and $l(m''_{i+1}, R_{m_i})$ and $!!(m_i, R_{m_i})$
S16	$l(m''_{i+1}, R_{m_i})$ and $l(m''_{i+1}, R_{m'_i})$ and $!!(m_i, R_{m_i})$ and $!!(m'_i, R_{m'_i})$
S17	$l(m_{i+1}, R_{m_i}) \mid Sim(m_{i+1}, R_{m_i})$
S18	$!!(m_i, R_{m_i}) \mid \nexists Sim(m_{i+1}, R_{m_i})$
S19	$l(C'_{i+1}, R_{C_i})$ and $!!(C_{i+1}, R_{C_i})$
S20	*null*
S21	$l(C'.m_i, R_{m_i})$ and $!!(C.m_i, R_{m_i})$
S22	$l(C'.m_i, R_{m_i})$ and $!!(C.m_i, R_{m_i})$
S23	$l(C'.m_i, R_{m_i})$
S24	$l(m'_{i+1}, R_{m_i})$

3. Define trace link generation heuristics

Integrate the user in the loop

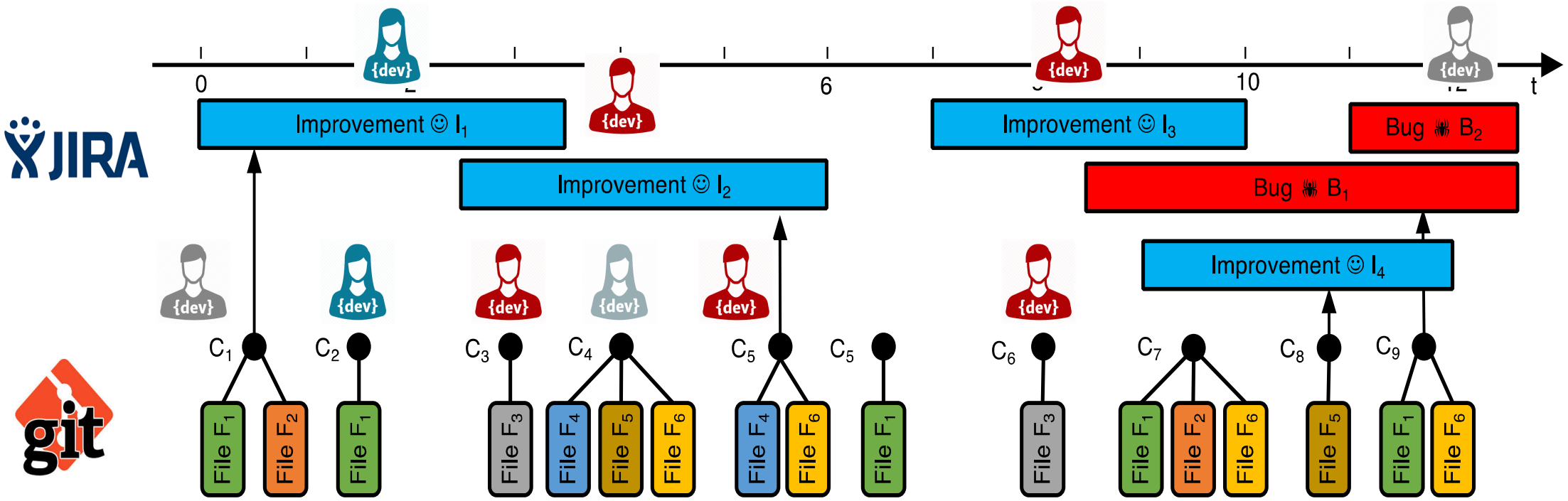


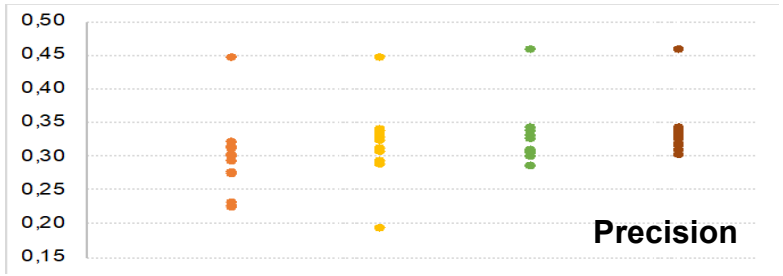


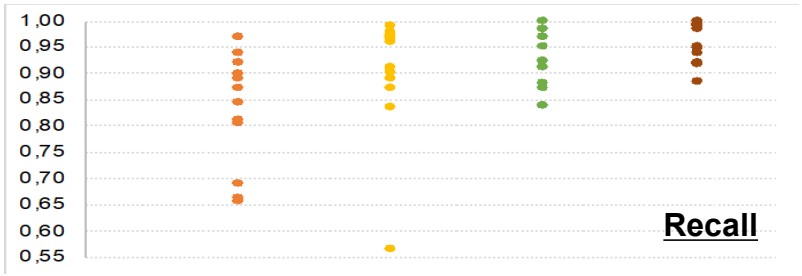
In our experiments the effort required by humans to confirm or deny TLE links was minimal – with few decision points per day.

Evolving software trace links between requirements and source code. Mona Rahimi, Jane Cleland-Huang: Empirical Software Engineering 23(4): 2198-2231 (2018)

Leverage the Project Environment







Traceability in the wild: automatically augmenting incomplete trace links.
Michael Rath, Jacob Rendall, Jin L. C. Guo, Jane Cleland-Huang, Patrick Mäder:
ICSE 2018: 834-845

- Mining project repositories
- Temporal properties
- Machine learning
- Project exhaust!