Assurance Cases

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Charles B. Weinstock
December 2008

Overview

Assurance

A justified measure of confidence that a system will function as intended in its environment of use

Assurance

A justified measure of confidence that a system will function as intended in its environment of use

- What level of confidence do we have as a result of various assurance activities?

Assurance

A justified measure of confidence that a system will function as intended in its environment of use

- Why should we have a particular level of confidence?
- What evidence is there to support this level of confidence?
- Why do we believe the evidence?

Assurance

A justified measure of confidence that a system will function as intended in its environment of use

- Not just the intended environment of use — the actual environment of use

Assurance

A justified measure of confidence that a system will function as intended in its environment of use

- "as intended" by the system's users as they are actually using it
  - Different usage patterns possible by different sets of users

Assurance

A justified measure of confidence that a system will function as intended in its environment of use

- Includes evaluating mitigations of possible causes of critical failures
  - Minimize impact of unusual (or unexpected) operational conditions
  - Minimize impact of vulnerabilities that can be exploited by hostile entities

Assurance Cases

A means of increasing well-founded confidence that a system will behave as intended

- Augments testing where testing by itself is infeasible or too costly
  - Cannot demonstrate system safety/security/performance solely by testing
- Can reduce the number of tests needed to assure a desired system capability because analysis results can complement testing results
- Typically used for safety cases (in Europe)
- Increasing interest in US
  - ISO 15026-2 "Assurance Case" [under development]
  - NRC Report: "Software for Dependable Systems: Sufficient Evidence?"

Purpose of Presentation

Show what Assurance Case technology is and how it can provide

- Help in leveraging scarce assurance resources
- Help in producing a reviewable artifact providing
  - Assurance of mission-critical properties
  - Go/No-go criteria at different stages of development
- Assessment of the impact of changes to a system
- Development of and support for an engineering culture for software
  - Explicit articulation and verification of assurance claims
  - Focus on evidence and its significance

Overview

Definition and Purpose

The System Assurance Problem

Introduction to Assurance Cases

Hazard Analyses and Assurance Cases

Assurance Case Development

Conclusions

Recognition of the Assurance Problem

OSD (AT&L) Software Engineering and System Assurance

- SW test/evaluation lacks rigor and breadth

NDIA Top Software Issues (August 2006)

- 5. Traditional SW verification techniques are costly and ineffective for dealing with the scale of complexity of modern systems
- 6. There is a failure to assure correct, predictable, safe, secure execution of complex SW in distributed environments
- 7 (in part). SW assurance issues are not adequately addressed for COTS/ NDI

NDIA 5: Ineffective SW Verification

Over-reliance on testing rather than robust software verification techniques applied across the life cycle

Current testing techniques scale poorly

Compliance-based testing is inadequate

NDIA 6: Distributed Systems

Assurance of systems of systems cannot be easily inferred from component level assurance

Current techniques for specifying, building, demonstrating, and verifying assured components are inadequate

Exhaustive testing to rule out vulnerabilities is not feasible

NRC Report on SW Dependability

Software for Dependable Systems: Sufficient Evidence?

- Experts from industry and academia assessed current practices for developing and evaluating mission-critical software
- Goals:
  - Identify the kinds of system properties for which certification is desired
  - Identify how certification is obtained today
  - Address system certification, examining a few different application domains (e.g., medical devices and aviation systems) and their approaches to software evaluation and assurance
  - Identify design and development methods, including methods for establishing evidence of trustworthiness

NRC: Dependability Definition

"Dependability: the ability to deliver service that can justifiably be trusted"

- Safety
- Reliability/Availability
- Security

"Dependability is not a local property of software that can be determined module by module

- [It] has to be articulated and evaluated from a systems perspective that [includes] the context of usage"

NRC: Testing Alone Is Insufficient

Testing is indispensable **BUT**

- "A rigorous development process in which testing and code review are the only verification techniques [cannot] justify claims of extraordinarily high levels of dependability"
  - "Rigorous process is essential for preserving the chain of dependability evidence but is not per se evidence of dependability."
- "Execution of even a large set of end-to-end tests, even [with] high levels of code coverage, in itself says little about the dependability of the system as a whole."
- "Credible claims of dependability are usually impossible or impractically expensive to demonstrate after design and development [are complete]"

NRC: Analysis Gives Meaning to Testing

"For testing to be a credible component of a [case for dependability], the relation between testing and properties claimed will need to be explicitly justified"

- Well-reasoned argument
- Formal proofs
- Static code checkers
- Known properties of system internals
- Operational profile
- Quality of the development staff

NRC Recommendation

Assurance that a system is dependable requires the construction and evaluation of a "dependability case"

- Claims
- Arguments
- Evidence
- Expertise

SW Impact on System Dependability

Positive

- Software can sometimes be used to compensate for hardware failures by switching out hardware that's failing
- SW can also detect impending HW failures, signaling the need for preventive maintenance
- Neither of these contributions to overall system dependability is related to detected SW defect rates

Negative

- SW failure rates for complex systems are usually underestimated

SW Failure: Random or Deterministic?

All software failures are deterministic in the sense that they occur every time certain conditions are met but

- After egregious software faults have been removed, failure occurrences become a function of usage patterns and history, neither of which is deterministic
  - Race conditions
  - Accumulation of error
  - Memory leaks
  - Capacity limits
  - Dependence on history
  - Vulnerability exploitation

For complex systems, the software is never perfect

- A system needs to be designed to recover from (currently unknown) critical faults whose effects are encountered only rarely

NRC: Reliability Analysis — SW vs. HW

Feasibility of testing

- HW: continuity of physical phenomena
  - allows inferences to be drawn from a few sample points (tests)
- SW: discontinuous
  - Limited SW testing can rarely provide compelling evidence of behavior under all conditions

Process/product correlation

- HW statistical process control
  - Product samples give evidence of process quality, which determines quality of unsampled items
- SW: process/product quality correlation is generally weak

Dependability Improvement Programs

[Dependability] growth is the improvement in a [dependability] parameter over a period of time due to changes in product design or the manufacturing process. It occurs by surfacing failure modes and implementing effective corrective actions [AMSAA Reliability Growth Guide, TR-652]

- HW reliability improvement activities include the identification of unexpected failure modes and the identification of stress points that are likely points of failure
- For SW, funding is allocated to find and remove code faults, but often there is no software FMEA followed by design modifications to ensure that even when critical software components fail, the likelihood of a critical failure is reduced

Where are the SW Dependability Improvement Programs?

Deficiencies in SW Dependability Approaches

Dependability modeling and analysis is hardly ever done prior to code development

Work focused on improving the robustness of a design, when done, is hardly ever considered a part of product improvement activities

More can be done now than simply developing software dependability goals or making plans about how to use collected defect data to predict field dependability

Overview

Definition and Purpose

The System Assurance Problem

Introduction to Assurance Cases

Hazard Analyses and Assurance Cases

Assurance Case Development

Conclusions

NRC Recommendation

Assurance that a system is **dependable** requires the construction and evaluation of a "**dependability** assurance case"

- Claims
- Arguments
- Evidence
- Expertise

Assurance Cases

An assurance case presents an argument that a system is acceptably safe, secure, reliable, etc. in a given context

- A system could be physical, a combination of hardware and software, or procedural (e.g., operating rules)

Experience with assurance cases has mainly been for safety cases

Argument and Evidence

An assurance case requires claims, evidence, and an argument linking evidence to claims:

- Evidence
  - Results of observing, analyzing, testing, simulating, and estimating the properties of a system that provide fundamental information from which the presence of some system property can be inferred
- High Level Argument
  - Explanation of how the available evidence can be reasonably interpreted as indicating acceptable operation, usually by demonstrating compliance with requirements, sufficient mitigation of hazards, avoidance of hazards, etc.

Problems in System Assurance

Size and complexity
- Increasingly complex systems

Structure/clarity
- Large amounts of relevant information
- Little high level structure, so can be hard to understand
- In many cases, evidence is compelling, but the reader has to work very hard to extract and confirm the rationale for why it is compelling
- Argument and Evidence are confused
  - what is assumed; what is empirically observed?
  - where are subjective judgments made?
- Activities and Argument are confused
  - "What we did" vs. "Why that makes it safe, secure, reliable, etc."

Consistency
- Coordinating effort from many sources (design analyses, development tests, specialty engineering, IV&V efforts)

Approaches to Establish Confidence in Systems

Standards-Based
- Evaluate developer competence based on conformance to process standards

Product-Based
- Claims about product behavior supported by evidence based on product analysis
- Evidence linked to claims by argument
- Beginning to be called an "assurance case" approach

Standards-Based Assurance Approach

Examples: DO-178B for avionics safety; Common Criteria for security

Development processes are evaluated against a standard

- Adherence to good development processes is evidence of ability to produce good products
- Product X has been developed using good development practices
- Therefore Product X is sufficiently safe, secure, reliable, etc.

Product-Based Approach

Example: safety case in UK

Developer creates an assurance case with

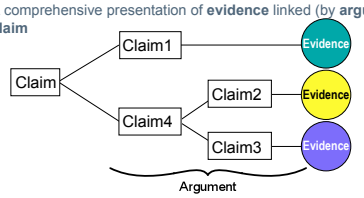
- Explicit claims about system behavior
- Supporting evidence for claims
- Arguments linking evidence to the claims

The case is evaluated by independent assessors

What is an Assurance Case?

A structured demonstration that a system is acceptably safe, secure, reliable, etc.

- A comprehensive presentation of **evidence** linked (by **argument**) to a **claim**



IF ● **THEN** Claim1; **IF** ○ **THEN** Claim2; **IF** ● **THEN** Claim3;
IF Claim2 **and** Claim3 **THEN** Claim4; **IF** Claim1 **and** Claim4 **THEN** Claim

Goal Structuring Notation

Goal Structuring Notation (GSN) was developed to help organize and structure Safety Cases in a readily reviewable form

GSN has been used in Safety Case development for over a decade. A brief overview of its history is in [Kelly 04]

GSN has been successfully used to document Safety Cases for systems such as aircraft avionics, rail signaling, air traffic control, and nuclear reactor shutdown

We have used it to build cases showing that other attributes of interest (e.g., security) have been met.

[Kelly 04] Tim Kelly, and Rob Weaver. The Goal Structuring Notation — A Safety Argument Notation. http://www-users.cs.york.ac.uk/~rob/papers/DSN04.pdf

Goal Structuring Notation

To show how **claims** ▭ are broken down into sub-claims,

and eventually supported by **evidence** ◯

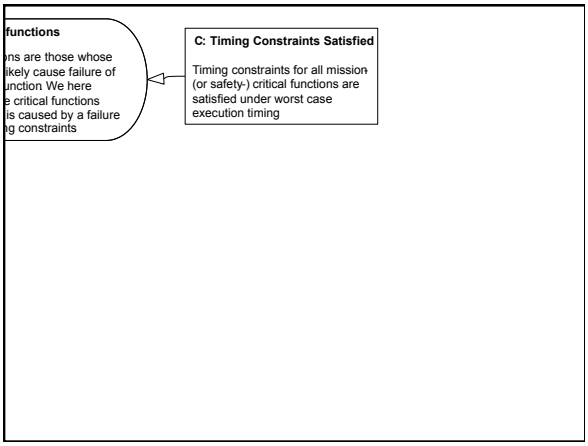
while making clear the argumentation **strategies** ▱ adopted,

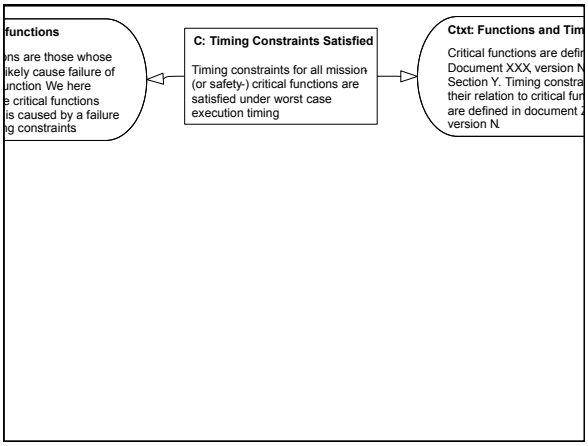
the rationale for the approach (**assumptions, justifications**) ⬭ A/J

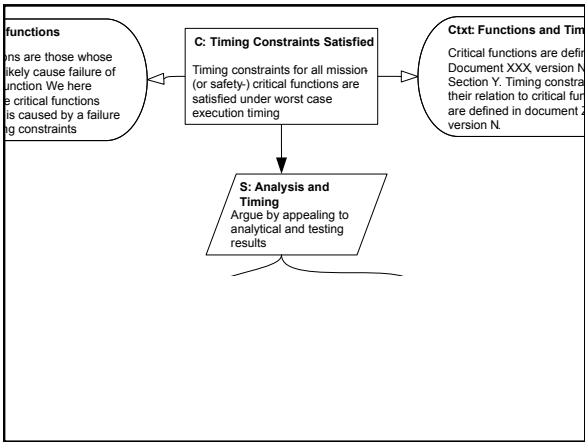
and the **context** ⬭ which claims are stated

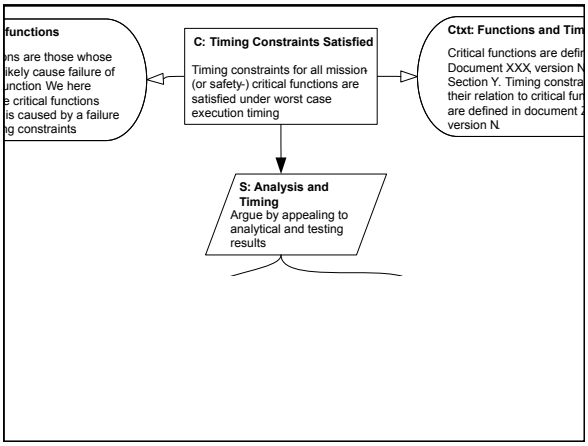
C: Timing Constraints Satisfied

Timing constraints for all mission (or safety-) critical functions are satisfied under worst case execution timing









Presenting Clear Cases

Basic structure

- Claim: what we want to show
  - A proposition: either true or false
- Argument: why we believe the claim is met, based on
- Evidence: test results, analysis results, etc.

In general, arguments are structured hierarchically

- Claim, argument, sub-claims, sub-arguments, evidence
- Easy to show graphically, although can be done in a document or tabular structure

Potential Assurance Case Benefits

Improves comprehension of existing arguments

Improves discussion and reduces time-to-agreement on what evidence is needed and what the evidence means

(Having identified argument structure up front) focuses activities towards the specific end-objectives

Recognition and exploitation of successful (convincing) arguments becomes possible (assurance case patterns)

Supports monitoring of project progress towards successful certification

Overview

Definition and Purpose

The System Assurance Problem

Introduction to Assurance Cases

Hazard Analyses and Assurance Cases

Assurance Case Development

Conclusions

GSN Principles

GSN has methodological rules that help in creating well-structured assurance cases

Cases can be reviewed against these rules to improve their quality

Cases are also reviewed to determine that the content is sound and convincing

GSN Advantages

Captures the elements most critical for arguing a case (claims, evidence, argument strategy, assumptions, relation of claims to subclaims and evidence)

Is reviewable

Can be used at various stages of system development

Gaining increasing interest in the US

- NRC report
- DHS workshop
- ISO draft standard in development
- Medical devices (FDA interest)
- Is required by some organizations in EU and UK

GSN Disadvantages

Lack of familiarity

Lack of examples in non-safety domains

Lack of clarity about costs vs. added value

Concerns about scalability

How to Present Assurance Cases

Possible approaches to inclusion of graphic diagrams:

- In full as Appendix / Annex to document
- Integrated within body of document
  - Claim structure (1 level), Text, Claim structure, Text …
- As 'Executive Summary' at beginning of document
  - Maximum 2 pages of structure, 2-3 levels of decomposition
- As separate, stand-alone, Index Document
  - e.g. to explain argument distributed across many safety case documents

Step 1 – Identify Claims: Top Claims

What is the overall objective of the argument?

- The 'take-home' message for the reader
- Reading the claim structure should convince someone that ...
  - e.g. "System X is Safe"

Things to watch out for

- Jumping ahead
  - "Interlocks fitted to machinery" rather than "Risk associated with Hazard X reduced"
  - The fact that risk is reduced is probably of more interest (to the reader) than how risk has been reduced

Step 1 – Identify Claims: Top Claims

Things to watch out for (cont.)

- Over-simplification
  - "System X is safe" vs.
    "System X is acceptably safe to operate in operating context Y"

The top claim is the seed from which the argument can develop

- If it doesn't contain the right concepts (e.g. acceptability) or jumps–ahead the scope & usefulness of the argument presented can be limited

Not always appropriate to start from 1st principles (depends on audience)

Step 1 – Identify Claims: Phrasing

Claims should be phrased as propositions

- Statements that can be said to be TRUE / FALSE (e.g. "The sky is blue" or "Pittsburgh is a beautiful city")
  - NB: not limited to statements that can be objectively proven
- Statement should be expressed as a single statement (1 sentence) of in the form:
  - <NOUN-PHRASE><VERB-PHRASE>
  - Noun-Phrase identifies the subject of the claim
  - Verb-Phrase defines a predicate over the subject

Step 1 – Identify Claims: Phrasing

Noun-Phrases

- Entities or attributes associated with the argument being made
- For safety arguments, from the following areas:
  - System development – the design method, coding, requirements activities, etc.
  - System design – physical & functional properties of design
  - System operation and maintenance – procedures, roles, etc.
  - Testing, Safety and Hazard Analyses – e.g. fault trees, test results
- Example subjects
  - "Module XYZ123", "Fault Tree for Top Event Y",
    "Testing Activity Z"

Step 1 – Identify Claims: Phrasing

Verb-Phrases

- Predicates over the subjects (qualification)
  - e.g. (over previously listed subjects) "… was developed to Integrity Level 4 guidelines", "… shows probability of event Y occurring < A", "… achieved 80% coverage of system functions"

Tense

- Claims should be written in a tense appropriate for the intended time of reading
  - e.g. for initial certification – past tense for development, present tense for system attributes, future tense for operation and maintenance
  - "System was written in SPARK-ADA subset",
    "Likelihood of hazard X is 1x10-6",
    "Maintenance will be carried out every Y days"

Step 1 – Identify Claims: Phrasing

Tense (cont.)

- Claims should be phrased as positive statements of objectives achieved – not requirements or aspirations
  - "Failure Rate is less than $1x10^{-6}$" rather than
    "Failure Rate must be less than $1x10^{-6}$"
- Why?
  - For the final argument, it is what the reader wants to know
  - To avoid (necessarily) having to re-phrase the claim structure
  - When developing the structure, a statement of achievement can be read as a claim to be achieved (i.e. stating a claim as an achievement doesn't pose a significant problem early in the lifecycle)

Step 1 – Identify Claims: Phrasing

The following are examples of correctly stated claims:

Subject <Noun-Phrase>	Predicate <Verb Phrase>
Component X	has no critical failure rates
All identified hazards for System Y	have been sufficiently mitigated
Non-destructive examination of weld-site Z	has been performed
Design A	employs triple modular redundancy

Step 1 – Identify Claims: Phrasing

The following are examples of *incorrectly* stated claims:

Claim:	Reason:
"Hazard Log for System Y"	*Noun Phrase — describes an entity — not a statement*
"Fault Tree for Hazard H-1"	*As above*
"Perform Fault Tree Analysis of Hazard H-1"	*Verb Phrase — an action — not a statement*
"How many failure modes does component X have?"	*Question — not a statement*

Test: can we say claim is TRUE/FALSE?

Step 1 – Identify Claims: Phrasing

Difficulties in Summarizing Claims:

- Sometimes, it can be difficult to 'sum-up' a claim in a single succinct statement
  - e.g. a safety requirement that is described in 1/2 page in the standard
- One approach, is to reference out:
  - e.g. "Requirement 6.3 has been met"
- However, we would still like to make the claim structure readable, so even the briefest description helps
  - e.g. "Requirement 6.3 (Defence in Depth) has been met"

Step 2 – Define basis for claims: Context

Having presented a claim, make clear (unambiguous) the basis on which that claim is stated

- When a claim talks of hazards, components, requirements, fault trees, acceptability, sufficiency … is it clear what is being referred to?

Claims are rarely objective 'context-free' statements (especially when terms such as tolerable and negligible are used)

The aim is to ensure that both writer and reader have same understanding

For example, it is not helpful to state the claim…

- "Requirement 6.3 has been met"

… if it is unclear to the reader what "Requirement 6.3" refers to

Step 3 – Identify strategy

Next step is to work out how to substantiate the stated claim

- "What reasons are there for saying the claim is TRUE?"
- "What statements would convince the reader that the claim is TRUE?"

Aiming for statements that are easier to support than the larger claim

- Breaking into a number of smaller claims – i.e. Divide-and-Conquer
- Relating claim more closely to specific application in question (e.g. for a generic requirement)

Step 3 – Identify strategy: Phrasing

The role of a strategy node is to clearly explain the relationship between a claim and a set of sub-claims

An analogy:

Strategy

$3xy^3 + 2x^2y^2 + 5xy = 17y$ *(Divide both sides by y)*

$3xy^2 + 2x^2y + 5x = 17$

Strategy statement should succinctly describe the argument approach adopted, ideally in the form:

- "Argument by … <approach>"

Step 3 – Identify strategy

Where do strategies come from?

For a safety argument, sources of information are:

- The design itself
- Analysis and Testing results
- NB: not just supporting claims, but also structuring argument

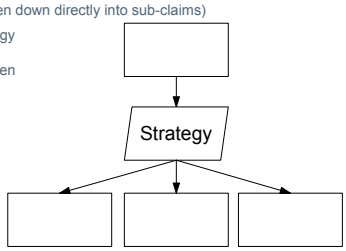
However, can also simply be a question of presentation

- Especially true at the high levels
- (Analogous with Fault Trees – many different trees produce the same cutsets)
- In these cases
  - Look within the argument (claims, context) already outlined
  - Bottom line: aiming for a clear breakdown

Step 3 – Identify strategy:

Strategies can be implicit or explicit

- Implicit examples already shown (Claims broken down directly into sub-claims)
- Explicit strategy is a node placed between parent and child claims



Step 3 – Identify strategy: Phrasing

Example statements:

- "Argument by appeal to test results"
- "Argument by consideration of historical data"
- "Quantitative argument using simulated run data"

Noun-Phrase descriptions of the claim decomposition

- Not actually part of the argument themselves
- But help in understanding the argument presented

Step 3 – Identify strategy

Q: When is it necessary to explicitly introduce a strategy node?

A1: Whenever you wish to explain the relationship between a claim and its sub-claims

- Ask yourself whether the reader will understand how you have broken down the claim into sub-claims

A2: Whenever the strategy requires additional (contextual) information, justification or assumptions

Step 3 – Identify strategy: Phrasing

Things to watch out for:

- Strategies should not be imperative verb-phrases
  - e.g. "Use Historical Data"
- Strategies should be expressed from the perspective of the argument approach, not the design, testing, or analysis approach
  - e.g., "Argument by appeal to interlock" rather than "Interlocks used"
- Strategies should not contain claims
  - Should be possible to remove strategy nodes and not affect the argument being made

Step 4 – Define basis for strategy

In the same way as is necessary for claims, must examine what contextual information (including models) is required

Same process – examine strategy for terms or concepts introduced but not 'bound'

- e.g. for sub-system breakdown strategy the term 'All identified sub-systems" is used

Ask what information is required in order to expand or fulfill strategy outlined

Step 4 – Define basis for claims: Rationale

Additionally, it may be useful to explicitly represent the following:

- Assumptions – are there any assumptions on which the strategy | claim is being put forward as a solution to the parent claim?
- Justification – why that particular strategy | claim is being put forward as a solution to the parent claim?

Both assumptions and justifications are statements (like claims) and therefore should be expressed as NOUN-PHRASE VERB-PHRASE

Step 5 – Elaborate strategy

Having identified an approach, it is necessary to lay out the claims that fulfill that approach, e.g.

- for strategy ranging over all sub-systems – expand for claims over each individual sub-system
- for quantitative results strategy – provide quantitative claim statements

In elaborating the strategy, define claims

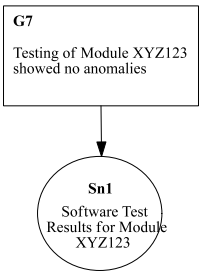
If strategy, and basis of strategy, are clear – this step can be straightforward

Step 6 – Identify Solutions

Eventually, faced with a claim that doesn't need further expansion, refinement, explanation …

In such cases, simply have to reference out to information that supports claim by means of solution

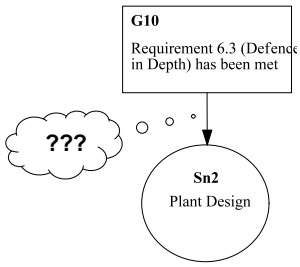
As references, solutions should be NOUN-PHRASEs



Step 6 – Identify Solutions

Possible Mistake:

- 'Jumping' to a solution too soon
- The relationship between claim and the referenced information should be obvious to both writer and reader



Overview

Definition and Purpose

The System Assurance Problem

Introduction to Assurance Cases

Hazard Analyses and Assurance Cases

Assurance Case Development

Conclusions

The System Assurance Problem

Systems are getting more complex and more dependent on software

- Reaching sound conclusions about dependability is getting harder

Traditional methods for evaluating dependable behavior are increasingly inadequate

- Too costly (in time and money) to test complex systems well
- Testing is not the best way of showing impact of subtle, but critical errors
- Constraining interactions among system components can make it easier to increase dependability but may be hard to find constraints consistent with required functionality

We need better ways of integrating a variety of analyses (evidence) into assurance cases, i.e.,

- We need better means of showing how certain evidence supports conclusions about system properties

Final Thoughts

Testing by itself is inadequate for reaching valid conclusions about SW dependability in complex systems

Assurance case must

- Integrate design analyses focused on SW hazards and FMEA
- Be reviewable

Assurance case evaluation criteria are currently subjective

- Need more data on which subtle defects are worth analysis efforts
- Need more understanding of what makes dependability arguments sound

Assurance case patterns hold promise of capturing "valid" arguments and guiding dependability improvement efforts

Purpose of Presentation

Show what Assurance Case technology is and how it can provide

- help in leveraging scarce assurance resources
  - what part of case is most important to gaining assurance?
  - where can work be leveraged (e.g., aspects of requirements analysis)
- help in producing a reviewable artifact providing
  - assurance of mission-critical properties (the claims)
  - go/no-go criteria at different stages of development
- assessment of the impact of changes to a system
  - which part of the case is affected
- development of and support for an engineering culture for software
  - explicit articulation and verification of assurance claims
  - focus on evidence and its significance

Conclusions

Within conventional assurance reports the 'chain of argument' can often get lost

- But the argument is more important than the document!

GSN (and assurance cases) have been found to be a useful basis for mapping out and evolving the structure of the Arguments

- Provides a roadmap for a document or set of documents
- Provides a basis for discussion among engineers and between developers and assessors
- Creating outline arguments at the beginning of a project helps show progress towards a final solution

Contact Information

Charles B. Weinstock
System of Systems
Software Assurance Project
Telephone: +1 412-268-7719
Email: weinstock@sei.cmu.edu

U.S. mail:
Software Engineering Institute
Customer Relations
4500 Fifth Avenue
Pittsburgh, PA 15213-2612
USA

World Wide Web:
www.sei.cmu.edu

Customer Relations
Email:
customer-relations@sei.cmu.edu
Telephone: +1 412-268-5800

Software Engineering Institute | Carnegie Mellon

Overview

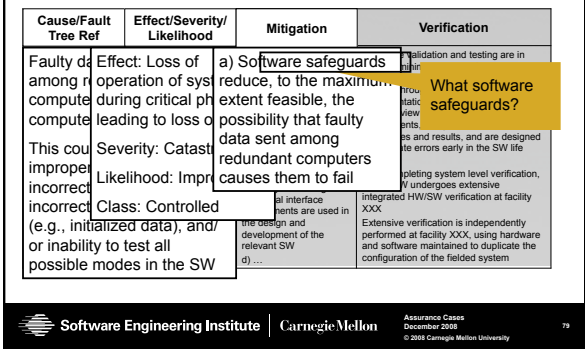
Definition and Purpose
The System Assurance Problem
Introduction to Assurance Cases
Hazard Analyses and Assurance Cases
Assurance Case Development
Conclusions

From Hazard Report to Assurance Case

Cause/Fault Tree Ref	Effect/Severity/ Likelihood	Mitigation	Verification
Faulty data exchanged among redundant computers causes all computers to fail. This could occur because of Improper requirements, incorrect coding of logic, incorrect data definitions (e.g., initialized data), and/ or inability to test all possible modes in the SW	Effect: Loss of operation of system during critical phase, leading to loss of life. Severity: Catastrophic Likelihood: Improbable Class: Controlled	a) Software safeguards reduce, to the maximum extent feasible, the possibility that faulty data sent among redundant computers causes them to fail b) Program Development Specifications and Functional SW Requirements c) Subsystem design and functional interface requirements are used in the design and development of the relevant SW d) ...	Extensive validation and testing are in place to minimize generic SW problems. The contractors must perform rigorous reviews throughout the SW definition, implementation, and verification cycles. These review processes cover requirements, design, code, test procedures and results, and are designed to eliminate errors early in the SW life cycle. After completing system level verification, critical SW undergoes extensive integrated HW/SW verification at facility XXX Extensive verification is independently performed at facility XXX, using hardware and software maintained to duplicate the configuration of the fielded system

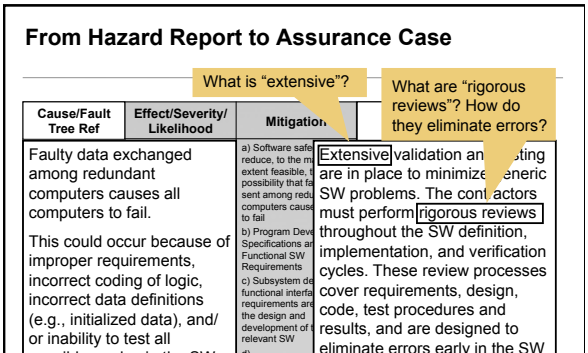
Portion of a typical, traditional, software-related hazard report

From Hazard Report to Assurance Case



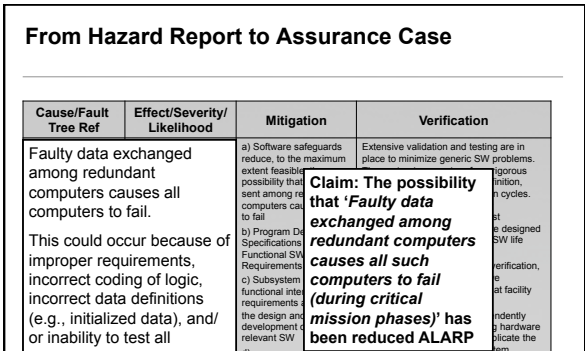
From Hazard Report to Assurance Case

From Hazard Report to Assurance Case



From Hazard Report to Assurance Case

From Hazard Report to Assurance Case



From Hazard Report to Assurance Case

From Hazard Report to Assurance Case

Cause/Fault Tree Ref	Effect/Severity/ Likelihood	Mitigation	Verification
Faulty data exchanged am redundant computers cau all computers This could occ because of im requirements, incorrect codin logic, incorrect definitions (e.g initialized data or inability to t possible mode the SW	Effect: Loss of operation of system during critical phase, leading to loss of life. Severity: Catastrophic Likelihood: Improbable Class: Controlled	are safeguards to the maximum basible, the ty that faulty data ong redu ers cause am Deve ations and nal SW ments ystem design and al interface ents are used in the design and development of the relevant SW d) …	Extensive validation and testing are in place to minimize generic SW problems. The contractors must perform rigorous reviews throughout the SW definition, on cycles. st re designed to eliminate errors early in the SW life cycle. After completing system level verification, critical SW undergoes extensive integrated HW/SW verification at facility XXX Extensive verification is independently performed at facility XXX, using hardware and software maintained to duplicate the configuration of the fielded system

Claim context and assumptions

AC to Resolve These Problems

C: Data Exchange Haz Mitigated

The possibility that 'Faulty data exchanged among redundant computers causes all such computers to fail (during critical mission phase)' has been reduced ALARP

C: Reliable Backup Computing

The likelihood of complete failure of primary and backup computers during a critical mission phase has been reduced ALARP

Ctxt: Critical Phases

Critical phases of system operation are X, Y, and Z (as defined in document DD)

C: Reliable Backup Computing

The likelihood of complete failure of primary and backup computers during a critical mission phase has been reduced ALARP

Ctxt: Effect/Severity/Likelihood

Effect: Loss of operation of system during critical phase, leading to loss of life
Severity: Catastrophic
Likelihood Improbable

C: HW Hazards Mitigated

HW hazards leading to complete failure of all redundant computers have been reduced ALARP

C: SW Hazards Mitigated

SW hazards leading to complete failure of all redundant computers have been reduced ALARP

Ctxt: Hazards

Faulty data exchanged among the redundant computer set; incorrect data initializations …

C: Data Exchange Haz Mitigated

The possibility that 'Faulty data exchanged among redundant computers causes all such computers to fail (during critical mission phase)' has been reduced ALARP

C: Other SW Hazards

Other SW hazards are (e.g., data initialization coding leading to fault transmissions, etc.)

S: Da

Argue data ex redund why the be redu

a Exchanges

need to exchange redundant

ngine

C: Minimal Data Exchange

Data exchanges among

eMellon

Assurance Case Benefits

It's not that we don't already produce this data, but we don't present it effectively

- The motivation for some activities and decisions is not evident to outside reviewers or new personnel

Effective presentation can

- Motivate consideration of different designs
  - Ways of eliminating data exchanges (requirements scrub)
- Explain why certain evidence is critically important
  - Code reviews showing that data validation checks are done
  - Test results showing invalid data is rejected
- Help prioritize activities that contribute to the argument

From Hazard Report to Assurance Case

Cause/Fault Tree Ref	Effect/Severity/ Likelihood	Mitigation	Verification
Faulty data exchanged among redundant computers causes all computers to fail. This could occur because of improper requirements, incorrect coding of logic, incorrect data definitions (e.g., initialized data), and/ or inability to test all possible modes in the SW	Effect: Loss of operation of system during critical phase, leading to loss of life. Severity: Catastrophic Likelihood: Improbable Class: Controlled	a) Software safe reduce, to the m extent feasible, t possibility that fa sent among redu computers caused them to fail b) Program Development Specifications and Functional SW Requirements c) Subsystem des functional interfac requirements are the design and development of the relevant SW d) …	Extensive validation and testing are in place to minimize generic SW problems. … requirements, design, code, test procedures and results, and are designed performed at facility XXX, using hardware and software maintained to duplicate the configuration of the fielded system

Evidence should point to each test and test result related to verification of each safeguard

When to Visualize the Argument?

Q: At what stage in a project is it worth visualizing the argument?

Answers:

- Early on (high level) to get a clear picture (and gain agreement) of argument structure
  - Useful as a scoping exercise and effort allocation
- As project is progressing, in order to monitor status towards completion of an acceptable argument
- At end of project in order to present the final argument and evidence that exists