# Evaluating Hazard Mitigations with Dependability Cases

John B. Goodenough, Ph.D.[1]
*Software Engineering Institute, Pittsburgh, PA, 15123*

Matthew R. Barry, Ph.D.[2]
*Software-Intensive Systems, Inc., Altadena, CA, 91001*

**There is growing interest in using a structure of claims, arguments, and evidence to explain why all critical software hazards have been eliminated or adequately mitigated in mission-critical and safety-critical systems. Such a structure has been called a dependability case, an assurance case, or a (goal-structured) safety case. Dependability cases are sometimes viewed as adding no extra value, e.g., given an existing hazard analysis, what is the added value of a dependability case showing how the hazard is mitigated? In this paper we present an example to show the value a dependability case adds to a traditional hazard analysis.**

## I. Introduction

A structure of claims, arguments, and evidence is being increasingly recommended as a good way to explain why all critical software hazards have been eliminated or adequately mitigated in mission-critical and safety-critical systems. A recent National Research Council report, "Software for Dependable Systems: Sufficient Evidence?"[1]strongly recommended this approach. In addition, the increasing interest in this approach has led to the development of an ISO technical report currently undergoing review as a Committee Draft.[2] The claims-argument-evidence approach has been called a *dependability case*,[1] an *assurance case*,[2] or a (goal-structured) *safety case*.[3] The approach has been used in the UK for more than a decade to express safety cases for avionics, nuclear, and railway control systems.[4]

In the best practice, an engineering organization will start a dependability case early in the development life cycle, using the case's structure to influence assurance-centered actions throughout the life cycle. Constructing a case early in development can help determine what claims are most critical and hence, what evidence and assurance-related activities are most needed to support such claims. Developing a dependability case early can also help guide design decisions that simplify the case, e.g., by making it easier to develop a convincing argument or convincing evidence that an important claim holds. (This might be called *assurance-guided* engineering.) In any case, the resulting product is useful for supporting certification (and re-certification) decisions, managing dependability resources and activities (by showing which activities have the most payoff for claims of particular importance), estimating the impact of design and requirements changes (by showing which portions of the case may be affected), and focusing and communicating engineering expertise.

Many safety-oriented development organizations already conduct some form of software hazard analysis. In this paper, we provide an extended example showing how a dependability case adds useful information to a hazard analysis. The example shows the power of the claims-arguments-evidence structure to clarify the hazards and show why the selected mitigations are effective. Reviewers and project engineers can better understand why a mitigation is effective, see the supporting evidence, and have more trust in the behavior of the system during operation. The example also shows how the engineering, safety, and dependability organizations can migrate toward the use of a common analysis product that addresses the needs of each. Although we focus in this paper on a particular form hazard analysis, a dependability case can similarly complement other forms such as fault tree analyses, failure modes and effects analyses, Systems-Theoretic Accident Modeling and Processes (STAMP)-based hazard analyses (STPA), etc.

A traditional software hazard report has several parts. For each identified hazard, the report includes a description of the cause of the hazard, a description of the effect of the cause along with a report of its severity and likeli-

[1] Institute Fellow and System of Systems Software Assurance Initiative Lead, Software Engineering Institute, 4500 Fifth Avenue, Pittsburgh, PA, 15213, USA.
[2] President, Software-Intensive Systems, Inc., 1575 Homewood Drive, Altadena, CA, 91001, USA.

hood, an enumeration of the controls emplaced to mitigate the hazard, and a description of any verifications of the controls. In the past, the narrative or analysis in the report may have suffered from disconnects with actual engineering activities. The controls may in fact be implemented differently than described, or described in such a way that the connection to architecture, design, or test features is unclear. Accordingly, the verification description is likewise disconnected and potentially difficult to repeat. It remains implicit that the controls address the hazard. The result is an analysis that may lack credibility while purportedly offering trustworthiness. The creation of an explicit dependability case can correct these problems.

The remainder of this paper is structured around an example dependability case, starting with analysis of an example hazard report. We first discuss the hazard report and then explain the notation we will be using to document an associated dependability case. We then develop the case and discuss its implications.

## II. An Example

We consider a portion of a software-related hazard report (Table 1) extracted from an actual product, with certain irrelevant identifying details omitted. The report deals with a system in which the possibility of a computer failure is so serious that the computers are replicated — if one computer fails, one of several backup computers will take over. The system controlled by these computers goes through a number of mission phases; failure of all the computers would be more critical in some phases than others.

For the backup computers to be able to take over for a failing computer, they must exchange some data (e.g., to determine which computer is currently in control). The potential fault of concern is the possibility that one computer could send bad data to one or more of the other computers, eventually causing all interconnected computers to fail.

To determine whether this hazard has been mitigated, we analyze the hazard report from the viewpoint of understanding what claims, arguments, and evidence are relevant to the hazards, controls, and verification actions. This analysis will show certain inadequacies in the report, but more importantly, will lead to a dependability case structure that is more convincing in explaining why certain controls and verification activities are relevant.

**Table 1. Example Hazard Analysis.**

| Cause/Fault Tree Reference | Effect/Severity/ Likelihood | Controls | Verification |
|---|---|---|---|
| Faulty data exchanged among redundant computers causes all computers to fail.<br><br>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 software (SW) | Effect: Loss of operation of system during critical phase, leading to loss of life.<br><br>Severity: Catastrophic<br><br>Likelihood: Improbable<br><br>Class: Controlled | a) Software safeguards reduce, to the maximum extent feasible, the possibility that faulty data sent among redundant computers causes them to fail<br><br>b) Program Development Specifications and Functional SW Requirements<br><br>c) Subsystem design and functional interface requirements are used in the design and development of the relevant SW<br><br>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.<br><br>After completing system level verification, critical SW undergoes extensive integrated HW/SW verification at facility XXX<br><br>Extensive verification is independently performed at facility XXX, using hardware and software maintained to duplicate the configuration of the fielded system |

### A. Analyzing the Report

The hazard/fault to be addressed is "Faulty data exchanged among redundant computers causes all computers to fail." The second column in the report specifies the significance of this fault, namely, "Loss of operation of system during a critical phase, leading to loss of life." The standard ways of characterizing the impact and likelihood of the fault follow, together with a statement about the disposition of the hazard — "Controlled," i.e., the system has been analyzed to ensure that this potential hazard is under control.

The "Controls" column summarizes several approaches to eliminating the fault or reducing its likelihood or consequence of occurrence. The first control states "a) Software safeguards reduce, to the maximum extent feasible, the possibility that faulty data sent among redundant computers causes them to fail." The report does not indicate what these software safeguards are or how they might be effective in controlling the hazard, nor does the report point to a place where this information might be found. Even if the report were more specific about the nature of the safeguards, it would not necessarily be clear about what is being defended against. We will return to this point when we discuss the dependability case developed to address this hazard.

The second and third controls are less informative:

> "b) Program Development Specifications and Functional Software Requirements"
> "c) Subsystem design and functional interface requirements"

How do specifications and requirements control for this hazard? Which specifications? Which requirements?

The verification paragraphs in the report are also uninformative:

> "Extensive validation and testing are in place to minimize generic software problems. The contractors must perform rigorous reviews throughout the software 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 software life cycle."

What does "extensive validation and testing" mean? What kinds of "rigorous reviews" will be conducted, and how are we to know whether the reviews eliminate the possibility of design or implementation errors leading to fatal exchanges of faulty data? What are the reviewers supposed to look for, and how could we be certain that the reviews have been done with sufficient care and completeness?

Someone who is sufficiently well versed in the described system surely knows more than what the words in the report convey. The answers may be available elsewhere, in other documents, or in other parts of this hazard report, but we are left to find them for ourselves. Moreover, the controls and verification actions are only implicitly related to the cause/fault. But these ambiguities and suggestive connections make the certification assessment more difficult. The missing relationships can be made more explicit by constructing claims and evidence in a dependability case.

## B. The Notion of a Dependability Case

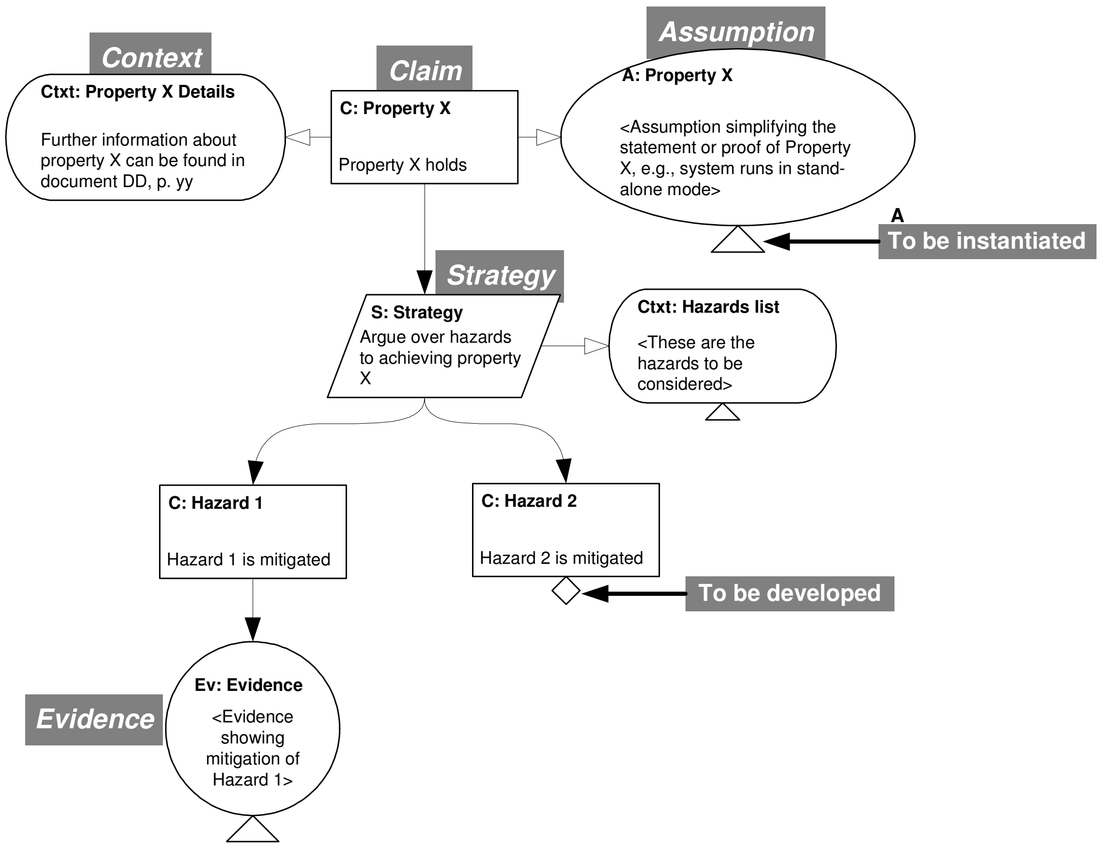
A dependability case consists of a top-level claim supported by subclaims. Each subclaim is further decomposed into sub-subclaims, and so on, until a claim is directly supported by evidence, i.e., data that is sufficient to support a claim without further argument. Such evidence might consist of test results, analyses, information about the competency of personnel, etc. The quality of a case (i.e., its soundness and the extent to which it is convincing in supporting its top-level claim) depends on the claim structure and, of course, the quality of the presented evidence.

A dependability case is somewhat similar in nature to a legal case. In a legal case there are two basic elements. The first is evidence, be it witnesses, fingerprints, DNA, etc. The second is an argument given by the attorneys as to why the jury should believe that the evidence supports (or does not support) the claim that the defendant is guilty (or innocent). A jury presented with only an argument that the defendant is guilty, with no evidence that supported that argument, would certainly have reasonable doubts about the guilt of the defendant. A jury presented with evidence without an argument explaining why the evidence was relevant would have difficulty deciding how the evidence relates to the defendant.

The dependability case is similar. Suppose there are analyses and test results developed to support a claim of safety. Without an argument as to why the analyses and test results support the safety claim, an interested party could have difficulty seeing its relevance or sufficiency. With only an argument that a system was safe, but without supporting analyses and test results, again it would be hard to certify the system's safety.

In the dependability case example we will present, the top-level claim will be that the hazard addressed in Table 1 has been adequately mitigated. From that claim will flow an argument that supports this mitigation claim. The argument will consist of one or more subsidiary claims that taken together give confidence in the truth of the top level claim. These lower-level claims will themselves be supported by additional claims until finally a subclaim is justified directly by evidence that clearly shows the subclaim to be true.

To document a dependability case, we use Goal Structuring Notation (GSN).[3] Fig. 1 illustrates the notation. *Claims* are specified in rectangular elements and are stated as predicates (i.e., true or false statements). Stating claims as predicates is very important—it sharpens thinking to state precisely what is to be proved with further argument or evidence. In the Figure, the top-level claim is labeled "Property X" and the claim is "Property X holds." We will give system-specific claims in the example that is discussed later.



Context
Ctxt: Property X Details
Further information about property X can be found in document DD, p. yy
Claim
C: Property X
Property X holds
Assumption
A: Property X
<Assumption simplifying the statement or proof of Property X, e.g., system runs in stand-alone mode>
A
To be instantiated
Strategy
S: Strategy
Argue over hazards to achieving property X
Ctxt: Hazards list
<These are the hazards to be considered>
C: Hazard 1
Hazard 1 is mitigated
C: Hazard 2
Hazard 2 is mitigated
To be developed
Evidence
Ev: Evidence
<Evidence showing mitigation of Hazard 1>

**Figure 1. Example showing GSN shapes and their meaning.**

The "Property X" claim is amplified by a *context* element (a rounded rectangle). Context elements provide additional information to help others understand what is being stated, allowing the claim to be stated more simply, e.g., in a fashion that is clear to someone knowledgeable of the system. With respect to the "Property X" claim, the context element provides a pointer to documents where the property is discussed in more detail. Some dependability case tools provide a hyperlink to such additional information.

The "Property X" claim is also associated with an *assumption* element (an oval labeled with an "A"). These elements are useful for documenting assumptions that might otherwise not be obvious to reviewers. Also, characterizing something as an assumption simplifies the case by allowing the argument to focus on claims that require more attention and work to support. If a reviewer does not think an assumption is appropriate, then additional argumentation can be requested and documented in a separate dependability case in which the assumption is the top claim. Another value in documenting assumptions is that when the system changes, the assumptions can be reviewed to see if they are still valid in light of the changes.

In Fig. 1, the assumption element contains a meta-statement describing what a real assumption statement could contain. To show clearly that this is a meta-statement, we enclose the statement in brackets (<>) and indicate that the statement needs to be instantiated with an actual assumption by putting a triangle at the bottom of the element. The notion of instantiation is particularly useful when describing dependability case patterns, i.e., argumentation structures that occur often in constructing cases in certain domains, but it is also useful when giving an example case that is divorced from an actual system or for which actual details are not yet known.

The "Property X" claim is supported by further argumentation. A *strategy* element (a parallelogram) describes the argumentation approach that will be used in supporting the "Property X" claim. A strategy element is not actually part of the argument; instead, it helps the reader understand the line of thinking supporting a claim. Strategy elements are introduced when considered helpful in guiding reviewers. In the Figure, we propose to argue that each hazard relevant to property X has been eliminated or controlled. To further explain the strategy, we associate a context element with the strategy, listing the hazards to be considered or referring to the portion of a document in which each of the hazards is discussed in more detail. Alternatively, we could associate the strategy with a *justification* element (not present in Fig. 1). A justification element summarizes an argument explaining why the chosen strategy is considered appropriate. An example will be shown later. A justification element is an oval (like an assumption element), but labeled with “J” instead of “A.”

The argument showing that Property X holds consists of two claims asserting that the two hazards considered relevant to Property X have been mitigated. For Hazard 1, *evidence* is provided that is considered to be sufficient to demonstrate that the hazard has been adequately mitigated. Evidence elements are shown as circles. For Hazard 2, a further argument needs to be developed. This is indicated by the diamond hanging off the bottom of the claim element.

There are other GSN shapes and structures, but they are not needed for the dependability case given in this paper.

## C. A Dependability Case for the Hazard

To develop a dependability case for the hazard described in Table 1, we start by developing a top-level claim stating that the hazard is mitigated (see Fig. 2). Since claims are expressed as predicates expressing what we want to be true about a system, we state what we mean when we say that the hazard has been mitigated, namely:

> The possibility that faulty data exchanged among redundant computers causes all such computers to fail (during critical mission phases) has been reduced ALARP [ALARP means “as low as reasonably practicable,” a term used to indicate that further effort to reduce risk would be significantly disproportionate to the amount of risk reduction that can be achieved.[5]]

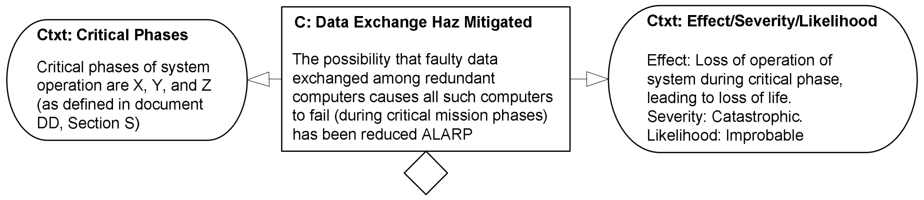

Ctxt: Critical Phases
Critical phases of system operation are X, Y, and Z (as defined in document DD, Section S)
C: Data Exchange Haz Mitigated
The possibility that faulty data exchanged among redundant computers causes all such computers to fail (during critical mission phases) has been reduced ALARP
Ctxt: Effect/Severity/Likelihood
Effect: Loss of operation of system during critical phase, leading to loss of life. Severity: Catastrophic. Likelihood: Improbable

**Figure 2. Initial top-level claim.**

We have restated the Table 1 hazard slightly to say that we are primarily concerned with the occurrence of this hazard during “critical” mission phases. The intended implication is that there are some periods of system operation in which failure of all computers would not be life threatening. By constraining the claim to certain phases of the system’s operation, we ensure that assurance efforts are focused where they are needed the most.

We use context elements in Fig. 2 to state additional relevant information, namely, where to find the definition of “critical” mission phases and the effect, severity, and likelihood of the hazard. Note how one of the context elements provides a pointer to a detailed discussion of what phases are critical. If later, during the project, there is a change to the set of critical phases, this context element provides a hook to show that support for this claim may have to be revisited.

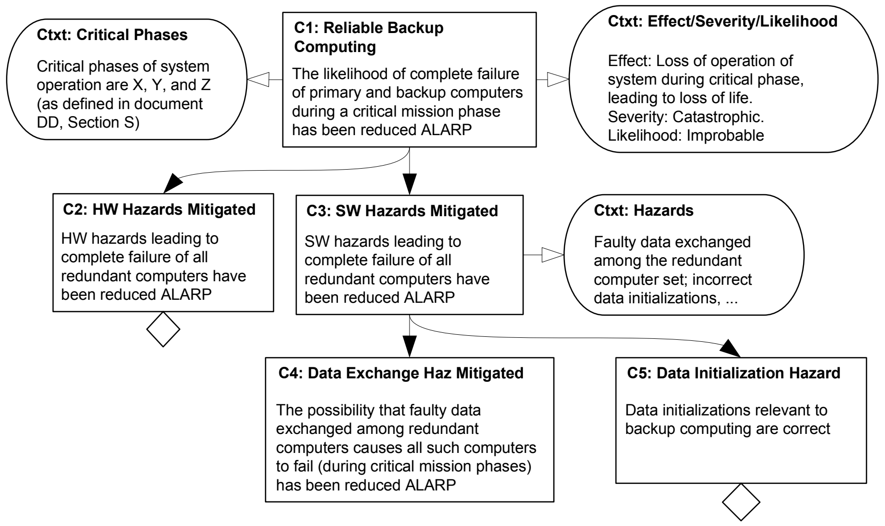

Ctxt: Critical Phases
Critical phases of system operation are X, Y, and Z (as defined in document DD, Section S)
C1: Reliable Backup Computing
The likelihood of complete failure of primary and backup computers during a critical mission phase has been reduced ALARP
Ctxt: Effect/Severity/Likelihood
Effect: Loss of operation of system during critical phase, leading to loss of life. Severity: Catastrophic. Likelihood: Improbable
C2: HW Hazards Mitigated
HW hazards leading to complete failure of all redundant computers have been reduced ALARP
C3: 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, ...
C4: Data Exchange Haz Mitigated
The possibility that faulty data exchanged among redundant computers causes all such computers to fail (during critical mission phases) has been reduced ALARP
C5: Data Initialization Hazard
Data initializations relevant to backup computing are correct

**Figure 3. A more comprehensive top-level claim (C1) showing the context in which the “C4: Data Exchange Haz Mitigated” claim sits.**

Although this claim and its associated context elements suffice to capture key parts of the hazard report, this claim is only part of a larger claim about the survivability of the redundant computer set. To show the context in which the data exchange hazard control exists, we formulate a new top-level claim: "The likelihood of complete failure of primary and backup computers during a critical mission phase has been reduced ALARP." We will attach Fig. 2's context elements to this new top-level claim (see claim C1 in Fig. 3), and then address hardware and software hazards in separate claims (C2 and C3). The original data exchange hazard (C4) is now shown to be only one of the software hazards relevant to achieving reliable backup computing.

The structure in Fig. 3 shows a general stylistic pattern that is useful — the context for the claim "SW Hazards Mitigated" names the hazards. This allows a reviewer to consider if all relevant hazards have been identified before the reviewer gets into deciding if each hazard is adequately controlled individually. Alternatively, for complicated situations, the context element might refer to a separate dependability case supporting the claim, "All critical software hazards have been identified."

All software hazards that cannot be eliminated and are worth controlling are addressed under the "SW Hazards Mitigated" claim. The example shows two software hazard mitigation claims, C4 and C5: one deals with faulty data exchanges and the other concerns data initialization hazards. The dependability case for mitigating the data exchange hazard is shown in Fig. 4. We'll walk through this diagram to explain the argument.

The two subclaims (C6 and C7) under the top-level claim C4 (Data Exchange Haz Mitigated) show the selected approach, which is based on an actual approach taken in one project.[6] The approach is two-fold: 1) reduce the amount of data that needs to be exchanged among the redundant computer set, particularly during critical mission phases ("C6: Minimal Data Exchange"), and 2) build in safeguards that help ensure that generated faulty data will be rejected ("C7: Faulty Data Detected").

The justification for minimizing data exchanges is shown in an attached "justification" element ("J: Minimize Data Exchanges"). This element explains why we believe that minimizing data exchanges is part of a reasonable approach for controlling the data exchange hazard, namely, that by reducing the amount and need for data exchanges we reduce the opportunity for such exchanges to cause a cascading failure of the redundant computers. We can't eliminate all data exchanges — a certain amount of data must be exchanged to allow a backup computer to take over

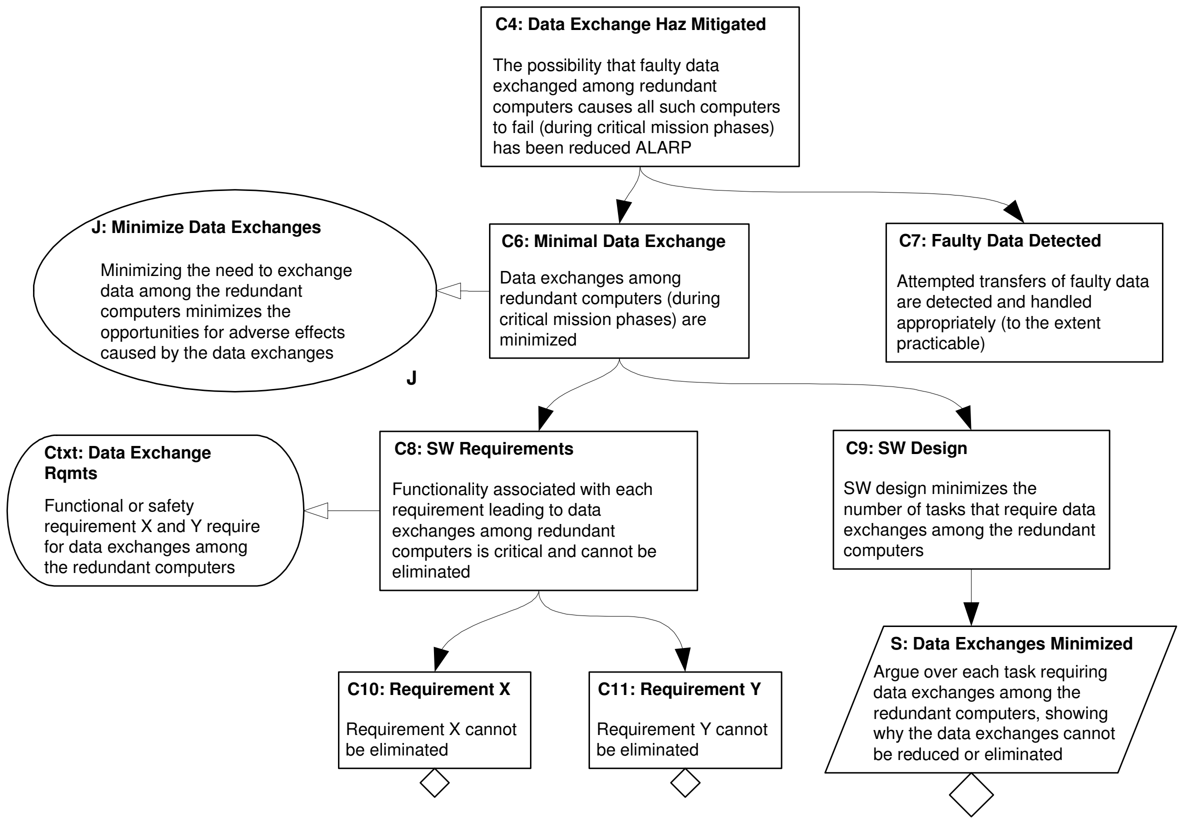

C4: Data Exchange Haz Mitigated
The possibility that faulty data exchanged among redundant computers causes all such computers to fail (during critical mission phases) has been reduced ALARP
J: Minimize Data Exchanges
Minimizing the need to exchange data among the redundant computers minimizes the opportunities for adverse effects caused by the data exchanges
J
C6: Minimal Data Exchange
Data exchanges among redundant computers (during critical mission phases) are minimized
C7: Faulty Data Detected
Attempted transfers of faulty data are detected and handled appropriately (to the extent practicable)
Ctxt: Data Exchange Rqmts
Functional or safety requirement X and Y require for data exchanges among the redundant computers
C8: SW Requirements
Functionality associated with each requirement leading to data exchanges among redundant computers is critical and cannot be eliminated
C9: SW Design
SW design minimizes the number of tasks that require data exchanges among the redundant computers
S: Data Exchanges Minimized
Argue over each task requiring data exchanges among the redundant computers, showing why the data exchanges cannot be reduced or eliminated
C10: Requirement X
Requirement X cannot be eliminated
C11: Requirement Y
Requirement Y cannot be eliminated

**Figure 4. Controls to minimize data exchanges among redundant computers.**

for a failing primary computer.

We now develop the argument further. To reduce the number and amount of data exchanges, we need to show that *requirements* causing data exchanges during critical mission phases are minimized (C8). In addition, we need to ensure that the software design does not introduce unnecessary data exchange tasks during critical mission phases ("C9: SW Design"). Because we list only these two claims in support of "C6: Minimal Data Exchange," we imply that the truth of C8 and C9 is sufficient to support the claim that data exchanges among redundant computers are minimized.

Next consider the argument supporting the requirements claim, C8. The context element for this claim lists the requirements that have been determined to require data exchanges among the redundant computers during critical mission phases. (In this illustrative example, we suppose that only two requirements are relevant.) Subclaims C10 and C11 address each of these requirements in turn. The argument supporting each of these claims needs to be further developed to show why each requirement cannot be eliminated. One might also add a claim that all relevant requirements have been identified; the evidence supporting such a claim might consist of the process used to identify data exchange requirements.

The determination that certain requirements cannot be eliminated is probably what was meant when the hazard report mentioned "requirements" as a control. In any event, this dependability case shows how renegotiating requirements can be a means to mitigate hazards when the requirements have an undesirable effect on safety, and this was the approach taken by Hammett, et. al.[6]

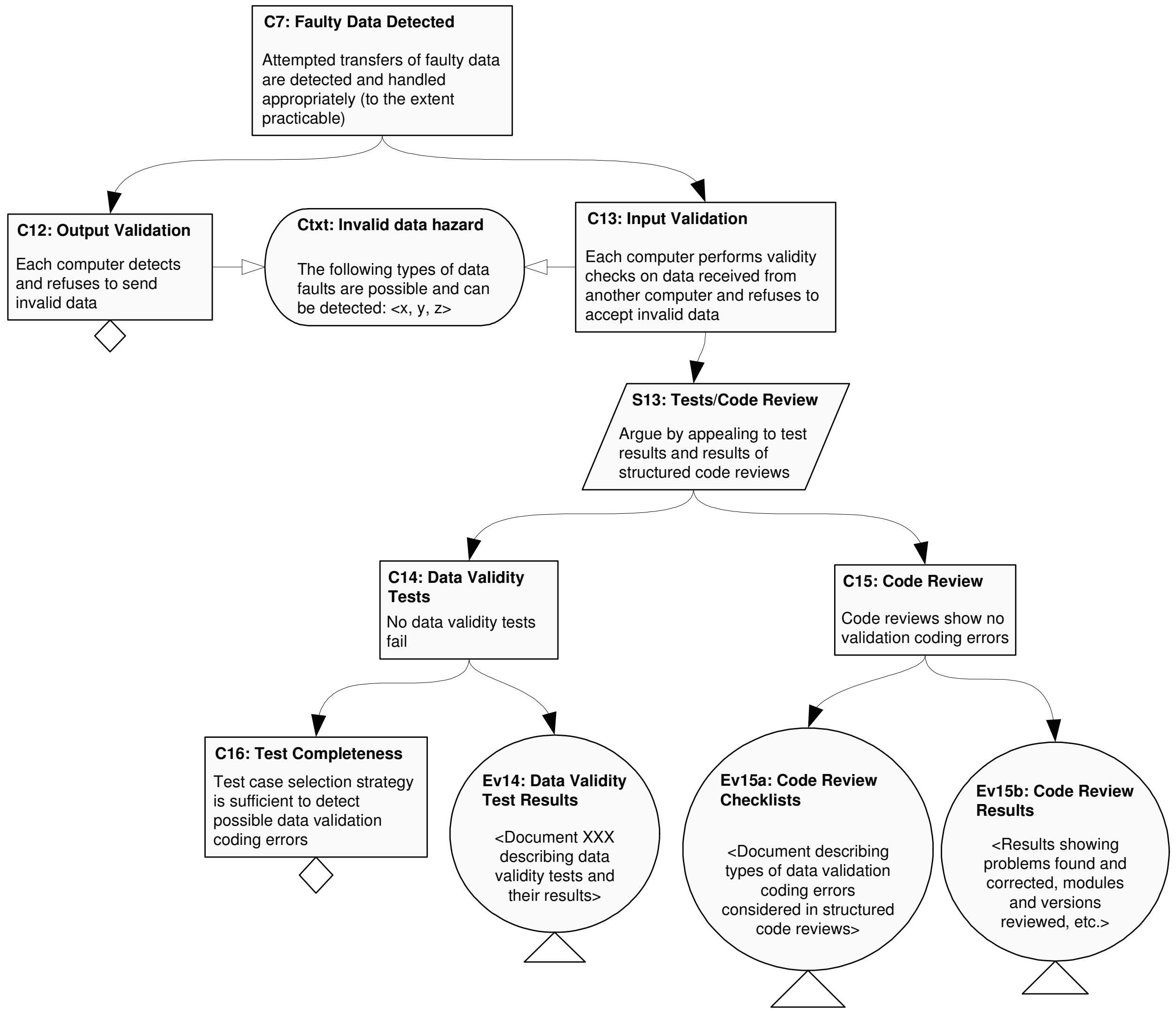

C7: Faulty Data Detected
Attempted transfers of faulty data are detected and handled appropriately (to the extent practicable)
C12: Output Validation
Each computer detects and refuses to send invalid data
Ctxt: Invalid data hazard
The following types of data faults are possible and can be detected: <x, y, z>
C13: Input Validation
Each computer performs validity checks on data received from another computer and refuses to accept invalid data
S13: Tests/Code Review
Argue by appealing to test results and results of structured code reviews
C14: Data Validity Tests
No data validity tests fail
C15: Code Review
Code reviews show no validation coding errors
C16: Test Completeness
Test case selection strategy is sufficient to detect possible data validation coding errors
Ev14: Data Validity Test Results
<Document XXX describing data validity tests and their results>
Ev15a: Code Review Checklists
<Document describing types of data validation coding errors considered in structured code reviews>
Ev15b: Code Review Results
<Results showing problems found and corrected, modules and versions reviewed, etc.>

**Figure 5. Argument and supporting evidence for detecting faulty data transmissions.**

The “C9: SW Design” claim similarly addresses whether the design has minimized the number of data exchanges required during critical mission phases. This claim is supported by requiring an analysis of each of the activities (tasks) requiring data exchanges, presenting claims and supporting evidence that the activity is necessary and cannot be eliminated (or the amount of data to be exchanged cannot be reduced).

We now turn our attention to the second subclaim in Fig. 4, “C7: Faulty Data Detected” (see Fig. 5). Claim C7 expresses an approach to mitigating the hazard of transmitting or acting on faulty data. The chosen approach is straightforward enough in principle — refuse to send or accept faulty data. Of course, not all data faults can be detected, so a context element is used to capture what types of data faults are possible and can be detected. The idea is that data output validation or input validation will address only these detectable faults. In the diagram, we only develop the argument for C13, the “Input Validation” claim.

What claims and evidence are needed to validate “C13: Input Validation?” The argument put forth in the Figure takes a dual approach (as expressed in the strategy element “S13: Tests/Code Review”) — support the claim by considering both test results and the results of code reviews. The idea is that a combination of analysis (code reviews) and testing provides stronger support for the claim than either approach by itself.

The principle claim dealing with testing is that no tests fail (“C14: Data Validity Tests”). This claim is supported by *evidence* (Ev14), namely, test results provided in some document; a reviewer of the case could go look at this document to see that the “No data validity tests fail” claim is indeed supported by the documented results. Of course, the lack of test failures is only significant to the extent the tests are sufficiently thorough. The subclaim, “C16: Test Completeness” addresses this point, namely, that the selected tests are sufficiently thorough to detect possible errors in the code that performs the validity checks. Support for this claim is to be developed further. Such further development would include providing a context element explaining what “sufficient” might mean, e.g., that tests exist for all detectable data errors, that tests check subtle aspects of the validation algorithms, etc. Putting the “Test Completeness” claim in our dependability case shows that attention is being paid to test quality as well as test results.

Finally, the second part of our approach for demonstrating that the validity checks are adequate is to perform code reviews to see if the checks for each type of data validity error are adequate (C15). The evidence in support of this claim is, of course, the results of the code review, E15b. This evidence element describes the information that we want to see as part of the code review results, e.g., we don’t want to know just that the review was done, but that it was done on the appropriate coding modules. It is also useful to know if the review found any problems, since this provides some indication that the review was effective. Since we know that structured code reviews are the most effective way of ensuring that the reviews detect problems, another item of evidence (E15a) is the code review instructions that indicate what kinds of errors are to be looked for, the review process, etc. Such evidence increases our confidence that the code reviews were conducted effectively, and hence, the result that no errors were found is significant.

## D. Discussion

When we introduced the hazard report shown in Table 1, we pointed out a number of deficiencies in the description of controls for the hazard and the approach to be used for verifying the adequacy of the controls. In particular, the original report mentioned analysis of requirements and specifications as a control, but was non-specific about how this analysis would contribute to mitigating the hazard. We suggested that the use of a dependability case would allow this information to be presented in a cogent, detailed manner. The example case we presented showed more specifically how the analysis of certain requirements can contribute to mitigating the hazard (namely, by minimizing the number of requirements leading to data transfers among the redundant computers). The case showed how a similar approach to analyzing the design can contribute to a reduction in data transfers (by minimizing the number of tasks that require data transfers). The hazard report mentioned “incorrect coding” and “inability to test” as contributors to the fault without being specific about what types of coding errors needed to be addressed. The report mentioned “validation and testing” and “rigorous reviews” as a means of eliminating errors, without being specific about what needed to be done. Again, the use of a dependability case allows this information to be communicated succinctly and in a manner that allows for review and discussion to determine whether the approach is adequate.

The creation of a dependability case does not necessarily cause new information (evidence) to be created, but it does organize an explanation of why certain analyses and test results are relevant to mitigating a hazard. The case explicitly documents claims that are otherwise implicit or unformulated, and thereby clarifies thought about what evidence needs to be developed (before the system has been completed) or what evidence exists and its probative value (after the system has been developed). The dependability case organizes information in a structure that can be reviewed during system certification. It provides a vehicle for assessing the impact of proposed changes to a system by showing what aspects of the argument may need to be revisited or revalidated.

By organizing information (connecting claims to evidence via argument), the dependability case provides an explicit and coherent rationale documenting why one should believe that critical hazards have been adequately mitigated. A dependability case is not necessarily needed for every aspect of a system, but for those aspects that are critical, it provides a succinct and understandable structure for documenting engineering rationale.

A dependability case does not always have to be developed from scratch. There are certain patterns of rationale that occur again and again in particular domains. Documenting these patterns in dependability case templates provides an assurance-based engineering practice and provides a basis for constructing specific cases more quickly and at lower cost.

## III. Conclusions

A dependability case documents why a hazard analysis should be considered adequate and allows for independent evaluation of the adequacy of hazard mitigations or controls. Because dependability cases capture and document the argument in support of a claim, they can improve the rigor of an organization's hazard analysis by focusing efforts on explaining why hazards are adequately mitigated, thereby contributing concretely to the trustworthiness of a safety-critical system. The case complements rather than replaces legacy engineering and safety analyses; it need not upend existing processes and practices. Finally, when it comes time to make changes to such systems, the dependability case provides information that helps in determining what aspects of the system cannot be affected by the change, thus potentially reducing the cost and effort involved in recertifying a system.

## Acknowledgements

We thank Chuck Weinstock for constructive comments helping us to improve this paper.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University[a] for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

## References

[1] Jackson, D., Thomas, M., and Millett, L.I. (eds.), *Software for Dependable Systems: Sufficient Evidence?,* The National Academies Press, Washington, DC, 2007, URL: http://www.nap.edu/catalog.php?record_id=11923 [cited February 20, 2009].

[2] ISO/IEC CD 15026-2.2, "Systems and software engineering — Systems and software assurance — Part 2: Assurance case," Committee Draft, 2009.

[3] Kelly, T.P. "Arguing Safety – A Systematic Approach to Safety Case Management." Ph.D. Dissertation, Department of Computer Science, University of York, York, UK, 1998.

[4] Kelly, T. P. and Weaver, R. A. "The Goal Structuring Notation — A Safety Argument Notation," *Proceedings of the Dependable Systems and Networks 2004 Workshop on Assurance Cases*, July 2004, URL: http://www-users.cs.york.ac.uk/~rob/papers/DSN04.pdf [cited February 20, 2009].

[5] ALARP definition and discussion, Wikipedia, URL: http://en.wikipedia.org/wiki/ALARP [cited February 20, 2009].

[6] Hammett, R., Schwartz, R. G., and Smithgall, W., "Preventing Data Pollution in the Space Shuttle Cockpit", *Digital Avionics Systems Conf*erence, IEEE, Piscataway, NJ, 2003, URL: http://ieeexplore.ieee.org/iel5/8816/27907/01245807.pdf?tp=&isnumber=&arnumber=1245807 [cited February 20, 2009].