White Paper on Limitations of Safety Assurance and Goal Structuring Notation (GSN)

Prof. Nancy Leveson
Aeronautics and Astronautics
MIT

Abstract: There has been a lot of effort spent on figuring out how to assure a system is safe after the system design is completed. This white paper presents some of the difficulties and alternatives to emphasizing after-the-fact assurance of safety. There are two basic limitations of safety assurance:

1. System properties such as safety must be built into the system, hopefully from the beginning. If the property is not already there, then it is clearly not possible to assure that it is without using an incorrect argument. Emphasis should be on designing the system to be safe from the start, not trying to prove it is safe after the design is complete.
2. If the assurance process is not successful, i.e., the system is found to have serious flaws with respect to safety or security, it is usually too late to fix it adequately after the design and construction process is complete. Systems today have millions of parts and tens of millions of lines of software (and sometimes hundreds of millions). Making changes without introducing other problems is incredibly difficult. The remaining alternatives are to
   (a) Add in expensive retrofits, such as redundancy, which are costly and are unlikely to be very effective; it is almost always too late to start over with a better basic concept and design.
   (b) Provide an argument that the design *is* safe as is, and then try to convince everyone that the argument is correct when it is not.

There have been suggestions that if the safety assurance argument is put into boxes with arrows, such as Goal Structuring Notation, the argument will somehow be possible to make and will "prove" that the system is safe. Unfortunately, all of the examples I have seen of such arguments have been logically incorrect, even those in widely reviewed government standards or published papers: Logically flawed arguments do not assure the safety of any system. For large and complex systems, any such argument is doomed from the beginning, at the least because it is practical.

But there is a much better alternative to this process, which is to build in safety from the beginning by

1. Using analysis to identify the system safety functional requirements and constraints during concept development,
2. Trace the safety requirements and constraints from the system level to the component level before the components are designed, and
3. Ensure that the safety requirements are satisfied by the design as the design process proceeds, usually involving more analysis to assist in making the lower-level design decisions.

The assurance process is thus spread throughout the development process. At the end, the argument has been made while the design was created. If the design is evaluated throughout the development process, there is much less chance that at the end one will be confronted by the necessity of an enormously expensive redesign process. Note that this "safety by construction" alternative is almost always cheaper and more effective, and I am astonished that so many people are still trying to assure safety into a system after the system has been designed. If safety-by-construction is used, then the assurance (certification) argument at the end is trivial and simply requires documentation of what was done.

This paper expands on this argument and provides examples from recent literature.

The Basic Problem

System properties such as safety must be built into the system from the beginning. It is not possible to assure a completed system has a particular property (without using a flawed argument) when it does not actually have it.

Safety is a system (or emergent) property. What that means is that one cannot just evaluate each component of the system for its safety and then combine these evaluations to determine whether the system as a whole will be safe. Emergent properties are a function of the interactions among the components and not just individual component behavior. In almost all cases, new aspects of safety arise in the interactions that are not in the individual components.

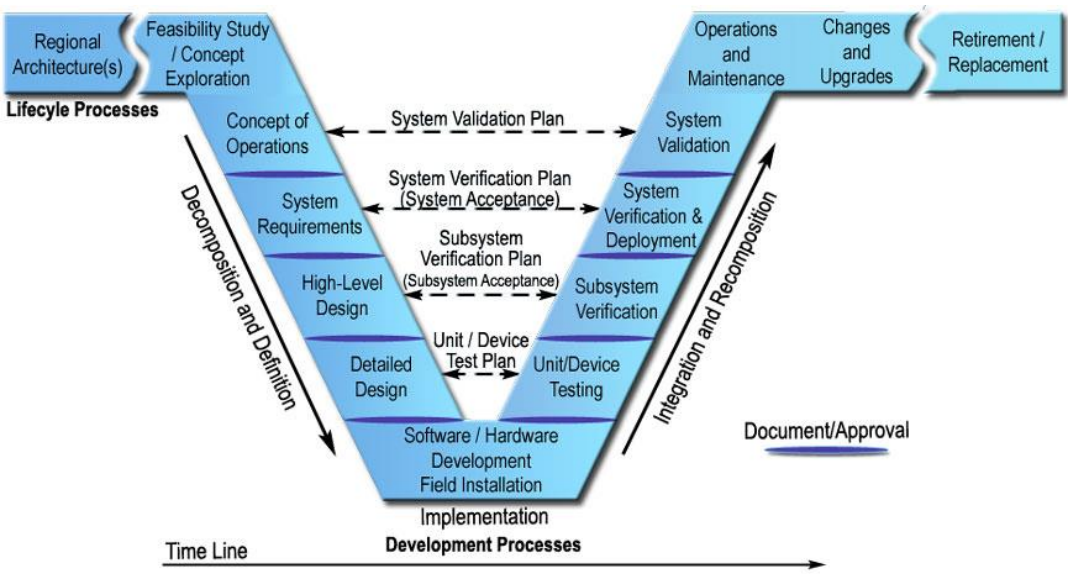
Another way of saying this is that system hazards may be different than the hazards associated with the components. For example, the hazards associated with a valve may involve such things as sharp edges that could cut a person working with the valve. When the valve is used in a braking system, then the relevant hazards are different. They involve the behavior of the braking system as a whole, including the interactions among the components of that system. System safety is related to how the components interact and fit together as a whole. Therefore, assessing safety at any point during or after development requires a top-down process that considers the system design and operation as a whole and not just the individual components. Bottom-up processes will not be effective or practical.

Now let's look at the assurance process in particular. The two ways to assure a system is safe are (1) to perform after-the-fact assurance or (2) to assure that the system is safe as it is being constructed. The latter is similar to what in mathematics is called *proof by construction*.

Almost all the emphasis in proving a system is safe in some fields, particularly software engineering, is on the first, i.e., on an after-the-fact assurance process. It is also the most common approach in software engineering for all qualities, i.e., to wait and test or verify the software has the desired property after it has already been constructed. To do this for a purely logical abstraction, like software, of course, is theoretically possible, but it is so wasteful and involves so much redundant effort that it is usually not practical. For large and complex software, as can be found today, it is simply not feasible.

In the after-the-fact approach, a system is constructed and then assurance and certification are based on an analysis that the already constructed system will be acceptable in terms of safety. There is an assumption here that effective efforts will be used to make the system safe while it is being constructed, but often the method to be used to do this is either not specified or, if specified, only the results are evaluated. The process looks like:

The standard V-model for system engineering is an example, where the left side identifies the development activities and the right side represents assurance (verification) and manufacturing:



One of the problems with an emphasis on assurance of the completed design is that if the system is not actually safe and that fact is determined during the assurance activity, then the only solution is to either start again (usually impossible), patch the completed design the best one can (usually ineffective and costly), or try to compensate for the unsafe aspects during operations (also usually ineffective and impractical).

The assurance approach appears to rest on the belief that almost all systems are safe when engineered (no matter how that engineering is done), and there is only a need to provide some extra argument about this assumption to convince management, the public, and/or a regulatory authority. If some safety flaws are found, then it is assumed they will be easy to fix before the system is used. This belief belies all evidence and the known limitations of engineering.

There are other problems with this approach. Because of the desire to get the system into operation and other cost and political pressures, there is huge pressure for the assurance activity to be successful, that is, that the result will provide the desired level of confidence. Often assurance processes, because they are late in the process, are cut short when budgets are depleted and delivery dates loom. In addition, if problems are found, going back to the drawing board is just not realistic for most systems nor is abandoning the system. This leads to the appearance of a psychological concept called *confirmation bias*.

The identification of heuristic biases in psychology has been called one of the most important psychological developments of the last century. Daniel Kahnemann won a Nobel prize, partially because of this work. *Heuristics* are simple strategies or mental processes that humans use to quickly form judgments, make decisions, and find solutions to complex problems. Heuristics are important and very useful, but they can also involve biases or systematic errors. One of the most relevant heuristic biases (sometimes called *cognitive biases*) involved in the assurance process is *confirmation bias*.

Confirmation bias is the tendency to interpret new evidence as confirmation of one's existing beliefs or theories. If the goal is to show that something is safe, then people tend to look for data that will support that goal, i.e., an argument that the system as designed is safe. We pay more attention to information that satisfies our biases while, at the same time, ignoring information that contradicts the goal.

> *"Still a man hears what he wants to hear and disregards the rest …"* (from The Boxer by Simon and Garfunkle)

As an example, a person may believe that left-handed people are more creative than right-handed people. Whenever this person encounters a person that is both left-handed and creative, they place greater importance on this "evidence" that supports what they already believe. This person might believe other "proof" that further backs up this belief while discounting examples that don't support the hypothesis. If our goal is to show that a system is safe, then it is easy to provide evidence or even to prove that it is safe by ignoring evidence to the contrary. Only positive proof is sought.

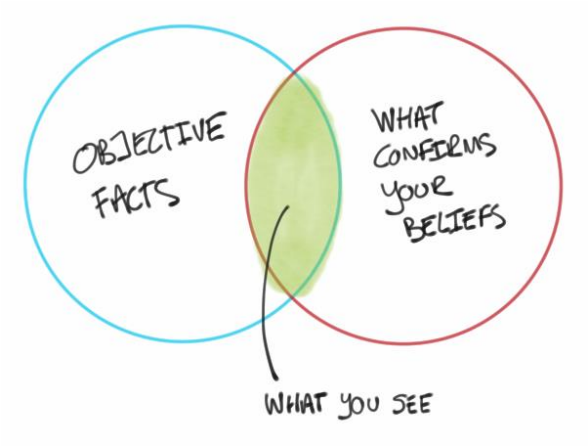


Figure 1: Confirmation Bias

How can we overcome confirmation bias? One way is to look for ways to challenge your goal, not to achieve it. For example, look for arguments that the system is *not* safe rather than arguments that it *is* safe. Making an argument for safety is always subject to confirmation bias. Instead you need to focus on gathering evidence that the system is unsafe. Actively seek out and consider contradictory evidence. This is the standard process used in hazard analysis.

Note that structuring the confirmatory arguments in some notation made up of boxes and arrows or arcane mathematical symbols does not make them less problematic; it simply provides false confidence that they are correct and unbiased. In some sense, structured arguments may be *more* dangerous.

Limits of Safety Assurance in Physical Systems

Now we are ready to get back to the limitations of safety assurance. Assurance is typically done in several ways: (1) testing (including simulation and other executable assurance activities), (2) formal mathematical analysis that does not rely on execution, and (3) informal argumentation, perhaps using notations such as Goal-Structuring Notation (GSN). GSN is a graphical argument used to "document and present proof that safety goals have been achieved in a clearer format than plain text" [Wikipedia: https://en.wikipedia.org/wiki/Goal_Structuring_Notation].

Testing and Simulation

The problem with testing is that the complexity of our systems today precludes the possibility of exhaustive testing. This limitation has always been true for software engineering. A common aphorism in computer science is that testing can only show the presence of errors, not their absence.

In addition, safety problems almost always stem from system engineering design flaws, including those related to false assumptions about the required behavior of the system to be safe and about the operational environment. Accidents occur when simulation and test equipment include those same false assumptions. Assurance activities are, at best, able to show that the system design correctly implements the requirements in the assumed operational environment, but cannot show that (1) the requirements are correct or have a desired operational property such as safety or (2) that the assumptions underneath the testing or simulation are correct including those about the assumed environment and changes that might occur in the future.

Testing has always been the major emphasis for assurance in software engineering, although somewhat less so in system engineering. The problem is that if software engineers wait until the end to do system testing (so called "big bang" testing), the approach becomes very costly and difficulty arises in fixing any errors that are found. Large projects have found that fixing one error often involves introducing at least one new error. In fact, the most successful form of software testing starts at the beginning of software development by testing the interfaces using "stubs." Then, once the system interfaces have been evaluated, the software components are developed and integrated system testing is done in parallel. The interfaces here include human-software interfaces as well as interfaces between the software components and between the software and hardware. Doing system testing at the end is not a cost-effective approach compared to catching errors early, when they can more easily be fixed.

And, once again, testing always makes assumptions about the environment in which the system is used, which may turn out to be invalidated over time. Most accidents occur after a change in the system or its environment, usually unanticipated during system development.

The argument here is not that testing and or simulation are not useful, only that they are not an adequate way to assure safety. Testing and simulation are effective means for achieving other goals.

Formal mathematical/logical arguments

A second way to create assurance is to use formal mathematical arguments. When the arguments are based on physical principles, such as aerodynamics or fluid dynamics, then strong assurance

arguments can be made combining both mathematics and testing, where testing is used to show that the assumptions underlying the mathematics are correct in this particular design. This process, of course, does not preclude mistakes when the assumptions about the physical properties are violated or new structures are used for which the physics is not entirely understood.

Starting as far back as 50 years ago, computer scientists have suggested that mathematical arguments can be used to show that software has certain properties. In practice, this has meant that the software is treated as a mathematical/logical specification. Formal logic is used to demonstrate the consistency of the software logic with the requirements (which are also specified mathematically). Mathematical/logical arguments in such assurance, therefore, are a way to show that two mathematical specifications are logically equivalent.

Unfortunately, almost all accidents in complex systems stem from inadequate/flawed requirements so showing consistency between the specified requirements and the software implementation of those requirements does not provide assurance of safety. In addition, such proofs are not applicable to the physical parts of the system or the human components, where the hazards actually lie. Software by itself is an abstraction and cannot cause any physical harm. Only when the software controls something physical does safety come into play.

For important systems today, formal verification is just not realistic. There are potential benefits from mathematical argumentation for some properties that can be easily represented in mathematical form. Unfortunately, this does not include safety.

Informal Argumentation such as Goal Structuring Notation (GSN)

A third possibility is to use informal arguments for assurance. The problem is that informal arguments are subject to heuristic bias as described above. Experiments have repeatedly shown that people tend to test hypotheses in a one-sided way, by searching for evidence consistent with the hypothesis they hold at a given time.[1,2] Rather than searching through all the relevant evidence, they ask questions that are phrased so that an affirmative answer supports their hypothesis. A related aspect is the tendency for people to focus on one possibility and ignore alternatives, obviously biasing the conclusions that are reached.

Confirmation biases are not limited to the collection of evidence. The specification and interpretation of the information is also critical. Fischoff, Slavin, and Lichtenstein conducted an experiment in which information was left out of fault trees. Both novices and experts failed to use the omitted information in their arguments, even though the experts could be expected to be aware of this information.[3] Fischoff *et.al.* attributed the results to an "out of sight, out of mind" phenomenon. In related experiments, an incomplete problem representation impaired performance because the subjects tended to rely on it as a comprehensive and truthful representation—they did not consider important factors omitted from the specification.[4] It is very likely that this same non-recognition of omissions in the argument will occur when informal safety assurance arguments are created and evaluated.

Putting the argument into a graphical format has been suggested as somehow better than simply writing the argument out in natural language. Unfortunately, graphical formats do not eliminate the bias

---

[1] Ziwa Kunda, *Social Cognition: Making Sense of People*, MIT Press, 1999.

[2] Raymond Nickerson, Confirmation Bias: A Ubiquitous Phenomenon in Many Guises, *Review of General Psychology* , Educational Publishing Foundaton, 2(2):175-220, 1998.

[3] B. Fischoff, P. Slovic , and S. Lichtenstein. Fault Trees: Sensitivity of Estimated Failure Probabilities to Problem Representation, Journal of Experimental Psychology: Human Perception and Performance, vol. 4, 1978.

[4] Kim Vicente and Jens Rasmussen, Ecological Interface Design: Theoretical Foundations, IEEE Trans. on Systems, Man, and Cybernetics, 22(4), July/August, 1992.

and, in fact, can make it worse as the creators and readers of the argument can be biased by the fact that there is a "structure" to the argument and therefore it must be "correct."

One widely marketed structuring method today, Goal Structuring Notation (GSN), is a graphical notation for presenting the structure of (safety) arguments. GSN primarily acts as a communication means to describe how a particular claim has been shown to be true by means of evidence. Remember, however, that any argument based on evidence for the claim is subject to confirmation bias by selecting evidence that supports the claim while ignoring evidence that does not. There is no way to show that all possible relevant evidence has been included. Examples are shown below.

The principal purpose of the goal structure notation is to show how goals (claims about the system) are successively broken down into ("solved by") sub-goals until a point is reached where claims can be supported by direct reference to available evidence (solutions). Basically, an *argument* is made that supposedly demonstrates how the set of evidence combines together to demonstrate the top claim, e.g. that the system is acceptably safe to operate in a particular operating environment. It sounds very scientific, but it is not.

The major problem with informal or even formatted arguments is, as stated, that they are subject to confirmation bias. In fact, every GSN example I have seen has contained confirmation bias and been logically incorrect.

In addition, informal arguments are often based on omission: if a hazard cause is not identified, then it is assumed to not exist. This type of argument is similar to the argument that not finding faults during testing means that faults do not exist. But remember, testing can only *find* faults, it cannot show that they do not exist (unless the testing is exhaustive, which is impossible for real systems). The same is true for arguments specified in a goal structuring notation. It is easy to provide evidence to support any argument if only the evidence that supports it is considered.

You may be saying to yourself that people will not do that. So let's look at some of the published GSN examples in papers and standards. The examples demonstrate varying uses of GSN including an example of the assurance of the safety of a simple control system where the hazard is a hand getting caught in the press. This GSN does not provide a logically correct argument although it purports to do so. The second shows the structuring of the argument for the safety of a nuclear power plant, which simply documents the standard process used in such certification efforts. It is not clear what value is added here. The third example is from a U.K. government standard on what is required for a Safety Management System for the components of the NATS (National Airspace Transportation System). This GSN simply states the components required, which would have been more readable as a simple list. Complexity is added without any real benefit and some drawbacks. The final example is of a security assurance case, which again contains logic flaws.

The first example below was produced and published by one of the creators of the technique.

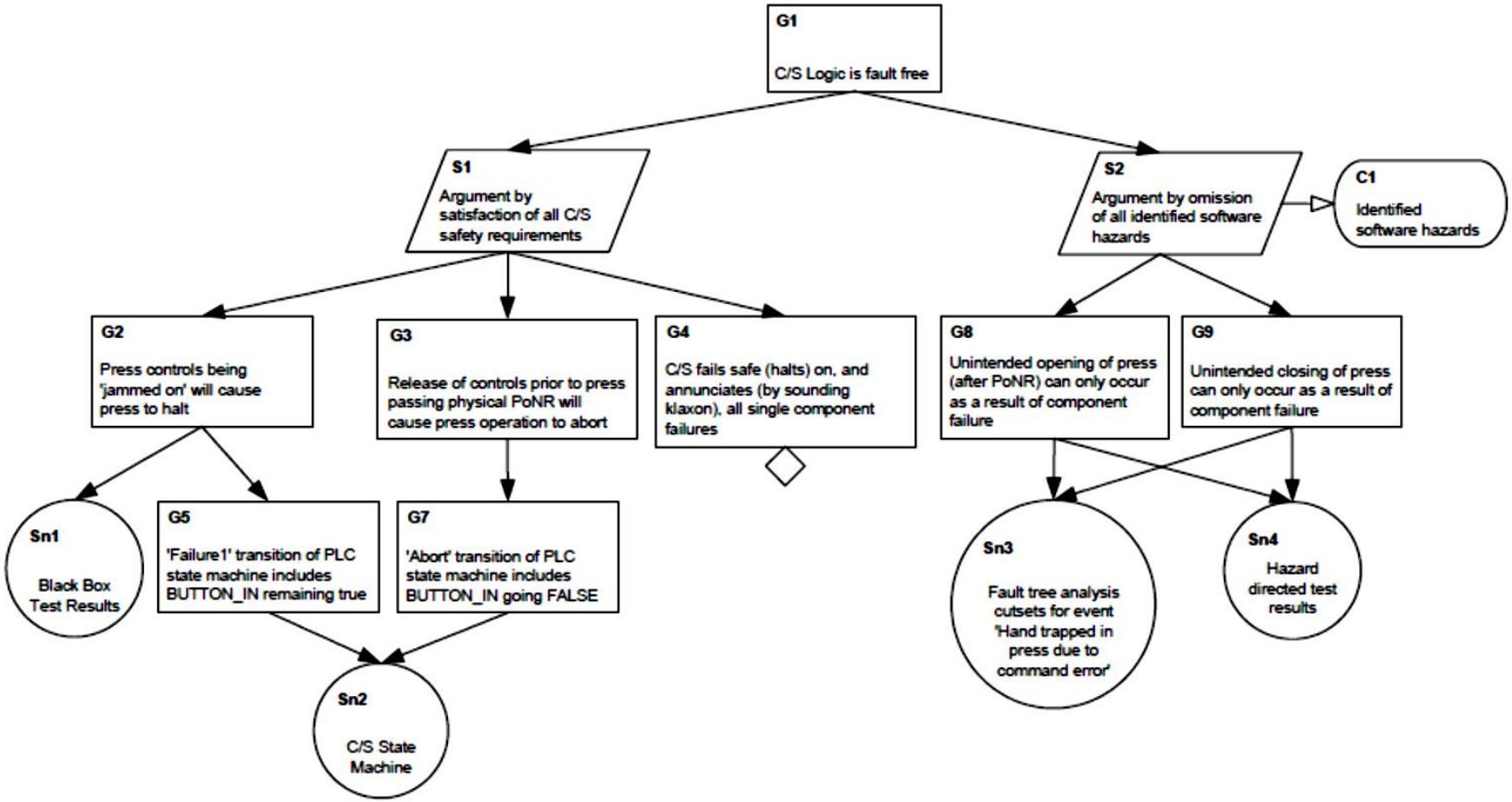


Figure 2: GSN for a Press [Tim Kelly and Rob Weaver, The Goal Structuring Notation—A Safety Argument Notation, *DSN* 2004]

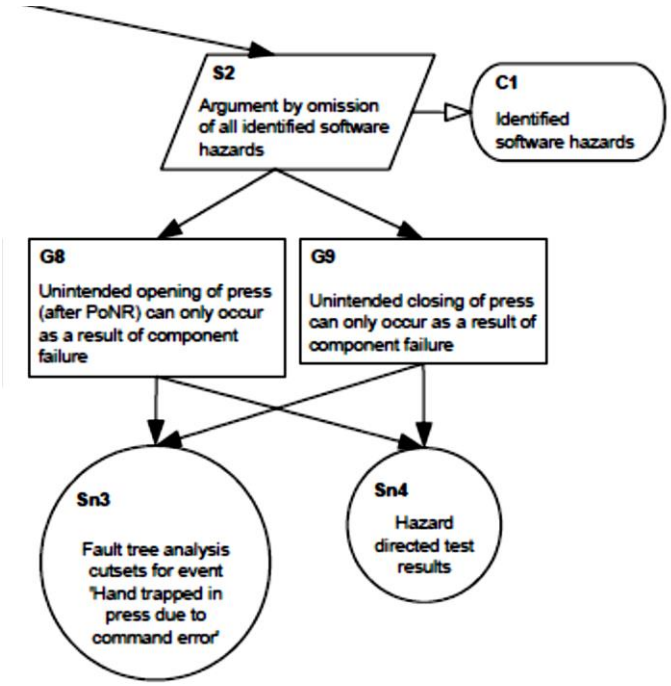
The top box states the overall claim being supported, which is that the "C*/S (Control System) logic is fault free*." In this system, the controller is implemented in software, but in over 40 years of work in software engineering in hundreds of real systems, I have never found any software that turned out to be "fault free."

Indeed, the term itself is nonsensical. Even if the logic can be shown to be consistent with the stated requirements, faults occur with respect to the use of the software. Used in a different environment, it can lead to a safety problem (e.g., the Ariane 5 and other reused software involved in accidents).

So the GSN in Figure 2 starts with trying to show something that is impossible. Surprisingly, the GSN purports to have done it. Luckily, this claim also is not necessary for the software to be safe, i.e. that the system logic is fault free. Software and systems can have faults in non-critical functions and still be completely safe. And safety (not freedom from faults) appears to be the real goal as shown in the boxes below the top one. Why the top goal was chosen is not clear. In the end, however, this GSN neither assures that the software is fault free nor that it will be safe.

The argument on the left side of the GSN involves hardware while that on the right considers software. The hardware argument seems to be based on assumptions about how the hardware will behave using a state machine (model) and test results. This is problematic, but the argument on the right for software provides a simpler example of the problems with this general approach.

The problems start to appear with the argument by omission (S2) branch of the network/tree which is directly at the end of an arrow from "*C/S (Control System) is fault free*"and thus is suggested to be support for the statement. Once again, note that even if one could show that there are no software errors, that does not prove that the software is fault free. But let's get back to the basic safety argument.



G8 and G9 claim that "Unintended opening (or closing) of press can only occur as the result of component failure." Arguments by omission are always problematic. Just because something was not found does not mean it does not exist, particularly if confirmation bias exists and only supporting

evidence is considered. In fact, this statement is demonstrably false, either from the incorrect argument below it or in general.

Using STPA, we have found that unintended behavior not only can be but often is the result of unsafe software or system behavior that does not result from component failure but, instead, by system design flaws. There are now hundreds of examples of accidents in systems where no components failed. By the argument in this GSN example, those accidents never happened because there was no component failure.

In this case, there are two statements that are considered evidence to support the (false) statement, i.e., that unintended opening or closing of the press being controlled can only occur as a result of a component failure. Theoretically, this conclusion is impossible as accidents occur all the time where there was no component failure, as noted. But perhaps it is true for this particular case? What is the evidence for this conclusion?

The first evidence is testing, which we already know is not proof that something *canno*t happen during operation even if it *does not* happen during testing. Testing can only show the presence of errors or faults, not their absence. What is "hazard-directed" test results? How does one design test cases to show that something that is supposedly not present in the system actually is present without exhaustive testing (which is impossible if the system contains software)?

The second claim is based on the fault tree analysis not containing any paths (cutsets) to "hand caught in press due to command error." There is no way to show that a particular fault tree is complete or correct, so not being in the fault tree is evidence of nothing. Because a fault tree does not include something does not imply that the "something" cannot occur. Those creating the fault tree may simply not considered software errors. It is like the argument, "I looked at your chest x-rays and did not find anything wrong so therefore there is nothing wrong with you." But, in fact, you have a broken leg. More important, fault trees identify scenarios containing failures. They are very poor at finding design errors or software errors. In fact, I am confident that an STPA analysis of this system would very likely find a way for a hand to be caught in a press due to a command error.

We have empirical evidence to demonstrate my assertions about fault trees and STPA for real and complex systems where a full hazard analysis (many following government standards such as MIL-STD-882) had already been performed and STPA was also performed. In these dozens of cases for real systems, the people doing STPA did not look at the traditional hazard analysis before they did their own analysis. Some of these systems were commercial or military aircraft that had gone through and passed an extensive certification process, which usually includes fault tree analysis and often FMECA. In every case (and we know of dozens), STPA found paths (scenarios) to accidents that were not identified in the fault trees, usually involving software (or humans). In several cases, the extra scenarios found by STPA were not fixed and serious accidents later occurred.

Would the problems with the GSN argument be fixed by using STPA instead of fault trees? Unfortunately, the answer to this question is no. There are no perfect safety analysis techniques. STPA will find more ways for accidents to occur, but there is no guarantee that all will be found. And, of course, the knowledge and skill of the analysts will always be important in the results of any analysis as well as the resources spent on the analysis. These factors, of course, do not seem to have been taken into account in the example GSN shown. But the bottom line is that the argument using GSN is false.

There are other arguments that are more theoretical and less empirical that GSN and structured arguments can be illogical and false. Simply putting something into boxes and arrows (or even more standard mathematical notation) does not make an argument correct. The GSN examples that I have seen published can usually be shown by formal logic to be fallacious.

A *logical fallacy* is the use of flawed reasoning in the construction of an argument. Consider the example above again. The GNS format relies on a logical statement of the form: A → B or A implies B. The truth of this logical statement depends on whether A is true.

For example, the statement "unintended opening or closing of the press can only occur as a result of component failure" is false so that any argument that claims to support this statement must be flawed. Even if the statement were true, the boxes below the boxes labeled G8 and G9 do not support this claim. Instead they assume it is true and then do not look for other means of unintended operation.

Here are some other examples of a logical fallacy: "I drank bottled water and now I am sick, so the water must have made me sick" or "Every member of the investigative team was an excellent researcher and therefore it was an excellent investigative team." In the latter case, the first statement does not prove the second because the team members may not work well together.

Consider another example, shown in Figure 3, where GSN is used to document a standard for certification of a nuclear reactor[5]. The top box (thing to be proven) says "The control system is acceptably safe to operate" (G1). This claim is "proven" by the two subclaims: "All identified hazards have been eliminated or sufficiently mitigated" (G2) and "Software in the control room has been developed to SIL appropriate to hazards involved" (G3). Let's examine each of these goals (conclusions) and how they are supported.

The first (G2) is that all identified hazards have been eliminated or mitigated. This is then elaborated as an argument for each identified hazard. What is particularly strange is the inclusion of claim A1. An oval or claim in GSN (like A1) is defined as "A statement asserted within the argument that can be assessed to be true or false." Here the claim is that "All hazards have been identified." This claim is so obviously unable to be assessed as "true or false," it is strange that it was included in the argument for the safety of the system: It is, of course, impossible to prove that all hazards have been identified. And it is not part of the argument anyway, which only refers to the identified hazards. Why is it included except perhaps to make the whole argument appear to be more complete?

Under the argument for each of the hazards being eliminated or mitigated, there are three boxes: (1) the hazard has been eliminated; (2) probability of the hazard occurring $< 1x10^{-6}$ per year, and (3) probability of hazard occurring $< 1x10^{-3}$ per year. This argument simply rewrites the requirements for certifying a nuclear power plant in a different notation. In fact, probability estimates for a hazard occurring are impossible to provide. Note that a hazard is not a failure. In many cases, it is possible to calculate the probability of a component failure occurring, but accidents can occur without component failures. I have looked for a long time but have only found two attempts to validate probabilistic risk analysis on real designs—one in the 1980s and a more recent one at Riso Labs in 2002. In both, the calculations of risk by experts for the same design have been up to 3 to 4 orders of magnitude different. That is not very comforting when the risk of a catastrophic accident is involved.

The other argument for the safety of this nuclear power plant is based on Safety Integrity Level (SIL). There are lots of problems with the SIL concept, including the fact that it is based on reliability estimates and that it does not apply to complex systems, particularly those that contain software. The SIL argument is not improved by putting it into a tree/network. There is no experimental evidence that higher SIL levels result in systems that are any safer, particularly for safety functions that are implemented in software.

My argument here is not that the method for certifying nuclear power plants is flawed, although I believe that is the case, but that putting it into a tree structure provides no value added. And the inclusion of a statement that "all hazards have been identified" is concerning as that can never be shown to be true and, in fact, probably is almost always <u>not</u> true.

[5] GSN Community Standard Version 1. 2011. Origin Consulting Limited. York. Available: http://www.goalstructuringnotation.info/documents/GSN_Standard.pdf

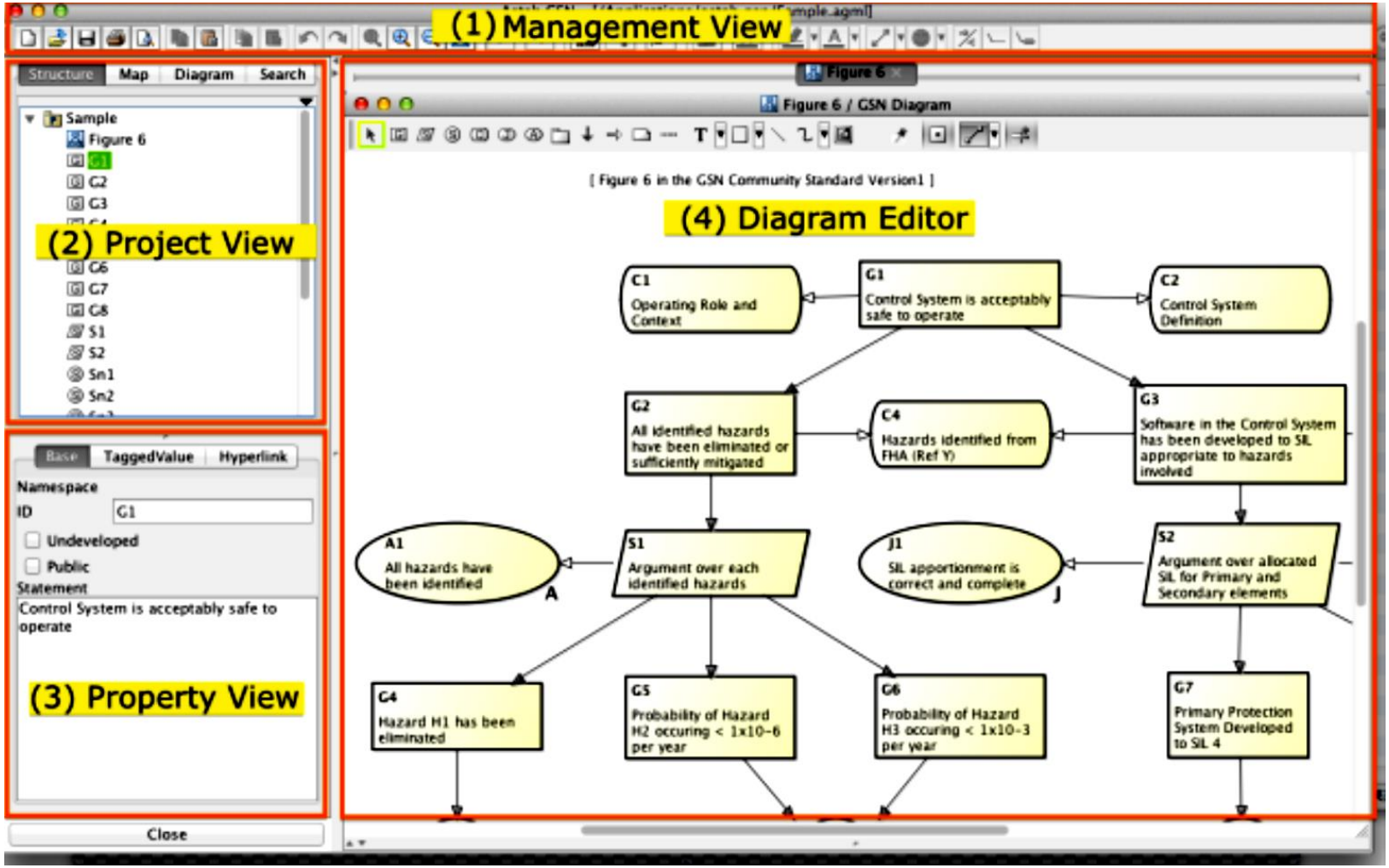


Figure 3: A GSN for a Nuclear Power Plant [Joonas Linnosmaa, Structured Safety Case Tools for Nuclear Facility Automation, Tampere University of Technology, M.S. thesis, December 2015.

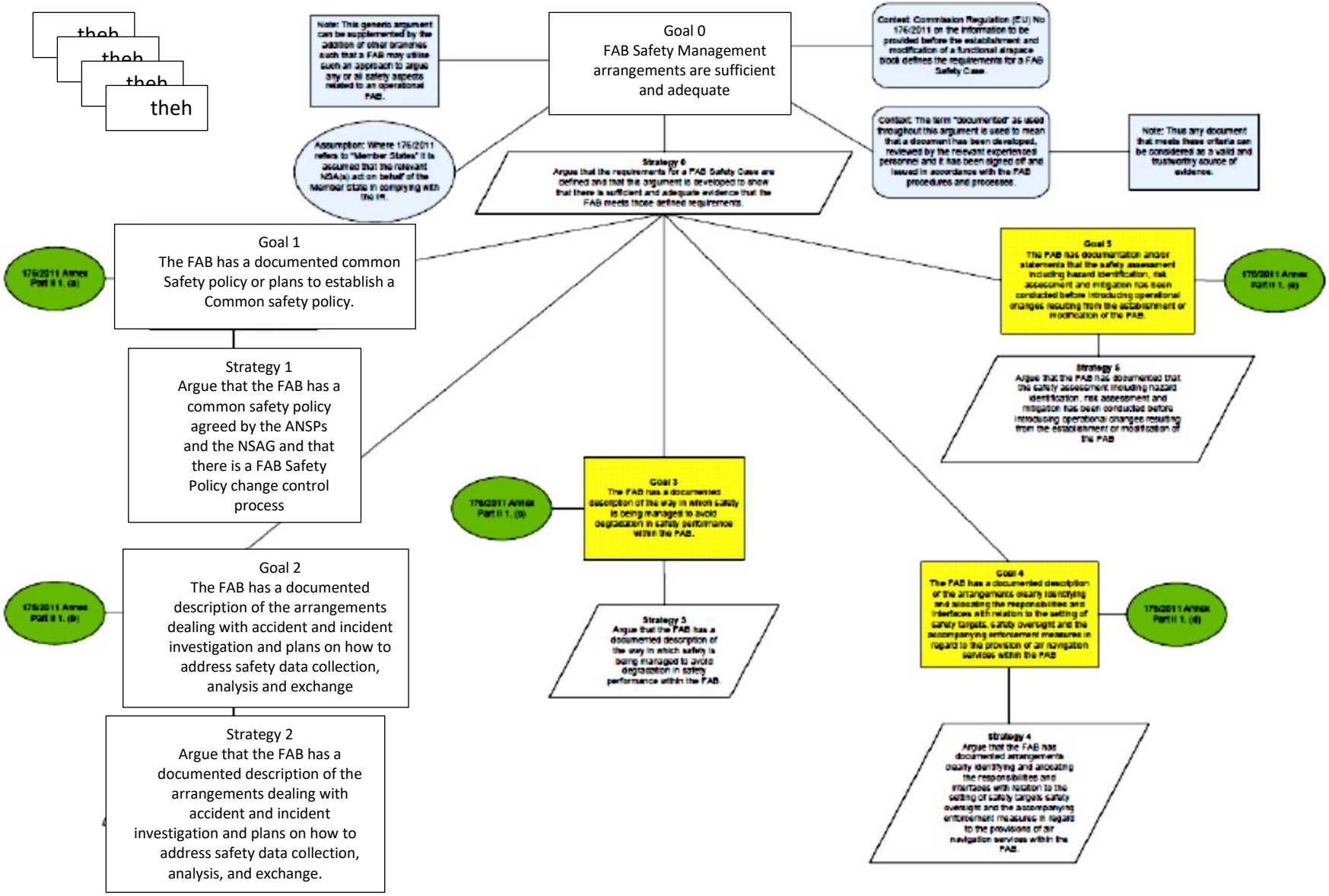


Figure 4: U.K NATS GSN for Safety Management

Some government agencies appear to be using GSN to assure that safety management systems and processes are adequate. This use is also very problematic. Figure 4 shows an example from a U.K. NATS (National Air Transportation System) standard [January 2012] for safety management. Parts of the figure are redrawn because the original was not readable—which is another problem with trying to use graphical models for even simple systems, let alone complex ones.

Goal 1 says that "The FAB [Functional Airspace Block] has a documented safety policy or plans to establish a common safety policy." Under that there is a "Strategy" that states to "Argue that the FAB has a common safety policy agreed by <its components> and that there is a FAB safety policy change control process."

Arguing that a group has a safety policy does not mean the safety policy is good or will be carried out or even can or will be carried out when other organizational goals conflict with it. Perhaps more information is somewhere else? The point is that putting what is essentially a list of the required components of an SMS into a bunch of boxes does not make it any better than simply writing a list of requirements for safety management systems. Where is the added value?

Worse, most of the contents are repetitive and require shifting to multiple pages of the GSN to get the necessary information. Consider Goal 2, which is "The FAB has a documented description of the arrangements dealing with accident and incident investigation and plans on how to address safety data collection, analysis and exchange. The Strategy for Goal 2 is "Argue that the FAB has a documented description of the arrangements dealing with accident and incident investigation and plans on how to address safety data collection, analysis, and exchange." What extra information has been added here?

How this argument should be made is in a different part of this large GSN. It is shown in Figure 5. It is not clear how having a documented description for dealing with accident investigation implies that good accident investigations will occur, which surely is the real goal. What is needed is an *effective* method for accident investigation, not just any documented method. But perhaps that information is in the references in the blue and red circles? But the real problem is that all this seems to be saying is that ANSPs and NSAs must have a documented description for dealing with accident investigations. Couldn't that just be written in one sentence?

Putting the "argument" (i.e., requirement) in this type of notation seems to imply that the statement is stronger or more meaningful than it is. Written in plain English might encourage most people to see the limitations of the policy. And again, a simple list of the things that a safety management system must have—for example, a documented description for dealing with accident investigation—would be as useful and more readable. If the references in the blue and red circles provide the more detailed information required, then I will need to look somewhere else for any useful information. Again, this could all be stated in one line of a list that said "The FAB and ANSP must have a documented description for dealing with accident investigations" along with the appropriate references provided for the details (the references in the blue and red circles in the example piece of the GSN). I cannot find anything in the GSN provided that could not be stated in a list of 3 or 4 items, but that, of course, would not give the false impression that all of this was somehow logically complete.

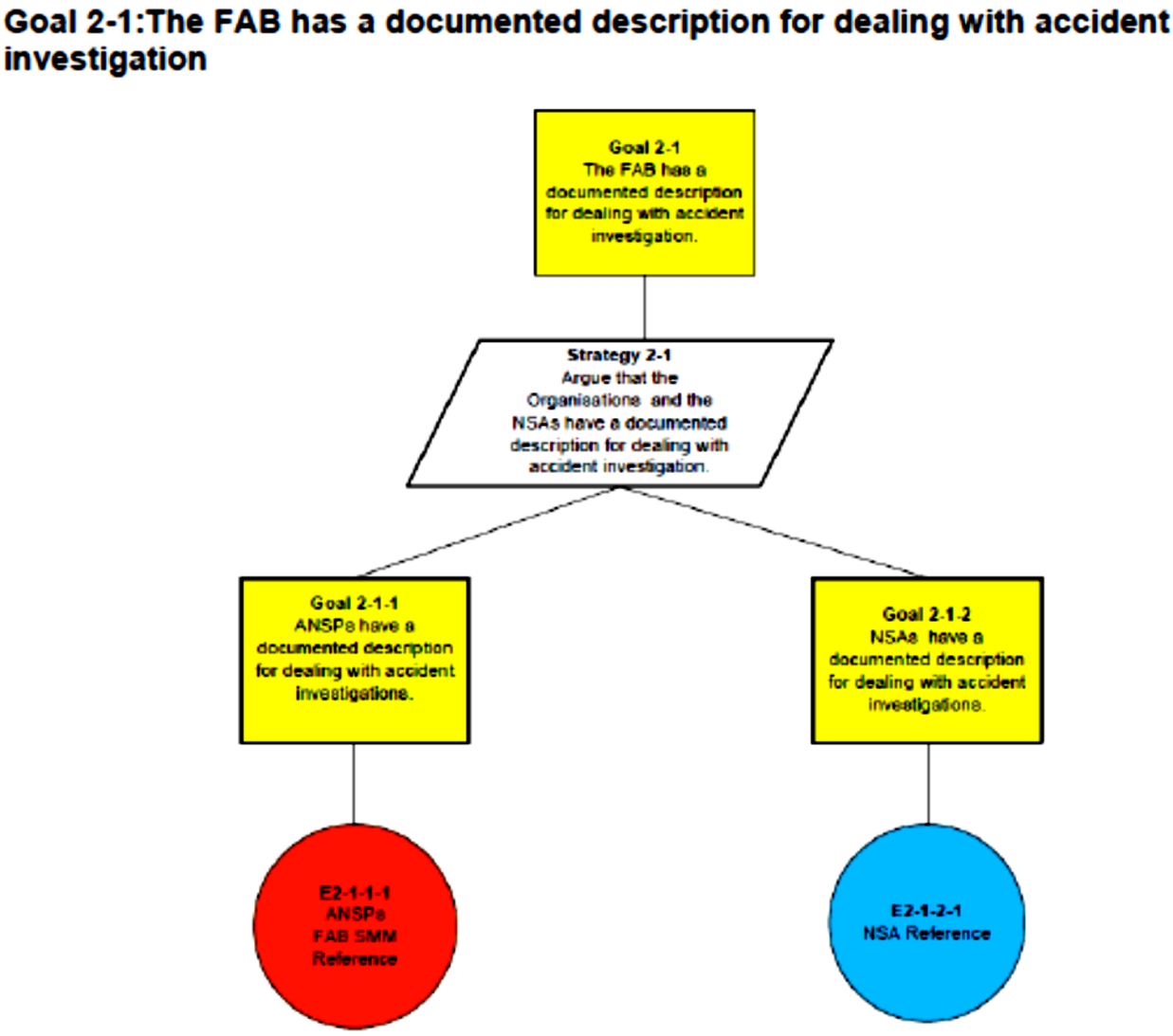


Figure 5: Another part of the UK NATS GSN for Safety Management

Figure 6 shows a last example. It is a security assurance case that focuses on buffer overflow. It is simply a specification of one process for handing buffer overflow. There is no reason to put it into this GSN framework beyond trying to imply that the process is complete and correct. The notation does not do that. In fact, writing it as process steps would more likely illustrate the limitations of the process. The bottom three circles provide the actual important data, i.e., the results from running a static analysis tool, the results of buffer overflow tests, and evidence showing why each warning of a buffer overflow possibility by the analysis tool was a false alarm. This data is used to "prove" the statement that "There are no buffer overflow possibilities in the code." Again, testing or static analysis cannot prove the absence of errors such as buffer overflow or the absence of such possibilities in the code. Is the static analysis tool perfect such that it never misses any case? How has that been proven? Was the testing exhaustive?

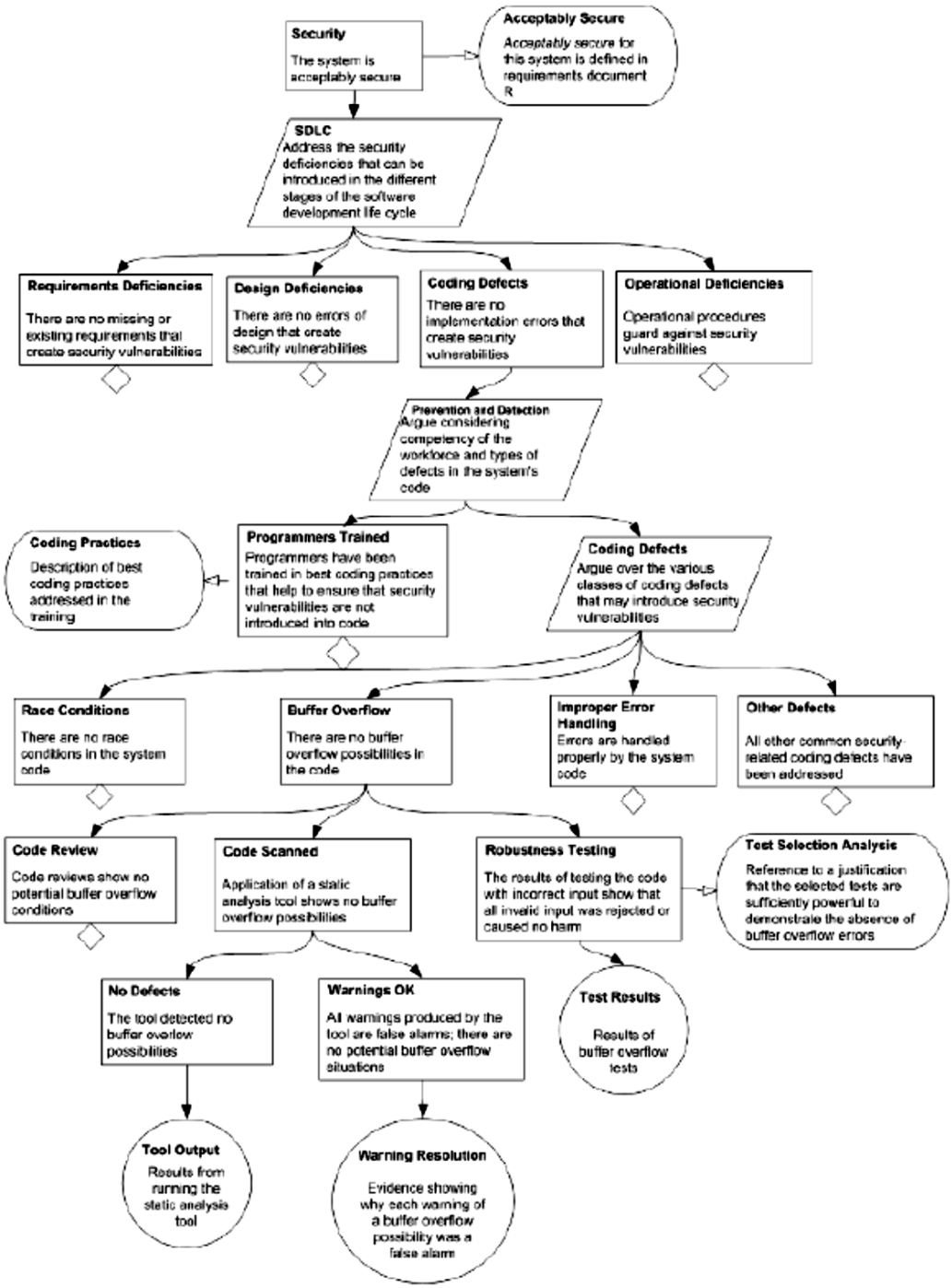


Figure 6: Partially expanded security assurance case that focuses on buffer overflow [Charles Weinstock, Howard Lipson, and John Goodenough. Arguing Security—Creating Security Assurance Cases, Software Engineering Institute, CMU, Jan. 2007]

In fact, most of the GSN "assurance" arguments seem to include just a process to be used without any real proof that the process is adequate, although the implication that it is logical and adequate by the use of GSN is dangerously misleading. GSN itself seems to add nothing useful.

Before adopting any engineering technique, there should be extensive experimental evidence collected by scientific methods to ensure that the technique is effective. I have not been able to find any such careful evaluation of GSN. Most papers show examples or describe how to prepare GSN diagrams, but they do not provide scientific data on whether it is effective. The prevalence of confirmation bias in all GSN arguments I have seen seems to reduce the likelihood that such evidence could be found.

What about empirical data? How do GSN arguments and safety assurance work in practice? There also is little of this type of data, but there is an interesting finding in the accident reports for Nimrod aircraft crash in Afghanistan in 2006, which had an assurance case performed on it before the crash. Haddon-Cave, the author of the Nimrod accident report, concluded that "Care should be taken when utilizing techniques such as Goal Structure Notation or 'Claims-Arguments-Evidence' to avoid falling into the trap of assuming the conclusion (the platform is safe) or looking for supporting evidence for the conclusion."

What is the alternative to these attempts at safety assurance?

The alternative has been used in system safety engineering for decades. In the standard System Safety approach, hazard analysis is performed during system development to identify flaws in the system. This process provides the "value-added" of system safety engineering. The system engineers have already created arguments for why their design is safe when they were designing it. Assurance cases simply repeat the same thing and do not really provide any value. System safety engineers find safety flaws by applying the opposite mindset from that of the developers, i.e., thinking about why the system may be *unsafe* and how to prevent the scenarios leading to hazards.

Traditional hazard analysis, however, usually comes after a design is essentially complete, a process that has some of the same drawbacks as after-the-fact safety assurance. By using modern hazard analysis techniques, such as STPA, safety can be designed into the system from the beginning.

An improved process, which is less resource intensive and more practical, identifies hazards and eliminates them or controls them in the design or operation of the system. Essentially, safety is built into the design throughout the system development process, starting from early concept development. Any final assurance process at the end would simply review the documentation of what was done for quality and for compliance with the required standards; it is not an argument that the system is safe, only that the hazards identified by the stakeholders have been properly handled.

Figure 7 shows the process that is being suggested.

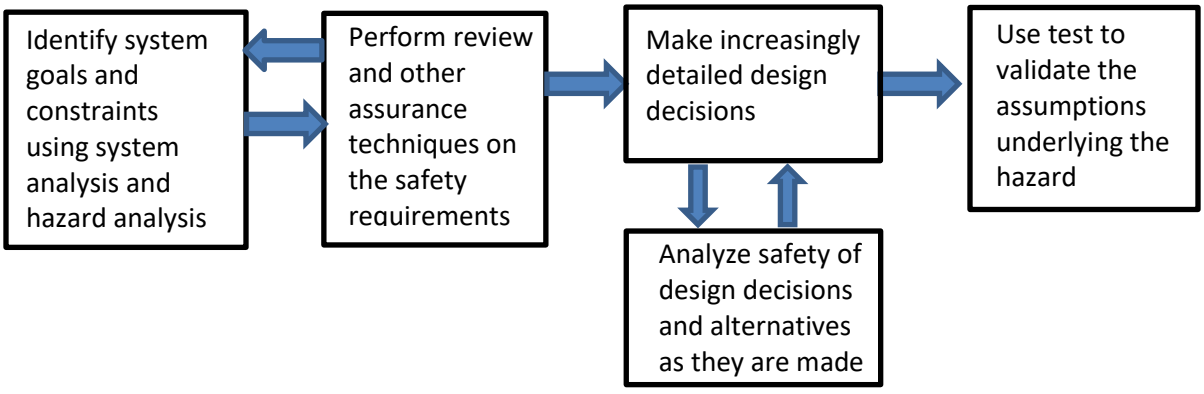


Figure 7: An improved process that integrates hazard analysis into the development process.

Note that test is used to validate the assumptions underlying the hazard analysis, which is doable.

Essentially, this proposed process incorporates safety "assurance" (analysis) as the system is being developed, not after-the-fact. Advantages include the ability to catch problems earlier and make better design decisions from the beginning of concept and design development, that is, identify the detailed system and component safety requirements before designs are created and catch problems early when it is still practical to fix them.

Engineers do not wait until the plane is finished and undergoing flight test before they do design analysis, such as using Navier-Stokes equations to validate the wing design. If they did, there would be too much backtracking (and loss of test pilots) to make such a process reasonable.

But what if you are a government regulatory agency with a responsibility to certify systems for safety? This responsibility does not need to imply that all activities must occur once the system is completed and ready for use. The design activities and analyses can be inspected as they occur. In fact, this will make the certification effort feasible for complex systems, which it currently is not unless the assurance involves the kind of hand-waving involved in the use of GSN or the use of probabilistic risk assessment numbers that are almost always incorrect, require unacceptable assumptions (independence), and have little to do with safety. But that needs to be the subject of a different paper.

Conclusions

At best, GSN and other formatted notations for arguments add nothing except the cost of putting the argument in this form. At worst, they are misleading and dangerous and subject to confirmation bias. And simply documenting standard processes in a tree structure rather than textually provides no added value.

Instead, I believe that our efforts should be directed at identifying hazards early in the concept development phase of complex systems and constructing the system to eliminate or mitigate those hazards from the beginning. Safety must be built into a complex system; it cannot be assured at the end. Instead of trying to argue that systems are safe, the most effective safety programs for complex systems have historically (for at least the past 60 years) instead put their effort and resources into identifying hazards and eliminating or controlling them.

At the end, final assessment might consist of documentation and review of the analyses and decisions made during the development process to ensure they were of high quality and follow documented processes. This final assurance documentation is similar to what is included in the MIL-STD-882 SAR (Safety Assessment Report). It is much too late to try to argue that a system is safe (build a safety case) once the system is completed. Even when trying to increase safety during development, the goal should be to figure out how it will be unsafe and not to talk ourselves into a belief that it is already safe.