

Safety Analysis as a Software Tool

Blair T. Whatcott

Northrop Grumman Information Technology

As a software development tool, an independent software safety analysis by trained analysts reduces losses of development resources and schedule, improves product quality, and prevents costly mishaps that occur during the operational phase of the system life cycle. The key issues of an effective and efficient software safety analysis include (1) financial and managerial independence from the software development activity, (2) trained and qualified personnel to perform the analysis, and (3) a disciplined process that focuses the analysis effort, by priority, on the more safety critical areas.

Performing an independent system and software safety analysis on embedded software saves overall life-cycle cost and schedule resources, and provides a better overall product. The primary objective of safety analysis is to find and remove embedded safety related hazards in the hardware and software systems before a mishap occurs.

Finding these embedded hazards early in the development cycle reduces cost, safeguards schedules, and improves product quality [see Figure 1]. The reduction in added costs and schedule slips due to problems found late in the development cycle and the improvement in product quality justify the cost of performing an independent software safety analysis. Additionally, preventing a single catastrophic mishap by removing an embedded hazard could more than pay for the independent safety analysis effort many times over, depending upon the system.

This article identifies key terms associated with system and software safety, provides a process for performing software safety analysis, specifies the required environment for efficiently and effectively performing safety analysis, provides cost and schedule savings rationale, and identifies issues that delay or prevent effective safety analyses. Although this article emphasizes performing safety analysis on software, a thorough software safety analysis includes a system safety analysis as many of the embedded hazards occur at interfaces between system components.

When developing software systems, a tool enables a developer to build better systems quicker. The systems are more effective, more efficient, and safer. Involving an independent software safety analysis contributes to these attributes and becomes a tool that should be used in today's complex system development efforts.

Software Safety Analysis Process

An effective process for performing a software safety analysis includes four pri-

mary steps (see Figure 2):

- **Step 1.** Identify safety-critical areas and system safety hazards.
- **Step 2.** Trace implementation of safety-critical requirements to the design and its corresponding code.
- **Step 3.** Verify correct system use and implementation of safety-critical data and processing.
- **Step 4.** Track identified hazards throughout the system life cycle.

Each step is discussed in the following paragraphs.

Step 1

The first step is to list all system requirements, the relative level of safety criticality for each system requirement with respect to the other requirements, and the applicable documented hazards for safety-critical requirements. Each system/sub-system requirement must be evaluated for criticality with respect to potential hazards. The safety criticality of a requirement depends upon the identified hazards and other items such as remoteness, contributory impact, redundancies, and human intervention.

System hazards vary depending upon the function and use of the system. These hazards must be identified and documented. Sub-hazards that contribute to higher-level hazards need to be specified to a sufficient detail. An example of a hazard might be *erroneous activation of release*. Examples of corresponding contributing sub-hazards could be (1) erroneous release signal, (2) erroneous status display, and (3) malfunctioning safety lock. This list of system requirements becomes the plan that is used to perform software safety analysis.

Step 2

Using the safety criticality priority established by the list from Step 1, the second step is to evaluate requirements. This step correlates the functional requirements to (1) the top level and detailed level design descriptions and (2) the source code

implementation. The code is verified for correct implementation of the requirements as well as for correct syntax and safe coding practices. This analysis also includes identifying specific safety-critical data items and processing within the reviewed code module for use in the next step. An example of a safety-critical data item could be a weapon release variable. An example of safety-critical processing could be the processing required to set or reset the release signal.

Requirements of higher safety criticality are evaluated before those of lesser criticality. The intent of safety analysis is to find the more critical hazards that would cause mishaps of higher severity. As there will always be some residual mishap risk, particularly when software is involved, the task of performing a complete and thorough software safety analysis would be both cost prohibitive as well as impossible. Consequently, items of lesser safety criticality may not be

Figure 1: Software Safety Analysis Benefits

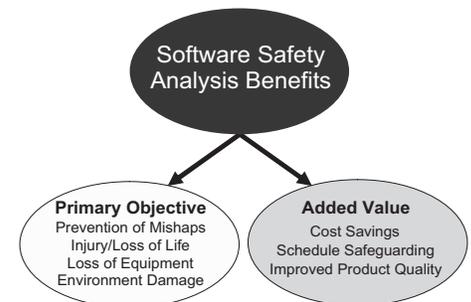
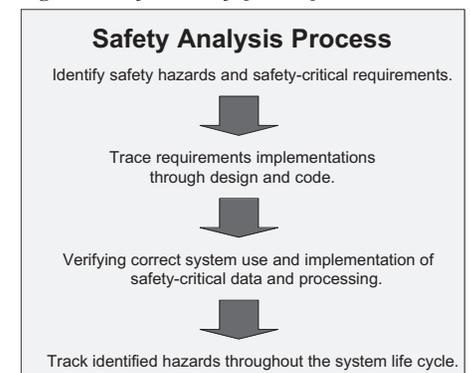


Figure 2: Software Safety Analysis Process



reviewed so that items that are more safety-critical can be more deeply and completely reviewed.

Step 3

The third step is to evaluate the safety-critical data and processing identified in Step 2 in the context of the system. Particular attention is given to the interfaces between subsystems, sequencing of state changes, and timing windows of vulnerability. This type of analysis is oftentimes not given sufficient attention during software development and testing as well as peer reviews because of the added complexity and time required to be thorough.

Step 4

In the fourth step, identified hazards are documented and communicated to the development organization. The safety analysis effort tracks the identified hazard until it is removed from the software. As software hazards tend to be repeated in other areas and applications, the hazard is added to the *lessons learned* software safety analysis database. The hazards from this database, as well as hazards identified on industry *generic* safety lists, are used for training of safety analysis engineers and for performing evaluation checklists when future modifications are made to the software being analyzed or other software in other systems.

An example of a generic safety list can be found in Appendix E of the Joint Software System Safety Committee's "Software System Safety Handbook" [1]. This combined list of software-specific hazards is very large. It would be cost

prohibitive to analyze every line of code against every item in the hazard list. The implementation freedom that software allows precludes an all-comprehensive automated tool that checks every line of code for every possible hazard. We have found that a trained analyst who is current with the list of software hazards is more efficient and effective in performing the safety analysis. Software utilities and tools can and are often used to help the analyst more quickly locate similar patterns, occurrences, and uses.

Software Safety Analysis Environment

To perform effective and efficient software safety analysis, an environment of three components is required: (1) financial and managerial independence from the software development activity, (2) trained and qualified personnel to perform the analysis, and (3) a disciplined process that focuses the analysis effort, by priority, on the more safety-critical areas [see Figure 3].

Each of these components is necessary for a successful software safety analysis. The absence of any undermines the effort and the strength of the other components. For example, without the financial and managerial independence from the development activity, the analyst may be directed in a way that inhibits fully performing the analysis, or the discipline process is circumvented because of management direction caused by a need to use resources in areas other than safety analysis. Similar examples can be drawn from the absence of the other com-

ponents.

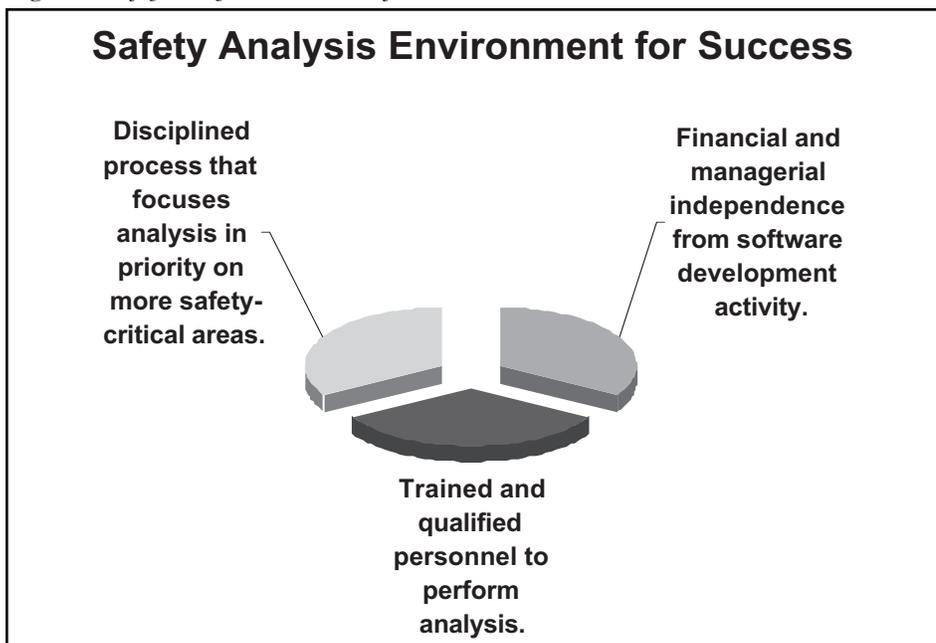
Financial and managerial independence ensures that specific resources will be used for performing the software safety analysis and that there is no conflict of interest between the development activity and the software safety analysis activity. An example of a conflict of interest could be the program office or development organization controlling the work of the safety analysis by direction toward or away from specific hazards or risks. The criticality list from Step 1 is the road map that identifies the priority of safety-critical areas to be reviewed. The resources are used in accordance with this priority, which means that safety-critical areas of lower priority may not be reviewed or analyzed because of the need to use resources to analyze safety-critical areas of higher priority.

An example of implementing an independent safety analysis effort would be for the program office to contract separate activities to the development organization and the software safety analysis organization. As such, reports to the program office from the software safety analysis organization are independent of the software development activity. Another example would be to have the safety office independently contract to the software safety analysis organization for work being developed by the program office that is contracted to the software development organization.

Performing a successful software safety analysis requires a technical staff that is qualified to perform safety analyses and enjoys doing this type of work. Most engineers prefer building new systems and being part of the development process of major systems. Many engineers find no interest in digging through systems to find embedded hazards that are not only difficult to find and understand, but also have not evidenced themselves. It becomes the proverbial looking for a needle in the haystack, except there is really no clue that the needle is even in the haystack or even in a number of haystacks. Finding engineers who can do this type of work and want to do it for many years is difficult. Some can do analysis for a year or so; however, the strength of a software safety analysis organization comes from analysts who have done analysis for many years on many systems.

Two pitfalls are seen when companies set up software safety organizations. First, individuals who are used to performing software safety analysis activities

Figure 3: *Safety Analysis Environment for Success*



Safety Terminology

For the purposes of this article, the following safety-related terms are provided from [2].

- A **mishap** is an unplanned event that results in death, injury, occupational illness, equipment or property damage or the loss of, or environmental damage.
- **Hazards** are conditions that cause a mishap.
- The **ultimate goal of a system safety program** is to design systems that contain no hazards. However, since the nature of most complex systems makes it impossible or impractical to design them completely hazard-free, a successful system safety program often provides a system design where there exist no hazards resulting in an unacceptable level of mishap risk.
- **Mishap risk** is an expression of the possibility/impact of a mishap in terms of hazard severity and hazard probability.
- **Residual mishap risk** is the remaining mishap risk after all mitigation techniques (techniques used to remove or lessen the hazard) have been implemented or exhausted.
- **Safety** is the freedom from hazards, which cause death, injury, occupational illness, equipment or property damage or loss, or environmental damage.
- The **objective of a safety analysis** is to achieve acceptable mishap risk through a documented systematic approach to hazard analysis, risks assessment, and risk management.
- **System safety** is the application of engineering and management principles, criteria, and techniques to achieve acceptable mishap risk, within the constraints of operational effectiveness and suitability, time, and cost, throughout all phases of the system life cycle.

are not correctly screened with respect to ability and desire. Oftentimes, they are selected from those who have not been successful doing other code development activities because of work habit issues or lack of ability. The fallacy of doing this is that performing successful analysis on code that is generated by the development activity requires individuals that are better trained and more capable than those who have developed the code. They must be able to find embedded hazards missed by other development reviews and testing that could result in a mishap during the operational phase of the system life cycle.

The second pitfall of setting up software safety organizations is that the development activity expects that the code developer should be able to generate good code that contains no safety hazards. Consequently, the development activity either sees no value in performing an independent safety analysis or limits safety analysis resources to the point that little can be done to effectively accomplish a thorough safety analysis. They fail to understand that there is a basic difference between engineers who develop code and engineers who analyze code for embedded hazards. Engineers who develop code are success oriented. They move from the implementation of one requirement to the next. They are driven by a typically over-budget schedule and are always anxious to *catch up*. Conversely, safety engineers analyze code in the context of finding failures. They move from analyzing one module to the next only when they are convinced that there are no embedded safety concerns.

Complexity Warrants Additional Safety Analysis

With our mentality of getting the most out of software development budgets combined with the mindset that developers can generate hazard-free code, managers pressure software developers to generate code faster and more efficiently. They insist that better processes, pride of workmanship, better compilers and development environment tools, code walk-throughs, and peer reviews should sufficiently guarantee safe code. It is true that compilers and development environment tools are becoming more powerful, but at the same time they are becoming more complex. This additional complexity warrants additional safety analysis. It is true that code walk-throughs, peer reviews, and other process improvements result in better code; however, they rely upon peers who are also behind

schedule, over-budget, and anxious to get their own work done. These distractions lessen the effectiveness of the peer review. Additionally, peer reviews tend to focus on the module level and place less emphasis at the system level where many of the embedded safety hazards reside.

Software developers must be safety conscious as they develop code. However, in light of the above and their success orientation, developers continue to introduce embedded hazards in the software development process; it is very difficult for them to see their own errors. E-mails are a vivid example. E-mail authors re-read their own e-mails over and over to verify correctness. They send them out only to later find a glaring error in the most awkward place that they missed during multiple reviews. Some well-known examples of software failures resulting in mishaps are described in Appendix F of the Joint Software System Safety Committee's "Software System Safety Handbook" [1].

Engineers who effectively analyze code for embedded hazards are convinced that all software contains embedded hazards and that it is only a matter of time and circumstance before the hazard(s) causes a mishap. The quality and quantity of analysis is a function of the analyst's safety experience and understanding of the code under inspection within the context of the system. Tangible products of the analysis may be

misleading as amount and quality of product does not necessarily prove that the right analysis was performed. On the other hand, the tangible products of a development effort do prove the efforts of the development engineer.

From the development activity perspective, if the code successfully performs its intended function and matches documented code standards, then return on investment is evident. Failure to identify embedded hazards does not confirm that quality analysis has not been performed any more than the identification of some embedded hazards ensures that all hazards have been found. The analyst decides the correct amount of effort spent in the analysis of a safety-critical area for hazards without evidence of their existence based on his or her experience and understanding.

In summary, development engineers are good at building new systems in the context of a driving schedule. Software safety engineers are good at evaluating code for embedded hazards. Requiring development engineers to constantly evaluate their code with the understanding that *something is wrong and there are embedded hazards* seriously takes away from the success orientation that enables forward progress. Software safety analysis engineers are attuned to the identification of embedded hazards and the amount of resources required to fully analyze a safety-critical area of code. Just

as letting off an automobile's gas pedal does perform some slowing, and letting off the brake pedal allows continued movement, both the gas pedal and the brake pedal are required for efficient handling. A combination of development engineers and software safety engineers in an independent environment provides a product that is synergistically more than if either were to do both tasks.

The software safety analysis process combines the people and the resources to produce the most effective and efficient product possible. The process ensures that priorities are followed, products are produced, and schedules are met. The four primary steps of a software safety analysis process have been described. Necessary products of the safety analysis include a criticality analysis report from Step 1, problem reports from all steps, and a software safety analysis report – including testing and analysis summaries – from Step 2 through Step 4 of the process. When a thorough and conscientious software safety analysis is complete, and safety hazards have been identified and removed, the resulting summary report becomes a tangible product that indicates with a high level of confidence that

the examined software will not be the source of a system mishap.

Issues That Hinder Software Safety Efforts

There are many reasons why organizations mistakenly choose not to include a software safety analysis activity early in the code development cycle (see Table 1). These include the following:

1. Organizations erroneously believe that performing software safety analysis only needs to be done when code has been generated. They believe that they can conserve resources during the requirements definition and design disclosure phases by waiting until code is released to involve the software safety analysis effort. They fail to understand the importance of evaluating system and functional requirements with respect to safety prior to design, and of evaluating the design disclosure with respect to safety prior to coding. Safety concerns found during the implementation phase after the code has been generated require re-evaluation of the requirements, redesign, and recoding. This results in wasted resources and schedule slips because

of the necessary review and rework. This is further impacted by the software safety analyst's need for time to become familiar with the function, requirements, design, and code of the software under analysis. If this need is put off until code is released, then safety concerns are, consequently, identified later in the implementation phase, resulting in additional wasted resources because testing must also be repeated due to reworked requirements, design, and code.

2. Government organizations find it difficult and time consuming to establish a contract with an independent organization to do software safety analysis. It is important to start the process early to take into account the lead times as well as the need for either contracting directly with the software safety analysis company or using a contract vehicle already in place by the contractor.
3. The organization erroneously believes that a good code development process will preclude all embedded safety hazards. Mishaps caused by software occur in fielded systems that were developed under good processes. As described earlier, an independent software safety analysis can find embedded hazards and prevent mishaps when trained and experienced analysts are used and the software safety analysis process is followed.
4. Organizations fail to factor into their budget the software safety analysis activity when cost projections are supplied to planning activities. Upon program execution, they severely limit or do not fund software safety activities because of the difficulty of finding unbudgeted resources to cover safety. Including software safety analysis activities in the master budget plan is critical to software safety.
5. The lack of mishap evidence gives the program manager a false impression of the safety state of the software being developed. If an embedded hazard is found and removed, there is no evidence that the mishap would have ever occurred. Embedded hazards cause catastrophic mishaps only when a set of combining circumstances simultaneously occurs. A thorough analysis covers areas and combinations of events that are either difficult to test or are not tested because of limitations due to test time and tester expertise.
6. Organizations have difficulty finding

Table 1: *Mistaken Reasons Why Software Safety Is Not Included During Early Phases of the System Development Life Cycle*

Mistaken Reasons Why Software Safety Is Not Included During Early Phases of System Software Development Life Cycle	
Reason	Actual Need
Conserve funds because there is no code to review.	Early evaluation of requirements and design precludes costly coding and testing errors.
Difficulty establishing a contract with an independent organization to do safety.	Most major defense contractors have General Services Administration-type contracts that could support safety efforts.
Misconception that good coding processes preclude embedded safety hazards.	Success orientation of development engineers results in missed errors; development environment pressures prevent thorough system and interface analysis.
Failure to budget in software safety analysis activities.	Software safety analysis needs to be budgeted independently of development activity budgets.
Lack of mishap evidence if hazard found and removed.	The objective of software safety is to remove hazard before mishap; prevention of one catastrophic mishap more than pays for safety analysis effort.
Lack of software safety analysis expertise and processes.	Evaluate potential safety analysis organizations on track record, processes, and staff.
Problems found in safety analysis cause additional work impacts.	Early identification of safety problems saves resources by preventing redesign, recode, retest, and prevents mishap.

software safety analysis expertise and processes. As described earlier, effective analysis is a function of the expertise and experience of the analyst. Qualified sources for software safety expertise will probably be more costly because of the need to employ this level of expertise and experience.

- Organizations are concerned that problems found by software safety analysis will cause additional work that impacts schedule and resource needs. Reputable organizations do not generate unsafe software. However, because of the nature of embedded hazards that result in mishaps, there is always the concern that large amounts of resources are spent to prevent mishaps that have a very low probability of occurring. These organizations fail to understand that providing a small level of software safety analysis can greatly lower the probability of a mishap occurring.

Each of these mistaken reasons is real. Together they may discourage using software safety analysis as a tool to generate a better product for less cost. Finding these embedded hazards early in the development cycle reduces cost, safeguards schedules, and improves product quality. Our experience shows that a requirements problem that is not found until the test phase of the software development cycle results in the loss of 70 percent of the time used to design, code, and test the implementation of that requirement.

Summary

We live in a world that is averse to unsafe conditions. We also live in a world that applies heavy pressure to building the *better and faster* more efficiently. The conflicts between these two mindsets are profit and risk. The courts of the land insist daily upon the responsibility of the product provider. Flashy packaging and brand-name recognition oftentimes erroneously instill within us a false sense of trust. And if we are harmed, our loss of productivity and capability demands compensation in order to survive.

Software safety analysis as a tool results in a safer and better product at a cost and schedule savings. Early involvement is critical to an efficient and effective analysis effort. Software requirements hazards will be found and removed during the requirements phase. Hazards found during the other software development phases will be found during the correct phase, preventing loss of

resources and schedule.

Software development teams want to generate a quality product, but are hesitant to have independent activities perform analysis on their product. A change of mindset will result in a synergistic team that produces a superior product. Development engineers will be able to do what they do best in a success-oriented environment within their resources and schedules. Software safety analysis engineers will provide the necessary checks and balances that result in a superior product, free of embedded hazards. When these work as a team, software development will cost less and be provided on schedule in our world of continuous change and improvement. ♦

References

- Joint Software System Safety Committee. Software System Safety Handbook. Washington, D.C.: Department of Defense, Dec. 1999 <www.egginc.com/dahlgren/files/ssshandbook.pdf>.
- Department of Defense. "Standard Practice for System Safety." MIL-STD 882D. Washington, D.C.: DoD, 10 Feb. 2000 <www.safetycenter.navy.mil/instructions/osh/milstd882d.pdf>.

About the Author



Blair T. Whatcott is the lead engineer and program manager of the Software and System Safety Analysis Program Office in the Information Solutions Department of Northrop Grumman Information Technology. He has managed and been lead engineer on independent verification and validation and software safety analysis projects for more than 18 years. These projects have focused on detailed analysis and testing of embedded software in military aircraft and weapon systems. He has a bachelor's degree in electrical engineering from Brigham Young University, Provo, Utah.

**Northrop Grumman
Information Technology
1530 N Layton Hills PKWY
STE 200
Layton, UT 84041-5683
Phone: (801) 773-5274 ext. 13
Fax: (801) 773-5262
E-mail: blair.whatcott@ngc.com**

CALL FOR ARTICLES

If your experience or research has produced information that could be useful to others, **CROSSTALK** can get the word out. We are specifically looking for articles on software-related topics to supplement upcoming theme issues. Below is the submittal schedule for four areas of emphasis we are looking for:



Cost Estimation

April 2005

Submission Deadline: November 15, 2004

Configuration Management

June 2005

Submission Deadline: January 17, 2005

Software: More Than Just Code

August 2005

Submission Deadline: March 14, 2005

Software Safety/Security

September 2005

Submission Deadline: April 19, 2005

Please follow the Author Guidelines for **Crosstalk**, available on the Internet at <www.stsc.hill.af.mil/crosstalk>. We accept article submissions on all software-related topics at any time, along with Letters to the Editor and BackTalk.