# Software Safety for Model-Driven Development

Timothy J. Trapp
*Raytheon*

Donald S. Hanline II
*U.S. Army AMCOM*

Howard D. Kuettner, Jr., and William A. Christian
*APT Research, Inc.*

*With software applications becoming increasingly complex and the demand for rapid deployment (including rapid prototyping) of software applications increasing, automated tools and updated methods for software development have become necessary. It follows that these new software development processes require new approaches for software safety. One company's 15-element Software Safety Process has now been adapted to a model-driven, spiral software development effort. This process provides an open working relationship to incrementally identify the causes of hazards at different levels.*

Software applications are being called upon to perform ever-increasing, safety-critical activities. To that end, software engineering is increasingly using automated tools and updated methods to sustain and gain better intellectual manageability over these solutions. The Missile Defense Agency's (MDA) Global Engagement Manager (GEM) development is an example of integrating these software methodology constraints. At the same time, software safety assurance requirements remain unchanged and mature software safety processes exist. The strategy used in the GEM software development methodology was to leverage the pedigree of existing software safety methods and adapt them to model-driven software development. The advancing software development methodologies lend themselves to a rich, comprehensive approach to safety analysis.

## Software Applications Are Becoming Increasingly Complex

The MDA has the mission to provide mechanisms that protect the homeland, deployed forces, friends, and allies from a ballistic missile attack. To frame the problem, ballistic missile defense has a global scale. Threats can originate from any region and be directed at any destination. Since the battlespace will most likely cross multiple areas of responsibility, coordination among command echelons is critical to prioritize the available radar and interceptor resources. To that end, the MDA's Command, Control, Battle Management, and Communications (C2BMC) program has been developing a series of products to enable the DoD to integrate individual sensor and weapon system elements into their Ballistic Missile Defense System (BMDS). One of the C2BMC's products is the GEM.

The GEM's objective is to provide the warfighter with execution-time decision aids to enable them to think globally while acting locally, thereby effectively using the BMDS element resources for the globally integrated active missile defense. Generally speaking, the warfighter's tasks are to:

- Maintain a deep understanding of the active defense design with its branches and sequels.
- Monitor the ballistic missile battlespace.
- Assess gaps from differences in the anticipated and actual enemy courses of action.

> *"The strategy ... was to leverage the pedigree of existing software safety methods and adapt them to model-driven software development."*

- When appropriate, manage by exception[1].

Present warfighter doctrine involves centralized planning while executing in a decentralized fashion. Anticipating the use of automated battle management capabilities to support decisions, warfighters must plan for retaining control over the automation as weapon systems/sensors join and leave the BMDS network. So, the deployment planning process for the BMDS will need to account for effectively integrating technology, processes, and personnel. As the BMD operations are readied for alert, the commander's intent and engagement priorities are configured in the GEM. As suspect tracks are detected and tracked, additional sensor resources can be utilized to gather more information on objects of interest. If tracks are assessed to be a threat, a layered defense based on priority will be calculated. Under an operator's control, weapon-system tasking will be issued to BMDS weapon systems. One can expect responses such as *will comply*, *cannot comply*, etc. This can be due to battlefield effects or conflicts among missions within a multi-mission platform. If an element is unable to support an engagement, the operators can immediately assess and task other elements within the layered defense. Suspect tracks will be prompted to the operators for disposition. If they are promoted to threat status, they will be prioritized and assigned appropriate interceptors. This workflow continues throughout battle.

In addition to being a decision aid, the GEM has a second responsibility, a system of systems (SoS) challenge. In looking at SoS research, failures occur when one system's failure cascades across connected systems or when properly working systems interact in unanticipated ways. The present state of practice is to have a controller manage emergent behavior. Even though fire control remains with the weapon systems engagement function, it is envisioned that the GEM will be assigned the SoS controller role for the BMDS. To globally control effective, execution-speed coordination (in the face of battlefield effects), unanticipated adversary actions, and multiple command structures, the GEM is to be built to have a level of predictability, dependability, and correct behavior that the warfighter can depend on during the *fog of war*.

## Software Development Increasingly Uses Automated Tools and Updated Methods

To gain the warfighter's trust in such a mission-critical environment, the GEM decision aid must have predictable behaviors across the broad environmental conditions in which the product may find

itself operating. To achieve that end, the Advanced Battle Management (ABM) development process was crafted from carefully selected modern software development methodologies. The key principles came from model-driven development, model-based acquisition, service orientation, and Agile software development methodologies (as shown in Tables 1 and 2). They were applied to the different activities in the development life cycle (i.e., specification, domain analysis, design, implementation, and testing activities).

The goal is to develop and maintain a common model of the product's behavior so that it optimizes errors and defect exposure at the time they are introduced in the software. A prime contributor to errors and defects are language barriers in systems. This approach strives to create a common understanding across stakeholders by mapping and relating perspectives and points of view into a composite specification. It relates:

- Human-machine interface tasks.
- Functional threads of behavior.
- Use cases to capture desired behavior.
- Collaboration-like activity diagrams, state charts, and algorithm definitions to capture design.
- Code-generated executable models.
- Supporting software to create a dynamic specification that can be assessed.
- Verification cases for further analysis and test.

A second key source of defects is related to the misinterpretation of statically written requirements. This process captures the specification in a way that can be exercised to see if the desired behavior really does occur. This is called an executable model. So, by analyzing this model, one can both evaluate the proper behavior and also reason about the predictable responses of the system when it is faced with faults or conditions beyond its operational bounds.

A complicating factor in software development is identifying errors as they are introduced during implementation. A dominant Agile development methodology tenet used in the ABM development process is short iterations. The design, implementation, and analysis is performed on a small incremental portion of the GEM's required behavior. An iteration consists of completed feature sets with automated test suites. That is, iterations deliver working code with working tests every two weeks. In general, development activities are broken into three major release categories (shown in Figure 1):

1. **Iterations:** Two weeks of development (approximately five iterations per cycle).
2. **Cycles:** Approximately two months of development (approximately six cycles per increment).
3. **Increments:** Approximately one year of development.

## Example of Integrating These Software Methodology Constraints

To initially determine and capture the correct dominant behavior, use cases are used as the documentation tool capturing functional behavior from warfighters, analysts, and subject matter experts. Use case extensions are used to reason through behavior under adverse conditions, such as load shedding and fault handling. Safety, performance, mission assurance, human factors, and information assurance concerns are analyzed and incorporated into the use cases. With this behavioral definition captured, detailed design incorporates architectural constraints and further elaborates finer-grained behavior that is only exposed with detailed design.

At GEM's foundation is an architecture that follows well-established business logic. The GEM uses the *kill chain*: detect, track, assign, engage, and assess. Second, from that stable and well-understood structure, behavior that is more apt to change is isolated into components. These components interact only with the stable structure, never with each other. Third, a reactive system of this type demands predictable performance. Between components, the GEM utilizes data queue and blackboard structures—called data stores—for information exchange. In the GEM's case, components must operate concurrently. Each component uses input from prescribed data stores. The safety framework monitors information that flows through the data stores, leveraging policies that are captured in assertions to trigger the safety executive and safety kernel functions into action.

The approach starts with defining policies that capture key behavioral requirements. These policies represent critical behavior that assesses proper operation of the GEM. Many policies are then converted into assertions in logic, linear temporal logic, and/or state charts. By monitoring these assertions in a run-time monitor, fault conditions can be detected, mitigated, or possibly avoided. For safety, critical safety assertions are monitored in the safety exec-

Table 1: *Selected Model-Driven Attributes Used in the ABM Development Process*

| Principles from Model-Driven Development | Principles from the Model-Based Acquisition Approach |
|---|---|
| • Develop a ubiquitous language.<br>• Remove grey-matter translations.<br>• Institute shared understanding of the customer, user, subject matter expert, engineer, software engineer, and tester.<br>• Incorporate model checking and code generation (to allow the specification to be tested without significant investment in implementation). | • Verification of GEM behavior in the specification phase.<br>• Verification of GEM computation code from third-party vendors.<br>• Component-based software engineering.<br>• Verification of modules prior to integration.<br>• Verification of safety components that ensure continuity of operations against all run-time faults.<br>• Separation of computational code from behavior code:<br>◦ Design by contract.<br>◦ Verification of assertions in the specification phase.<br>◦ Asynchronous messaging among software modules.<br>• Test oracles with automated test procedures:<br>◦ Verified distributed system behavior in the specification phase.<br>◦ Unified process approach with no more than two-month cycles.<br>◦ Validation of implementation against the GEM model at the conclusion of each cycle.<br>◦ Verification of meeting hard real-time deadlines. |

Table 2: *Selected Agile and Service Orientation Attributes Used by the ABM Development Process*

| Principles From Agile Practices | Principles From Service Orientation |
|---|---|
| • Short iterations<br>• Automated test<br>• Continuous builds<br>• Re-factoring<br>• Retrospectives<br>• Daily stand-ups<br>• Feature-based development | • **Service reusability:** Logic divided into services with the intention of promoting reuse.<br>• **Service contract:** Services adhere to a communications agreement, as defined collectively by one or more service description documents.<br>• **Service loose coupling:** Services maintain a relationship that minimizes dependencies and only requires they maintain awareness of each other.<br>• **Service abstraction:** Beyond what is described in the service contract, services hide logic from the outside world.<br>• **Service composability:** Collections of services can be coordinated and assembled to form composite services.<br>• **Service autonomy:** Services have control over the logic they encapsulate.<br>• **Service statelessness:** Minimize retaining information specific to an activity.<br>• **Service discoverability:** Services are designed to be outwardly descriptive so they can be found and assessed by discovery mechanisms. |

utive. Should the assertion trigger, mitigating action will be taken by the safety kernel. Additionally, assertions are used for mission assurance and at interfaces between components. By establishing assertions about the pre-conditions, post-conditions, and invariants, the errors, defects, and faults can be detected and addressed.

With behavior defined and assertions established, the executable model is created. Code is generated directly from the base logic state charts and activity diagrams. Then, this behavioral logic capability is augmented with the action code. The functionality is layered in during nine-week cycles. It is done in short iterations so all incremental functionality can be evaluated. The proper behavior is analyzed as the executable model functionally grows by monitoring the assertions from an orthogonal view. The test environment is a key analytical tool that is equivalent to an engineer's workbench. Portions of the assertion base are in the test harness and are independent from the implementation. They are always checking to see if a new functionality has broken what had been previously built. It allows for exercising the executable model to analyze and demonstrate anticipated behavior across the broad dynamic range of the battlefield environment. At the end of a cycle, the product leaves the analysis phase and moves into testing. Testing activities that occur at this time are looking for defects that escaped that phase of development.

The ability to economically develop software in this model-driven fashion is made possible by the advancing state of practice in software engineering. These practices continue to make mainstream computer-aided software engineering tools. Though mission assurance and safety concerns are moving into the software development culture in the form of reliability and safety engineering constraints, software safety training is important. Seasoned developers/engineers often have the tools but not the experience and do not know what is sufficiently complete or correct when it comes to these concerns. Some examples of definitions developed for GEM Software Safety training include:

- **Sufficient Completeness.** The consensus of all of the qualified reviewers that the specifications and developments for each part of the presented system and subcomponents are full expressions to the extent that is foreseeable in regards to intended behavior, intended performance, and intended environment.
- **Sufficient Correctness.** The consensus of all qualified reviewers that a software system and its components are free from foreseeable faults in its specification, design, and implementation in regards to intended behavior, intended performance, and intended environment.
- **Intended Behavior.** The planned aggregate of response, reactions, or movements made by a system in any situation. This conversely includes the planned prevention of undesired responses, reactions, or movements.
- **Intended Environment.** Conditions of the elements external to the system that are planned to be affected by or are currently effecting the employment or deployment of the system.
- **Intended Performance.** The metrics of system behavior over time. Examples are latency, throughput, availability, and utilization.

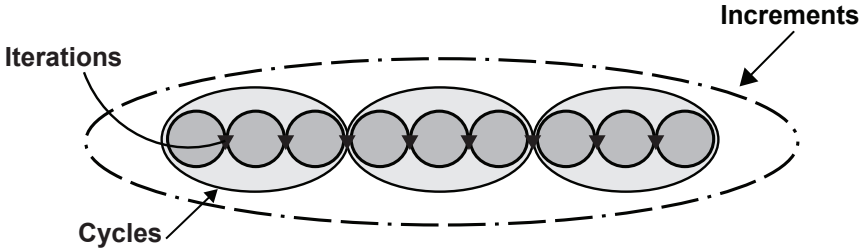


Figure 1: *Three Major Release Categories for Software Functionality*

*"Though mission assurance and safety concerns are moving into the software development culture ... software safety training is important."*

## Requirements and Processes

### *Safety Assurance Requirements Remain Unchanged*

MIL-STD-882D [1] forms part of the basis for the MDA's safety guidance. Awareness of these requirements is the beginning point for software safety training. MIL-STD-882D guidance applicable to software development includes:

- Unacceptable conditions that are considered unacceptable for development efforts. Positive action and verified implementation is required to reduce the mishap risk associated with these situations to a level acceptable to the program manager [1].
- Acceptable conditions are considered acceptable for correcting unacceptable conditions and will require no further analysis once mitigating actions are implemented and verified [1].

### *A Safety Process for Waterfall Software Development*

When software is developed using the classic Waterfall development process, assuring that the software is safe is sometimes difficult. Nevertheless, the software safety process applied to classical software development is understood and practiced by experts today. The 15-element Software Safety Process (developed by APT Research, Inc.), shown in Figure 2 (see next page), is an example of a mature approach.

The fundamental premise is to focus the effort that is needed to perform software system safety. This is classically done by further focusing efforts on the safety subset of the system software. The safety-critical functions are identified from the system requirements documents, the prime development specifications, and the preliminary hazard analysis. Those safety-critical functions with direct or indirect software control are then identified and become the focus for the software system safety effort. The safety-critical software requirements that flow from the safety functional requirements are then identified. The software safety personnel perform this step while coordinating with system safety and software developers.

## Adaptation for Model-Driven Software Development

The system-level preliminary hazard analysis provides the framework for reasoning about sub-system (software in this case) hazard analysis in the form of candi-

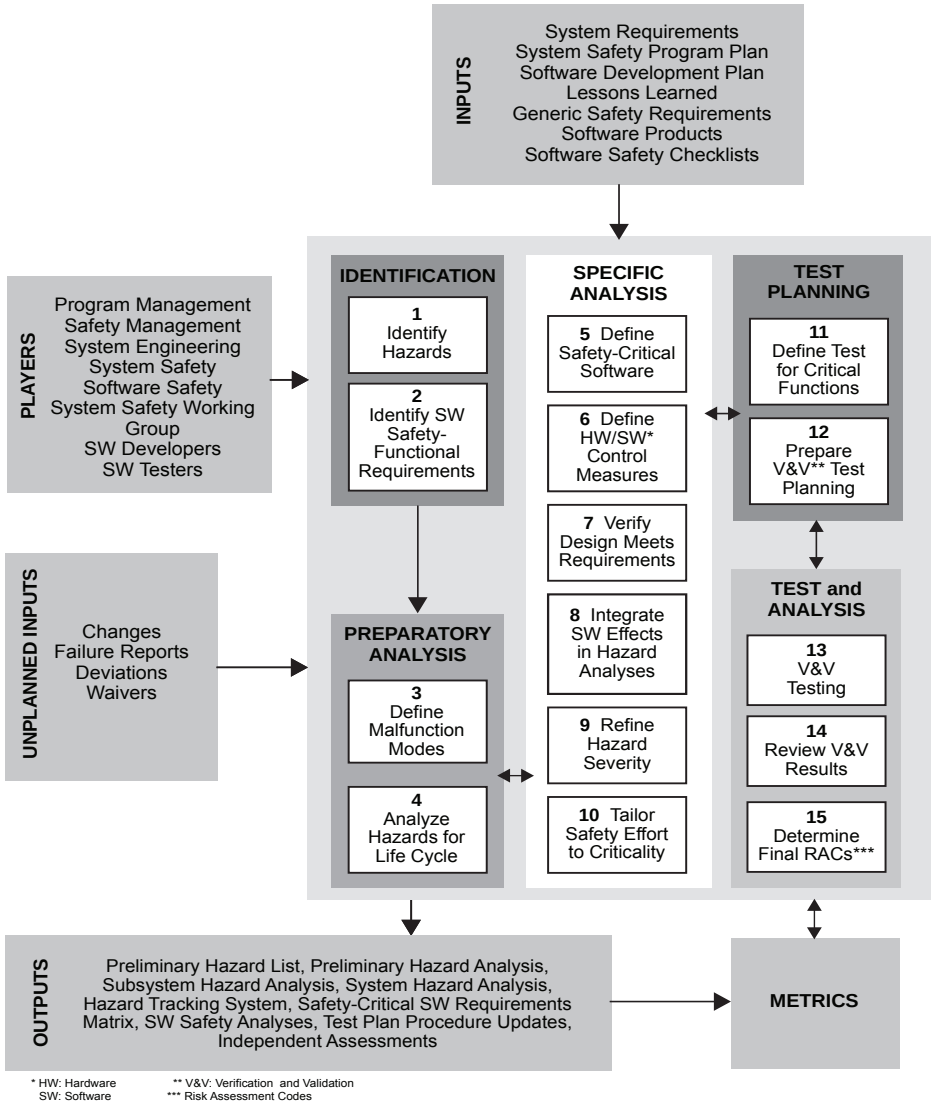


Figure 2: *15-Element Software Safety Process* [2]

| GEM Safety Process | 15-Element Software Safety Process | Phase |
|---|---|---|
| Continuous Throughout Increment | 1. Identify Hazards<br>2. Identify SW Safety-Functional Requirements | Identification |
| | 3. Define Malfunction Modes<br>4. Analyze Hazards for Life Cycle | Preparatory Analysis |
| | 5. Define Safety-Critical Software | Specific Analysis |
| | 8. Integrate SW Effects in Hazard Analyses<br>9. Refine Hazard Severity | |
| | 10. Tailor Safety Effort to Criticality | |
| 1.0 Perform Initial Survey of the Capabilities of the Cycle | 6. Define HW/SW Control Measures | |
| 2.0 Create and Document Policies for Domain Analysis – Use Cases | 6. Define HW/SW Control Measures<br>7. Verify Design Meets Requirements | |
| 3a. Realization of Controls and Mitigation in Collaborations/Algorithms<br>3b. Implement Controls and Mitigation | 6. Define HW/SW Control Measures<br>7. Verify Design Meets Requirements | |
| 4.0 Select Verification Evidence Approach for Use-Case Policies | 11. Design Test for Critical Functions<br>12. Prepare V&V Test Planning | Test Planning |
| 5.0 Evaluate Verification Cases for Software Safety | 12. Prepare V&V Test Planning | |
| 6.0 Safety Analysis of Verification Results | 13. V&V Testing<br>14. Review V&V Results<br>15. Determine Final RACs | Test and Analysis |
| 7.0 Create and Maintain the GAR (input to C2BMC Safety Assessment Report) | Updates to Safety Artifacts and Input to C2BMC Safety Assessment Report | N/A |

Table 3: *Mapping APT's 15 Elements to ABM Software Safety/Mission Assurance Analysis Activities*

date causes, contributors, controls, and policies within the context of a given required behavior. The differences start here. The analysis activity at the development level is not limited to the safety-critical software functions at first. During each cycle, the incremental specification and implementation draws the whole development staff into identifying broad environmental impacts and captures them in the design artifacts and in a development-level software assessment report called the GEM Assessment Report (GAR). This highlights the value of incorporating the software safety training program into the GEM program. The result is a highly collaborative, fully integrated involvement for:

- Identification of hazards and concerns.
- Evaluation of causes.
- Establishment of detection logic and mitigation policies.
- Selection of mitigator strategies.
- Verification case definition, development, and execution.
- Analysis of results.
- Collection of mitigation assessment and test evidence.

The combination of hazards and concerns coupled with the incremental development activities enhances the defect detection and avoidance mentality. The safety subject matter expert(s) can provide substantial leverage when mentoring the development staff during these activities.

Each of the 15-element Software Safety Process portions was examined for application to each of the development activities. In many cases, the elements were revisited and incrementally built up over the development cycles (see Table 3).

### *GEM Safety Activities Overview*

The GEM Software Safety Activities to support a model-driven, spiral software development effort are shown in Figure 3. The end result is a software safety program that includes all of the fundamental software safety elements, and incrementally grows and matures as the executable model evolves. The beginning of the software development process is the safety entry point into the GEM architecture and product. For the approach to work, it is critical to get buy-in from all stakeholders prior to beginning development.

All members of the GEM development staff should analyze their work in the context of the program's concerns. The following GEM safety principles are integrated into the software development process:

- Reasoning through safety mitigation

options modifies design and implementation trade space.
- Stringent coding standards on safety-critical software are required by MDA safety requirements.
- Analysis is reviewed incrementally by safety staff.
- Safety activities and artifacts are incorporated into the design from the beginning.

Use cases are the mechanisms for specifying GEM safety-required behavior within the GEM behavioral specification. Just as use cases provide the required behavior view, the GEM assessment report provides a view of the key concerns, their causes, contributors, controls, policies, and verifications as they have been defined.

The following stakeholders are to perform safety activities:
- Domain analysts.
- Model designers/architects.
- Model developers/implementers.
- Testers/test analysts.
- Safety subject matter experts.

During the domain analysis, model design, model development, and test, the stakeholders must be alert for the introduction of new hazard causes. New hazard causes may require additional hazard controls and verifications. Additionally, new hazard causes, controls, and verifications must be traced to the implementation and GEM assessment report. In the end, all stakeholders must become familiar with key safety artifacts: hazard logs (or database), hazard causes and controls, and the traceability of required behavior.

## System-Level Safety-Critical Functionality Assessment Is Still Fundamental

The fundamental safety premise still holds: Clear enumeration of the agreed-upon safety-critical functions and the assessment of the level of mitigation that exists in the implementation is needed. The safety-critical functions are identified and analyzed in context with the broader environment for which they will operate. Those safety-critical functions with direct or indirect software control are then identified and become the focus for the software system safety effort. The safety-critical software requirements flow from the system-level safety-functional requirements. The software safety personnel perform this step while coordinating with system safety and software developers. However, by using the GEM software safety activities, there is now a rich cause/contributor assessment captured in the development artifacts. Determining the appropriate control to mitigate the causes of the identified hazards has both a top-down and a bottom-up component.

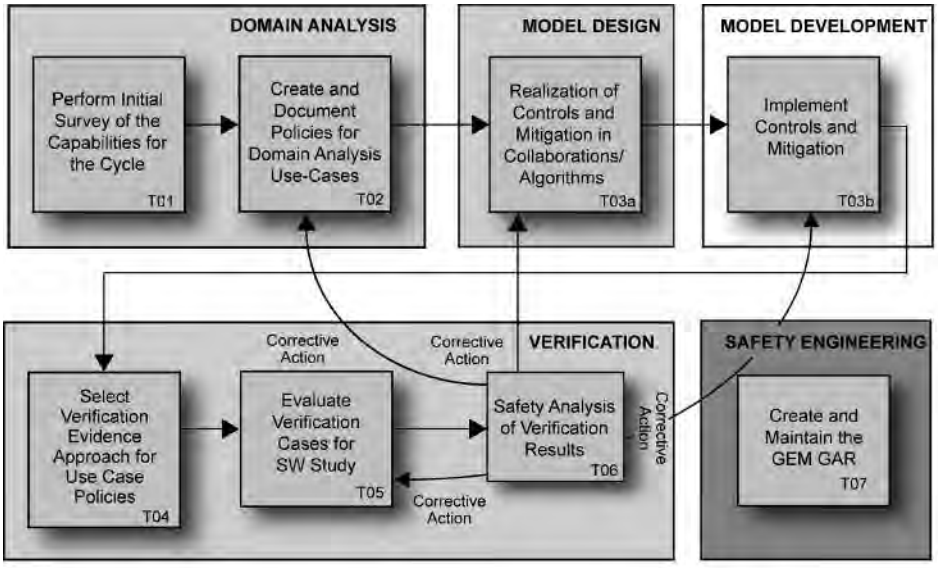


Figure 3: *GEM Software Safety Activities*

## Observations and Conclusions

Advancing software development methodologies lend themselves to a rich, comprehensive approach to safety analysis. It provides an open working relationship to incrementally identify the following causes at various tiers of granularity:
- Selection of architecture principles.
- Design and implementation of strategies for mitigators.
- Selection of verification cases that both enhance analysis of properly operating functions and mitigation mechanisms.
- Collection of the needed evidence to satisfy safety review boards.

The GEM software development approach maintains verification logic as additional functionality is added to the growing product base. This software safety analysis complements system-level safety analysis, as is currently being practiced.

In conclusion, a comparison of software safety in Waterfall development projects versus those that use Agile/spiral approaches is found in Table 4.

Additionally, when it comes to modifying the application of software safety analysis for model-driven, spiral-developed software (using the GEM development as an example) this article draws four observations:

1. State of practice in software engineering continues to make mainstream computer-aided engineering tools. Mission assurance and safety concerns are moving into the culture in the form of reliability and safety engineering constraints. Training is important, as seasoned developers/engineers do not know what is sufficiently complete or

Table 4: *Comparison of Waterfall and Agile/Spiral Approaches*

| Waterfall Development | Agile/Spiral |
|---|---|
| Requires complete sets (requirements, high-level design, low-level design, code, and test cases). | Proceeds with partial sets of overall development. |
| The focus on smaller details is achieved in later phases. | The overall viewpoint is achieved in later phases. |
| Testing begins later in the development. | Partial sections are tested sooner. |
| Lessons learned acquired in later phases. | Lessons learned acquired in earlier phases. |
| The size of the set of changes for correction/ enhancement tends to be larger. | The size of the set of changes for correction/ enhancement tends to be smaller and occur incrementally. |
| Configuration management is easier since the initial set of requirements tend to be fixed. | Configuration management is harder due to needed response to growing and varying requirements. |
| It is in later phases that the complete set of details come together that influence safety concerns. | It is in later phases that the overall viewpoint for efficient mitigation implementation can be selected. |

correct when it comes to these concerns.

2. System-level preliminary hazard analysis provides a framework for reasoning about software causal analysis in the form of candidate causes, contributors, controls, and policies within the context of a given required behavior. The use of incremental specification and implementation draws the whole development staff into identifying full environmental impacts that are captured both in the design artifacts and a development-level software assessment report. The importance of software safety training cannot be underestimated. The approach enhances the defect detection and avoidance mentality and allows the safety subject matter expert to mentor the development staff and have a high impact during these activities.
3. Safety-critical functions at the system level are used to define safety-critical software functions. These functions are reasoned through at the system level, and use the candidate policies to identify which policies will be tracked as safety-critical. The total approach provides both a top-down and bottom-up assessment.
4. Active safety subject-matter expert involvement is required in software development phase sometimes as the lead, sometimes as a mentor. This allows the program to gain the values afforded by advancing engineering techniques.◆

## References


1. DoD. "Standard Practice for System Safety." MIL-STD-882D, Appendix A. 10 Feb. 2000 <http://safetycenter.navy.mil/instructions/osh/milstd882d.pdf>.
2. APT Research, Inc. "The Safety Engineering and Analysis Center." 15 Oct. 2007 <www.apt-research.com/pages/about/S-07-00100_SEAC_Booklet.pdf>.


## Note

1. If manage-by-exception actions are warranted, the warfighters are to adaptively direct sensor and weapon system activities in coordination with the element commander. Event-triggered automated actions for elements are coordinated with similar automated actions by the GEM decision aid.

## About the Authors


**Timothy J. Trapp** is the global engagement manager chief engineer with the Missile Defense National Team/C2BMC. He has 25 years experience with the design, development, management, and operations of DoD and commercial interactive systems that are integrated with communications infrastructures. He holds a bachelor's degree in electrical engineering from Purdue University and a master's degree in engineering management from George Washington University. Trapp also holds a patent on the use of multicast-based distribution for timely and real-time data.

**Raytheon**
**2611 Jefferson Davis HWY**
**STE 700**
**Arlington, VA 22202**
**Phone: (703) 418-4288**
**E-mail: tim.j.trapp@mdnt.com**



**Howard D. Kuettner, Jr.** is the software safety lead for a major system development program at APT. He has more than 35 years experience in systems and software development, systems and software test, and systems and software safety. Kuettner has been a member of the System Safety Society (Tennessee Valley Chapter) since 2000, and was named their Engineer of the Year for 2003. He has a bachelor's degree in physics, and has co-authored papers presented at prior International System Safety Conferences.

**APT Research, Inc.**
**4950 Research DR**
**Huntsville, AL 35805**
**Phone: (256) 327-3383**
**E-mail: hkuettner@apt-research.com**



**Donald S. Hanline II** is a system safety engineer for the U.S. Army Aviation and Missile Command (AMCOM). He has more than 25 years of experience in the aerospace industry, with nine years of systems safety and software systems safety experience on command and control and weapons system development programs. Hanline serves the AMCOM Safety Office on independent safety review boards, the development of Army and AMCOM software system safety requirements, and software system safety training. He has bachelor's degrees in chemistry and mechanical engineering from the University of Alabama in Huntsville.

**U.S. Army AMCOM**
**ATTN: AMSAM-SF-A**
**Redstone Arsenal, AL 35898-5000**
**Phone: (256) 842-3248**
**E-mail: donald.s.hanline@us.army.mil**



**William A. Christian** is currently a software safety engineer for the GEM at APT. He has more than 30 years experience in the hardware and software requirements, design, implementation, and test. Christian has co-authored a paper on reviewing code for requirements verification and has spent more than 20 years in developing software for instrumentation, intercom systems, databases, and testing radio frequency applications.

**APT Research, Inc.**
**4950 Research DR**
**Huntsville, AL 35805**
**Phone: (256) 883-3474**
**E-mail: bchristian@apt-research.com**