Annex B

Glossary of Techniques

Current proven techniques that can be used to achieve and assess different aspects of information assurance throughout the life of a system are listed in Exhibits 2 through 5:

Exhibit 2 lists IA analysis techniques
Exhibit 3 lists IA design techniques and features
Exhibit 4 lists IA verification techniques
Exhibit 5 lists IA accident/incident investigation techniques

The exhibits identify (1) which techniques and groups of techniques are complementary or redundant; (2) the primary focus of each technique: safety, security, or reliability; and (3) the generic life-cycle phases in which the techniques can be used. Each project team should choose a complementary set of analysis, design, and verification techniques that are appropriate for the size, complexity, criticality, cost, and duration of their specific project/system. The results obtained from using these techniques is included as part of the evidence in the IA integrity case.

Many techniques used during the concept and development phases can also be used during the operational phase to (1) verify that the required IA integrity level is being maintained, (2) investigate an accident/incident, and (3) determine why a system did not achieve its stated IA goals. Likewise, many techniques serve multiple purposes. For example, some IA analysis techniques can also be used during verification and accident/incident investigation. Techniques that serve multiple purposes are noted in the exhibits.

Following each exhibit, a description of the technique* is provided in the following format:

* Annex B of *Software Safety and Reliability: Techniques, Approaches, and Standards of Key Industrial Sectors*, by Debra S. Herrmann, IEEE Computer Society Press, 1999, lists tools that are commercially available tools to automate many of these techniques.

Exhibit 1 Legend for Exhibits 2 through 5

Column	*Code*	*Meaning*
Type	SA	Technique primarily supports safety engineering
	SE	Technique primarily supports security engineering
	RE	Technique primarily supports reliability engineering
	All	Technique supports a combination of safety, security, and reliability engineering
C/R	Cx	Groups of complementary techniques
	Rx	Groups of redundant techniques; only one of the redundant techniques should be used.

- **Purpose:** summary of what is achieved using the technique; why the technique should be used.
- **Description:** summary of the main features of the technique and how to implement it.
- **Benefits:** how the technique enhances IA integrity or facilitates assessment; any cost benefits derived from using the technique.
- **Limitations:** factors that may limit the use of the technique, affect the interpretation of the results obtained, or impact the cost-effectiveness of the technique.
- **References:** sources for more information about the technique.

Exhibit 1 explains the codes used in Exhibits 2 through 5.

B.1 IA Analysis Techniques

B.1.1 Bayesian Belief Networks (BBNs)

Purpose: To provide a methodology for reasoning about uncertainty as part of risk analysis and assessment.

Description: Bayesian belief networks (BBNs) are graphical networks that represent probabilistic relationships among variables (events or propositions). The nodes represent uncertain variables and the arcs represent the cause/relevance relationships among the variables. The probability tables for each node provide the probabilities of each state of the variable for that node, conditional on each combination of values of the parent node.[5] As new knowledge or uncertainties are discovered, this information can be propagated through the BBN.

Benefits: BBNs provide the ability to combine logical inferences, objective evidence, and subjective expert judgment in one complete model. The graphical nature of BBNs improves communication among different stakeholders, developers, and assessment teams.

Limitations: The use of an automated tool, such as Hugin's BBN tool, is required to develop the models in a reasonable amount of time.

Exhibit 2 Information Assurance Analysis Techniques

IA Analysis Techniques	*C/R*	*Type*	*Life-cycle Phase in which Technique is Used*		
			Concept	*Development*	*Operations*
Bayesian belief networks (BBNs)[a]	C1	All	x	x	x
Cause consequence analysis[a,b]	R1/C1	SA, SE	x	x	x
Change impact analysis	C1	All		x	x
Common cause failure analysis[b]	C1	All	x	x	x
Develop operational profiles, formal scenario analysis	C1	All	x	x	x
Develop IA integrity cases	C1	All	x	x	x
Event tree analysis[a,b]	R1/C1	All	x	x	x
Functional analysis	C1	SA, SE	x	x	x
Hazard analysis	C1	SA, SE	x	x	x
HAZOP studies[a,b]	C1	SA, SE	x	x	x
Highlighting requirements likely to change	C1	All	x		
Petri nets[a,b]	C1	SA, SE		x	x
Reliability block diagrams	C1	RE	x	x	x
Reliability prediction modeling	C1	RE	x	x	
Response time, memory, constraint analysis	C1	All		x	x
Software, system FMECA[a,b]	C1	All	x	x	x
Software, system FTA[a,b]	R1/C1	SA, SE	x	x	x
Sneak circuit analysis[a,b]	C1	SA, SE		x	x
Usability analysis	C1	SA, SE	x	x	x

[a] These techniques can also be used during verification.

[b] These techniques can also be used during accident/incident investigation.

Source: Adapted from Herrmann, D., *Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.

References:

1. Bouissou, M., Martin, F., and Ourghanlian, A., Assessment of a safety-critical system including software: A Bayes-belief network for evidence sources, *Proceedings of the Annual Reliability and Maintainability Symposium (RAMS'99)*, IEEE, 1999, 142–150.
2. Jensen, F., *An Introduction to Bayesian Belief Networks*, Springer-Verlag, 1996.
3. Neil, M. and Fenton, N., Applying BBNs to critical systems assessment, *Safety Systems*, 8(3), 10–13, 1999.

4. Neil, M., Littlewood, B., and Fenton, N., Applying BBNs to system dependability assessment, *Safety-Critical Systems: The Convergence of High Tech and Human Factors*, Springer-Verlag, 1996, 71–94.
5. www.agena.co.uk; BBN articles and tutorials.

B.1.2 Cause Consequence Analysis

Purpose: To enhance IA integrity by identifying possible sequences of events that can lead to a system compromise or failure.

Description: Cause consequence analysis is a hybrid technique that combines fault tree analysis and event tree analysis. Beginning with a critical event, a cause consequence diagram is developed backward and forward from that event. The backward portion of the diagram is equivalent to a fault tree. The forward portion of the diagram is equivalent to an event tree in which possible consequences of the sequence of events are identified. Standard symbols have been defined for cause consequence diagrams so that propagation conditions, timing information, and probability of occurrence can be recorded and analyzed.

Benefits: Cause consequence diagrams are particularly well suited to studying start-up, shutdown, and other sequential control problems.[2] They facilitate analysis of combinations of events and alternative consequence paths.[2]

Limitations: Separate diagrams are required for each initiating event.[2]

References:

1. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
2. Leveson, N., *Safeware: System Safety and Computers*, Addison-Wesley, 1995.
3. Nielsen, B., The Cause Consequence Diagram Method as a Basis for Quantitative Accident Analysis, RISO-M-1374, 1971.
4. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.

B.1.3 Change Impact Analysis

Purpose: To analyze *a priori* the potential local and global effects of changing requirements, design, implementation, data structures, and/or interfaces on system performance, safety, reliability, and security; prevent errors from being introduced during enhancements or maintenance.

Description: Changing or introducing new requirements or design features may have a ripple effect on a current or proposed system. A change or fix can be applied to one part of a system with detrimental or unforeseen consequences to another part. Change impact analysis evaluates the extent and impact of proposed changes by examining which requirements and design components are interdependent. The analysis evaluates whether or not the proposed change could invoke a vulnerability/threat, affect a threat control measure, increase the likelihood or severity of a vulnerability/threat, adversely affect IA-critical or IA-related software, or change the criticality of a software component.[5] Change impact analysis should be conducted when[2]:

- The operational environment has changed
- System components are being modified or replaced
- The system is to be used for a new or different application than it was originally designed
- Changes are proposed to the requirements, design, implementation
- Preventive, corrective, or adaptive maintenance is being performed

Change impact analysis can also be used to support analysis of alternatives, by highlighting which alternative can be implemented most efficiently, and to identify the extent of reverification and revalidation needed. (*See also* Regression Testing.)

Benefits: The potential for uncovering latent defects or introducing new errors when implementing changes or enhancements is minimized.

Limitations: The scope of the analysis determines its effectiveness.

References:

1. Arnold, R. and Bohner, S., *Software Change Impact Analysis*, IEEE Computer Society Press, 1996.
2. DEF STAN 00-55, Requirements for Safety-Related Software in Defence Equipment, Part 1: Requirements, U.K. Ministry of Defence (MoD), August 1, 1997.
3. DEF STAN 00-55, Requirements for Safety-Related Software in Defence Equipment, Part 2: Guidance, U.K. Ministry of Defence (MoD), August 1, 1997.
4. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
5. NASA GB-1740.13.96, *Guidebook for Safety-Critical Software — Analysis and Development*, NASA Glenn Research Center, Office of System Safety and Mission Assurance, 1996.
6. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.

B.1.4 Common Cause Failure (CCF) Analysis

Purpose: To enhance IA integrity by identifying scenarios in which two or more failures or compromises could occur as the result of a common design defect.

Description: Common cause failure (CCF) analysis seeks to identify intermediate and root causes of potential failure modes. The results of CCF analysis are often documented graphically by event trees. This information is analyzed to determine failures that could result from common design defects, hardware failures, or operational anomalies and to propose the requisite mitigating actions, such as the need for diversity. CCF analysis includes hardware, software, and communications equipment. It is essential that fault-tolerant designs be verified through CCF analysis. (*See also* Diversity, Redundancy, and Root Cause Analysis.)

Benefits: Common cause failure analysis results in a more robust system architecture.

Limitations: The extent to which the analysis is carried out (e.g., how far back intermediate and root causes are identified) determines its effectiveness.

References:

1. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
2. Space Product Assurance: Safety, European Space Agency, ECSS-Q-40A, April 19, 1996.
3. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.

B.1.5 Develop Operational Profiles and Formal Scenario Analysis

Purpose: To identify operational profiles; capture domain knowledge about MWFs and MNWFs; understand human factors safety, reliability, and security concerns.

Description: A scenario-based test model is developed from the analysis of operational profiles, user views, and events. Operational profiles are an ordered sequence of events that accomplishes a functional requirement specified by an end user.[1] User views are a set of system conditions specific to a class of users.[1] Events are particular stimuli that change a system state or trigger another event.[3] Operational profiles are recorded in a formalized tree notation, similar to that used for finite state machines. Probabilities are assigned to each potential set of operations.[1,2]

Benefits: The development of operational profiles and formal scenario analysis helps to identify deadlock, nondeterministic conditions, incorrect sequences, incorrect initial and terminating states, and errors caused by an incomplete understanding of the domain knowledge. (*See also* Usability Testing.)

Limitations: The development of operational profiles and formal scenario analysis is somewhat labor intensive; both developers and end users are involved.

References:

1. Herrmann, D., *Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.
2. Hsia, P., *Testing the Therac-25: A Formal Scenario Approach, Safety and Reliability for Medical Device Software*, Herrmann, D. (Ed.), Health Industries Manufacturers Association (HIMA) Report No. 95-8, 1995, tab 6.
3. Lyu, M. (Ed.), *Handbook of Software Reliability Engineering*, IEEE Computer Society Press, 1996.
4. Pant, H., Franklin, P., and Everett, W., A structured approach to improving software-reliability using operational profiles, *Proceedings of the Annual Reliability and Maintainability Symposium*, IEEE, 1994, 142–146.

B.1.6 Develop IA Integrity Case

Purpose: To collect, organize, analyze, and report information to prove that IA integrity requirements have been (or will be) achieved and maintained.

Description: An IA integrity case is a living document throughout the life of a system, from initial concept through decommissioning. An IA integrity case consists of seven components (see Chapter 7, Exhibit 9):

- IA goals
- Assumptions and claims
- (Current) evidence

- Conclusions and recommendations
- Outstanding issues
- Approval, certification records
- Backup, supporting information

The preliminary IA integrity case provides a justification that the recommended architecture and threat control measures will achieve specified IA goals. It can be prepared during the proposal or BAFO stage. The interim IA integrity case collects ongoing evidence that indicates a project is on track for meeting specified IA requirements. It is prepared during the development and assessment stage. The operational IA integrity case is a complete set of evidence that the specified IA integrity requirements were met and are being maintained. (*See also* Review IA Integrity Case.)

Benefits: The structure imposed by developing an IA integrity case helps system designers and developers to be more thorough when addressing IA integrity issues. Organized and complete IA integrity cases help certifying authorities to perform a more effective and thorough assessment.

Limitations: None.

References:

1. DEF STAN 00-42, Reliability and Maintainability Assurance Guides, Part 2: Software, U.K. Ministry of Defence (MoD), September 1, 1997.
2. DEF STAN 00-55, Requirements for Safety-Related Software in Defence Equipment, Part 1: Requirements, U.K. Ministry of Defence (MoD), August 1, 1997.
3. DEF STAN 00-55, Requirements for Safety-Related Software in Defence Equipment, Part 2: Guidance, U.K. Ministry of Defence (MoD), August 1, 1997.
4. Herrmann, D., *Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.
5. Herrmann, D. and Peercy, D., Software Reliability Cases: The bridge between hardware, software and system safety and reliability, *Proceedings of the Annual Reliability and Maintainability Symposium (RAMS'99)*, IEEE, 1999, 396–402.
6. JA 1002, Software Reliability Program Standard, Society of Automotive Engineers (SAE), 1998.
7. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.

B.1.7 Event Tree Analysis

Purpose: To enhance IA integrity by preventing defects through analysis of sequences of system events and operator actions that could lead to failures, compromises, or unstable states.

Description: Event trees organize, characterize, and quantify potential system failures in a methodical manner.[5] An event tree is developed in a graphical notation following a six-step process:

1. Identify all possible events (accidental and intentional) that could initiate a system compromise or failure.
2. Identify the system response.
3. Identify the mitigating threat control measure(s).
4. Group initiating events with their corresponding responses.

5. Identify initiating event/response branches that will lead to a system compromise or failure.
6. Assign probabilities to each branch in the event tree.

This process is repeated until all initiating events and threat control measures have been evaluated. Note that it is possible for some responses to act as new initiating events.

Benefits: Event tree analysis is considered to be one of the more exhaustive analysis techniques and particularly well suited for high-risk systems.[5]

Limitations: The effectiveness of this technique is proportional to the ability to anticipate all unwanted events and all of the potential causes of these events.[3] Use of an automated tool is necessary. Event tree analysis can be very time-consuming if not focused correctly.

References:

1. Bott, T., Evaluating the risk of industrial espionage, *Proceedings of the Annual Reliability and Maintainability Symposium (RAMS'99)*, IEEE, 1999, 230–237.
2. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
3. Leveson, N., *Safeware: System Safety and Computers*, Addison-Wesley, 1995.
4. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.
5. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.

B.1.8 Functional Analysis

Purpose: To identify safety and security hazards associated with normal operations, degraded mode operations, incorrect usage, inadvertent operation, absence of function(s), and accidental and intentional human error.

Description: Functional analysis is conducted to identify potential hazards that could result from correct or incorrect functioning and use of the system. Functional analysis is conducted throughout the life cycle, from concept definition, requirements specification and design, to implementation and operation. The first step is to diagram relationships between components and their functions, including lower level functions; for example[2]:

- Principal functions
- Subsidiary functions
- Warning functions
- Operator indication and control functions
- Protection functions
- Human operator initiated functions
- Failure mitigation functions

Accidental and intentional, random and systematic failures are examined. All functional modes are evaluated, including:

- Normal operation
- Abnormal operation
- Degraded mode operations

- Incorrect operation
- Inadvertent operation
- Absence of functionality
- Human error that causes functions to be operated too fast, too slow, or in the wrong sequence

Benefits: Functional analysis is a comprehensive technique. It highlights the contribution of low-level functions to hazards and complements FTA, FMECA, and HAZOP studies.

Limitations: To employ this technique effectively, all stakeholders must be involved in the analysis, and the analysis must be carried out to the lowest-level functions.

References:

1. DEF STAN 00-56, Safety Management Requirements for Defense Systems Containing Programmable Electronics, Part 1: Requirements, U.K. Ministry of Defence (MoD), December 13, 1996.
2. DEF STAN 00-56, Safety Management Requirements for Defense Systems Containing Programmable Electronics, Part 2: General Application Guidance, U.K. Ministry of Defence (MoD), December 13, 1996.

B.1.9 Hazard Analysis

Purpose: To enhance IA integrity by identifying potential hazards associated with using a system so that appropriate mitigation features can be incorporated into the design and operational procedures.

Description: Hazard analysis is a category of techniques used to identify hazards so that they can be eliminated or mitigated. FTA, event tree analysis, sneak circuit analysis, and Petri nets are all examples of hazard analysis techniques.

Hazard analyses are performed throughout the life cycle to explore safety, reliability, and security concerns. A preliminary hazard analysis is performed based on the requirements specification and concept of operations. Subsequent hazard analyses are performed on the design, source code, operational profiles, and the operational system. All anomalies and recommended corrective action are noted as part of the hazard analyses and tracked to closure. Examples of items to evaluate during a hazard analysis include:[2,3,4]

- Cause(s) of a hazard or vulnerability, whether accidental or intentional
- Severity of the consequences of a hazard
- Likelihood of a threat triggering a hazard
- Alternative threat control strategies
- Effective exception handling
- Effective handling of errors
- Efficient transitioning to degraded mode operations, fail safe/secure, or fail operational when needed
- Conditions that could cause a system to enter an unknown or unsafe/unsecure state, such as transient faults, excessive interrupts, and saturation

Benefits: Potential hazards are identified early during the life cycle, when it is easier and cheaper to eliminate or mitigate them. The hazard analysis process, by identifying the severity and likelihood of hazards, facilitates the efficient assignment of resources to the most critical hazards.

Limitations: The comprehensiveness of the hazard analyses determines their utility.

References:

1. CE-1001-STD Rev. 1, Standard for Software Engineering of Safety-Critical Software, CANDU Computer Systems Engineering Centre for Excellence, January 1995.
2. Leveson, N., *Safeware: System Safety and Computers*, Addison-Wesley, 1995.
3. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.
4. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.

B.1.10 HAZOP Studies

Purpose: To prevent potential hazards (accidental and intentional, physical and cyber) by capturing domain knowledge about operational environment, parameters, modes/states, etc. so that this information can be incorporated in the requirements, design, and operational procedures.

Description: A hazard and operability (HAZOP) study is a method of discovering hazards in a proposed or existing system, their possible causes and consequences, and recommending solutions to minimize the likelihood of occurrence.[1] The hazards can be physical or cyber, and result from accidental or malicious intentional action. Design and operational aspects of the system are analyzed by an interdisciplinary team. A neutral facilitator guides the group through a discussion of how a system is or should be used. Particular attention is paid to usability issues, operator actions (correct and incorrect, under normal and abnormal conditions), and capturing domain knowledge. A series of guide words are used to determine correct design values for system components, interconnections and dependencies between components, and the attributes of the components.

Benefits: This is one of the few techniques to focus on (1) hazards arising from the operational environment and usability issues and (2) capturing domain knowledge from multiple stakeholders.

Limitations: The facilitator must be adequately trained in the methodology for the sessions to be effective.

References:

1. DEF STAN 00-58, HAZOP Studies on Systems Containing Programmable Electronics, Part 1: Requirements, U.K. Ministry of Defence (MoD), interim, July 25, 1996.
2. DEF STAN 00-58, HAZOP Studies on Systems Containing Programmable Electronics, Part 2: General Application Guidance, U.K. Ministry of Defence (MoD), interim, July 25, 1996.
3. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
4. Leveson, N., *Safeware: System Safety and Computers*, Addison-Wesley, 1995.
5. Redmill, F., Chudleigh, M., and Catmur, J. *System Safety: HAZOP and Software HAZOP*, John Wiley & Sons, 1999.
6. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.
7. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.

B.1.11 Highlighting Requirements Likely to Change

Purpose: To enhance the maintainability of threat control measures and IA integrity.

Description: During concept definition and requirements specification, time is taken to identify requirements that are likely to change in future system releases, due to anticipated enhancements, upgrades, and changes or additions to the operational mission. Attention is focused on IA-critical and IA-related functions/entities. This information is then fed into the design process. Components that are likely to change are partitioned from those that are more stable.

Benefits: Future maintainability and supportability is simplified. The cost of implementing and planning for new system releases is reduced. The likelihood of disrupting a threat control measure when performing adaptive maintenance is reduced.

Limitations: None.

References:

1. CE-1001-STD Rev. 1, Standard for Software Engineering of Safety Critical Software, CANDU Computer Systems Engineering Centre for Excellence, January 1995.
2. DEF STAN 00-41/Issue 3, Reliability and Maintainability, MoD Guide to Practices and Procedures, U.K. Ministry of Defence (MoD), June 25, 1993.

B.1.12 Petri Nets

Purpose: To identify potential deadlock, race, and nondeterministic conditions that could lead to a system compromise or failure.

Description: Petri nets are used to model relevant aspects of system behavior at a wide range of abstract levels.[2] Petri nets are a class of graph theory models that represent information and control flow in systems that exhibit concurrency and asynchronous behavior.[2,3,6] A Petri net is a network of states and transitions. The states may be marked or unmarked; a transition is enabled when all the input places to it are marked.[2,3,6] When enabled, it is permitted but not obliged to fire. If it fires, the input marks are removed and each output place from the transition is marked instead.[2,3,6] These models can be defined in purely mathematical terms, which facilitates automated analysis, such as producing reachability graphs.[2]

Benefits: Petri nets can be used to model an entire system, subsystems, or subcomponents at conceptual, top-level design, and implementation levels.[2] They are useful for identifying deadlock, race, and nondeterministic conditions that could lead to a system compromise or failure.

Limitations: The production of Petri nets can be time-consuming without the use of an automated tool.

References:

1. Buy, U. and Sloan, R., Analysis of real-time programs with simple time Petri nets, *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, ACM Press, 1994, 228–239.
2. Herrmann, D., *Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.
3. Jensen, K., *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, Springer-Verlag, Vol. 1, 1996; Vol. 2, 1995.

4. Lindemann, C., *Performance Modelling with Deterministic and Stochastic Petri Nets*, John Wiley & Sons, 1998.
5. NASA GB-1740.13.96, *Guidebook for Safety-Critical Software — Analysis and Development*, NASA Glenn Research Center, Office of System Safety and Mission Assurance, 1996.
6. Peterson, J., *Petri-Net Theory and the Modeling of Systems*, Prentice-Hall, 1981.
7. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.

B.1.13 Reliability Block Diagrams

Purpose: To enhance IA integrity by diagrammatically identifying the set of events that must take place and the conditions that must be fulfilled for a system or task to execute correctly[1,2]; support initial reliability allocation, reliability estimates, and design optimization.

Description: Reliability block diagrams illustrate the relationship between system components with respect to the effect of component failures on overall system reliability. These relationships generally fall into four categories:

- A serial system
- A dual redundant system
- M out of n redundant systems
- A standby redundant system

Reliability block diagrams are annotated to show[1]:

- The reliability and maintainability values assigned to each block, such as MTBF and MTTR
- Assumptions about each component
- Operational profiles
- Item criticality
- Dependencies between blocks that are not apparent from the diagram
- Development risk

Benefits: Reliability block diagrams are useful for analyzing systems that are composed of multiple diverse components, such as hardware, software, and communications equipment.

Limitations: A reliability block diagram does not necessarily represent the system's operational logic or functional partitioning.[4]

References:

1. DEF STAN 00-41/Issue 3, Reliability and Maintainability, MoD Guide to Practices and Procedures, U.K. Ministry of Defence (MoD), June 25, 1993.
2. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
3. IEC 61078(1991), Analysis Techniques for Dependability — Reliability Block Diagram Method.
4. O'Connor, P., *Practical Reliability Engineering*, 3rd ed., John Wiley & Sons, 1991.
5. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.

B.1.14 Reliability Prediction Modeling

Purpose: To predict the future reliability of a software system.

Description: The failure probability of a new program, usually one that is under development, is predicted in part by comparing it to the known failure probability of an existing operational program. The criteria for determining the degree of similarity include: design similarity, similarity of service use profile, procurement and project similarity, and proof of reliability achievement. The generic reliability prediction process also involves estimating the fault density per KSLOC. This value is then used to predict the number of errors remaining in the software and the time it will take to find them.

Benefits: This model can be executed at any time during the life cycle.

Limitations: The validity of the prediction depends on the similarity between the program, its operational environment and operational profile(s), and that to which it is compared. None of the current reliability prediction models incorporate data from qualitative assessments or static analysis techniques.

References:

1. ANSI/AIAA R-0133-1992, Recommended Practice for Software Reliability.
2. BS5760, Part 8: Guide to the Assessment of Reliability of Systems Containing Software, British Standards Institution (BSI), October 1998.
3. DEF STAN 00-42, Reliability and Maintainability Assurance Guide, Part 2: Software, U.K. Ministry of Defence, 1998.
4. IEEE Std. 982.1-1988, IEEE Standard Dictionary of Measures to Produce Reliable Software.*
5. IEEE Std. 982.2-1988, IEEE Guide for the Use of the Standard Dictionary of Measures to Produce Reliable Software.
6. Lyu, M. (Ed.), *Handbook of Software Reliability Engineering*, IEEE Computer Society Press, 1996.
7. Musa, J., *Software Reliability Engineering*, McGraw-Hill, 1999.
8. Peters, W., *Software Engineering: An Engineering Approach*, John Wiley & Sons, 1999.
9. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.

B.1.15 Response Time, Memory, Constraint Analysis

Purpose: To ensure that the operational system will meet all stated response time, memory, and other specified constraints under low, normal, and peak loading conditions.[3]

Description: Constraint analysis evaluates restrictions imposed by requirements, the real world, and environmental limitations, as well as the design solution.[2] Engineering analyses are conducted by an integrated product team to evaluate the system architecture and detailed design. The allocation of response time budgets between hardware, system software, application software, and communications equipment are examined to determine if they are realistic and comply with stated requirements. An assessment is made to determine if the available memory is sufficient for the system and application software. Minimum

* Note that this standard began an update cycle in late 1999.

and maximum system throughput capacity under low, normal, peak, and overload conditions is estimated. Timing and sizing analysis for IA-critical and IA-related functions/entities are evaluated against maximum execution time and memory allocation, particularly under worst-case scenarios. Items to consider when quantifying timing/resource requirements include[2,4]:

- Memory usage versus availability
- I/O channel usage (load) versus capacity and availability
- Execution time versus CPU load and availability
- Sampling rates versus rates of change of physical parameters
- Sensor/actuator accuracy and calibration
- Physical time constraints and response times
- Minimum time required to transition between modes/states
- Minimum time required for human response/action
- Off-nominal environments

Benefits: Design deficiencies, which could cause safety and security vulnerabilities are uncovered before full-scale development.

Limitations: This static analysis technique should be supplemented by performance and stress testing.

References:

1. Briand, L. and Roy, D., *Meeting Deadlines in Hard Real-Time Systems: The Rate Monotonic Approach*, IEEE Computer Society Press, 1999.
2. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
3. Leveson, N., *Safeware: System Safety and Computers*, Addison-Wesley, 1995.
4. NASA GB-1740.13.96, *Guidebook for Safety-Critical Software — Analysis and Development*, NASA Glenn Research Center, Office of System Safety and Mission Assurance, 1996.
5. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.

B.1.16 Software, System FMECA

Purpose: To examine the effect of accidental and intentional, random and systematic failures on system behavior in general and IA integrity in particular.

Description: A failure mode effects criticality analysis (FMECA) identifies the ways in which a system could fail accidentally or be made to fail intentionally, and thus impact IA integrity. All stakeholders are involved in an FMECA to ensure that all aspects of a failure are adequately evaluated. FMECAs are conducted and refined iteratively throughout the life of a system. There are three types of FMECA: functional FMECA, design FMECA, and interface FMECA.[7] FMECA can and should be conducted at the system entity level (hardware, software, communications equipment, human factors) and at the system level. FMECAs (1) help to optimize designs, operational procedures, and fault tolerance strategies, (2) uncover operational constraints imposed by a design, and (3) verify the robustness of IA design techniques/features or the need for corrective action.[1]

The procedure for conducting a software FMECA is straightforward.[1–3,6,7,9] The software is broken into logical components, such as functions or tasks. Potential worst-case failure modes are predicted for each component. The cause(s) of these failure modes and their effect on system behavior is (are) postulated. Finally, the severity and likelihood of each failure mode are determined. In general, quantitative likelihoods are used to estimate random failures, while qualitative likelihoods are used to estimate systematic failures. Reliability block diagrams and the system operation characterization are used as inputs to an FMECA. Type failure modes examined include[1]:

- Premature operation
- Failure to operate at a prescribed time
- Intermittent operation
- Failure to cease operation at a prescribed time
- No output, wrong output, partial output
- Failure during operation

The effect of each failure mode is evaluated at several levels, such as[1]:

- Local effect
- Effect at the next higher level of assembly or function
- Effect on the system and its operational mission

The effect of failures is examined at different levels to (1) optimize fault containment strategies and (2) identify whether or not a failure at this level creates the conditions or opportunity for a parallel attack, compromise, or failure. The principle data elements collected, analyzed, and reported for each failure mode are:

- System, entity, and function
- Operational mission, profile, and environment
- Assumptions and accuracy concerns
- The failure mode
- Cause(s) of the failure
- Likelihood of the failure occurring
- Severity of the consequences of the failure
- Responsible component, event, or action
- Current compensating provisions: anticipate/prevent, detect/characterize, and respond/recover
- Recommended additional mitigation

Benefits: The results of an FMECA can be used to prioritize and verify threat control measures and as input to an FTA.

Limitations: An FMECA only captures known potential failure modes; it does not accommodate reasoning about uncertain or unknown failure modes.[4,5] The development of a software FMECA can be labor intensive unless an automated tool is used.[1]

References:

1. DEF STAN 00-41/Issue 3, Reliability and Maintainability, *MoD Guide to Practices and Procedures*, U.K. Ministry of Defence (MoD), June 25, 1993.
2. IEC 60812(1985), Analysis Techniques for System Reliability — Procedure for Failure Modes Effects Analysis (FMEA).
3. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
4. Leveson, N., *Safeware: System Safety and Computers*, Addison-Wesley, 1995.
5. NASA GB-1740.13.96, *Guidebook for Safety-Critical Software — Analysis and Development*, NASA Glenn Research Center, Office of System Safety and Mission Assurance, 1996.
6. Raheja, D., *Assurance Technologies: Principles and Practices*, McGraw-Hill, 1991.
7. SAE Recommended Best Practices for FMECA, (draft) March 1999.
8. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.
9. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.

B.1.17 Software, System FTA

Purpose: To identify potential root cause(s) of undesired system events (accidental and intentional) so that mitigating features can be incorporated into the design and operational procedures.

Description: FTA (fault tree analysis) aids in the analysis of events, or combinations of events, that will lead to a physical or cyber hazard.[2] Starting at an event that would be the immediate cause of a hazard, the analysis is carried out backward along a path.[3,4] Combinations of events are described with logical operators (AND, OR, IOR, EOR).[3,4] Intermediate causes are analyzed in the same manner back to the root cause.[3,4] A software FTA follows the same procedure as a hardware or system FTA to identify the root cause(s) of a major undesired event.[2] An FTA should be developed iteratively throughout the life cycle and in conjunction with an FMECA.[2]

Benefits: A software FTA can be merged with a hardware or system-level FTA. FTA complements FMECA.[1] The effects of nontechnical failures can be analyzed, such as human error, weather, etc.[1] All possible component failure combinations are identified.[1]

Limitations: A fault tree only captures known potential faults; it does not accommodate reasoning about uncertain or unknown faults.[4,5] The development of a software FTA can be labor intensive unless an automated tool is used.[1,4]

References:

1. DEF STAN 00-41/Issue 3, Reliability and Maintainability, *MoD Guide to Practices and Procedures*, U.K. Ministry of Defence (MoD), June 25, 1993.
2. Herrmann, D., *Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.
3. IEC 61025(1990), Fault Tree Analysis.
4. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
5. Leveson, N., *Safeware: System Safety and Computers*, Addison-Wesley, 1995.

6. NASA GB-1740.13.96, *Guidebook for Safety-Critical Software — Analysis and Development,* NASA Glenn Research Center, Office of System Safety and Mission Assurance, 1996.
7. Raheja, D., *Assurance Technologies: Principles and Practices*, McGraw-Hill, 1991.
8. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.
9. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.

B.1.18 Sneak Circuit Analysis

Purpose: To identify hidden unintended or unexpected hardware or software logic paths or control sequences that could inhibit desired system functions, initiate undesired system events, or cause incorrect timing and sequencing, leading to a system compromise or failure.[3]

Description: Sneak circuits are latent conditions that are intentionally or accidentally designed into a system, which may cause it to perform contrary to specifications and affect safety, reliability, and security.[2,5,6] Maintenance assist modes are examples of intentional benign sneak circuits; however, if these techniques are not implemented correctly, they can have unintended negative consequences.[7] Trap doors and Trojan horses are examples of malicious intentional sneak circuits.

The first step in sneak circuit analysis is to convert the design into a topological network tree, identifying each node of the network.[1–3] The use and interrelationships of instructions are examined to identify potential sneak circuits.[2] All possible paths through a software component or circuit are examined because sneak circuits can result from a combination of hardware, software, and operator actions. Categories of sneak circuits that are searched for include[1]:

- **Sneak paths,** which cause current, energy, data, or logical sequence to flow along an unexpected path or in an unintended direction
- **Sneak timing,** in which events occur in an unexpected or conflicting sequence
- **Sneak indications,** which cause an ambiguous or false display of system operating conditions and thus can result in an undesired action by an operator or process
- **Sneak labels,** which incorrectly or imprecisely label system functions or events, such as system inputs, controls, displays, buses, etc. and thus may mislead an operator into applying an incorrect stimulus to the system

Hardware sneak circuits include[3] sneak paths, sneak opens, sneak timing, sneak indications, and sneak labels. Software sneak circuits include[3] sneak outputs, sneak inhibits, sneak timing, and sneak messages.

The final step is to recommend appropriate corrective action to resolve anomalies discovered by the analysis.[5]

Benefits: Unintended, unauthorized logic paths and control sequences are identified and removed prior to a system being fielded. These defects are not normally found by other testing and analysis methods.[1]

Limitations: Sneak circuit analysis is somewhat labor intensive and should only be applied to IA-critical and IA-related functions/entities. Use of an automated tool is required.[1]

References:

1. DEF STAN 00-41/Issue 3, Reliability and Maintainability, *MoD Guide to Practices and Procedures*, U.K. Ministry of Defence (MoD), June 25, 1993.
2. Herrmann, D., *Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.
3. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
4. Raheja, D., *Assurance Technologies: Principles and Practices*, McGraw-Hill, 1991.
5. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.
6. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.
7. Whetton, C., Maintainability and its influence on system safety, *Technology and Assessment of Safety-Critical Systems*, Springer-Verlag, 1994, 31–54.

B.1.19 Usability Analysis

Purpose: To enhance operational IA integrity by ensuring that software is easy to use so that effort by human users to obtain the required service is minimal[1]; prevent accidental induced or invited errors that could lead to a system failure or compromise.

Description: Human error is a principal cause of accidental serious system failures and compromises. The likelihood of such errors can be influenced by IA design features/techniques.[2] Usability analysis is a method of analyzing a system design to identify ways to eliminate or reduce the likelihood of accidental induced or invited errors. This method consists of three steps[2]: hierarchical task analysis, human error identification, and error reduction. During hierarchical task analysis, all human tasks, activities, and steps are identified for administrators, end users, and maintenance staff. This information is analyzed and recorded in a tabular format:

- **Stimulus** human received to take some action
- **Action** human takes in response to the stimulus
- **Feedback** human receives from taking the action

Next, each stimulus/action/feedback scenario is examined for opportunities or factors that could contribute to human error. Specific items examined include[2]:

- **Information presentation;** for example, the distinctiveness of different types of parameters and commands when they are displayed and entered, and the clarity of units of measure
- **Distractions** in the operational environment; for example, lighting, noise, motion, or vibration
- **Human factors;** for example, skill level, mental and physical fatigue, boredom, overload, and stress, ease of learning how to use the system, adequacy of warnings, alarms, and operator feedback

Finally, action is taken to eliminate or mitigate errors that have the highest likelihood and severity, by[2]:

- Implementing defense in depth
- Improved ergonomics
- Improved operational procedures and contingency plans
- Improved training

Many of the attributes evaluated relate to human factor engineering issues. Attributes should be evaluated under all operational modes/states and profiles. Usability analysis should be conducted on the requirements, design, and implementation of IA-critical and IA-related functions/entities. All stakeholders should be involved.

Benefits: Usability analysis is one of the few techniques to consider the operational environment and as such helps to reduce the likelihood of accidental induced or invited errors.

Limitations: Participation by the end users and human factor engineers is critical.

References:

1. CE-1001-STD Rev. 1, Standard for Software Engineering of Safety Critical Software, CANDU Computer Systems Engineering Centre for Excellence, January 1995.
2. DEF STAN 00-41/Issue 3, Reliability and Maintainability, *MoD Guide to Practices and Procedures*, U.K. Ministry of Defence (MoD), June 25, 1993.
3. Hackos, J. and Redish, J., *User and Task Analysis for Interface Design*, John Wiley & Sons, 1998.
4. Hix, D. and Hartson, H., *Developing User Interfaces: Ensuring Usability Through Product and Process*, John Wiley & Sons, 1993.
5. Nielsen, J. and Mack, R., *Usability Inspection Methods*, John Wiley & Sons, 1994.

B.2 IA Design Techniques/Features

B.2.1 Access Control

Purpose: To protect IA-critical and IA-related systems, applications, and data by preventing unauthorized and unwarranted access to these resources.

Description: Access control is a set of design features that is implemented to control access to system resources, such as networks, computer systems, individual software applications, data, utilities, and peripherals such as printers. Access control consists of two main components: (1) access rights that define which people and processes can access which system resources, and (2) access privileges that define what these people and processes can do with or to the resources accessed.[2] Examples of access privileges are read, write, edit, delete, execute, copy, print, move, forward, distribute, etc. Access control rights and privileges can be defined on a need-to-know basis or by a security classification scheme. Access control rights and privileges are generally defined in a matrix format by user name, user roles, and local or global user groups. Access control is usually implemented through a combination of commercial operating system utilities and custom code. Two important aspects of implementing access control are

Exhibit 3 Information Assurance Design Techniques and Features

IA Design Techniques and Features	*C/R*	*Type*	*Life-cycle Phase in which Technique is Used*		
			Concept	*Development*	*Operations*
Access control	C2	SA, SE	x	x	x
Rights					
Privileges					
Account for all possible logic states	C2	SA, SE		x	x
Audit trail, security alarm	C2	SE	x	x	x
Authentication	C2	SA, SE	x	x	x
Biometrics					
Data origin					
Digital certificates					
Kerberos					
Mutual					
Peer entity					
Smartcards					
Unilateral					
Block recovery	C2	All		x	x
Confinement	C2	SA, SE		x	x
DTE					
Least privilege					
Wrappers					
Defense in depth	C2	All	x	x	x
Defensive programming	C2	All		x	x
Degraded-mode operations, graceful degradation	R2/ C2	All		x	x
Digital signatures	C2	SE		x	x
Nonrepudiation of origin					
Nonrepudiation of receipt					
Diversity	C2	SA, SE	x	x	x
Hardware					
Software					
Encryption	C2	SE	x	x	x
Asymmetric					
Symmetric					
Block					
Stream					
Hardware					
Software					
Error detection, correction	C2	All		x	x
Fail safe/secure, fail operational	R2/ C2	SA, SE		x	x
Fault tolerance	C2	All		x	x
Firewalls, filters	C2	SA, SE		x	x
Formal specifications, animated specifications	C2	SA, SE	x	x	x
Information hiding	C2	SA, SE		x	x

Exhibit 3 Information Assurance Design Techniques and Features (continued)

IA Design Techniques and Features	*C/R*	*Type*	*Life-cycle Phase in which Technique is Used*		
			Concept	*Development*	*Operations*
Intrusion detection, response	C2	SA, SE		x	x
Partitioning	C2	SA, SE	x	x	x
Hardware					
Software					
Logical					
Physical					
Plausibility checks	C2	All		x	x
Redundancy	C2	RE	x	x	x
Reliability allocation	C2	RE	x	x	
Secure protocols	C2	All		x	x
IPSec, NLS					
PEM, PGP, S/MIME					
SET					
SSL3, TLS1					
Virus scanners	C2	All			x

Source: Adapted from Herrmann, D., *Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.

(1) determining who has permission to define/change access control rights and privileges and (2) protecting the table that defines the access control rights and privileges from unauthorized manipulation and corruption (see Chapter 6, Exhibit 14).

Benefits: Access control provides a first layer of defense in protecting critical system resources.

Limitations: Effective implementation of access control depends on (1) taking the time to define a comprehensive set of access control rights and privileges, including permissions to create/change these definitions; (2) protecting the table containing these definitions from unauthorized manipulation and corruption; and (3) a robust authentication capability.[2]

References:

1. Blakley, B., *CORBA Security: An Introduction to Safe Computing with Objects*, Addison-Wesley, 1999.
2. Denning, D., *Information Warfare and Security*, Addison-Wesley, 1999.
3. Gollmann, D., *Computer Security*, John Wiley & Sons, 1999.
4. Gong, L., *Inside Java™ 2 Platform Security: Architecture, API Design and Implementation*, Addison-Wesley, 1999.
5. ISO/IEC 10164-9(1995-12), Information Technology, Open Systems Interconnection — Systems Management: Objects and Attributes for Access Control.
6. ISO/IEC 10181-3(1996-09), Information Technology, Open Systems Interconnection — Security Framework for Open Systems: Access Control Framework.
7. Rozenblit, M., *Security for Telecommunications Network Management*, IEEE, 1999.

B.2.2 Account for All Possible Logic States

Purpose: To prevent a system from entering unknown or undefined states, and thus potentially unstable states, which could compromise IA integrity.

Description: Mission-critical systems act upon and respond to a variety of inputs and commands that come from operators, sensors, actuators, and processes. Any given parameter can be in a finite number of states. The same holds true for a combination of parameters. These states can be specified in a truth table. For example, if two parameters are monitored together and they both can only be "on" or "off," there are four possible logic states that could be encountered. Hence, the software monitoring these two parameters should be designed to respond to each of the four logic states, no matter how unlikely they are to occur. This is easily accomplished through the use of a CASE statement. An extra layer of safety and security is provided by including an OTHERWISE clause to trap exceptions (see Chapter 6, Exhibit 15).

Benefits: This technique helps to ensure accurate and predictable system responses to all possible conditions and states, thereby lowering the likelihood of anomalous behavior. This technique is also useful for uncovering missing or incomplete requirements specifications and trapping transient faults.

Limitations: Some additional system resources are used by including the logic to handle all possible logic states. However, the cost is trivial when compared to the consequences of the potential hazards thus prevented.

References:

1. Herrmann, D., *Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.

B.2.3 Audit Trail, Security Alarm

Purpose: To capture evidence of the system resources accessed by a user or process to aid in tracing from original transactions forward or backward to their component transactions.

Description: An audit trail is a design feature that provides an ongoing system monitoring and logging function. An audit trail serves four purposes. First, it captures information about which people and processes accessed what system resources and when they did so. Second, it captures information about system states and transitions, the availability and loading of system resources, and the general "health" of the system. When abnormal events are logged, they trigger warnings and alarms so that action can be taken to prevent or minimize the effects of hazardous events. For example, an alarm may trigger the shutdown of an unstable nuclear power plant or the blocking of an intrusion attempt. The alarms may trigger a combination of automatic processes and operator alerts. Third, audit trail data is used to develop normal system and user profiles as well as attack profiles for intrusion detection systems. Fourth, audit trails are also used to reconstruct events during accident/incident investigation (see Chapter 6, Exhibit 16).

Benefits: An audit trail provides real-time and historical logs of system states, transitions, and resource usage. It is essential for safe, secure, and reliable

system operation and for performing trend analysis and pattern recognition of anomalous events.

Limitations: The completeness of the events/states recorded and the timeliness in responding to anomalous events determine the effectiveness of the audit trail. An audit trail consumes system resources; thus, care should be exercised when determining what events to record and how frequently they should be recorded. A determination also has to be made about the interval at which audit trails should be archived and overwritten.

References:

1. Gollmann, D., *Computer Security*, John Wiley & Sons, 1999.
2. Rozenblit, M., *Security for Telecommunications Management*, IEEE, 1999.

B.2.4 Authentication

Purpose: To establish or prove the validity of a claimed identity of a user, process, or system.

Description: Authentication is a design feature that permits the claimed identity of a user, process, or system to be proven to and confirmed by a second party. Authentication is invoked prior to access control rights and privileges. A combination of parameters can be used to establish an identity, such as user name, password, biometric information, location, and traffic source. There are weaknesses associated with each of these parameters; thus, it is best to use a combination of parameters and not rely on any one alone. To protect the user and the system, authentication should be bidirectional; that is, the user should be authenticated to a system and a system should be authenticated to a user. The latter is an important step in preventing site switching and other security compromises while connected to the Internet.

Benefits: A strong authentication strategy is essential for implementing effective access control rights and privileges.

Limitations: The effectiveness of an authentication strategy is determined by (1) the selection of parameters to be verified, and (2) how stringent the verification process is. The goal is to minimize the number of false positives and false negatives.

References:

1. Blakley, B., *CORBA Security: An Introduction to Safe Computing with Objects*, Addison-Wesley, 1999.
2. Gollmann, D., *Computer Security*, John Wiley & Sons, 1999.
3. ISO/IEC 9594-8(1995-09), Information Technology — Open Systems Interconnection — The Directory: Authentication Framework, 2nd ed.
4. ISO/IEC 9798-1(1991-09), Information Technology — Security Techniques — Entity Authentication Mechanism, Part 1: General Model.
5. ISO/IEC 10181-2(1996-05), Information Technology — Open Systems Interconnection — Security Framework for Open Systems: Authentication Framework.
6. Oppliger, R., *Authentication Systems for Secure Networks*, Artech House, 1996.
7. Rozenblit, M., *Security for Telecommunications Management*, IEEE, 1999.
8. Tung, B., *Kerberos: A Network Authentication System*, Addison-Wesley, 1999.

B.2.5 Block Recovery

Purpose: To enhance IA integrity by recovering from an error and transitioning the system to a known safe and secure state.

Description: Block recovery is a design technique that provides correct functional operation in the presence of one or more errors.[2] For each critical module, a primary and secondary module (employing diversity) are developed. After the primary module executes, but before it performs any critical transactions, an acceptance test is run. This test checks for possible error conditions, such as runtime errors, excessive execution time, or mathematical errors, and performs plausibility checks.[4] If no error is detected, normal execution continues. If an error is detected, control is switched to the corresponding secondary module and another acceptance test is run. If no error is detected, normal execution resumes. However, if an error is detected, the system is reset either to a previous (backward block recovery) or future (forward block recovery) known safe and secure state.

In backward block recovery, if an error is detected, the system is reset to an earlier known safe state. This method implies that internal states are saved frequently at well-defined checkpoints. Global internal states can be saved or only those for critical functions. In forward block recovery, if an error is detected, the current state of the system is manipulated or forced into a future known safe state. This method is useful for real-time systems with small amounts of data and fast-changing internal states[2] (see Chapter 6, Exhibit 17).

Benefits: A system is quickly transitioned to a known safe state and the consequences of a failure are contained.[3]

Limitations: Potential vulnerability to common cause failures must be clearly understood in order to transition the system forward or backward far enough[4] (see also common cause failure analysis).

References:

1. Herrmann, D., *Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.
2. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
3. Leveson, N., *Safeware: System Safety and Computers*, Addison-Wesley, 1995.
4. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.

B.2.6 Confinement

Purpose: To restrict an untrusted program from accessing system resources and executing system processes.

Description: Confinement refers to a set of design features that purposely limit what an untrusted program can access and do. The intent is to prevent an untrusted program from exhibiting unknown and unauthorized behavior, such as:

- Accidentally or intentionally corrupting data
- Accidentally or intentionally triggering the execution of critical sequences

- Initiating a trapdoor or Trojan horse through which executables are misused or corrupted
- Opening a covert channel through which sensitive data is misappropriated

This is accomplished by giving the untrusted program access to the minimum set of system resources it needs to perform its function and no more.[4] Default settings and optional features are disabled. Confinement is particularly useful when COTS products are employed, given the prevalence of undocumented features.

Least privilege, domain and type enforcement (DTE), and wrappers are examples of confinement. In least privilege, child processes do not inherit the privileges of the parent processes. DTE is a confinement technique in which an attribute (called a domain) is associated with each subject (user or process) and another attribute (called a type) is associated with each object (system resource). A matrix is defined that specifies whether or not a particular mode of access to objects of type x is granted to subjects in domain y.[2] Wrappers encapsulate untrusted software to control invocation and add access control and monitoring functions.[3]

Benefits: Potential hazards resulting from the use of untrusted programs are minimized.

Limitations: Thorough analysis is needed to determine how to restrict the untrusted program and what to restrict it to. The effectiveness of confinement is dependent on this up-front analysis.

References:

1. Badger, L., Sterne, D., Sherman, D., and Walker, M., A domain and type enforcement UNIX prototype, *Usenix Computing Systems*, Vol. 9, 1996.
2. Fraser, T. and Badger, L., Ensuring continuity during dynamic security policy reconfiguration in DTE, *IEEE Symposium on Security and Privacy*, 1998, 15–26.
3. Fraser, T., Badger, L., and Feldman, M., Hardening COTS software with generic software wrappers, *IEEE Symposium on Security and Privacy*, 1999.
4. Lindquist, U. and Jonsson, E., A Map of Security Risks Associated with Using COTS, *Computer (*IEEE Computer Society), 31(6), 60–66, 1998.

B.2.7 Defense in Depth

Purpose: To provide several overlapping subsequent limiting barriers with respect to one safety or security threshold, so that the threshold can only be surpassed if all barriers have failed.[3]

Description: Defense in depth is a design technique that reflects common sense. In short, everything feasible is done to prepare for known potential hazards. Then, acknowledging that it is impossible to anticipate all hazards, especially unusual combinations or sequences of events, extra layers of safety and security are implemented through multiple complementary design techniques and features such as those cited in Exhibit 3. For example, partitioning, information hiding, plausibility checks, and block recovery could be implemented in a system; four layers of protection are better than one (see Chapter 6, Exhibit 18).

Benefits: Defense in depth is one of the few techniques that targets potential unknown and unforeseen hazards.

Limitations: There are some additional resources used when implementing defense in depth. However, the cost is trivial when compared to the consequences of the potential hazards thus prevented.

References:

1. CE-1001-STD Rev. 1, Standard for Software Engineering of Safety Critical Software, CANDU Computer Systems Engineering Centre for Excellence, January 1995.
2. Herrmann, D., *Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.
3. IEC 60880(1986-09), Software for Computers in Safety Systems of Nuclear Power Stations.

B.2.8 Defensive Programming

Purpose: To prevent system failures or compromises by detecting errors in control flow, data flow, and data during execution and reacting in a predetermined and acceptable manner.[1]

Description: Defensive programming is a set of design techniques in which critical system parameters and requests to transition system states are verified before acting upon them. The intent is to develop software that correctly accommodates design or operational shortcomings. This involves incorporating a degree of fault/failure tolerance using software diversity and stringent checking of I/O, data, and commands.[2] Defensive programming techniques include[1]:

- Plausibility and range checks on inputs and intermediate variables that affect physical parameters of the system
- Regular automatic checking of the system and software configuration to verify that it is correct and complete
- Plausibility and range checks on output variables
- Monitoring system state changes
- Checking the type, dimension, and range of parameters at procedure entry

Benefits: Defensive programming results in a more robust system architecture and protection from software design errors and failures in the operational environment.

Limitations: Defensive programming increases the complexity of software supportability.

References:

1. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
2. NASA GB-1740.13.96, *Guidebook for Safety-Critical Software — Analysis and Development*, NASA Glenn Research Center, Office of System Safety and Mission Assurance, 1996.
3. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.

B.2.9 Degraded-Mode Operations, Graceful Degradation

Purpose: To ensure that critical system functionality is maintained in the presence of one or more failures.[1]

Description: High-integrity, mission-critical systems can rarely cease operation when an error situation is encountered; they must maintain some minimum level of functionality, usually referred to as degraded-mode operations or graceful degradation of service. During the design and development of high-integrity, mission-critical systems, this minimum required set of functionality should be identified, along with the conditions under which the system should transition to this mode. Degraded-mode operations should include provisions for the following items at a minimum[2]:

- Notifying operational staff and end users that the system has transitioned to degraded-mode operations
- Error handling
- Logging and generation of warning messages
- Reduction of processing load (execute only core functionality)
- Masking of nonessential interrupts
- Signals to external world to slow down inputs
- Trace of system state to facilitate post-event analysis
- Specification of the conditions required to return to normal operations

Systems should be designed to ensure that the specified functionality set will be operational in the presence of one or more failures. A maximum time interval during which a system is allowed to remain in degraded-mode operations should be defined.

Benefits: Degraded-mode operations provides an intermediate state between full operation and system shutdown. This allows the minimum priority system functionality to be maintained until corrective action can be taken.

Limitations: Degraded-mode operations only provides a temporary response to system or component failures.

References:

1. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
2. NASA GB-1740.13.96, *Guidebook for Safety-Critical Software — Analysis and Development*, NASA Glenn Research Center, Office of System Safety and Mission Assurance, 1996.
3. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.

B.2.10 Digital Signatures

Purpose: To provide reasonable evidence of the true sender of an electronic message or document.

Description: A digital signature is a unique block of data that is generated according to a specific algorithm and then attached to an electronic document or message. The block of data is associated with a particular individual.

Therefore, the recipient or an independent third party can verify the sender. Digital signatures establish the source of a message or document, and provide a reasonable degree of nonrepudiation. Digital signatures are created using public key encryption, such as a RSA hashing function. A signature generation algorithm and a signature verification algorithm are involved. The initial Digital Signature Standard (DSS) was established in FIPS PUB 186 in May 1994.[2]

Benefits: Digital signatures, while not 100 percent foolproof, provide a reasonable degree of confidence about the true sender of an electronic message or document.

Limitations: Digital signatures help to establish the identity of a sender of a document or message. However, they do not necessarily prove that the sender created the contents of the document or message.[1] For example, it is very easy to edit forwarded e-mails. Digital signatures consume additional system resources and require that a reliable key management process be followed.

References:

1. Denning, D., *Information Warfare and Security*, Addison-Wesley, 1999.
2. FIPS PUB 186, Digital Signature Standard (DSS), National Institute of Standards and Technology (NIST), U.S. Department of Commerce, May 1994.
3. Ford, W. and Baum, M., *Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Encryption*, Prentice-Hall, 1997.
4. ISO/IEC 9796(1991-09), Information Technology — Security Techniques — Digital Signature Scheme Giving Message Recovery.
5. Rozenblit, M., *Security for Telecommunications Network Management*, IEEE, 1999.

B.2.11 Diversity

Purpose: To enhance IA integrity by detecting and preventing systematic failures.

Description: Diversity is a design technique in which multiple different means are used to perform a required function or solve the same problem. Diversity can be implemented in hardware or software. For software, this means developing more than one algorithm to implement a solution. The results from each algorithm are compared and, if they agree, the appropriate action is taken. Depending on the criticality of the system, 100 percent agreement or majority agreement may be implemented; if the results do not agree, error detection and recovery algorithms take control.[4] Diversity can be implemented at several stages during the life cycle[1]:

- Development of diverse designs by independent teams
- Development of diverse source code in two or more different languages
- Generation of diverse object code by two or more different compilers
- Implementation of diverse object code by using two or more different linking and loading utilities

Benefits: Diversity limits the potential for common cause and systematic failures.

Limitations: Diversity may complicate supportability issues and synchronization between diverse components operating in parallel.[1]

References:

1. DEF STAN 00-55, Requirements for Safety-Related Software in Defence Equipment, Part 1: Requirements, U.K. Ministry of Defence (MoD), August 1, 1997.
2. DEF STAN 00-55, Requirements for Safety-Related Software in Defence Equipment, Part 2: Guidance, U.K. Ministry of Defence (MoD), August 1, 1997.
3. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
4. Herrmann, D., *Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.
5. Leveson, N., *Safeware: System Safety and Computers*, Addison-Wesley, 1995.
6. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.

B.2.12 Encryption

Purpose: To provide confidentiality for information while it is stored and transmitted.

Description: Encryption provides one layer of protection to sensitive data by making the data unintelligible to all but the intended recipients. Encryption consists of a mathematically based algorithm, which specifies the steps involved in transforming the data, and a key, which represents a specific instance of the algorithm. The keys may be public/private (asymmetric) or secret (symmetric), and are changed frequently; in contrast, the algorithm remains constant. Encryption can be implemented in hardware or software and through the use of block or stream ciphers. Encryption predates computers and can be implemented manually. A variety of different encryption algorithms with varying key types and lengths are available today. The goal is to select the encryption algorithm and mode appropriate for the specific application, operational environment, and level of confidentiality/protection needed (see Chapter 6, Exhibits 19 and 20).

Benefits: Given that the Internet is basically a big party-line, encryption provides one means of protecting the confidentiality of information that traverses it. One challenge is to determine the correct layer(s) in the ISO OSI and TCP/IP reference models in which to implement encryption.

Limitations: Encryption consumes additional system resources. Effective implementation requires staff training and following a reliable key management process. Note, however, that encryption provides temporary confidentiality because all encryption algorithms and keys can be broken — it is just a matter of time. With today's rapid increases in processing power, the times are getting shorter and shorter. Also, encryption does not ensure data integrity.

References:

1. Denning, D., *Cryptography and Data Security*, Addison-Wesley, 1982.
2. ISO/IEC 9797(1994-04), Information Technology — Security Techniques — Data Integrity Measures Using a Cryptographic Check Function Employing a Block Cipher Algorithm.
3. ISO/IEC 9798-2(1994-12), Information Technology — Security Techniques — Entity Authentication Mechanisms — Part 2: Mechanisms Using Symmetric Encipherment Algorithms.

4. ISO/IEC 9798-3(1993-11), Information Technology — Security Techniques — Entity Authentication Mechanisms — Part 3: Entity Authentication Using a Public Key Algorithm.
5. ISO/IEC 9798-4(1995-03), Information Technology — Security Techniques — Entity Authentication Mechanisms — Part 4: Mechanisms Using a Cryptographic Check Function.
6. ISO/IEC 10118-1(1994-10), Information Technology — Security Techniques — Hash Functions — Part 1: General.
7. ISO/IEC 10118-2(1994-10), Information Technology — Security Techniques — Hash Functions — Part 2: Hashing Functions Using an n-bit Block Cipher Algorithm.
8. ISO/IEC 11770-1(1997-01), Information Technology — Security Techniques — Key Management — Part 1: Framework.
9. ISO/IEC 11770-2(1996-04), Information Technology — Security Techniques — Key Management — Part 2: Mechanisms Using Asymmetric Techniques.
10. Menezes, A., Van Oorschot, P., and Vanstone, S., *Handbook of Applied Cryptography*, CRC Press, 1996.
11. Kippenhahn, R., *Code Breaking: A History and Exploration*, Overlook, 1999.
12. Schneier, B., *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd ed., John Wiley & Sons, 1995.
13. Stallings, W., *Cryptography and Network Security*, 2nd ed., Prentice-Hall, 1998.

B.2.13 Error Detection/Correction

Purpose: To increase data integrity.

Description: Error detection/correction algorithms are used to increase data integrity during the transmission of data within and among networks and system integrity during execution of application software. At the network level, error detection/correction algorithms examine data to determine if any data was **accidentally** corrupted or lost, and to discover if any unauthorized changes were **intentionally** made to the data.[3] These errors are compensated for by self-correcting codes at the receiving end or requests for retransmission. At the application software level, error detection/correction algorithms detect anomalous or illegal modes/states, parameters, etc. and initiate the appropriate error handling routines. It is unlikely that corrective action will be implemented for all potential error conditions due to program size, response time, and schedule and budget constraints; hence, the focus should be on IA-critical and IA-related functions/entities.

Benefits: The severity of the consequences of an error, fault, or failure is minimized by early detection and recovery. Automated error detection and correction is faster and generally more reliable than that which involves humans.

Limitations: The effectiveness of this technique is directly proportional to the thoroughness by which potential error conditions have been identified and compensated for by the design.

References:

1. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
2. Knight, J., Elder, M., and Du, X., Error recovery in critical infrastructure systems, *Computer Security, Dependability, and Assurance: From Needs to Solutions*, IEEE, 1999, 49–71.

3. Morris, D., *Introduction to Communications Command and Control Systems*, Pergamon Press, 1977.
4. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.

B.2.14 Fail Safe/Secure, Fail Operational

Purpose: To ensure that a system remains in a known safe and secure state following an irrecoverable failure.

Description: Fail safe/secure and fail operational are IA design techniques that ensure that a system remains in a known safe and secure state following an irrecoverable failure. To fail safe or secure means that a component automatically places itself in a safe and secure mode/state in the event of a failure. In many instances, known safe and secure default values are assumed. Then, the system is brought to a safe and secure mode/state by shutting it down. To fail operational means that a system or component continues to provide limited critical functionality in the event of a failure; in some instances, a system cannot simply shut down.

Fail safe/secure and fail operational ensure that a system responds predictably to failures by making proactive design decisions. The first step is to identify all possible failure modes. This is done by developing transaction paths and using IA analysis techniques such as FTA, FMECA, and HAZOP studies. Next, the appropriate response to each failure is specified so that the system will remain in a known safe and secure state.

Benefits: Planning for and implementing provisions for fail safe or fail operational modes reduces the likelihood that unplanned events will occur.

Limitations: A comprehensive set of potential failure modes must be identified, particularly those that effect IA-critical and IA-related functions and entities.

References:

1. Bishop, P. and Bloomfield, R., *The SHIP Safety Case Approach*, Adelard, 1995.
2. Herrmann, D., *Software Safety and Reliability: Techniques, Approaches, and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.
3. Leveson, N., *Safeware: System Safety and Computers*, Addison-Wesley, 1995.
4. McDermid, J., Issues in the development of safety-critical systems, *Safety Critical Systems*, Chapman & Hall, 1993, 16–42.
5. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.

B.2.15 Fault Tolerance

Purpose: To provide continued correct execution in the presence of a limited number of hardware or software faults.[1–5]

Description: Fault tolerance is a category of IA design techniques that focuses on containing and mitigating the consequences of faults, rather than preventing them. There are three types of fault tolerance: system fault tolerance, hardware fault tolerance, and software fault tolerance. Hardware fault tolerance is usually implemented through redundancy, diversity, power-on tests, BIT, and other monitoring functions. The concept is that if a primary component fails, a secondary component will take over and continue normal

operations. Software fault tolerance is usually implemented through block recovery, diversity, error detection/correction, and other IA design techniques. The basic premise of software fault tolerance is that it is nearly impossible to develop software that is 100 percent free of defects; therefore, IA design techniques should be employed to detect and recover from errors while minimizing their consequences. System fault tolerance combines hardware and software fault tolerance, with software monitoring the health of both the hardware and the software. System fault tolerance should be employed for IA-critical and IA-related functions.

Benefits: Fault tolerant design is an effective method to increase system reliability and availability.

Limitations: Fault tolerance potentially increases the size, weight, and power consumption of a system that may conflict with specified constraints.

References:

1. IEC 60880(1986-09), Software for Computers in Safety Systems of Nuclear Power Stations.
2. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
3. Levi, Shem-Tov and Agrawala, A., *Fault Tolerant System Design*, McGraw-Hill, 1994.
4. Lyu, M. (Ed.), *Software Fault Tolerance*, John Wiley & Sons, 1995.
5. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.

B.2.16 Firewalls, Filters

Purpose: To block unwanted users, processes, and data from entering a network while protecting legitimate users, sensitive data, and processes.

Description: A firewall functions as a security gateway between two networks. A firewall can be implemented in software or a combination of hardware and software. It uses a variety of techniques, such as packet filtering, application level gateways, and circuit level gateways to prevent unauthorized users, processes, and data from entering the network. At the same time, a firewall protects legitimate users, processes, and data and allows them to interact with resources outside the firewall. In effect, a firewall implements access control between networks. The main functions of a firewall are[3,4]:

- Performing access control based on sender/receiver addresses
- Performing access control based on the service requested
- Hiding the internal network topology, addresses, and traffic from the outside world
- Checking incoming files for viruses
- Performing authentication based on traffic source
- Logging internet activities
- Blocking incoming junk e-mail and outgoing connections to objectional Web sites

Firewalls should be tested against potential threats, known vulnerabilities, content-based attacks, and specified protection profiles.[4]

Benefits: Firewalls are useful for preventing accidental or malicious intentional traffic from entering a network. The usefulness of firewalls in preventing intentional malicious traffic from entering a network appears to be more in the area of delaying its entry than preventing it altogether.[4] Consequently, firewalls must be used in conjunction with other defensive design features.

Limitations: Firewalls provide one layer of protection for IA-critical and IA-related systems and data. However, they are not 100 percent foolproof.

References:

1. Chapman, D. and Zwicky, E., *Building Internet Firewalls*, 1st ed., O'Reilly & Associates, 1995.
2. Cheswick, W. and Bellovin, S., *Firewalls and Internet Security*, Addison-Wesley, 1994.
3. Denning, D., *Information Warfare and Security*, Addison-Wesley, 1999.
4. Gollmann, D., *Computer Security*, John Wiley & Sons, 1999.

B.2.17 Formal Specifications, Animated Specifications

Purpose: To ensure correctness, consistency, completeness, and unambiguousness of the requirements and design for IA-critical and IA-related functions.

Description: Formal methods describe a system and its intended properties and performance using a fixed notation based on discrete mathematics, which can be subjected to mathematical analysis to detect incompleteness, inconsistencies, incorrectness, and ambiguousness.[6] The description can be analyzed by computer, similar to the syntax checking of a source program by a compiler, to display various aspects of system behavior.[6,7] Most formal methods provide a capability for stating assertions for pre- and post-conditions at various locations in a program.[3,4,8] Some of the more common formal methods used today include: B, calculus of communicating systems (CCS), communicating sequential processes (CSP), higher order logic (HOL), language for temporal ordering specification (LOTOS), OBJ, temporal logic, Vienna development method (VDM), and Z (see Chapter 6, Exhibit 21).

Benefits: The rigor imposed by formal methods exposes many gaps and inconsistencies in specifications and designs that would not be as visible when using other techniques. The animated models that can be developed from the specification and design help to clarify requirements and facilitate communication among different stakeholders.[7]

Limitations: The design team must be thoroughly trained in the formal method to implement it correctly. The use of an automated tool is required. Development life-cycle costs are equivalent to traditional development methods; however, more resources are used earlier in the life cycle. Given that more errors are found earlier in the life cycle with formal methods, when it is easier and cheaper to fix them, overall life-cycle costs are less.

References:

1. Bowen, J. and Hinchey, M., *High Integrity System Specification and Design*, IEEE Computer Society Press, 1999.
2. DEF STAN 00-55, Requirements for Safety-Related Software in Defence Equipment, Part 1: Requirements, U.K. Ministry of Defence (MoD), August 1, 1997.

3. Diller, A, *Z: An Introduction to Formal Methods*, 2nd ed., John Wiley & Sons, 1994.
4. Harry, A., *Formal Methods Fact File: VDM and Z*, John Wiley & Sons, 1996.
5. Heitmeyer, C. and Madrioli, D., *Formal Methods for Real-Time Computing*, John Wiley & Sons, 1996.
6. Herrmann, D., *Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.
7. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
8. Ince, D., *An Introduction to Discrete Mathematics, Formal System Specification, and Z*, Oxford University Press, 1992.
9. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.

B.2.18 Information Hiding

Purpose: To (1) prevent **accidental** access to and corruption of software and data, (2) minimize introduction of errors during maintenance and enhancements, (3) reduce the likelihood of CCFs, and (4) minimize fault propagation.

Description: Information hiding is an IA design technique developed by Dr. David Parnas that minimizes the interdependency or coupling of modules and maximizes the independence or cohesion of modules.[3] System functions, sets of data, and operations on that data are localized within a module. The interface to each software module is designed to reveal as little as possible about the module's inner workings.[4] This is accomplished by making the logic of each module and the data it utilizes as self-contained as possible.[3] In this way, if it is necessary to change the functions internal to one module, the resulting propagation of changes to other modules is minimized.

Benefits: The likelihood of common cause failures is reduced, fault propagation is minimized, and future maintenance and enhancements are facilitated.[3] Object-oriented designs are well suited for information hiding.[1,2]

Limitations: Information hiding requires more time up-front to analyze the design of modules and precise module and interface specifications.

References:

1. DEF STAN 00-55, Requirements for Safety-Related Software in Defence Equipment, Part 1: Requirements, U.K. Ministry of Defence (MoD), August 1, 1997.
2. DEF STAN 00-55, Requirements for Safety-Related Software in Defence Equipment, Part 2: Guidance, U.K. Ministry of Defence (MoD), August 1, 1997.
3. Herrmann, D., *Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.
4. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
5. Parnas, D., On the criteria to be used in decomposing systems into modules, *Communications of the ACM*, December, 1053–1058, 1972.

B.2.19 Intrusion Detection and Response

Purpose: To recognize and respond to a security breach either as it is happening or immediately afterward.

Description: Intrusion detection is a design feature that takes over where firewalls leave off to provide another layer of protection for IA-critical and IA-related functions and data. Intrusion detection and response software looks for both insider and outsider attacks. Three types of algorithms are used to implement intrusion detection[3]:

1. **Statistical anomaly detection** analyzes audit trail data for abnormal system or user behavior.
2. **Rules-based detection** analyzes audit trail data for patterns that match known attack profiles.
3. **Hybrid detection** employs a combination of statistical and rules-based detection algorithms.

Intrusion detection algorithms search for indications of unusual activity that point to past, present, or impending misuse of system resources. Audit trails, keystroke trapping, traffic source, login history, and packet sniffers are employed to assist intrusion detection. Another approach to intrusion detection is to set up decoy servers and LANs that legitimate users would never access (*see also* Audit Trail).

Benefits: Intrusion detection and response systems, while not 100 percent foolproof, provide an extra layer of protection beyond firewalls, access control, and authentication. Intrusion detection and response systems serve as an early warning system, alerting operators and systems so that action can be taken quickly to prevent an attack or minimize its damage.

Limitations: Intrusion detection systems consume additional system resources. Care should be exercised in selecting the events to be monitored. The accuracy of the "normal" profiles determines the percentage of false positives and false negatives generated from statistical anomaly detection. Only known attack profiles are intercepted with rules-based detection.

References:

1. Escamilla, T., *Intrusion Detection: Security Beyond the Firewall*, John Wiley & Sons, 1998.
2. Gollmann, D., *Computer Security*, John Wiley & Sons, 1999.
3. Herrinshaw, C., Detecting attacks on networks, *Computer* (IEEE Computer Society), 30(12), 16–17, 1997.
4. Lehtinen, M. and Lear, A., Intrusion detection: managing the risk of connectivity, *IT Professional*, 1(6), 11–13, 1999.

B.2.20 Partitioning

Purpose: To enhance IA integrity by preventing non-IA-related functions/entities from **accidentally** or **intentionally** corrupting IA-critical functions/entities.

Description: Partitioning is an IA design technique that can be implemented in hardware or software. In the case of software, partitioning can be logical or physical. IA-critical and IA-related functions/entities are isolated from non-IA-related functions/entities. Both design and functionality are partitioned to prevent accidental and intentional interference, compromise, and corruption originating from non-IA-related functions/entities. Partitioning is often referred to as separability in the security community. Several national

and international standards either mandate or highly recommend the use of partitioning.[1–7]

Benefits: Well-partitioned systems are easier to understand, verify, and maintain. Partitioning facilitates fault isolation and minimizes the potential for fault propagation. Partitioning helps to identify the most critical components so that resources can be more effectively concentrated on them.

Limitations: Partitioning requires complete interface specifications.

References:

1. CE-1001-STD Rev. 1, Standard for Software Engineering of Safety Critical Software, CANDU Computer Systems Engineering Centre for Excellence, January 1995.
2. DEF STAN 00-55, Requirements for Safety Related Software in Defence Equipment, Part 1: Requirements, U.K. Ministry of Defence (MoD), August 4, 1997.
3. EN 50128:1997, Railway Applications: Software for Railway Control and Protection Systems, The European Committee for Electrotechnical Standardization (CENELEC).
4. Development Guidelines for Vehicle Based Software, The Motor Industry Reliability Association (MISRA™), November 1994.
5. Herrmann, D., *Software Safety and Reliability: Techniques, Approaches, and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.
6. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
7. NASA GB-1740.13.96, *Guidebook for Safety-Critical Software — Analyses and Development*, NASA Glenn Research Center, Office of Safety and Mission Assurance, 1996.

B.2.21 Plausibility Checks

Purpose: To enhance IA integrity by verifying the validity and legitimacy of critical parameters before acting upon them; detect errors early in the execution cycle to prevent them from progressing into system failures or compromises.

Description: Plausibility checks are an IA design technique. The basic approach is simple: checks are performed on parameters, before critical operations are performed, to verify that the value of the parameters are plausible and legal. Plausibility checks can be used to enhance safety, security, and reliability. Examples of checks that can be performed to enhance safety and reliability include[1,4]:

- Parameter size (number of bits, bytes, digits, etc.)
- Array bounds
- Counter values
- Parameter type verification
- Legitimate called from routine
- Timer values
- Assertions about parameter value, operational mode/state, and pre- and post-conditions
- Range checks of intermediate results

The specific parameters checked will vary by application. However, all parameters that affect IA-critical and IA-related functions/entities should be checked.

Benefits: Plausibility checks enhance the operational integrity of the system.

Limitations: None.
References:

1. DEF STAN 00-55, Requirements for Safety-Related Software in Defence Equipment, Part 1: Requirements, U.K. Ministry of Defence (MoD), August 1, 1997.
2. DEF STAN 00-55, Requirements for Safety-Related Software in Defence Equipment, Part 2: Guidance, U.K. Ministry of Defence (MoD), August 1, 1997.
3. Herrmann, D., *Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.
4. IEC 60880(1986-09), Software for Computers in Safety Systems of Nuclear Power Stations.
5. SEMSPLC Guidelines, Safety-Related Application Software for Programmable Logic Controllers, IEE Technical Guidelines 8:1996.

B.2.22 Redundancy

Purpose: To enhance hardware reliability and system availability.
Description: Redundancy is a fault tolerance design technique in which redundant hardware components are employed to increase hardware reliability and system availability. Secondary or redundant components function as hot or cold standbys, ready to assume primary functionality should the primary component fail or exhibit anomalous behavior. Redundancy is employed at the level and to the extent that is meaningful and practical for a given system and the criticality of its operation. This may include redundant memory, disk drives, servers, printers, processors, etc. Many real-time process control operations, especially those involving PLCs, employ triple modular redundancy (TMR).
Benefits: Redundancy helps to eliminate single points of failure.
Limitations: (1) Redundancy does not compensate for design flaws inherent in a component; all redundant components will contain the same error. (2) Redundancy is not applicable to software; the same design errors are simply replicated. Instead, diversity is employed.[2,3,6] (3) Care should be taken to ensure that redundant components are not subject to common cause failure modes.
References:

1. DEF STAN 00-41/Issue 3, Reliability and Maintainability, *MoD Guide to Practices and Procedures*, U.K. Ministry of Defence (MoD), June 25, 1993.
2. Herrmann, D., *Software Safety and Reliability: Techniques, Approaches, and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.
3. Leveson, N., *Safeware: System Safety and Computers*, Addison-Wesley, 1995.
4. O'Connor, P., *Practical Reliability Engineering*, 3rd ed., John Wiley & Sons, 1991.
5. SEMSPLC Guidelines, Safety-Related Application Software for Programmable Logic Controllers, IEE Technical Guidelines 8:1996.
6. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.

B.2.23 Reliability Allocation

Purpose: To distribute reliability and maintainability requirements, derived from IA goals, among system entities.

Description: Reliability requirements are generally specified at the system level early in the life cycle. During architectural analysis, system reliability requirements are allocated to individual system components, including hardware, software, and communications equipment. It is usually necessary to perform trade-off studies to determine the optimum architecture that will meet reliability requirements. This may involve reassigning functionality between hardware and software components. FTA, FMECA, HAZOP studies, and reliability block diagrams provide input to the reliability allocation process. Where appropriate, separate reliability requirements may be specified for different types and consequences of failure[1]:

- The severity of the consequences of the failure
- Whether or not recovery from the failure is possible without operator intervention
- Whether or not a failure causes corruption of software or data
- The time required to recover from a failure

Benefits: If sufficient analysis is conducted to support the reliability allocation, the likelihood that reliability requirements will be met is greater. Also, it is more cost-effective to analyze and allocate reliability requirements early in the life cycle than to wait until after a system is developed to find out that it does not meet reliability requirements.

Limitations: The distinction between random hardware failures and systematic software failures must be maintained when allocating reliability requirements.

References:

1. DEF STAN 00-41/Issue 3, Reliability and Maintainability, *MoD Guide to Practices and Procedures*, U.K. Ministry of Defence (MoD), June 25, 1993.
2. DEF STAN 00-42, Reliability and Maintainability Assurance Guides, Part 2: Software, U.K. Ministry of Defence (MoD), September 1, 1997.
3. O'Connor, P., *Practical Reliability Engineering*, 3rd ed., John Wiley & Sons, 1991.

B.2.24 Secure Protocols

Purpose: To enhance the confidentiality of distributed data communication.

Description: A variety of protocols have recently been developed or are under development to enhance the confidentiality of information exchanged among distributed systems. Some examples include IPSec, NLS, PEM, PGP, S/MIME, SET, SSL3, and TLS1. IPSec and NLS provide network-level security. PEM, PGP, and S/MIME provide e-mail security. SSL3 and TLS1 provide security for distributed client/server applications. SET provides e-Commerce security. Each of these protocols is designed for a specific function and environment.

Benefits: These protocols provide an extra level of confidentiality for Internet transactions. The robustness of the protocols and the level of confidentiality provided vary.

Limitations: None of these protocols is 100 percent secure; they are too new and still evolving. The seamlessness with which these protocols can be implemented within an existing communications architecture varies.

References:

1. Doraswamy, N. and Harkins, D., *IPSec: The New Security Standard for the Internet, Intranet, and Virtual Private Networks*, Prentice-Hall, 1999.
2. Garfinkel, S., *PGP: Pretty Good Privacy*, 1st ed., O'Reilly & Associates, 1994.
3. Kaufman, E. and Newman, A., *Implementing IPSec*, John Wiley & Sons, 1999.
4. Merkow, M., Breithaupt, J., and Wheeler, K., *Building SET Applications for Secure Transactions*, John Wiley & Sons, 1998.
5. Oppliger, R., *Internet and Intranet Security*, Artech House, 1998.
6. Rozenblit, M., *Security for Telecommunications Management*, IEEE, 1999.
7. www.setco.org.

B.2.25 Virus Scanners

Purpose: To automatically detect and remove computer viruses before they are activated.

Description: Virus scan software scans boot sectors, memory, and computer files, looking for the presence of potentially malicious hidden code; for example, executables hidden in document files. Virus scanners look for known viruses and patterns that resemble potential viruses. Most scanners have the option of marking or cleansing files suspected of being infected. On occasion, the cleansing operation may not be successful; the virus is not contained or the original file is not recoverable.

Benefits: If virus scan software is executed frequently and kept up-to-date, a reasonable degree of protection is provided against known viruses.

Limitations: (1) Virus scanners only detect the presence of known viruses or patterns that resemble potential viruses; a new virus strain may go undetected. (2) Virus scan software must be updated constantly. (3) Most users consider themselves too busy to run or update virus scan software; they cannot be relied upon to do so. Hence, the execution and updating of virus scan software must be automatically linked to external events: power-up, time of day, receipt of foreign files, etc.

References:

1. Cohen, F., *A Short Course on Computer Viruses*, 2nd ed., John Wiley & Sons, 1994.
2. Slade, R., *Guide to Computer Viruses*, Springer-Verlag, 1994.
3. http://service.symantec.com.
4. www.mcafee.com.

B.3 IA Verification Techniques

B.3.1 Boundary Value Analysis

Purpose: To identify software errors that occur in IA-critical and IA-related functions/entities when processing at or beyond specified parameter limits.

Description: During boundary value analysis, test cases are designed that exercise the software's parameter processing algorithms. The system's response to specific input and output classes is evaluated, such as:

Exhibit 4 Information Assurance Verification Techniques

IA Verification Techniques	*C/R*	*Type*	*Life-Cycle Phase in which Technique is Used*		
			Concept	*Development*	*Operations*
Boundary value analysis	C3	All		x	x
Cleanroom	C3	All		x	
Control flow analysis[a]	C3	All		x	x
Data or information flow analysis[a]	C3	All		x	x
Equivalence class partitioning	C3	All		x	x
Formal proofs of correctness	C3	SA, SE	x	x	x
Interface testing	C3	All		x	x
Performance testing	C3	All		x	x
Probabilistic or statistical testing	C3	All		x	x
Regression testing	C3	All		x	x
Reliability estimation modeling	C3	RE		x	x
(IA) requirements traceability	C3	All	x	x	x
Review IA integrity case[a]	C3	All	x	x	x
Root cause analysis[a]	C3	All		x	x
Safety/security audits, reviews, and inspections	C3	SA, SE		x	x
Stress testing	C3	All		x	x
Testability analysis, fault injection, failure assertion	C3	All		x	x
Usability testing	C3	All		x	x

[a] These techniques can also be used during accident/incident investigation.

Source: Adapted from Herrmann, D., *Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.

- Parameter below minimum specified threshold
- Parameter at minimum specified threshold
- Parameter at maximum specified threshold
- Parameter over maximum specified threshold
- Parameter within specified minimum/maximum range

Zero or null parameters tend to be error-prone. Specific tests are warranted for the following conditions as well[1]:

- Zero divisor
- Blank ASCII characters
- Empty stack or list
- Full matrix
- Zero entry table

Boundary value analysis should be used to verify processing of authentication parameters, parameters that control IA-critical and IA-related functions,

and potential buffer overflow conditions. Boundary value analysis complements plausibility checks.

The intent is to verify that the software responds to all parameters correctly, so that the system remains in a known safe and secure state. Error handling routines are triggered if a parameter is out of the specified range or normal processing continues if a parameter is within the specified range. Boundary value analysis can also be used to verify that the correct data type is being used: alphabetic, numeric, integer, real, signed, pointer, etc.

Benefits: Boundary value analysis enhances IA integrity by ensuring that data is within the specified valid range before operating upon it.

Limitations: None.

References:

1. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
2. IEC 61704(1995-06), Guide to Test Methods for Dependability Assessment of Software.
3. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.

B.3.2 Cleanroom

Purpose: To prevent defects from being introduced or remaining undetected in IA-critical and IA-related functions/entities through an evaluation of the completeness, consistency, correctness, and unambiguousness of requirements, design, and implementation.[1,2]

Description: Cleanroom is a full life-cycle verification process that supports the measurement and analysis of pre-release software reliability. Cleanroom analysis emphasizes the prevention of errors rather than their detection.[3] This approach takes a holistic view of software development by promoting top-down, stepwise refinement of the total design, with the correctness of that design being verified at each step.[3]

Benefits: The cleanroom process is cost-effective; it promotes the prevention and early detection of errors.

Limitations: Cleanroom analysis does not determine if performance and response time requirements will be met.

References:

1. DEF STAN 00-42/Issue 1, Reliability and Maintainability Guides, Part 2: Software, U.K. Ministry of Defense (MoD), September 1, 1997.
2. Dyer, M., *The Cleanroom Approach to Quality Software Development*, John Wiley & Sons, 1992.
3. Herrmann, D., *Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.
4. Prowell, S., Trammell, C., Linger, R., and Poore, J., *Cleanroom Software Engineering — Technology and Process*, Addison-Wesley, 1999.

B.3.3 Control Flow Analysis

Purpose: To uncover poor and incorrect program logic structures that could compromise IA integrity.

Description: Control flow analysis is a static analysis technique that examines the logical structure of a program. A digraph is used to represent the control flow through a software system. Unconditional jumps, unused and unreachable code, all of which could be used as an opening for an attack, are uncovered. The digraph is also reviewed for opportunities to optimize program structure and thereby enhance its maintainability. The emphasis is on verifying correct control flow to, from, and within IA-critical and IA-related functions/entities. Control flow analysis should be used in conjunction with data flow analysis to substantiate noninterference claims.

Benefits: Control flow analysis uncovers implementation errors before a product is tested or fielded. Inconsistencies between designs and implementations are also highlighted.

Limitations: Control flow analysis does not verify timing, capacity, or throughput requirements.

References:

1. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
2. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.

B.3.4 Data or Information Flow Analysis

Purpose: To uncover incorrect and unauthorized data transformations and operations that could compromise IA integrity.

Description: Data flow analysis is a static analysis technique that examines the access and change sequence of critical data elements. Using the digraph developed for control flow analysis, each distinct operation performed on a data element and each distinct transformation of that element are evaluated. Actual data flow is compared to required data flow to detect erroneous conditions and potential leakage, which could lead to a system compromise or failure. Examples of items to check during data flow analysis include[1,2]:

- Variables that are read before they are assigned a value
- Variables that are written more than once before they are read
- Variables that are written but never read
- Variables that are accidentally or intentionally overwritten
- Variables that are accidentally or incorrectly read (framing, addressing errors, etc.) or modified

The emphasis is on verifying correct data flow to, from, and within IA-critical and IA-related functions/entities. Data flow analysis should be used in conjunction with control flow analysis to substantiate noninterference claims.

Benefits: Data flow analysis uncovers incorrect and unauthorized data transformations and operations before a product is tested or fielded. Inconsistencies between designs and implementations are also highlighted.

Limitations: Data flow analysis does not verify timing, capacity, or throughput requirements.

References:

1. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
2. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.

B.3.5 Equivalence Class Partitioning

Purpose: To identify the minimum set of test cases and test data that will adequately test each input domain.

Description: During equivalence class partitioning, the set of all possible test cases is examined to determine which test cases and data are unique or redundant, in that they test the same functionality or logic path. The intent is to obtain the highest possible test coverage with the least possible number of test cases. Input partitions can be derived from the requirements and the internal structure of a program.[1] In the IA domain, at least one test case should be taken from each equivalence class for IA-critical and IA-related functions/entities.

Benefits: Testing activities are more efficient when equivalence class partitioning is employed.

Limitations: A thorough understanding of the system design and its functionality are needed to perform equivalence class partitioning. Several standard algorithms have been developed to assist this process.

References:

1. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
2. IEC 61704(1995-06), Guide to Test Methods for Dependability Assessment of Software.

B.3.6 Formal Proofs of Correctness

Purpose: To prove that the requirements, design, and implementation of IA-critical and IA-related functions/entities are correct, complete, unambiguous, and consistent.

Description: Formal mathematical proofs are developed from formal specifications to prove that the specifications and corresponding design and implementation are correct, complete, unambiguous, and consistent. The proofs demonstrate that a program transfers pre-conditions into post-conditions according to the set of specified logical rules. System behavior under normal, abnormal, and exception conditions is verified. The completeness of specifications in regard to logic states, data definitions and operations, timing, termination, etc. is demonstrated. Formal proofs are developed for IA-critical and IA-related functions/entities. For example, formal proofs could be developed to demonstrate that the access control rules do not allow any unintended inferred access control privileges or information flow. (As a historical note, the *Orange Book* required formal proofs of correctness for evaluation class A-1 systems.) The structure of the proof will correspond to the formal method chosen.

Benefits: Formal proofs are a comprehensive ongoing verification activity and provide evidence for the IA integrity case.

Limitations: The thoroughness and accuracy of the proof determines its effectiveness. An automated tool must be used in most cases.

References:

1. Bowen, J. and Hinchey, M., *High Integrity System Specification and Design*, IEEE Computer Society Press, 1999.
2. Diller, A., *Z: An Introduction to Formal Methods*, 2nd ed., John Wiley & Sons, 1994.
3. Harry, A., *Formal Methods Fact File: VDM and Z*, John Wiley & Sons, 1996.
4. Heitmeyer, C. and Mandrioli, D., *Formal Methods for Real-Time Computing*, John Wiley & Sons, 1996.
5. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
6. Ince, D., *An Introduction to Discrete Mathematics, Formal System Specification, and Z*, Oxford University Press, 1992.
7. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.

B.3.7 Interface Testing

Purpose: To verify that interface requirements are correct and that interfaces have been implemented correctly.

Description: Interface testing verifies that hardware/software, system software/application software, and application software/application software interfaces work correctly, as specified. Interface testing is used to verify that the interfaces between IA-critical, IA-related, and non-IA-related functions/entities are correct, especially if the design incorporates partitioning or information hiding. Different types of parameters are passed under varying system loads and states. Snapshots of pre- and post-conditions are examined. Examples of items to evaluate during interface testing include[2]:

- Detection and handling of failure modes
- Response to out-of-range values
- Response to not receiving a specified input
- Handling of time-out conditions
- Response to inputs that are received too early, too late, or out of sequence
- Responses to minimum and maximum input arrival rates
- Responses to masked or disabled interrupts
- Responses to outputs that are produced faster than specified
- Responses to inputs that are received during initialization, shutdown, or while a system is offline

Benefits: System integration errors are detected prior to a system being fielded.

Limitations: Interface testing must be conducted in the operational environment or a simulated operational environment to yield valid results.

References:

1. DEF STAN 00-55, Requirements for Safety-Related Software in Defence Equipment, Part 1: Requirements, U.K. Ministry of Defence (MoD), August 1, 1997.

2. DEF STAN 00-55, Requirements for Safety-Related Software in Defence Equipment, Part 2: Guidance, U.K. Ministry of Defence (MoD), August 1, 1997.
3. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
4. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.

B.3.8 Performance Testing

Purpose: To verify whether or not a system will meet stated performance requirements and that these requirements are correct.

Description: Performance testing exercises a system under varied loads and operational modes/states to determine if response time, capacity, and throughput requirements will be met. That is, the successful implementation of nonfunctional requirements and the absence of resource contention (memory, processor speed, capacity, I/O buses, communications bandwidth, storage, etc.) is verified. Items evaluated include[1,2]:

- Interactions between system processes
- Resource usage by each process
- Distribution of demands placed upon the system under average and worst-case conditions
- Mean and worse-case throughput and response times for individual system functions
- Real-time response time and throughput tests

Not meeting response time, capacity, and throughput requirements can have a major impact on IA integrity, such as causing a system failure or compromise. Hence, the emphasis in performance testing should be on IA-critical and IA-related functions/entities. Performance testing complements response time, memory, and constraint analysis and should be supplemented by stress testing.

Benefits: Performance shortfalls that may contribute to safety, security, and reliability failures are identified prior to a system being fielded.

Limitations: Performance testing must be conducted in the operational environment or a simulated operational environment to yield valid results.

References:

1. DEF STAN 00-42/Issue 1, Reliability and Maintainability Guides, Part 2: Software, U.K. Ministry of Defense (MoD), September 1, 1997.
2. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.

B.3.9 Probabilistic or Statistical Testing

Purpose: To provide a quantitative assessment of operational IA integrity; verify design integrity against operational profiles.

Description: During probabilistic or statistical testing, test cases are developed from operational profiles which reflect how different classes of users will use

a system, the type and frequency of transactions performed, the anticipated system loading, etc. That is, test cases are statistically similar to or mimic the in-service environment. Parameters and conditions that activate IA-critical and IA-related functions/entities, in particular fault tolerant, fail safe/secure, and fail operational features, are exercised. Continuous-mode and demand-mode functions are tested.[4] Test cases are designed to catch random and systematic failures.[3] The test interval must be several times longer than the estimated MTBF to yield valid results.[2] Probabilistic testing should be supplemented by simulation and specification animation.

Benefits: Probabilistic testing yields reliability measures that correspond to how a system is expected to be used. Greater weight is given to the correct operation of transactions that are performed frequently and considered essential than those that are performed infrequently and are not essential. This approach contrasts with typical software reliability models, which treat all errors equally. Other values can be derived from these measures as well, including[3]:

- Probability of failure free operation
- Probability of system survival
- System availability
- (Updated) MTBF
- Probability of safe and secure operation

Limitations: Effective probabilistic testing is dependent on an accurate and complete set of operational profiles.

References:

1. BS5760 Part 8, Guide to the Assessment of Reliability of Systems Containing Software, British Standards Institution (BSI), October 1998.
2. DEF STAN 00-42/Issue 1, Reliability and Maintainability Guides, Part 2: Software, U.K. Ministry of Defense (MoD), September 1, 1997.
3. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.

B.3.10 Regression Testing

Purpose: To verify that changes and enhancements have been implemented correctly and that they do not introduce new errors or affect IA integrity.

Description: After a change or enhancement is implemented, a subset of the original test cases is executed. The results are compared with the original results to ensure stable and predictable system behavior after the change. In particular, regression testing should verify that changes and enhancements have not had an adverse effect on threat control measures. Regression testing should be performed in conjunction with change impact analysis.

Benefits: Regression testing minimizes the potential for unexpected system behavior following changes and enhancements.

Limitations: Test cases must be selected carefully so that both local and global effects of the changes or enhancements are verified.

References:

1. Beizer, B., *Software Testing Techniques*, International Thomson Press, 1990.
2. Kaner, C., *Testing Computer Software*, 2nd ed., John Wiley & Sons, 1993.
3. Kung, D., Hsia, P., and Gao, J., *Testing Object-Oriented Software*, IEEE Computer Society Press, 1998.
4. NASA GB-1740.13.96, *Guidebook for Safety-Critical Software — Analysis and Development*, NASA Glenn Research Center, Office of System Safety and Mission Assurance, 1996.
5. Perry, W., *Effective Methods for Software Testing*, 2nd ed., John Wiley & Sons, 1999.

B.3.11 Reliability Estimation Modeling

Purpose: To estimate software reliability for the present or some future time.

Description: A generic ten-step process is followed for estimating software reliability; the process uses the outputs of several IA analysis (Exhibit 2) and verification techniques:

1. Identify the software application being evaluated.
2. Derive the reliability requirement for this software component from the system reliability allocation.
3. Define failure modes and conditions.
4. Define operational environment and profiles.
5. Define test cases and procedures that correspond to the operational environment and profiles.
6. Select appropriate software reliability models.
7. Collect data from test results.
8. Estimate parameters from historical data.
9. Validate the model.
10. Use the model to estimate reliability for this software component.

Some of the more common reliability estimation models include: Duane, general exponential, Musa basic, Musa logarithmic, Littlewood/Verrall, and Schneidwind.

Note that software reliability is necessary to achieve IA integrity. Software must be reliable for security and safety functions to perform correctly.

Benefits: These models are useful in estimating how much maintenance and support will be required once a product is fielded.

Limitations: All of these models are used late in the life cycle; the software must be operational. Several of them assume that no new faults are introduced during maintenance activities, which is rarely the case. None of these models accommodate data derived from qualitative assessments or static analysis techniques.

References:

1. ANSI/AIAA R-013-1992, Recommended Practice for Software Reliability.
2. BS5760 Part 8, Guide to the Assessment of Reliability of Systems Containing Software, British Standards Institution (BSI), October 1998.

3. DEF STAN 00-42, Reliability and Maintainability Assurance Guides, Part 2: Software, U.K. Ministry of Defence, 1998.
4. IEEE Std. 982.1-1988, IEEE Standard Dictionary of Measures to Produce Reliable Software.*
5. IEEE Std. 982.2-1988, Guide for the Use of the IEEE Standard Dictionary of Measures to Produce Reliable Software.
6. Lyu, M. (Ed.), *Handbook of Software Reliability Engineering*, McGraw-Hill, 1996.

B.3.12 (IA) Requirements Traceability

Purpose: To verify that (1) all safety, reliability, and security requirements derived from IA goals are correct; (2) verify that all safety, reliability, and security requirements have been implemented correctly; and (3) verify that no additional unspecified or unintended capabilities have been introduced.

Description: Requirements traceability demonstrates that all requirements have been satisfied at each milestone during the life cycle. Requirements traceability is bidirectional; one should be able to (1) trace each requirement forward to its implementation in the design and source code, and (2) trace backward from the source code through the design to the requirements specification. Requirements traceability is generally captured in a tabular format. Backward traceability analysis is useful for finding unspecified or unintended functionality that has been accidentally implemented, while forward traceability analysis identifies specified requirements that have not been implemented or are incorrect, incomplete, or inconsistent.

Benefits: Requirements traceability helps to ensure that the product delivered is the product specified. Requirements traceability facilitates the development of test cases, other verification activities, and change impact analysis.

Limitations: The use of an automated tool is required. The degree of specificity in citing design and source code modules that implement a given requirement determines the effectiveness of the traceability analysis.

References:

1. DEF STAN 00-55, Requirements for Safety-Related Software in Defence Equipment, Part 1: Requirements, U.K. Ministry of Defence (MoD), August 1, 1997.
2. DEF STAN 00-55, Requirements for Safety-Related Software in Defence Equipment, Part 2: Guidance, U.K. Ministry of Defence (MoD), August 1, 1997.
3. Kotonya, G. and Sommerville, I., *Requirements Engineering: Processes and Techniques*, John Wiley & Sons, 1998.
4. Sommerville, I. and Sawyer, P., *Requirements Engineering: A Good Practice Guide*, John Wiley & Sons, 1997.
5. Thayer, R. and Dorfman, M., *Software Requirements Engineering*, 2nd ed., IEEE Computer Society Press, 1997.

B.3.13 Review IA Integrity Case

Purpose: To determine if the claims made about IA integrity are justified by the supporting arguments and evidence.

* Note that this standard began an update cycle in late 1999.

Description: The IA integrity case demonstrates that all IA goals and requirements have been achieved or that appropriate progress is being made toward achieving them. Evidence is reviewed to verify that it is complete, accurate, and current, including:

- Results from safety and security audits
- Results from vulnerability and threat analyses
- Reviews of critical threat zones in relation to threat control measures
- Results from static analysis activities
- Results from statistical testing based on operational profiles
- Results from performance testing
- Claims based on previous in-service experience
- Analysis of the impact on IA integrity from ASICs and reused software[1]

Evidence is reviewed to verify that the system and software engineering process is appropriate, such as[1,2]:

- Claims made that the methods, techniques (Exhibits 2, 3, and 4), and procedures used were followed correctly and are adequate
- Claims made that the analysis and interpretation of results from static and dynamic analyses are correct
- Justification for the OS, utilities, compiler, and automated tools used
- Justification of personnel competency
- Justification of the adequacy of IA verification activities

The IA integrity case is evaluated throughout the life cycle, as evidence is accumulated, to monitor that progress is being made toward meeting IA goals or identify the need for corrective action (*see also* Develop IA Integrity Case).

Benefits: The formality of reviewing an IA integrity case helps system designers and developers be more thorough when addressing IA integrity issues. Organized and complete IA integrity cases help Certification Authorities perform a more effective and thorough assessment.

Limitations: IA integrity cases must be succinct and organized in a logical manner to be useful.

References:

1. DEF STAN 00-42, Reliability and Maintainability Assurance Guides, Part 2: Software, U.K. Ministry of Defense (MoD), September 1, 1997.
2. DEF STAN 00-55, Requirements for Safety-Related Software in Defence Equipment, Part 1: Requirements, U.K. Ministry of Defence (MoD), August 1, 1997.
3. DEF STAN 00-55, Requirements for Safety-Related Software in Defence Equipment, Part 2: Guidance, U.K. Ministry of Defence (MoD), August 1, 1997.
4. Herrmann, D. and Peercy, D., Software reliability cases: the bridge between hardware, software, and system safety and reliability, *Proceedings of the Annual Reliability and Maintainability Symposium (RAMS'99)*, IEEE, 1999, 396–402.
5. JA 1002, Software Reliability Program Standard, Society of Automotive Engineers (SAE), 1998.

B.3.14 Root Cause Analysis

Purpose: To identify the underlying cause(s), events, conditions, or actions that individually, or in combination, led to an accident/incident; determine why the defect was not detected earlier.

Description: Root cause analysis is an investigative technique used to determine how, when, and why a defect was introduced and why it escaped detection in earlier phases. Root cause analysis is conducted by examining a defect, then tracing back step by step through the design and the decisions and assumptions that supported the design to the source of the defect. Root cause analysis supports defect prevention, continuous process improvement, and accident/incident investigation.

Benefits: The process of conducting root cause analysis may uncover defects in other areas as well.

Limitations: Root cause analysis can be time-consuming on large complex systems.

References:

1. Latino, R. and Latino, K., *Root Cause Analysis: Improving Performance for Bottom Line Results*, CRC Press, 1999.
2. *Root Cause Analysis Handbook*, ABS Group, Inc., 1000 Technology Drive, Knoxville, TN 37932-3369, 1999.
3. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.

B.3.15 Safety and Security Audits, Reviews, and Inspections

Purpose: To uncover errors and mistakes throughout the life of a system that could affect IA integrity.

Description: Safety and security audits, reviews, and inspections comprise a static analysis technique that is used to find errors of commission and errors of omission. Requirements, designs, implementations, test cases, test results, and operational systems can be subjected to safety and security audits. Unlike other audits and reviews, these focus solely on issues that impact safety and security; for example, verifying that fault tolerance has been implemented correctly, access control rules have been specified correctly, or operational security procedures are being followed correctly. Checklists, in the form of a set of questions intended to stimulate a critical appraisal of all aspects of safety and security, can be used to guide the audits.[5] Any open issues or discrepancies are assigned a severity and tracked through resolution. Internal and independent audits should be conducted regularly. Safety and security audits complement IA requirements traceability.

Benefits: Communication among all stakeholders is facilitated. More and different types of errors are detected, due to the involvement of multiple stakeholders.

Limitations: Safety or security audits must yield repeatable results if their validity is questionable; objective criteria need to be evaluated.[1] Adequate preparation time is necessary.

References:

1. Gollmann, D., *Computer Security*, John Wiley & Sons, 1999.

2. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
3. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.

B.3.16 Stress Testing

Purpose: To determine (1) maximum peak loading conditions under which a system will continue to perform as specified and IA integrity will be maintained; and (2) system overload/saturation conditions that could lead to a system compromise or failure.

Description: Failure rates can be orders of magnitude greater when resource utilization is high.[2] Overload conditions can cause an operation not to complete on time, or at all. Erratic behavior often results from overload conditions, leading to a system compromise or failure. Stress testing, sometimes called avalanche testing, seeks to observe the impact of increasing system loads on IA integrity, in particular changes that take place when transitioning from normal to peak and overload conditions. Stress testing helps to verify the correct dimensioning of internal buffers, dynamic variables, stacks, lists, I/O bandwidth, etc.[3] During stress testing, system performance is monitored under low, normal, peak, and overload conditions. Interfaces, memory capacity, throughput capacity, and communication links of IA-critical and IA-related functions/entities are each "stressed." The system is subjected to extreme conditions and anomalous situations to verify that it performs correctly.[4] Examples of stress testing modes include[2,3]:

- Increasing data transmission/output and receipt/input rates
- Increasing message sizes
- Increasing the number of simultaneous users, processes, transactions, queries, etc.
- Increasing database size
- Increasing processor and I/O channel speeds

Stress testing should be conducted as an adjunct to performance testing.

Benefits: Stress testing provides a realistic assessment of how a system will perform in the operational environment. It also helps identify conditions that may cause the compromise or failure of an IA-critical or IA-related function/entity.

Limitations: None.

References:

1. Beizer, B., *Software Testing Techniques*, International Thomson Press, 1990.
2. DEF STAN 00-42/Issue 1, Reliability and Maintainability Guides, Part 2: Software, U.K. Ministry of Defense (MoD), September 1, 1997.
3. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
4. Kaner, C., *Testing Computer Software*, 2nd ed., John Wiley & Sons, 1993.
5. NASA GB-1740.13.96, *Guidebook for Safety-Critical Software — Analysis and Development*, NASA Glenn Research Center, Office of System Safety and Mission Assurance, 1996.
6. Perry, W., *Effective Methods for Software Testing*, 2nd ed., John Wiley & Sons, 1999.

B.3.17 Testability Analysis, Fault Injection, Failure Assertion

Purpose: To verify IA integrity by determining if a system design can be verified and is maintainable, and that it detects and responds correctly to erroneous data, conditions, and states.

Description: The goal of testability analysis is to derive an indicator of the testability of a software product from an analysis of the controllability and observability of internal nodes.[3] This indicator is based on measurements of the number of unique operators, number of unique operands, total occurrence of each operator, total occurrence of each operand, and number of unique logic paths.[3] This analysis uncovers unreachable nodes, unused nodes, and nondeterministic conditions, all of which could be used as an opening for an attack.[3] Since the original project, testability analysis has been expanded to include analyses of traceability, repeatability, predictability, accessibility, fault injection, and failure assertion.[2,5]

Fault injection, often referred to as error seeding, is used to uncover potential systematic failures. Experience is used to predict which input states are likely to cause errors,[6] and test cases are generated accordingly. Then, known errors are inserted into a program and the test cases are executed. Errors should be injected to trigger specific vulnerabilities/threats and, hence, verify the effectiveness of threat control measures.

The success rate at finding known errors provides (1) a measurement of the effectiveness of the testing effort[1,6] and (2) an estimate of the number of real errors remaining in the software[1,4]:

$$\frac{\text{Found injected errors}}{\text{Total injected errors}} \cong \frac{\text{Found real errors}}{\text{Total real errors}}$$

Independent teams are used to insert and test for injected faults.

Benefits: Testability analysis helps identify whether or not a system design can be verified, and if not, the components that need redesign. Potential attack points and ineffective threat control measures are identified.

Limitations: Testability analysis is most useful when applied to large complex systems. Injected faults must be representative of real faults.[1,6]

References:

1. DEF STAN 00-42/Issue 1, Reliability and Maintainability Guides, Part 2: Software, U.K. Ministry of Defense (MoD), September 1, 1997.
2. Friedman, M. and Voas, J., *Software Assessment: Reliability, Safety, and Testability*, John Wiley & Sons, 1995.
3. Herrmann, D., *Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors*, IEEE Computer Society Press, 1999.
4. IEC 61508-7, Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7: Overview of Techniques and Measures.
5. Parkinson, J., Classification of programmable electronic systems operation for testability, *Directions in Safety-Critical Systems*, Springer-Verlag, 1993, 67–83.
6. Storey, N., *Safety-Critical Computer Systems*, Addison-Wesley, 1996.
7. Voas, J. and McGraw, G., *Software Fault Injection*, John Wiley & Sons, 1998.

B.3.18 Usability Testing

Purpose: To determine if a system performs in the operational environment in a manner acceptable to and understandable by administrators and end users; verify that the design does not contribute to induced or invited errors which could lead to a system compromise or failure.

Description: Human factors engineering is an important aspect of achieving and maintaining IA integrity. Usability testing evaluates the human factors engineering characteristics of a system. Usability testing is conducted by a team of users. The focus of usability testing is to verify that domain knowledge has been captured and implemented correctly, in particular with regard to (1) how a system will be used, (2) what a system will be used for, and (3) how users expect to interact with a system. The potential for induced or invited errors is examined in the context of user expectations. The User's Manual and Operational Procedures are executed as part of usability testing.

Benefits: The potential for induced or invited errors and mismatches with user expectations are identified before a system is deployed. Errors in the User's Manual and Operational Procedures are uncovered.

Limitations: None.

References:

1. Beizer, B., *Software Testing Techniques*, International Thomson Press, 1990.
2. Kaner, C., *Testing Computer Software*, 2nd ed., John Wiley & Sons, 1993.
3. Perry, W., *Effective Methods for Software Testing*, 2nd ed., John Wiley & Sons, 1999.
4. Rubin, J., *Handbook of Usability Testing: How to Plan, Design and Conduct Effective Tests*, John Wiley & Sons, 1994.

Exhibit 5 Information Assurance Accident/Incident Investigation Techniques

IA Accident/ Incident Investigation Techniques	*C/R*	*Type*	*Life-Cycle Phase in which Technique is Used*		
			Concept	*Development*	*Operations*
Barrier analysis[a]	C4	SA, SE		x	x
Critical incident interviews	C4	SA, SE		x	x
Damage mode effects analysis[a]	C4	SA, SE		x	x
Event and causal factor charting	R4/C4	SA, SE		x	x
Scenario analysis	C4	SA, SE		x	x
Sequentially timed event plot (STEP) investigation system	R4/C4	SA, SE		x	x
Time/loss analysis (TLA) for emergency response evaluation	C4	SA, SE			x
Warning time analysis	C4	SA, SE			x

[a] These techniques can also be used during verification.

B.4 IA Accident/Incident Investigation Techniques

B.4.1 Barrier Analysis

Purpose: To ascertain which defensive layers failed or were missing or inadequate during an accident/incident.

Description: Barrier analysis helps to determine accident/incident causation by examining each defense in depth layer (or barrier) for accidental or intentional unwanted control, data, or information flow. Hazardous control and information flows to and from people and processes are uncovered and how they penetrated or bypassed existing defensive layers. Defensive layers that failed or were missing or inadequate are identified, as well as those that did not fail. In practice, IA design techniques/features are referred to as "hard barriers," while operational procedures and physical security measures are referred to as "soft barriers" (see Chapter 8, Exhibits 5 and 6).

Benefits: Barrier analysis highlights the need for new or modified defensive layers, and helps to pinpoint intermediate accident/incident causation.

Limitations: Barrier analysis does not evaluate an entire system, only the defensive layers.

References:

1. U.S. Department of Energy (DoE), Barrier Analysis, DOE-76-45/29, SSDC-29, Safety Systems Development Center, July 1985.
2. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.

B.4.2 Critical Incident Interviews

Purpose: To collect evidence about an accident/incident and previous related mistakes, anomalies, and near-misses from operational personnel.

Description: Key personnel with first-hand experience in developing, using, administering, and maintaining the system that failed or was compromised are interviewed. The interview focuses on experience with or observations about the system immediately before and during the accident/incident, as well as mistakes, anomalies, and near-misses experienced or observed in the past. Operator actions, system modes/states, conditions, functions, malfunctions, etc. are discussed. Printouts, server and workstation OS and memory dumps, audit trails, test results, network and system configuration reports, etc. are collected to support verbal accounts. This information is analyzed to expose potential immediate, intermediate, and chronic accident/incident precursors.

Benefits: People closest to and with the most experience using a system have invaluable insights that other people do not and that may not be readily apparent from technical evidence alone. They also assist in the accurate interpretations of events.

Limitations: Interviewers need to be careful to separate fact from opinion, subjective from objective data. Interviews must be conducted in an open, positive environment so that witnesses do not feel threatened, intimidated, coerced, or fearful of employment-related retaliation.

References:

1. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.

B.4.3 Damage Mode Effects Analysis

Purpose: To postulate which specific threat mechanisms caused an accident/incident from an analysis of the damage modes.

Description: Damage mode effects analysis is a deductive technique that provides an early assessment of survivability and the effect of an accident/incident on a system's mission/operation. Damage mode effects analysis is an extension of an FMECA. It examines the damage mode for each IA-critical and IA-related function, entity, component, and subcomponent; specifically[1]:

- Type of damage experienced
- Primary, secondary, and cascading damage effects on this and other functions, entities, and systems
- Variation in the damage mode by operational mode/state, profile, and mission
- Local, next higher level, and end effect(s) of the damage

The damage modes are analyzed to postulate which specific threat mechanisms caused an accident/incident.

Benefits: The survivability assessment provides essential input to recovery efforts and often exposes latent vulnerabilities. If legal action is pursued as the result of an accident/incident, a damage mode effects analysis must be performed.

Limitations: The effectiveness of this technique is proportional to the ability to analyze damage modes immediately during or after an accident/incident.

References:

1. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.

B.4.4 Event and Causal Factor Charting

Purpose: To graphically reconstruct the events, immediate, intermediate, and root cause(s) of an accident/incident.

Description: Event and causal factor charts depict a detailed sequence of the facts and activities that led to an accident/incident. The right-most block on the chart is the primary event — the accident/incident. The immediate cause is shown in the next block, on the left parallel to the first block. Reasons that permitted or contributed to the immediate causes are listed underneath. This process is continued backward to the underlying root cause(s)/event(s). Unknown events are shown as gaps (?) in the diagram and highlight areas needing further investigation. Causes are categorized as human or system actions. Cascading and lateral events are recorded as well so that all pertinent avenues of investigation are explored (see Chapter 8, Exhibit 7).

Benefits: Event and causal factor charts summarize what is known and unknown about an accident/incident in a format that is easily understood by all stakeholders. The sequential nature of the charts facilitates an unfolding investigation. Arrows connecting cause and event blocks represent potential primary and secondary prevention points; this information can be used to reinforce defensive layers.

Limitations: Event and causal factor charts do not capture the exact timing of events.
References:

1. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.

B.4.5 Scenario Analysis

Purpose: To develop avenues to investigate from causation theories and hypothetical event chains.

Description: During scenario analysis, all system entities, components, operational profiles, modes/states, environment, and missions as well as operator actions are examined by an interdisciplinary team. This team, under the guidance of a neutral facilitator, attempts to surmise all possible, credible, and logical scenarios that could have caused or contributed to an accident/incident. The starting point for the team is the fact that the accident/incident occurred. They do not examine evidence; rather, they develop causation theories and hypothetical event chains, based on experience and inductive reasoning, that become avenues to investigate.

Benefits: Scenario analysis, because it is not dependent on extensive evidence, is particularly well suited for investigating novel accidents/incidents for which little or no historical data exists.[1]

Limitations: Successful scenario analysis is dependent on an accurate understanding of the system that failed or was compromised, without letting that knowledge constrain visualization of potential threat scenarios.
(*Note:* Do not confuse this technique with formal scenario analysis discussed in Section B.2 and Chapter 6.)

References:

1. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.

B.4.6 Sequentially Timed Event Plot (STEP) Investigation System

Purpose: To expound a diagram of linked, sequentially timed events and their causal relationships, which demonstrates how an accident/incident occurred.

Description: The STEP investigation system is an analytical methodology that develops accident process descriptions. A diagram of sequentially timed, multilinear events depicts accident/incident causal relationships. Direct, converging, and diverging relationships of immediate, intermediate, and underlying events are illustrated. STEP diagrams visually display the sequence and timing aspects of accident/incident precursors. The event chain necessary to produce the accident/incident outcome is linked together; accident data is transformed into event building blocks.[3,4,4] Uncertainties or gaps in the event chain are highlighted for further investigation. Standard symbols and notation are used to develop a STEP diagram (see Chapter 8, Exhibits 8 through 11).

Benefits: The STEP investigation system supports an in-depth, thorough, and focused analysis of an accident/incident. STEP diagrams are easy to understand; consequently, they can be reviewed and verified by multiple stakeholders. An unlimited number of logical possibilities (accidental/intentional, human/computer action)

can be investigated.[344] STEP diagrams expose misunderstandings about how a system "should" versus "does" operate and deficiencies in operational procedures, contingency plans, and physical security practices.

Limitations: A skilled facilitator is needed to keep the analysis proceeding at a level that is meaningful and relevant to the investigation. The analysis should not be at too high or too low a level.

References:

1. Events Analysis Inc., *STEP Investigation Course*, 1978–1992.
2. Events Analysis Inc., *STEP Investigation Guides*, 1978–1992.
3. Ferry, T., *Modern Accident Investigation and Analysis*, John Wiley & Sons, 1988.
4. Hendrick, K. and Benner, L., *Investigating Accidents with STEP*, Marcel Dekker, 1988.
5. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.
6. U.S. Department of Transportation, Transportation Safety Institute, *Risk Assessment Techniques Manual*, August 1986.

B.4.7 Time/Loss Analysis (TLA) for Emergency Response Evaluation

Purpose: To evaluate the (1) effect of human intervention following an accident/incident, (2) controllability of an accident/incident, and (3) effectiveness of mitigating threat control measures over time.

Description: The results of TLA are recorded in TLA graphs. TLA graphs measure and compare actual versus natural loss following an accident/incident. Intervention data is recorded at vertical points on the *x*-axis timeline. Loss units (number of fatalities or injuries, property damage, financial loss, loss of productivity, environmental damage, etc.) are recorded on the *y*-axis. T_0 is when the accident/incident commences; T_{end} correlates to the time of the last directly related loss. The natural loss curve is estimated over time given no human intervention. The actual loss curve plots the sequential effect of each intervening action T_n. The slope between T_0 and T_1 is the same for both curves and represents the effectiveness of automatic mitigating (detect/characterize, respond/recover) threat control measures over time. The delta between the actual and natural loss curves from T_1 on is a function of the controllability of the accident/incident and the value of human intervention. The general shape of the curves is more important than precise data points[5] (see Chapter 8, Exhibit 12).

Benefits: TLA graphs can also be used to analyze alternative hypothetical intervention strategies[5] and contingency plans. Criteria for measuring loss units must be standardized and objective.

Limitations: TLA must be performed, or at least begun, promptly after an accident/incident because the evidence tends to dissipate.

References:

1. Benner, L., *Guide 4: A Guide for Using Time/Loss Analysis Method in Accident Investigations (TLA Guide)*, Events Analysis Inc., 1983.
2. U.S. Department of Energy, DOE 76-45/37 (SSOC-37), Time/Loss Analysis, 1987.
3. Driver, E. and Benner, L., Evaluating dangerous goods emergency response with time/loss analysis, *Proceedings, 6th International Symposium — Packaging and Transportation of Radioactive Materials*, 1980.

4. Driver, E. and Benner, L., Evaluating HAZMAT responses with time/loss analysis, *Fire Journal*, July 1981.
5. System Safety Society, *System Safety Analysis Handbook*, 2nd ed., July 1997.

B.4.8 Warning Time Analysis

Purpose: To investigate the delta between the available and actual response times (human and automatic) to an accident/incident and the contributing factors, such as erroneous, unforeseen, or unnecessary delays.

Description: Warning time analysis examines various intervals along the timeline from when an accident/incident occurred and recovery mechanisms were initiated. Specific intervals scrutinized include[1]:

- **Propagation time:** time from occurrence of initiating event to time when accident/incident occurred
- **Detection time:** time from occurrence of initiating event to earliest indication or alarm
- **Response time$_A$:** time for automatic corrective action
- **Response time$_H$:** time for human-initiated corrective action

(See Chapter 8, Exhibit 13.)

Benefits: Warning time analysis evaluates the effectiveness of anomaly detection, the time available for a response, and the adequacy of emergency operational procedures and contingency plans, especially when system reconfiguration or redundant switchover is needed. A comparison between the available and actual response times is made.

Limitations: None.

References:

1. Space Product Assurance: Policy and Principles, European Space Agency, ECSS-Q-40A, April 19, 1996.