DOT/FAA/AR-08/34

Air Traffic Organization
NextGen & Operations Planning
Office of Research and
Technology Development
Washington, DC 20591

Requirements Engineering Management Findings Report

May 2009

Final Report

U.S. Department of Transportation
Federal Aviation Administration

Technical Report Documentation Page

1. Report No. DOT/FAA/AR-08/34	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle REQUIREMENTS ENGINEERING MANAGEMENT FINDINGS REPORT		5. Report Date May 2009
		6. Performing Organization Code
7. Author(s) David L. Lempia and Steven P. Miller		8. Performing Organization Report No.
9. Performing Organization Name and Address Rockwell Collins, Inc. 400 Collins Road NE Cedar Rapids, IA 52498		10. Work Unit No. (TRAIS)
		11. Contract or Grant No. DTFACT-05-C-00004
12. Sponsoring Agency Name and Address U.S. Department of Transportation Federal Aviation Administration Air Traffic Organization NextGen & Operations Planning Office of Research and Technology Development Washington, DC 20591		13. Type of Report and Period Covered Final Report
		14. Sponsoring Agency Code AIR-120
15. Supplementary Notes The Federal Aviation Administration Airport and Aircraft Safety R&D Division COTR was Charles Kilgore.		

16. Abstract

This report describes the results of a study conducted for the Federal Aviation Administration Airport and Aircraft Safety Research and Development Division to determine methods to enable successful management, control, integration, verification, and validation of system and software requirements that may be developed by multiple entities.

In Phase 1 of the study, a survey of the literature was conducted to identify best practices in requirements engineering, and an industry survey was taken to determine current practices and the issues and concerns of practitioners. These sources were then combined to produce an overall list of safety and certification issues in requirements engineering management.

In Phase 2 of the study, the findings of Phase 1 were further refined and used to guide the selection of several best practices from the literature. This final report included the results of Phases 1 and 2. The best practices were integrated into a complete set of 11 recommended practices and documented in a Handbook of Requirements Engineering Management. These practices can be incrementally added to an organization's existing requirements engineering process to incorporate the best practices identified in the literature.

17. Key Words Requirements, Engineering, Avionics, Systems, Software		18. Distribution Statement This document is available to the U.S. public through the National Technical Information Service (NTIS) Springfield, Virginia 22161.	
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 216	22. Price

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AD	Architecture Design
ARINC	Aeronautical Radio, Incorporated
ARP	Aerospace Recommended Practices
CNS/ATM	Communication, Navigation, Surveillance, and Air Traffic Management
CoRE	Consortium Requirements Engineering
DER	Designated Engineering Representative
FAA	Federal Aviation Administration
FGS	Flight Guidance System
HMI	Human Machine Interface
IMA	Integrated Modular Avionics
MBD	Model-Based Development
MOPS	Minimum Operational Performance Standard
NASA	National Aeronautic and Space Administration
OEM	Original equipment manufacturer
OOD	Object-Oriented Development
OOTiA	Object-Oriented Technology in Aviation
PLD	Programmable logic device
REM	Requirements Engineering Management
RSML	Requirements State Machine Language
RSML^{-e}	Requirements State Machine Language Without Events
RTCA	Requirements and Technical Concepts for Aviation
SAE	Society of Automotive Engineers
SCR	Software Cost Reduction
SpecTRM	Specification Toolkit and Requirements Methodology
SpecTRM-RL	SpecTRM Requirements Language
SQA	Software Quality Assurance
SR	System Requirements
ST	System Test
SysML	Systems Modeling Language
TCAS	Traffic Alert and Collision Avoidance System
TSO	Technical Standard Order
UML	Unified Modeling Language
UR	User Requirements
V&V	Verification and validation

EXECUTIVE SUMMARY

This report describes the results of a study conducted for the Federal Aviation Administration Airport and Aircraft Safety Research and Development Division to determine methods to enable successful management, control, integration, verification, and validation of system and software requirements that may be developed by multiple entities.

In Phase 1 of the study, three tasks were completed. First, a survey of the literature was conducted to identify best practices in Requirements Engineering Management (REM). Second, an industry survey was taken to identify current practices and the issues and concerns of practitioners. Third, these sources were then combined to produce an overall list of safety and certification issues in REM.

In Phase 2 of the study, the findings of Phase 1 were further refined and used to guide the selection of several best REM practices identified the literature. This final report includes the results of Phases 1 and 2. The best practices were integrated into a complete set of recommended practices and documented in the Handbook of Requirements Engineering Management. To validate these practices, they were applied to two examples that were also used to illustrate the practices. The Handbook was also reviewed on a regular basis during its development by experts in requirements engineering, system safety analysis, and software certification.

The survey results show that the majority of respondents capture requirements in English text, tables, and diagrams using either Telelogic® DOORS® or a word processor. Modeling tools, such as MATLAB Simulink® and Esterel Technologies SCADE Suite™, are starting to be used for the specification of requirements, though there does not seem to be a consensus on what level of requirements these models represent. Object-oriented approaches, such as the Unified Modeling Language, are also starting to be used, though to a lesser extent and primarily for software requirements.

Perhaps the most surprising finding is that 16 years after the publication of DO-178B, there still seems to be many questions on how to specify and manage requirements within a DO-178B process. It also appears that the best practices identified in the literature search are being used only rarely. Even the object-oriented approaches for requirements elicitation and documentation, probably the best known of these practices, do not appear to be widely used by the survey respondents.

The Handbook of Requirements Engineering Management addresses these concerns by integrating 11 main-level recommended practices for requirements engineering. These can be incrementally added to an organization's existing requirements engineering process to incorporate the best practices identified in the literature.

1. INTRODUCTION.

The Federal Aviation Administration (FAA) Airport and Aircraft Safety Research and Development Division commissioned a research study examining Requirements Engineering Management (REM). The goal of this study was to determine methods that enable successful management, control, integration, verification, and validation of system and software requirements that may be developed by multiple entities.

The study was conducted in two phases. In Phase 1, three tasks were completed. First, a survey of the literature was conducted to identify best practices in REM. Second, an industry survey was taken to identify current practices and the issues and concerns of practitioners. Third, these sources were then combined to produce an overall list of safety and certification issues in REM.

In Phase 2 of the study, the findings of Phase 1 were further refined and used to guide the selection of several best practices from the literature. This final report includes the results of Phases 1 and 2. The best practices were integrated into a set of eleven recommended practices and documented in the Handbook of Requirements Engineering Management [1]. To validate these practices, they were applied to two examples that were also used to illustrate the practices in the Handbook. The Handbook was also reviewed on a regular basis during its development by experts in requirements engineering, system safety analysis, and software certification.

The Handbook is structured so that the recommended practices can be incrementally added to an organization's existing REM process to gradually incorporate the best practices identified in the literature.

1.1 BACKGROUND.

Incomplete, ambiguous, or rapidly changing requirements are well known to have a profound impact on the quality and cost of system and software development. Fred Brooks states the problem succinctly in reference 2.

> "The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements...No other part of the work so cripples the resulting system if done wrong. No other part is as difficult to rectify later."

Several studies have shown that the majority of software development errors are made during requirements analysis, and that most of these errors are not found until the later phases of a project. Other studies have shown that due to the amount of rework that has to be done, the cost of fixing a requirements error grows dramatically the later it is corrected [3 and 4]. In one well-known study conducted at TRW, Inc., it was found that it costs ten times as much to correct a requirements error during unit testing than during requirements analysis [3]. Correcting a requirements error after a product had been deployed increased the cost by 100 to 200 times [3]. Moreover, requirements errors are often the most serious errors. Investigators focusing on safety-critical systems have found that requirements errors are most likely to affect the safety of embedded system than errors introduced during design or implementation [5-9].

1.2 OVERVIEW OF THE MAIN STUDY FINDINGS.

This section provides an overview of the main findings of the study. The findings from the literature search, the industry survey, and the development of the Requirements Engineering Management Handbook are discussed.

1.2.1 Findings From the Literature Search.

Several best practices specifically developed for REM were discovered through the literature search. Each of these has its own strengths and weaknesses. For example, the Software Cost Reduction (SCR) method [10-13] and the Consortium Requirements Engineering (CoRE) method [14-16] provide a strong conceptual model for relating system and software requirements, by making heavy use of tables for ease of presentation and validation, and encouraging the specification of environmental assumptions, an often overlooked aspect of requirements. The Requirements State Machine Language (RSML) method [17 and 18] introduces additional tabular formats and graphical notations that make requirements specifications accessible to a still wider audience. The Specification Toolkit and Requirements Methodology (SpecTRM) [7-9, 19, and 20] adds to RSML techniques for capturing the intent, or rationale, of requirements, and for developing requirements for human factors.

The Unified Modeling Language (UML) [21 and 22] also provides several notations to support REM. While UML does not appear to provide the same level of rigor as the approaches described above, it is widely known, is supported by many commercial tools, and has its own useful notations for the elicitation and presentation of requirements. In particular, use cases [23-25] appear to be a very effective technique for the early validation of requirements and specifying the operational concepts that define how operators and other systems will interact with the system.

A more recent advance is the use of modeling tools, such as MATLAB Simulink® [26 and 27] and Esterel Technologies SCADE Suite™ [28], for the specification of requirements. Like the UML tools, these have strong support from their vendors and are increasingly being used for specifying system and software requirements. However, critics of these tools assert that they are actually being used to specify detailed design or even source code, not requirements.

Some of the most important issues in REM identified in the literature search include clearly defining the system boundary, documenting all environmental assumptions, supporting development by multiple entities, providing intent and rationale, supporting the validation of requirements, managing change and complexity, specifying Human Machine Interface (HMI) requirements, and supporting the verification of requirements.

1.2.2 Findings From the Industry Survey.

The survey results show that the majority of respondents capture requirements in English text, tables, and diagrams using either Telelogic® DOORS® or a word processor. Modeling tools such as MATLAB Simulink [26 and 27] and Esterel Technologies SCADE Suite [28] are starting to be used for the specification of requirements, though there does not seem to be a consensus on what level of requirements these models represent. Object-oriented approaches such as UML [21 and 22] are also starting to be used, though to a lesser extent and primarily for software requirements.

Perhaps the most surprising finding is that 14 years after the publication of DO-178B [29], there still seem to be many questions on how to specify and manage requirements within a DO-178B process. It also appears that the best practices identified in the literature search are being used only rarely. Even the object-oriented approaches for requirements elicitation and documentation, probably the best known of these practices, do not appear to be widely used by the survey respondents.

Finally, the survey respondents raised very few concerns related to several key issues identified in the literature search, including how to specify the system boundary, how to identify and document environmental assumptions, how specify the intent of requirements, and how to specify requirements for the HMI interface.

1.2.3 Development of the Requirements Engineering Management Handbook.

As a result of the findings summarized in the preceding sections, the recommendation made at the end of Phase 1 was to develop an incremental approach for improved REM that would allow practitioners to progress from their current state of practice by gradually introducing more of the best practices identified in the literature search. This approach is documented in the Requirements Engineering Management Handbook [1] developed in Phase 2 of the project.

The Requirements Engineering Handbook describes 11 main-level recommended practices gleaned from the literature on how to collect, write, and organize requirements:

1. Develop the System Overview
2. Identify the System Boundary
3. Develop the Operational Concepts
4. Identify the Environmental Assumptions
5. Develop the Functional Architecture
6. Revise the Architecture to Deal with Implementation Constraints
7. Identify the System Modes
8. Develop the Detailed Behavioral and Performance Requirements
9. Define the Software Requirements
10. Allocate the System Requirements to Subsystems
11. Provide Rationale

Introducing any of these practices into an REM process that does not already include them should improve the quality of the process. The recommended practices were mapped against the issues identified in the industry survey and the literature search and were found to significantly address almost all of the issues raised. To validate the recommended practices in the Handbook, two running examples were developed and used to illustrate the recommended practices. The Handbook was also reviewed on a regular basis during its development by experts in requirements engineering, system safety analysis, and software certification.

1.3 ORGANIZATION OF REPORT.

The remainder of this report is organized as follows:

- Section 2 surveys the current literature on requirements engineering and provides a brief synopsis of several of the best practices in the literature.
- Section 3 discusses the findings of an industry survey conducted to determine current practices.
- Section 4 discusses the safety and certification issues related to requirements engineering identified in the survey and the literature search.
- Section 5 describes how the safety and certification issues were consolidated and prioritized.
- Section 6 discusses how the recommended practices were developed and how they address the most important safety and certification issues.
- Section 7 presents the main findings and recommendations of the study.
- Section 8 presents the references.
- Section 9 presents a glossary of the four-variable model.

2. BEST PRACTICES IDENTIFIED IN THE LITERATURE.

The volume of information on REM is immense, making a complete survey beyond the scope of the project. As a result, this section focuses on a few of the better known approaches and provides a brief synopsis of each. References for these approaches are provided in section 8. A list of additional references not reviewed is presented in appendix A.

This section first reviews several key regulatory documents guiding the practice of requirements engineering in civil aviation. This provides background information needed to understand the survey results presented in section 3 and the safety and certification issues identified in section 4. Following this, a variety of notations, methods, and tools developed for requirements engineering are discussed. Finally, Integrated Modular Avionics (IMA) are discussed because safety, performance, and functional requirements developed for an IMA system may require strong coordination between diverse teams located in different companies.

2.1 DO-178B, DO-248B, AND DO-278.

Software for airborne civil aircraft must be developed in compliance with the Requirements and Technical Concepts for Aviation (RTCA) document DO-178B, “Software Considerations in Airborne Systems and Equipment Certification” [29]. Additional clarification of DO-178B can be found in RTCA document DO-248B, Final Report for Clarification of DO-178B “Software

Considerations in Airborne Systems and Equipment Certification" [30]. Similar guidance for software in Communication, Navigation, Surveillance, and Air Traffic Management (CNS/ATM) systems can be found in RTCA document DO-278, "Guidelines for Communication, Navigation, Surveillance, and Air Traffic Management (CNS/ATM) Systems Software Integrity Assurance" [31]. Much of the guidance in these documents is relevant to REM and has a profound impact on the practice of REM for civil avionics systems. This section summarizes the relevant portions of these documents, particularly DO-178B, which is the basis for DO-248B and DO-278.

DO-178B does not prescribe a preferred software life cycle, but does describe the separate processes that comprise most software life cycles and the interactions between them. The software life cycle processes defined in DO-178B include the software planning process, the software development process, and the integral processes of software verification, software configuration management, software quality assurance, and certification liaison.

The software development process begins with the software requirements process described in section 5.1 of reference 29, which uses the outputs of the system engineering process (e.g., as described in as described in the Society of Automotive Engineers (SAE) Aerospace Recommended Practices (ARP) document ARP 4754 [32]) and the safety assessment process (e.g., as described in ARP 4761 [33]) to develop the software high-level requirements. As part of the software requirements process, the developer is required to analyze the "system functional and interface requirements allocated to software for ambiguities, inconsistencies, and undefined conditions." Inadequate or incorrect inputs are reported back to the source process. The software high-level requirements are defined in the glossary of DO-178B as "… software requirements developed from analysis of system requirements, safety-related requirements, and system architecture."

The high-level software requirements must satisfy several criteria. Each system requirement that is allocated to software should be specified in the high-level requirements, and high-level requirements that address system requirements allocated to software to preclude system hazards should be defined. The software high-level requirements should be stated in quantitative terms, with tolerances when applicable, and should not describe design or verification detail except for specified and justified design constraints. From this, it seems clear that the high-level requirements will present a Black Box specification of the software behavior, except for those details called out in system design constraints. The developer is also required to show that the software high-level requirements comply with the system requirements, are accurate and consistent, are compatible with the target computer, are verifiable, conform to standards, are traceable to the system requirements, and have accurate algorithms.

With the software high-level requirements as a primary input, the developer can begin the software design process as stated in section 5.2 of reference 29, in which the "software high-level requirements are refined through one or more iterations… to develop the software architecture, and the low-level requirements that can be used to implement Source Code." Section 5.2.2 of DO-178B specifically states that this may involve one or more lower levels of requirements.

The software architecture is defined in the glossary of DO-178B as the "… structure of the software selected to implement the software requirements."

While DO-178B does not define the software architecture more than this, it can be inferred from other parts of the document that the software architecture is a white-box view of the software that encompasses what software components need to exist, the interfaces to those components, how components are invoked, and how information flows between the components. Partitioning of data and control also appears to be a function of the software architecture. The developer is required to show that the software architecture is compatible with the high-level requirements, is consistent, is compatible with the target computer, is verifiable, conforms to standards, and can confirm software partitioning integrity.

It seems clear that DO-178B provides the developer the freedom to refine the Black Box view of the software found in the high-level requirements into one or more layers of components, where the behavior of each component is, in turn, specified with a Black Box view of that component's behavior. This process terminates when the low-level requirements are fully developed. The low-level requirements are defined in the glossary of DO-178B as the "… software requirements derived from high-level requirements, derived requirements, and design constraints from which source code can be directly implemented without further information."

The developer is required to show that the low-level software requirements comply with the high-level software requirements, are accurate and consistent, are compatible with the target computer, are verifiable, conform to standards, are traceable to the high-level software requirements, and have accurate algorithms.

During the course of the software design process, it may be necessary to make design choices that change the behavior of a component or that have implications for system safety. These are documented as derived requirements. DO-178B defines derived requirements in the glossary as

> "Additional requirements resulting from the software development process, which may not be directly traceable to higher-level requirements."

Derived requirements are introduced during the software development process as the result of design decisions. Section 5.0 of DO-178B expands on this by stating

> "Each software development process may produce derived requirements. Derived requirements are requirements that are not directly traceable to higher-level requirements. An example of such a derived requirement is the need for interrupt handling software to be developed for the chosen target computer. High-level requirements may include derived requirements, and low-level requirements may include derived requirements. The effect of derived requirements on safety-related requirements are determined by the system safety assessment process."

This is further clarified in DO-248B 30 in FAQ #36

> "Essentially this means that during the software development process, requirements may be developed that do not trace directly to higher-level requirements. Development of these derived requirements is typically based upon the results of design, performance, architectural decisions, etc., and is therefore not directly traceable to higher-level requirements. Some of these requirements (the ones that are not directly traceable to a higher level requirement) are considered to be derived. Even though a derived requirement is not traceable to a higher level requirement, the derived requirement may affect the higher-level requirements."

Derived requirements may be either high-level derived requirements or low-level derived requirements depending on whether they affect the high- or low-level requirements.

While the software requirements and design processes described in DO-178B seem clear enough on first reading, there are some subtleties that cause confusion. The relationship between software requirements and the software design process seems to be one such source of confusion. As mentioned earlier, the software design process described in Section 5.2 of DO-178B clearly allows iterations with "one or more lower levels of requirements." However, the software verification process described in Section 6.0 of DO-178B only discusses high- and low-level software requirements. The exact nature of the requirements for the components at the intermediate levels is not clear. If the specification of the behavior of these components consists of high-level software requirements, it may be in conflict with the definition of high-level requirements as software requirements developed from analysis of system requirements, safety-related requirements, and system architecture. Conversely, if the specification of the behavior of the intermediate level components consists of low-level requirements, this seems to be in conflict with the definition of the low-level requirements as software requirements derived from high-level requirements, derived requirements, and design constraints from which source code can be directly implemented without further information.

Another source of confusion relates to the distinction between derived requirements and low-level requirements. Derived requirements are defined as additional requirements resulting from the software development process, which may not be directly traceable to higher-level requirements. However, the software design process is inherently one of decomposing a component into smaller components that work together to satisfy the requirements of the parent. Each component has its own set of requirements that are different from those of its parent. It is not always clear whether these can be directly traced to the higher-level requirements. Generally, derived requirements seem to be those that modify or further constrain the externally visible behavior of the system or that have safety implications, both of which need to be provided to the system safety process for review.

Several more safety and certification issues related to DO-178B were raised in the industry survey. These are discussed further in section 4.

2.2 STATECHARTS, STATEMATE, AND RHAPSODY.

Several notations have been developed for capturing requirements in tables or diagrams. One of the earliest modeling notations, statecharts, provides a powerful and intuitive graphical notation for describing the behavior of systems [34]. While statecharts can be used to specify requirements, it is a general purpose notation that can be used equally well during design. In this respect, it differs from notations such as SCR (section 2.3) and RSML (section 2.4) that were explicitly developed for requirements modeling. However, the use of statecharts is so widespread that variations of it have been incorporated into most graphical modeling languages. For example, RSML is itself a derivative of statecharts.

In a statechart model, the relevant states of a system are depicted as ovals and transitions between states are depicted as arrows connecting the states. Transitions can be labeled with events and guards that enable the transition, and actions that can generate new events or change the value of local or global variables. States can be hierarchically decomposed into more detailed statecharts, and concurrency can be described through the use of multiple state machines.

While the graphical notation of the statecharts model is powerful and intuitively appealing, its formal semantics can be quite complicated and tend to vary from tool to tool. The complexity of the semantics can have a significant impact both on how easily a model can be understood and on how much automated support can be used in validating and verifying the model. Included here is a brief summary of the semantics of two well known statechart tools, Telelogic StateMate® and Rhapsody®. These will be contrasted later with the semantics of other requirements modeling tools, such as SCR and RSML.

StateMate was one of the first commercially supported statecharts tool [35, 36, and 37]. An atomic step in StateMate is divided into a sequence of substeps. Computation begins when one or more external events are received from the environment. These events may enable transitions in the model to "fire," causing the state machines to change state and possibly generate additional events. Any generated events are queued up until all enabled transitions have "fired." Once all enabled transitions have fired, a substep is complete. If there are any queued transitions, the process repeats itself until a substep completes with no queued events. At this point, the step is complete and the system is ready to process another event from the external environment.

StateMate uses a priority mechanism in case multiple, conflicting transitions (i.e., transitions that start from the same initial state but lead to different final states) are enabled. StateMate evaluates from the outside in; that is, transitions in the outermost state machines are given priority over transitions in their children. However, for multiple conflicting transitions at the same level of the state hierarchy, the semantics of StateMate are explicitly nondeterministic. That is, there are situations where multiple transitions may be enabled leading to different final states. In this case, StateMate simply picks one of the transitions and fires it.

Rhapsody is a UML tool from Telelogic, the same company that makes StateMate. Rhapsody uses statecharts to specify the behavior of individual objects [38]. Like StateMate, Rhapsody works from the outside in, taking the first transition that is enabled starting from the outermost active state. Surprisingly, the semantics of object and event interactions is underdefined: the ordering of transitions within a statechart is implementation dependent. This adds an element of nondeterminism into specifications, as the same statechart specification can be interpreted in multiple ways by different implementations. Also, there are two different semantics for event broadcast: one that is treated as a function call to each receiving machines, and another that adds the event to a queue for each of the receiving machines. In the event of queuing, the response of the system depends on the threading model of the state machine objects, adding another source of nondeterminism to the statechart specification.

2.3 SOFTWARE COST REDUCTION AND THE FOUR-VARIABLE MODEL.

One of the earliest approaches for modeling requirements was the SCR methodology developed by Parnas and Madey [11] to specify the requirements for the U.S. Navy's A7E Corsair II aircraft [10]. Unlike statecharts, SCR was explicitly designed for the specification of software requirements. The SCR notation was built around a conceptual model called the four-variable model, shown in figure 1.

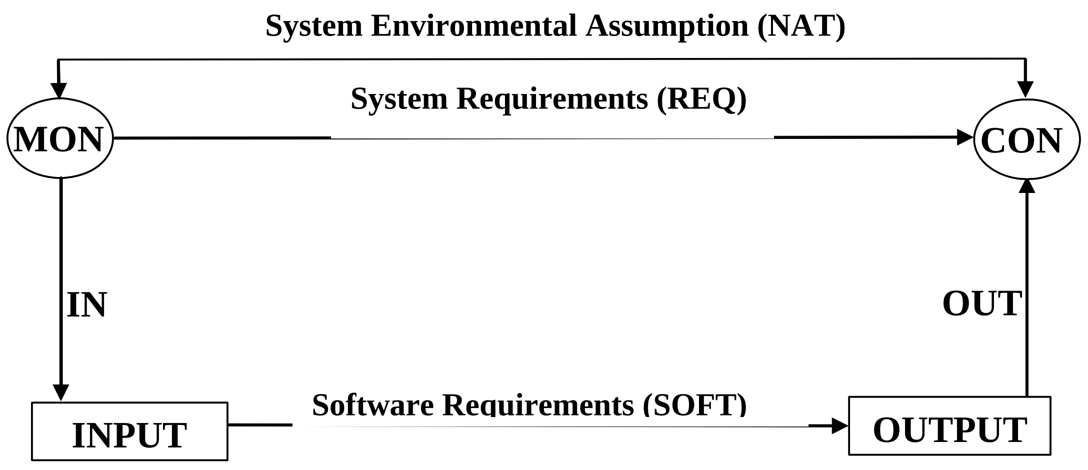
System Environmental Assumption (NAT)
System Requirements (REQ)
MON
CON
IN
OUT
INPUT
Software Requirements (SOFT)
OUTPUT

Figure 1. The Four-Variable Model

The variables in this model are continuous functions of time and include:

- Monitored variables (MON), which are quantities in the environment that the system monitors and responds to;
- Controlled variables (CON), which are quantities in the environment that the system will control;
- Input variables (INPUT), which are quantities in the software program that are provided by the system, and through which the software senses the MON, and
- Output variables (OUTPUT), which are quantities in the software program that are provided to the system, and through which the software changes the CON.

For example, monitored values might be the actual altitude of an aircraft and its airspeed, while controlled variables might be the position of a control surface, such as an aileron, or the displayed value of the altitude on the primary flight display. The corresponding input and output values would be Aeronautical Radio, Incorporated (ARINC)-429 bus words or discrete values that the software reads, or writes, to sense these quantities.

To complete the specification, five mathematical relations are defined between the variables:

- NAT defines the natural system constraints, or environmental assumptions, imposed by the environment, such as the maximum rate of climb of an aircraft;
- REQ defines the system requirements that specify how the controlled variables (CON) will respond to changes in the monitored variables (MON);
- IN defines the map between the input variables (INPUT) and the monitored variables (MON);
- OUT defines the map between the output variables (OUTPUT) and the controlled variables (CON); and
- SOFT defines the bounds of the true software requirements without specifying the software design.

NAT and REQ together define the constraints imposed on the software by the environment and the system requirements. The hardware interfaces surrounding the software are modeled by the IN and OUT relations that define how the input and output are related to the monitored and controlled environmental variables. Specification of the NAT, REQ, IN, and OUT relations implicitly bounds the allowed behavior of the software, shown in figure 1 as SOFT, without specifying its design.

Of course, for an actual system, there may be hundreds of such variables and the relationships between them will be very complex. In one application of this approach, the specification of the requirements for the C-130J avionics, there were over 1600 monitored and controlled variables [15]. Effectively organizing such a large specification can be a daunting task. To help improve readability, SCR makes heavy use of tables in defining REQ, NAT, IN, and OUT that could easily be checked with automated tools for completeness and consistency. To help organize the specification, SCR also allows the user to define internal variables, called terms, and system modes and transitions between the modes similar to statecharts. However, the use of mode transition diagrams in SCR is intended to capture major system states and is generally much simpler than that found in statecharts. SCR is also a fully synchronous language [39] in that the entire model changes its state in a single, atomic step. SCR assumes that only one input value can change in any step. While this helps to reduce the complexity of the specification, it leaves to the implementer the responsibility for ensuring that the one input assumption holds in the actual product. The SCR notation ensures that a given specification is complete (a valid next state and outputs are defined for every current state and inputs) and deterministic (only one next state and outputs are defined for each current state and inputs).

There are several fundamental concepts embedded in the four-variable model of SCR that are highly relevant to requirements engineering.

- The system requirements define relationships between quantities that exist in the environment external to the system under construction, i.e., the monitored and controlled variables.

- Identification of the system boundary is an essential step in requirements definition. In SCR, the system boundary is explicitly defined by the monitored and controlled variables.

- The system requirements should not only define what the system is to do by defining how the controlled variables will change in response to changes to the monitored variables (REQ), but also by the assumptions that the system makes about its environment (NAT). Failure to document such environmental assumptions has played a significant role in the failure of several systems [5 and 8].

- The relationship of the software requirements (SOFT) to the system requirements and assumptions (REQ and NAT) is clearly depicted by defining the IN and OUT relationships that map the INPUT and OUTPUT variables to the monitored and controlled variables.

There are variations of the four-variable model that can also be useful. For example, figure 1 is laid out as shown to emphasize that the INPUT and OUTPUT variables are at a lower level of abstraction than are the MON and CON variables, with the IN and OUT relationships mapping between these levels of abstraction. It can be helpful to position the IN and OUT relations into layers implementing different levels of abstraction. Another variation is to glue the controlled variables of one or more models to the monitored variables of another model to create a larger system specification, or to split a large model into several smaller models.

A weakness of the four-variable model is that it does not explicitly specify the software requirements, SOFT, but rather bounds it by specifying NAT, REQ, IN, and OUT. This leaves the software developer with the practical problem of how to structure the software and relate it to the system requirements. To address this, Miller and Tribble [40] proposed an extension to the four-variable model that extends the SOFT relationship into the relations IN', REQ', and OUT', as shown in figure 2.

Here, IN' and OUT' are nothing more than the specification of hardware drivers to be implemented in software. In addition to isolating the software from changes in the hardware, they also serve to recreate virtual versions of the monitored and controlled variables defined in the subsystem specification in the software.

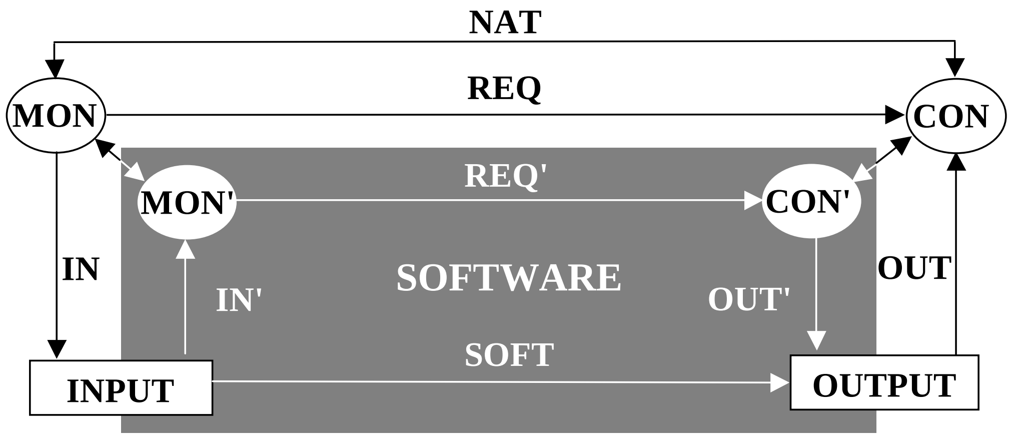

Figure 2. The Extended Four-Variable Model

The primary advantage of this model is that it makes the relationship between the system requirements and the software requirements straightforward. Each function defined in the system requirement REQ maps directly into an identical function in the software requirement REQ'. In similar fashion, IN' and OUT' map directly to the hardware specification.

It is important to note that MON' and CON' are not the same as the system level variables represented by MON and CON. Small differences in value are introduced both by the hardware and software, and differences in timing are introduced when sensing and setting the input and output variables. For example, the value of an aircraft's altitude recreated in software is always going to lag behind and differ somewhat from the aircraft's true altitude. In safety-critical applications, the existence of these differences must be considered. However, if they are well within the tolerances of the system, they can be treated as perturbations. Figure 2 provides an intuitive model relating the system requirements to the software requirements.

SCR has continued to evolve over the past two decades. The Software Productivity Consortium used SCR as the basis for the CoRE methodology [14-16]. CoRE extended the basic four-variable model to include object-oriented concepts to make the specification robust in the face of change and to support the development of product families. To specify real-time requirements, CoRE extended the basic four-variable model with constraints, such as the minimum and maximum time that a controlled variable must be updated after a monitored variable changes, or the periodicity that a controlled variable must be updated. These timing constraints can be specified as arbitrarily complex functions of the system modes, state, and inputs, allowing great flexibility in specifying real-time behavior. In the same manner, CoRE also allowed tolerances (allowed deviations from the ideal behavior) to be specified for each controlled variable to make it easier to define a range of acceptable behaviors. As mentioned earlier, CoRE was used to specify the software requirements for the C-130J aircraft [15]. A complete example of a CoRE specification for the mode logic of a flight guidance system can be obtained from reference 41.

The Naval Research Lab has also developed a number of tools for the specification and analysis of SCR models [12 and 13]. Experiences using an early version of the SCR tools can be found in reference 42.

2.4 THE RSML AND THE RSML^{-e}.

RSML is a requirements modeling language similar to SCR produced by Nancy Leveson's group at the University of California at Irving for specifying the behavior of process control systems [17]. One main design goal of RSML was readability and understandability by noncomputer professionals such as end users, engineers in the application domain, managers, and representatives from regulatory agencies. RSML was used to specify the Traffic Alert and Collision Avoidance System (TCAS) II, and this specification was ultimately adopted by the FAA as the official specification for TCAS II [18].

RSML is based on a conceptual model very similar to that of SCR. An RSML specification consists of a collection of input variables, state variables, input and output interfaces, functions, macros, and constants. Input variables are used to record the values observed in the environment. State variables are organized in a hierarchical fashion and are used to model various states of the control model. Interfaces act as communication gateways to the external environment. Functions and macros encapsulate computations, providing increased readability and ease of use.

RSML was heavily influenced by both SCR and statecharts. Like SCR, RSML makes extensive use of tables that can be automatically checked for completeness and consistency. An important contribution of RSML was the development of AND/OR tables that presented complex predicates in a form that could easily be read by domain experts, pilots, and regulators. RSML also made greater use of statecharts, to the extent that an RSML specification often looks very much like a statecharts model. The RSML tools automatically maintain traceability of all model elements that reference or are referenced by a model element. This makes the impact of change to a model element much easier to determine. In addition to simple checks of completeness and consistency, the feasibility of automating more complex safety analyses, such as fault tree analysis and deviation analysis, was also explored using RSML [43].

In process control systems, accidents can be caused by dependence on inputs describing environments that are either stale or wrong. For this reason, RSML variables can be assigned the value UNDEFINED on system start-up or when their value is in doubt. This encourages the specifier to consider cases where the system's information about its environment is obsolete. An RSML model also satisfies completeness criteria that ensure that safety information is captured about inputs, outputs, modes, and state variables. These include specification of the variables' expected range, granularity, units, load (anticipated frequency of update or output), exception handling information, and traceability information.

Like statecharts, RSML uses a notion of explicit event propagation in which events are queued for processing by the receiving state machines. In the course of developing the TCAS-II specification and its independent verification and validation (V&V) experiment, it became clear that the most common source of errors was this dependence on explicit events [44]. To reduce this problem, the Critical Systems group at the University of Minnesota developed the Requirements State Machine Language without Events (RSML^{-e}) [45]. As its name implies, RSML^{-e} eliminates the use of explicit events and is a fully formal synchronous language, much like SCR. The full specification of the mode logic of a Flight Guidance System (FGS) written in RSML^{-e} can be obtained in reference 46.

The elimination of explicit events and the transition to a fully synchronous language (i.e., one that changes its state in a single, atomic step [39]) made RSML^{-e} well suited for formal verification techniques such as model checking and theorem proving. As part of National Aeronautics and Space Administration (NASA) Langley's Aviation Safety and Security Program, translators were developed from the RSML^{-e} language to the NuSMV model checker [47] and the PVS theorem proving system [48], and numerous properties were mathematically proven to hold about a model of the RSML^{-e} specification of the FGS mode logic [49-52]. While it was highly feasible to prove properties about the RSML^{-e} specification of the mode logic, allowing variables to take on the value UNDEFINED made the formal verification considerably more difficult [53]. A better approach might be to assign a health status to each variable that has to be interrogated in the specification before it is used.

In the course of conducting these exercises, it became clear that two distinct styles of requirements specification had been developed to describe the FGS mode logic. The RSML^{-e} model provided a very detailed, constructive specification of the system requirements that was assembled from base types in the language (Booleans, enumerated types, integers, reals) into more complex structures using language constructs such as records and transitions. For this reason, the model was strongly biased towards a particular implementation, as evidenced by the fact that source code could automatically be generated from it. Like SCR, the syntax of the RSML^{-e} language ensured that the resulting specification was consistent (i.e., deterministic in that it defined a single next state and outputs for each current state and inputs) and complete (in that it defined a next state and outputs for every current state and inputs).

In contrast, the formal properties provided a more abstract, property-oriented or axiomatic, statement of the requirements in that they only specified relationships between the system inputs and outputs of the FGS mode logic without providing any details of the systems internal structure. In effect, they were more analogous to the traditional approach of specifying requirements as shall statements in a natural language, although with the advantage of being formally specified in a mathematical logic. This style of requirements specification was free of implementation bias, but it was not clear that the properties were consistent (did not contradict each other) and complete (there were enough properties provide to fully specify the required behavior). These observations are described more fully in reference 53.

2.5 SPECIFICATION TOOLKIT AND REQUIREMENTS METHODOLOGY AND INTENT SPECIFICATIONS.

SpecTRM is a methodology and toolset for developing safety-critical systems developed by Safeware Engineering Corporation [7-9, 19, and 20]. The Black Box behavior of the system is captured in the SpecTRM Requirements Language (SpecTRM-RL), which is very similar to RSML. A SpecTRM-RL specification systematically captures the information needed to ensure the requirements are complete, in that all possible combinations of state and inputs are handled, and consistent, in that each possible combination of state and inputs is handled in only one way. It also makes extensive use of tables and state transition diagrams that can be checked individually for consistency and completeness. Input and state variables can be declared obsolete, just as RSML variables can be declared UNDEFINED.

As with RSML, a SpecTRM-RL model satisfies completeness criteria that ensure safety information is captured about inputs, outputs, state information, and modes. Examples of the information collected include acceptable values, units, granularity, initiation delay, completion delay, load, hazardous timing behavior, exception handling, and feedback information that can be used to check that outputs are correctly affecting the environment [5].

However, a SpecTRM-RL model is embedded within a larger intent specification that "provides a seamless flow of rationale and reasoning from the highest level goals of the system, down through the SpecTRM-RL model all the way down to the implementation and operator training materials" [7-9, 19, and 20]. This provides essential information as new members are brought onto the engineering team and helps to prevent important decisions from being undone during development and maintenance. This can also be extremely helpful when distributed design teams work on a single project.

To support this, a SpecTRM specification is organized along both a part-whole dimension and an intent dimension. The part-whole dimension provides a fairly traditional view of the specification, breaking it down into parts so that the user can focus on detailed views within each level of the model. The part-whole dimension includes explicit sections for information about the environment and the human operators, two areas that are often overlooked in traditional requirements specification. The intent dimension organizes the specification into levels, each providing the intent (i.e., the why) information about the level below, so that the rationale for the system can be specified in increasing levels of detail.

The first level of the intent dimension (System Purpose) contains information useful to system engineers in reasoning about system-level goals, constraints, priorities, and tradeoffs. These include the safety constraints, which are related back to the system safety analysis. The second level (System Design Principles) contains information about the system in terms of physical principles and laws upon which the design is based. The third level (Black Box Behavior) is similar to the RSML specifications discussed earlier and supports reasoning about the system without dealing with implementation levels. The fourth level (Design Representation) provides information about the design of individual components, and the fifth level (Physical Representation) contains implementation detail such as code. Environmental assumptions similar to those discussed in CoRE are captured at all levels of the specification.

Numerous links are maintained in an intent specification between the levels in all dimensions, allowing the reader to easily navigate between the parts of the specification and between the goals, constraints, and design assumptions.

2.6 UNIFIED MODELING LANGUAGE.

UML [21 and 22] is a graphical design language used to specify, visualize, construct, and document the artifacts of software-intensive systems developed using Object-Oriented Development (OOD). The Object Management Group (OMG) adopted this technology in November 1997 as UML 1.1. The current release of UML is 2.0. The Systems Modeling Language (SysML) customizes UML 2.0 to support the specification, analysis, design, verification, and validation of complex systems. SysML reuses a subset of UML 2.0 diagrams and adds additional modeling constructs for systems engineering. For requirements engineering management, the significant SysML contributions include parametric models, time properties, requirements capture, and requirements traceability throughout the model. The UML diagrams used to capture, analyze, and track requirements information are: use case, activity, statechart, sequence, and structure.

The UML notation most often associated with requirements is the use case [23]. A use case describes sequences of actions a system performs that yield an observable result of value to a particular actor or user [24]. In other words, a use case describes a system's desired behavior by telling a story at one level of detail from the user's point of view [25]. The name of the use case is phrased as a goal to be achieved. The user's point of view is represented by the actor. The relationship between the actor and the use case is one of communication or interaction. The use case describes this interaction through the use of free form text, sequence diagrams, activity diagrams, or requirements. The free form text of the use case can include general comments, requirements, constraints, scenarios, and attributes. It often contains a pre- and post-condition and a textual story describing the use case.

Use cases can describe the functionality or tell the story as a sunny-day flow, as an alternate path flow, or as a misuse case flow [24]. The sunny-day flow describes the most common path traveled by the system or subsystem. This flow, or scenario, is a use case that describes a sequence of actions fulfilling a need or goal in the system. It includes what starts the use case, what ends the use case, and the steps and repeated steps of the flow.

The alternate path identifies different ways the system can be used beyond the typical uses. These include optional situations, odd cases, and variants. The alternate paths are viewed as a set of scenarios that branch from the sunny-day scenario. The alternate paths typically elicit requirements needed for system robustness.

The misuse cases identify ways the system can be abused or broken. The user is viewed as a hostile agent to the system or subsystem. This flow, or scenario, is a use case that describes a need or goal that is a threat to the system and should not occur. The misuse case scenario is described by the hostile agent. The misuse case elicits security and safety requirements and additional exceptions.

An attractive hierarchical, UML-based approach to specifying requirements for complex systems has been developed by Richard Stevens and his colleagues [54]. Their approach, illustrated in figure 3, uses use cases to specify the system functions and requirements in both a Black Box, user-oriented fashion and a white-box, system-oriented fashion. They divide the system development life cycle into the User Requirements (UR), Systems Requirements (SR), Architecture Design (AD), and System Test (ST) phases. These phases move along the horizontal axis from a sketchy description on the left to a more detailed analytic description on the right.

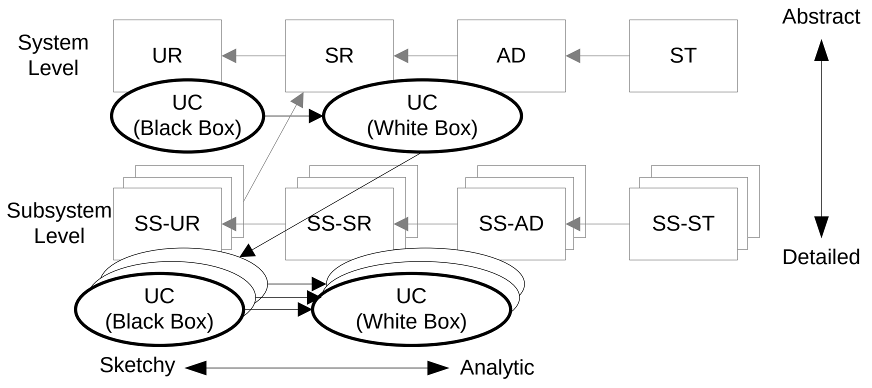

Figure 3. Hierarchy of Systems Engineering

The User Requirements are part of the Black Box, user-oriented use cases. Sequence diagrams are used to specify the interactions between the user and the system, and statecharts and activity diagrams are used to specify the Black Box user-observable behaviors. User Requirements are linked to or associated with the use case, the sequence diagrams, and the statecharts.

The System Requirements and the Architecture Design are associated with white-box, system-oriented use cases. These use cases often contain sequence diagrams to specify the communication needs between the subsystems. Activity diagrams are used to specify concurrency and sequences of interactions between subsystems. Sequence diagrams describe the communication and constraints between the subsystems. Statecharts are used to specify the Black Box behaviors of each of the subsystem. System Requirements are linked to or associated with the use case, the sequence diagrams, the activity diagrams, and the statecharts. The User Requirements and the System Requirements are then used to develop the System Tests.

This pattern is repeated recursively along the vertical axis for each subsystem. At the top level, the full system is described. Moving down to the subsystems, the requirements, architectures, and tests become more detailed. The white-box system level use cases are refined as Subsystem User Requirements (SS-UR) and Black Box use cases. At the subsystem level, the scope of detail narrows, the interfaces and communications become more important, and the requirements for end user interactions diminish. The subsystem requirements development is repeated using the same UML diagrams as in the specification of the systems requirements. Complex systems may require many levels of decomposition.

When applied to requirements, the UML diagrams and methodologies are frequently discussed in the context of eliciting or discovering the requirements for a system. This is in contrast to the methodologies of SCR, RSML, RSML^{-e}, and SpecTRM, which focus more on precisely documenting the behavior of the system. Conversely, the notations of UML do not appear to provide the same emphasis on completeness, consistency, and mathematical rigor as these approaches.

2.7 SYNCHRONOUS GRAPHICAL MODELING LANGUAGES.

Over the last decade, synchronous, graphical modeling languages have been increasingly used in the development of safety-critical systems, particularly for avionics and automotive systems, often referred to as Model Based Development (MBD). Two prominent examples include Simulink [26 and 27] and SCADE™ [28]. While originally developed for system and software design, these tools are increasingly being used for the specification of requirements. However, there does not appear to be widespread consensus of whether such models represent system requirements, high-level software requirements, low-level software requirements, or whether they represent requirements at all (see section 3.2.3.4). A brief description of these tools is included here, followed by a discussion of their use for requirements specification.

Simulink [26 and 27], sold by The Mathworks, Inc. is a popular platform for the modeling and simulation of dynamic systems. It provides an interactive graphical environment and a customizable set of block libraries that can be used to design, simulate, debug, implement, and test reactive systems. Users assemble a system specification by dragging and dropping blocks onto a pallet and connecting the outputs of one block to the inputs of another block. Blocks can be composed hierarchically from simpler blocks, allowing designers to organize complex system designs. New blocks can be defined by the developer and added to a reusable library. Blocks can also be parameterized. Control logic for representing system states and state transitions can be modeled with the integrated StateFlow® add-on. Simulink and StateFlow are both integrated with the MATLAB® environment, also marketed by The Mathworks, Inc., providing access to several additional tools for algorithm development, data analysis, data visualization, and numerical computation. Executable code can generated from a Simulink model using the Real-Time Workshop® add-on. An advantage of Simulink is that it can be simulated with fixed or variable-step solvers, allowing both the control system and the plant model (for example, the airframe) to be modeled within the same framework.

SCADE [28] is an environment for the development of safety-critical systems similar to Simulink. Originally developed for the design of aircraft systems, similar but separate versions are now marketed by Esterel Technologies for the automotive industry (SCADE Drive™) and the avionics industry (SCADE Suite). SCADE also provides an interactive graphical environment that allows users to assemble system specifications by dragging and dropping blocks onto a pallet and connecting the outputs of one block to the inputs of another. Control logic for representing system states and state transitions can be modeled with the integrated Safe State Machine© (SSM) add-on. Since the SCADE tools were explicitly created for the development safety-critical software and hardware, SCADE supports only fixed step simulation. For the same reason, the features and blocks supported by SCADE and SSM are restricted to those with an unambiguous mathematical representation. An advantage of SCADE is that its models are translated into the Lustre language, a synchronous data flow language with a precise formal semantics. C source code can be generated from SCADE using the KCG™ code generator, which has been qualified as a Level A software development tool in accordance with

DO-178B. The SCADE Suite also includes a gateway that can import Simulink models and a model checker called Design Verifier.

The use of modeling languages, such as Simulink and SCADE, in the development of safety-critical systems appears to be part of a long-term trend. Simulink is now widely used for the development of automotive systems in the United States and is increasingly being applied to avionics system. SCADE has been used extensively over the last decade for the development of avionics systems in Europe and is seeing increasing use in Europe for automotive systems. This popularity indicates that system developers see genuine value in their use.

However, unlike SCR [10-13], CoRE [14-16], RSML [17 and 18], and SpecTRM [7-9, 19, and 20], which were designed explicitly for the specification of requirements, these notations were developed originally for system and software design. As a result, they lack facilities for specifying environmental assumptions as do SCR, CoRE, and RSML, capturing intent, as in SpecTRM, or documenting human factors requirements, as in SpecTRM. They also do not provide support for notations for eliciting requirements, such as use cases, as UML does.

On the other hand, a model written in SCR, CoRE, RSML, or SpecTRM often contains much of the same information found in a Simulink or a SCADE model. Whether a Simulink or a SCADE model is a detailed specification of requirements or a detailed design seems to be largely a matter of opinion. The most common objections to the use of these tools for requirements specification are that they contain significant design and that they obscure the truly important requirements in a sea of detail. However, DO-178B explicitly states that low-level requirements may be derived from design constraints in addition to high-level requirements, and Simulink and SCADE models are frequently used as specifications of the low-level requirements.

As shown in the survey results regarding MBD (sections 3.2.3.3, 3.2.3.4, 4.4, and appendix F.3), identifying when and how these models can best be used to specify requirements appears to be an area for further work.

2.8 INTEGRATED MODULAR AVIONICS.

Traditionally, digital flight control systems have been implemented as federated architectures in which each major function resides on its own computing platform, loosely connected to other platforms by buses and other electrical paths. Such architectures provide a very high level of fault containment so that faults in one function are unlikely to propagate to another function on a different platform. More recently, the trend has been towards IMA architectures in which several functions are implemented on a single, fault-tolerant, computing platform [55 and 56]. Such systems are attractive in that they offer the promise of reducing space, power, weight, and other costs. However, the higher physical integration and distribution of system functions across organizational boundaries in IMA heighten concern about how their requirements are specified and managed.

An IMA platform is a complicated system that must satisfy numerous requirements. Chief among these is the requirement for partitioning. Partitioning prevents faults arising in one aircraft function from propagating to other functions executing on the same platform, providing in the IMA platform the protection provided through physical separation in a federated system. This is particularly important in isolating critical functions from lower criticality functions. The essential requirement identified by Rushby [57] for an IMA system is that "the behavior and performance of software in one partition must be unaffected by the software in other partitions," aside from the effects of intended communications between the functions in each partition. This can be broken down further into requirements for spatial and temporal separation. Spatial partitioning ensures that software in one partition cannot change the software or private data of another partition (either in memory or in transit), nor command the private devices or actuators of other partitions. Temporal partitioning ensures that the service received from shared resources by the software in one partition cannot be affected by the software in another partition, including the rate, latency, jitter, and duration of scheduled access to the shared resource.

An IMA platform must also satisfy stringent requirements for fault tolerance and health monitoring. Since so many functions are implemented on an IMA platform, the availability and reliability requirements for these functions often dictate that the platform be physically distributed and redundant, as well as implemented to a high level of criticality. IMA platforms also typically provide services for monitoring, managing, and reporting faults. These basic requirements on the IMA system itself can be implemented in a variety of different mechanisms, each of which introduces numerous issues for design of the IMA system. For a more detailed discussion, the reader is directed to references 55-57.

However, meeting the platform requirements does not ensure the safety needs of the system being implemented are met. One promise of IMA is that applications can be purchased, developed incrementally, or reused from other programs, where individual applications and the IMA platform may be developed by different companies. The overall functions for the aircraft are accomplished through the combined interactions of these application functions. The overall safety of the aircraft is an emergent property of the system as a whole and cannot be ensured by analyzing the applications in isolation. However, the physical distribution of the developers (not to mention their corporate and legal separation) demands a clear means of allocating the overall safety requirements among them in such a way that the overall safety requirements can be demonstrated to be met. Another way of stating this is to ask how requirements can be stated to facilitate incremental acceptance for certification.

One step toward achieving this is to provide a complete, precise, and unambiguous specification of the requirements to each product team. Several best practices discussed here, such as SCR, RSML, and RSML^{-e}, do this. However, with regard to safety requirements (and other requirements), it is important to convey the rationale, or the why, for the requirements of a particular application. This helps the application developer understand the emergent safety requirements at the system level and how they apply to their application. The intent specifications of SpecTRM address this by providing a format to provide all design teams with the system design decisions and rationale relevant to their application [9].

Another proposed approach is the use of safety assurance contracts [58]. A safety assurance contract identifies the key safety requirements between applications or between the application and the platform. These safety requirements are driven by the safety analysis and address failures within the platform, failures within an application, and failures in other applications.

Each safety contract consists of three parts, (1) the potential failure as identified in the safety analysis, (2) the supplier's guarantee, and (3) the client's reliance on the guarantee. The safety analysis identifies a failure, the supplier guarantees protection, and the client assesses whether these guarantees have an impact.

As an example, a failure occurs when an application gains direct access to another application's address space. The supplier guarantee is that the platform assigns an address space for each application and shuts down the application if the application accesses memory outside of its assigned space. The developer of the client application assesses this guarantee and does an analysis to ensure that memory read-and-writes the application performs are within the address space. If the developer cannot ensure that this requirement is met, then the client application may not be able to meet its safety guarantees. This may result in a redesign of the client application, renegotiation with the platform developer, or failure to meet the overall safety property.

Specifying requirements that facilitate incremental acceptance for certification and help developers ensure that the emergent safety properties of the overall system are satisfied appears to be one key issue for requirements engineering in IMA.

3. INDUSTRY SURVEY.

To assess how requirements are currently managed, a survey of industry, regulatory, and research organizations was conducted. The survey approach, findings, and conclusions are described in this section.

3.1 APPROACH.

An initial set of questions for the survey was developed by the team members. These consisted of a set of multiple-choice questions designed to assess the background of the respondents, a set of multiple-choice questions designed to determine what tools and methods were in common use, and a set of free-text questions designed to identify safety and certification issues relative to requirements engineering management. This initial survey was sent to selected individuals in industry, academia, and the FAA for their review. The survey was then revised per their comments to produce the final survey form shown in appendix B.

The finalized survey form was distributed on February 14, 2005, via the FAA software mailing list, the Aerospace Vehicle Systems Institute mailing list, and to individuals known to be working in the civil aviation industry. The survey was accompanied by the cover letter shown in appendix C. Recipients were invited to return their responses by March 10, 2005, either directly to Rockwell Collins or anonymously to the FAA.

Responses sent anonymously to the FAA had all identifying information removed by the FAA and were forwarded to Rockwell Collins. In addition, Rockwell Collins removed any identifying information from the responses that were sent directly to them, then collated the information from all the responses into the form presented in this report. Multiple-choice questions were tabulated as shown in appendix D and appendix E. The responses to free-text questions were collected and are shown in appendix F.

The responses to the free-text questions were carefully studied to identify the main safety and certification issues related to requirements engineering management. The responses and the issues were entered into a Telelogic DOORS database. Responses were linked to each relevant issue. This database was used to produce the list of Industry Survey Comments Organized by Issue in appendix G .

Finally, the list of safety and certification issues that the survey identified was supplemented with additional issues identified from the literature search described in section 2.

3.2 FINDINGS.

A total of 42 survey forms were returned. The findings from the survey are presented in the following sections, organized by background information, tools and methods, and comments from the respondents.

3.2.1 Background Information.

The purpose of the first section of the survey was to identify the types of projects and expertise represented by the survey respondents. The results of this part of the survey are summarized here. Additional details are presented in appendix D.

3.2.1.1 Regulatory Compliance.

An overwhelming majority of the respondents were familiar with DO-178B or ED-12B (refer to appendix D for a tabular list). Forty respondents stated that their current or most recent project was compliant with DO-178B, while 7 stated that their current or most recent project was compliant with ED-12B.

3.2.1.2 Project Size.

A variety of project sizes were sampled (refer to appendix D for a tabular list). Seventeen respondents stated that their current or most recent project consisted of 1 to 9 employees, 21 claimed 10 to 99 employees, and 3 stated that over 100 people worked on their current or most recent project.

3.2.1.3 Job Function.

A variety of job functions were represented (refer to appendix D for a tabular list). On their current or most recent project, 19 respondents selected product development/engineering as their job function, 14 selected regulatory (Designated Engineering Representative (DER), FAA liaison, etc.), 12 selected management, 8 selected process/tools support, 2 selected research, 1 selected marketing/sales, and 4 selected other (quality assurance, software quality assurance, and software quality assurance/DER).

3.2.1.4 Software Level.

DO-178B software levels A through E were also well represented (refer to appendix D for a tabular list). Twenty-nine of the respondents stated that level A applied to their current or most recent project, 16 stated that level B applied, 18 responded level C, 14 responded level D, and 8 responded level E. Note that more than one software level could apply to a project.

3.2.1.5 Company Roles.

Thirty-one respondents identified their company's role as a subsystem developer (refer to appendix D for a tabular list). Ten selected system integrator, and six selected airframe manufacturer. Ten selected the category of "other" for their company's role. The "other" category included FAA software development for radar systems, supplier, avionics manufacturer, software V&V, supplier of Technical Standard Order (TSO) avionics equipment, software developer, engine developer, reusable software development, and tool developer.

3.2.1.6 Primary Source of Requirements.

Twenty-six of the respondents stated they received their requirements from the airframe manufacturer, 20 said they received their requirements from the system integrator, and 9 specified the end customer (airlines, pilots, maintenance, etc.) (refer to appendix D for a tabular list). Additional sources of requirements included existing system requirements, the U.S. government, the FAA, company designed products, flight test pilots, advanced design specialist, internal systems group, original equipment manufacturers (OEM), industry Minimum Operational Performance Standards (MOPS), and subsystem suppliers.

3.2.2 Tools and Methods.

The purpose of the next section of the survey was to identify the types of tools and methods used to capture, manage, validate, and verify the requirements.

3.2.2.1 Requirements Format.

The first question of this section attempted to identify how requirements are being captured independent of specific tools. As shown in table 1, the majority of the respondents claimed that their requirements are captured as English text, tables, and diagrams. About one-fifth of the respondents claimed to be using executable models, such as MATLAB Simulink or Esterel Technologies SCADE Suite, to capture system and software requirements. UML notations, such as use cases, sequence diagrams, and state diagrams, are also being used to a lesser degree to capture both high- and low-level software requirements,. However, none of the respondents claimed to be using UML for system, data interconnect, or hardware requirements. A small number of respondents claimed to be using a proprietary database or Telelogic DOORS objects (distinct from use of the DOORS tool) to capture their requirements. A small number of respondents were using dataflow diagrams to capture system and software requirements.

Table 1. Survey Findings—Requirements Format

In what form are requirements captured on your current (or most recent) DO-178B/ED-12B compliant project?

Enter an "x" in every row/column cell that applies	System Requirements	Data Interconnect {ICD}	High-Level Software Requirements	Low-Level Software Requirements	Hardware Requirements
English Text or Shall Statements	39	27	36	32	29
Tables and Diagrams	31	30	30	19	18
UML Use Cases	1		2	4	
UML Sequence Diagrams			3	6	
UML State Diagrams			1	7	
Executable Models (e.g. Simulink, SCADE Suite, etc.)	7	1	8	8	1
Data Flow Diagrams (e.g. Yourdon)	4		6	9	
Other (Specify)-Proprietary Database, DOORS objects	1	4	2	2	1
Other (Specify)XML		1			
Operational models or prototypes	1	1			1
UML			1	1	

3.2.2.2 Tools for Requirements Capture.

The next survey question focused on identifying specific tools being used. As shown in table 2, the overwhelming majority of the respondents claimed to use the Telelogic DOORS tool or a word processor. A much smaller number of respondents claimed to use Requisite Pro®, MATLAB Simulink, Engenuity VAPS^{TM}, a generic database tool, or a generic spreadsheet. Somewhat surprisingly, very few UML tools were cited. It is also interesting to note that the most popular tools are used for capturing requirements at all levels.

Table 2. Survey Findings—Tools for Requirements Capture

What tools are used for requirements capture on your current (or most recent) DO-178B/ED-12B compliant project?

Enter an "x" in every row/column cell that applies	System Requirements	Data Interconnect {ICD}	High-Level Software Requirements	Low-Level Software Requirements	Hardware Requirements
Database (e.g., Microsoft Access)	3	4	3	3	
DOORS	23	13	22	18	12
Rational ROSE®			1	3	
RDD-100®					
Requisite Pro®	5	3	5	4	4
Rhapsody	1				
SCADE Suite	2		3	1	
Simulink	5	1	5	3	1
Slate	1		1	1	
Spreadsheet (e.g., Microsoft Excel)	5	4	5	4	3
Statemate					
Word Processor (e.g., Microsoft Word)	19	20	18	17	16
VAPS™		1	3	3	
Designer's Workbench™			1	1	
Proprietary Database, SCADE like pic tool		1	1		
Interleaf	1	1	1	1	1
BEACON	1	1	1	1	
CaliberRM	1	1	1	1	1
XM:		1			
Wiring diagram		1			1
Schematic capture	1				1
RTM	1		1	1	
Artisan			1	1	
Home-grown Autocoder			1		
Tau			1	1	

3.2.2.3 Tools for Requirements Traceability.

The next survey question (table 3) asked what tools were being used for traceability between different levels of requirements. Again, the most common answer was Telelogic DOORS. A generic word processor or spreadsheet was the next most frequent choice, followed by Requisite Pro® and generic database tools. As with requirements capture, the same tools seemed to be used equally often for systems, software, hardware, and interconnect requirements.

Table 3. Survey Findings—Tools for Requirements Traceability

What tools are used for requirements traceability/compliance on your current (or most recent) DO-178B/ED-12B compliant project?

Enter an "x" in every row/column cell that applies	System Requirements to High-Level Software Requirements	High-Level Software Requirements to Low-Level Software Requirements	Low-Level Software Requirements to Software Design	System Requirements to Hardware
Database (e.g., Microsoft Access)	4	3	3	2
DOORS	24	24	19	14
Rational ROSE				
Requisite Pro	5	4	5	4
Rhapsody				
Slate				
Spreadsheet (e.g., Microsoft Excel)	9	9	7	7
Word Processor (e.g., Microsoft Word)	10	7	7	8
Vbay DCM		1	1	1
Interleaf	1	1	1	1
Internally developed tool	1	1	1	1
RTM	1	1	1	1
Python Scripts		1	1	

3.2.2.4 Techniques for Requirements V&V.

The final multiple choice survey question asked what techniques were used to ensure that the requirements are accurate, consistent, and verifiable. As shown in table 4, the most common techniques were reviews and inspections, which were used equally often across all levels of requirements. Rapid prototyping and simulation were cited about one-fourth as often, but again fairly equally across all levels of requirements (a bit less often for hardware and data interconnect). This was closely followed by flight test and automated analysis. Aircraft integration and iron-bird testing were cited by only a few respondents.

Table 4. Survey Findings—Techniques for Requirements V&V

Enter an "x" in every row/column cell that applies	System Requirements	Data Interconnect {ICD}	High-Level Software Requirements	Low-Level Software Requirements	Hardware Requirements
Reviews/Inspections	34	30	33	32	24
Creation of Test Cases	23	17	34	33	20
Rapid Prototyping	9	5	12	10	7
Automated Analysis	6	4	5	6	5
Flight Test	15	6	6	5	5
Simulation	11	4	9	6	4
Aircraft Integration		1			
Other (Specify) Iron-bird Testing	1	1			1

3.2.3 Comments.

The remainder of the survey consisted of several questions to which the respondents were asked to provide free text answers. While not as straightforward to summarize, these responses proved invaluable in identifying the respondent's main safety and certification concerns. Most of these responses were carefully analyzed in section 4 on Safety and Certification Issues. A few highlights are presented here. A complete list of the responses is included in appendix F.

3.2.3.1 Ensuring Compatibility With the Target Computer.

The survey recipients were asked

> "What techniques are used to show that the high-level and low-level software requirements are compatible with the target computer on your current (or most recent) DO-17B/ED-12B compliant project?"

The responses to this question are provided in section F.1of appendix F. While summarizing these results is rather subjective, the majority of the responses generally fell into the categories of testing on the target platform, testing on an emulator, test case creation, prototyping, analysis, and reviews and inspections. An approximation of the number of times each technique was cited by a respondent is given in table 5.

Table 5. Techniques to Ensure Target Compatibility

Technique	Cited
Testing on the target platform	26
Testing on an emulator	4
Test case creation	3
Prototyping	4
Analysis	12
Reviews and inspections	18

3.2.3.2 Ensuring Conformance to Standards.

The survey recipients were asked

> "What techniques are used to show that the high-level and low-level software requirements conform to standards on your current (or most recent) DO-17B/ED-12B compliant project?"

The responses to this question are provided in section F.2 of appendix F. Again, summarizing the result is rather subjective, but the overwhelming majority of the responses cited reviews and inspections as their primary technique for ensuring conformance to standards. A few respondents distinguished between peer reviews and Software Quality Assurance (SQA) audits. A few respondents also cited the use of standard templates. Some respondents stated that the use of tools helped with enforcement of standards. An approximation of the number of times each technique was cited by a respondent is given in table 6.

Table 6. Techniques to Ensure Conformance to Standards

Technique	Cited
Reviews and inspections	34
SQA audit	3
Use of templates	2
Enforced by tools	3
Analysis	1
Testing	1

3.2.3.3 Issues Introduced by New Development Paradigms.

The survey recipients were asked

> "In your opinion, what new issues related to Requirements Engineering Management are introduced with new development paradigms/approaches such as:
>
> 1. Model-Based Development (the use of executable modeling tools such as Simulink, SCADE Suite, Statemate, etc., and their associated code generators.)
> 2. Integrated Modular Avionics
> 3. Object-Oriented Design
> 4. Other"

The responses related to MBD are provided in section F.3 of appendix F. Careful analysis of these responses helped to identify requirements engineering issues in MBD. The results of this analysis are presented in section 4.4. The greatest concerns appear to be those related to tool qualification, use, obsolescence, model validation, the relationship between requirements and design in MBD, and traceability.

The responses related to IMA are provided in section F.4 of appendix F. Careful analysis of these responses helped to identify requirements engineering issues in IMA. The results of this analysis are presented in section 4.5. The greatest concerns appear to be the risk of common mode and cascading failures, how application and platform services requirements should be specified, how to support incremental acceptance for certification, how to specify and meet system performance requirements, how to specify and meet dependencies between applications, and issues of traceability and configuration management.

The responses related to OOD are provided in section F.5 of appendix F. Careful analysis of these responses helped to identify requirements engineering issues in OOD. The results of this analysis are presented in section 4.6. The greatest concerns appear to be those related to traceability and the relationship between requirements and design.

The responses to the "other" question are provided in section F.6 of appendix F. These responses were also used in the determination of safety and certification issues discussed in section 4. Two issues were identified that had not been raised elsewhere. The first issue included the use of Engenuity VAPS^{TM} models for symbology and modeling of the human-machine interface. The second issue concerned closer integration between ARP 4754 and DO-178B and the use of MBD and OOD with shared services such as the software control library and software quality assurance. These are raised as specific issues in section 4.

3.2.3.4 What Level of Requirements do Models Represent?

The survey recipients were asked

> "In your opinion, what level(s) of requirements do the models represent in Model-Based Development? (Check all that apply)."

The responses of the recipients are shown in table 7. There does not appear to be a consensus on what level of requirements models can be used to represent.

Table 7. Requirements Represented by Models

Level of Requirements	Number of Respondents Agreeing
System requirements	20
Data interconnect	16
High-level software requirements	33
Low-level software requirements	25
Hardware requirements	9
The model does not represent requirements	7
Did not answer	4

3.2.3.5 What Are Your Favorite References on REM?

The survey recipients were asked

> "What are your favorite references on Requirements Engineering?"

The responses of the recipients are shown in section F.8 of appendix F. While several references were suggested, no reference other than the SEI Capability Maturity Model was cited repeatedly.

3.2.3.6 How is Existing Guidance Inadequate or Unclear?

The survey recipients were asked

> "With regard to Requirements Engineering Management, how is guidance provided by DO-178B/ED-12B, DO-254 and/or ARP 4754 inadequate or unclear?"

The responses of the recipients are shown in section F.9 of appendix F. These responses were also used in determining the safety and certification issues discussed in section 4, particularly those identified as general issues (section 4.1), safety issues (section 4.2), and certification issues (section 4.3). While four respondents stated that the current guidance was adequate, the majority of the respondents requested additional guidance. The most common comment was a request for clarification of the relationship of system requirements, high-level requirements, low-level requirements, and the software design process described in DO-178B. Several respondents asked for more guidance on issues related to traceability. Specifying the intent, or rationale, of requirements was also mentioned, though generally as a vague concern that the big picture was being lost in the detail. Three respondents asked what should be done differently with a safety requirement as opposed to other requirements. The reader is referred to section 4 for a more complete analysis of these responses.

3.2.3.7 Additional Comments.

In the final question of the survey, the recipients were asked

> "Please provide any additional comments on Requirements Engineering Management you feel are relevant to this survey."

The responses of the recipients are shown in section F.10 of appendix F. Although no specific trend was observed in these comments, they were also used in determining the safety and certification issues identified in section 4.

3.3 SURVEY CONCLUSIONS.

What do the results say about REM? First, the survey appears to represent respondents from a variety of backgrounds, including project size, criticality, job function, and company role. The one uniform characteristic of the respondents appears to be experience with DO-178B/ED-12B.

The majority of survey respondents capture requirements in English text, tables, and diagrams using either Telelogic DOORS or a word processor. Modeling tools, such as MATLAB Simulink and Esterel Technologies SCADE Suite, are starting to be used for the specification of requirements, though there does not seem to be a consensus on what level of requirements these models represent. Object-oriented approaches, such as UML, are also starting to be used, though to a lesser extent and primarily for software requirements. The most popular tools tend to be used for all levels of requirements (systems, data interconnect, high-level software, low-level software, and hardware).

Traceability between the requirements is managed primarily using Telelogic DOORS. To a lesser extent, generic word processors, spreadsheets, and databases are used for traceability, as is Requisite Pro®.

Requirements are checked primarily through reviews and inspections and the creation of test cases, and to a lesser extent, through rapid prototyping, simulation, flight test, and analysis.

Compatibility of the high-level and low-level software requirements with the target platform is shown primarily by testing, reviews and inspections, and analysis.

Conformance of the high- and low-level software requirements to standards is shown primarily by reviews and inspections.

The introduction of MBD, IMA, and OOD have raised several concerns related to REM. However, there appears to be an even more fundamental problem in that there are many questions regarding the relationship of system requirements, high-level software requirements, low-level software requirements, software design, and hardware requirements, as described in DO-178B.

Finally, the best practices identified in the literature search of section 2 appear to be used only rarely. Even the use of object-oriented approaches, probably the best known of these practices, do not appear to be widely used by the respondents. As a result, most of the benefits of these approaches, such as techniques for eliciting requirements, capturing the intent of requirements, documenting environmental assumptions, or defining a clear conceptual model of the relationship between system and software requirements, are probably not being used to their best advantage.

4. SAFETY AND CERTIFICATION CONCERNS.

This section discusses the safety and certification concerns that were identified in the survey and in the literature search. These are organized into categories of general, safety, certification, MBD issues, IMA, and OOD issues. The most useful sources for identifying issues were the comments provided by survey respondents. The comments were traced to major issues, and a list of the comments related to each major issue is provided in appendix G. In many cases, more detailed issues were identified within each major issue, and these are presented and discussed following the major issue. For example, two detailed issues for section 4.3.1 were recorded as (a) How should the boundary between the system and software be defined? and (b) How do system and software requirements differ? No attempt was made to trace the survey comments to the detailed issues, as many comments referred to several detailed issues. A few major issues were identified in the literature search, but they were not raised in the survey comments. These are included as opportunities for improvement. Several issues were raised in more than one category. For example, “What is the relationship of Software Requirements and Software Design?” was raised as a certification, MBD, and OOD issue. The chart shown in figure 4 combines these related issues for comparison. The issue “Clarification of DO-178B Terms and Guidance” has already included comments from all categories.

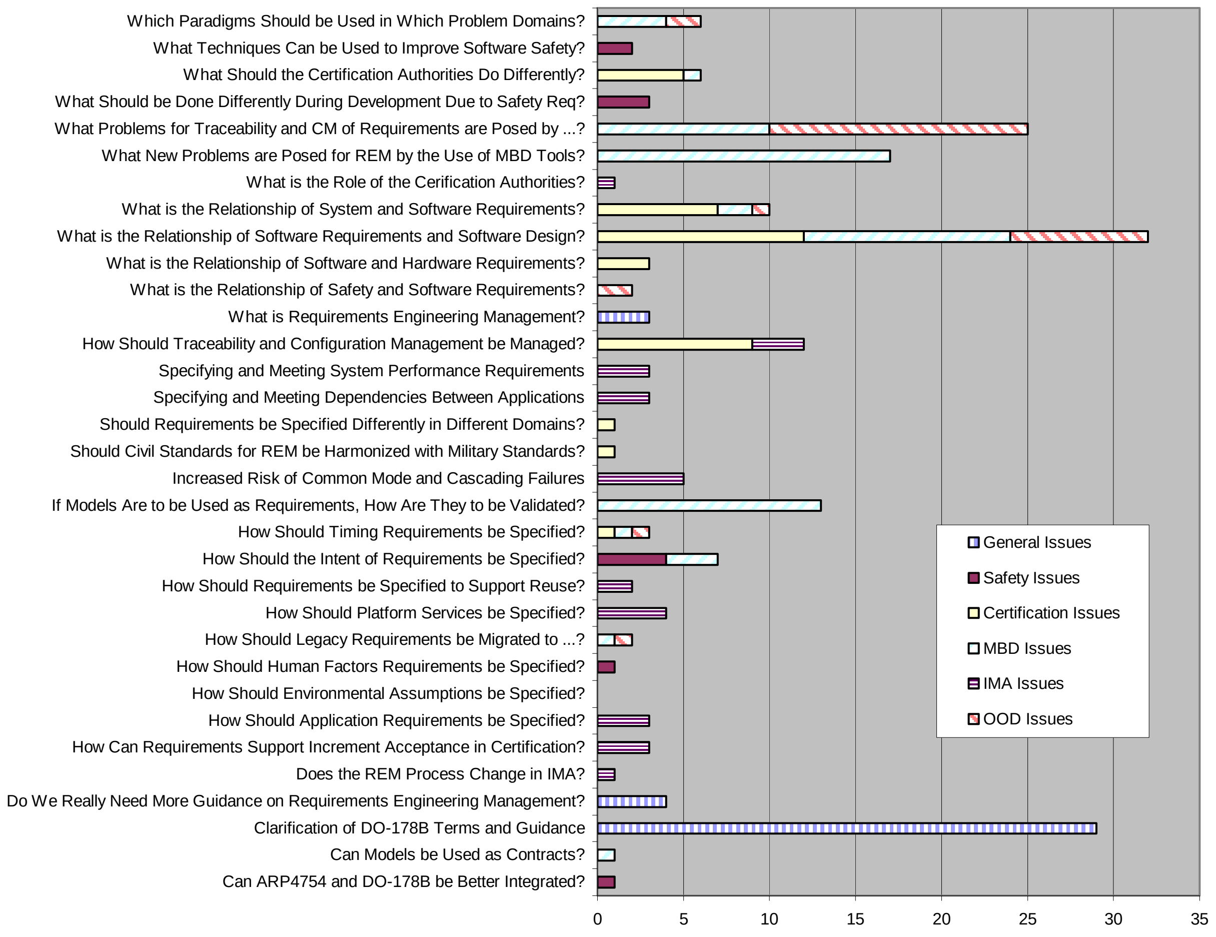

Figure 4. Survey Comments on Issues That Span Categories

4.1 GENERAL ISSUES.

A few issues were identified that span all the other categories of safety, certification, MBD, IMA, and OOD (for example, "Is more guidance on requirements engineering management really needed?"). These "general" issues are shown in figure 5, with the number of survey comments related to each. Each of these are discussed in the following sections.

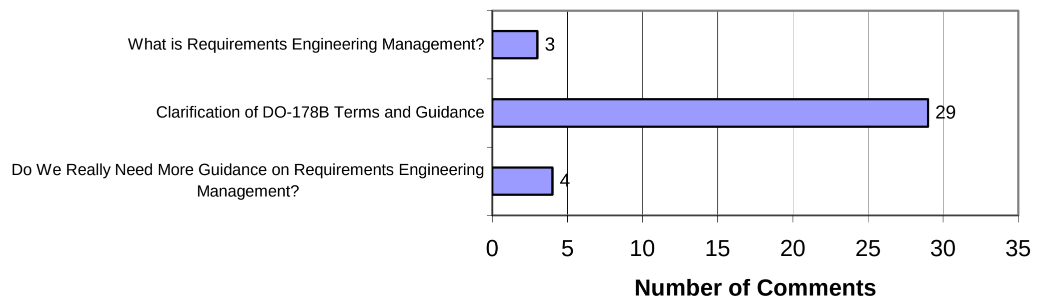

Figure 5. Survey Comments on General Issues

4.1.1 What is Requirements Engineering Management?

In the section of the survey asking what additional guidance would be helpful, one respondent asked for a definition of REM. This seems like an eminently reasonable request that deserves careful thought in the final handbook. Another respondent suggested topics that should be included in the handbook's scope and another asked for guidance on how to produce high-quality, testable requirements.

4.1.2 Clarification of DO-178B Terms and Guidance.

Reading through the survey comments, a concern that was raised repeatedly was for additional clarification of the guidance provided in DO-178B. Similar issues were raised in the comments relating to MBD and OOD that ultimately traced back to this issue. There appears to be a great deal of confusion on how to apply DO-178B in the area of REM. Although these concerns may be addressed by the RTCA SC-205/WG-71 committee currently revising DO-178B, it is important that the final handbook is compatible with DO-178C.

4.1.3 Is More Guidance Needed on Requirements Engineering Management?

Four respondents questioned whether more guidance was needed on REM. One identified the state of the art as one of chaos that seems to work never the less, two stated that the existing guidance is inadequate but acceptable, and the fourth felt that the existing guidance was adequate.

4.2 SAFETY ISSUES.

Comments strictly related to safety were less numerous than expected, although virtually all survey comments have safety implications. Several safety issues that were identified in the literature search (discussed in section 2) were added to the issues identified from the survey. Figure 6 shows the number of survey comments relating to each major safety issue. The greatest concerns appear to be for better guidance on what should be done differently for safety requirements (as opposed to other requirements) and for better integration between ARP 4754 [32] and DO-178B [29]. Important issues identified in the literature search that were not raised by the respondents included how to specify the "intent" of requirements, environmental assumptions, and how to specify human factors requirements. These issues are discussed in the following sections.

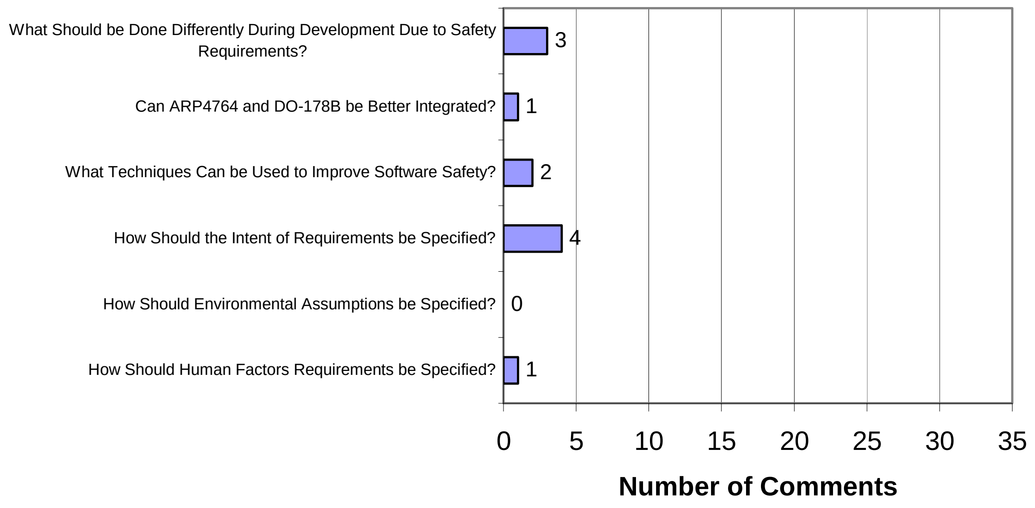

Figure 6. Survey Comments on Safety Issues

4.2.1 What Should be Done Differently During Development Due to Safety Requirements?

Three respondents asked for guidance on what should be done differently to verify and test safety requirements as opposed to other requirements.

4.2.2 Can ARP 4754 and DO-178B be Integrated Better?

One respondent stated that the "hand-off" from ARP 4754 to DO-178B could be improved.

4.2.3 What Techniques can be Used to Improve Software Safety?

One respondent requested guidance on specific methods that can be used to improve software safety, for example, implementing parity checking in software when it is not provided in hardware. A second respondent made a similar comment with respect to programmable logic devices (PLD).

4.2.4 How Should the Intent of Requirements be Specified?

Two respondents indirectly addressed the issue of capturing the intent of requirements. One respondent noted that "None of this tells you if … was the right thing to do in the first place," and another referred to the "role of commentary in clarifying requirements." Intent information, as discussed in section 2.5, helps to document the "why" of requirements and can play a significant role in safety.

4.2.5 How Should Environmental Assumptions be Specified?

Surprisingly, no respondents raised the issue of capturing assumptions made about the environment as part of the requirements. As discussed in sections 2.3 and 2.4, documenting any environmental assumptions is an essential part of requirements capture that has direct implications for system safety. For this reason, it has been added here as a major safety issue.

4.2.6 How Should Human Factors Requirements be Specified?

Only one respondent asked for guidance on specifying human factors requirements. However, human factors plays such a significant role in system safety that it is included as a major component of a SpecTRM specification (section 2.5).

4.3 CERTIFICATION ISSUES.

Many comments were provided regarding the guidance given in DO-178B and the certification issues in systems and software development. Many of these issues were raised again in connection with MBD and OOD. Figure 7 shows the number of survey comments related to each major certification issue. The main certification concerns appear to be the relationship of software requirements to software design, the management of traceability and configuration management of requirements, the relationship of system requirements to software requirements, and concerns with oversight by the certification authorities. These issues are discussed in the following sections.

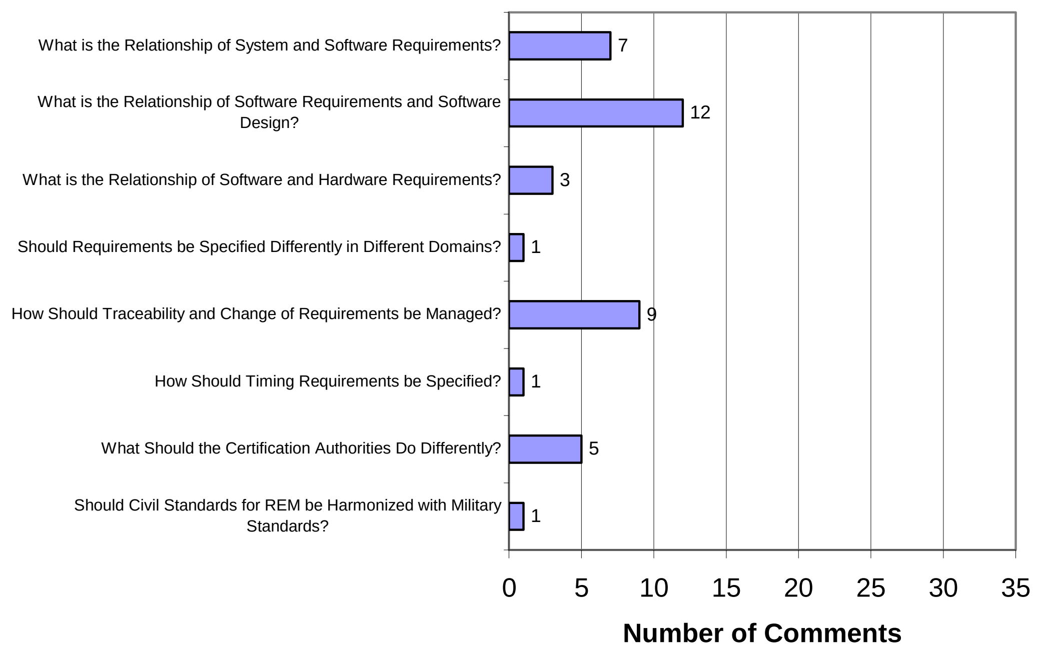

Figure 7. Survey Comments on Certification Issues

4.3.1 What is the Relationship of System and Software Requirements?

Clarification on the proper relationship of systems and software requirements was requested by several respondents. These can be summarized in the following, more detailed issues. These are also addressed by several of the REM methodologies discussed earlier, particularly SCR and CoRE (section 2.3). These issues are also explicitly addressed in reference 41. One respondent asked for a better version of ARP 4754, and two respondents stated that REM needed to be treated as a systems engineering activity.

The more detailed issues emanating from the survey question regarding the relationship between system and software requirements are as follows.

- How should the boundary between the system and software be defined?

  Three respondents expressed frustration with determining where system requirements stop and software requirements begin.

- How do system and software requirements differ?

  Three respondents asked for better guidance on what constitutes a system requirement, as opposed to a software requirement.

4.3.2 What is the Relationship of Software Requirements and Software Design?

Several comments dealt with the software design process described in DO-178B and how it relates to REM. Some of the more detailed issues raised included:

- Clarify terms used in DO-178B. What are high-level requirements? What are low-level requirements? What are derived requirements?

  Several of the respondents asked for more guidance on what constituted a high-level requirement, a low-level requirement, and a derived requirement.

- Why does DO-178B only identify two levels of requirements?

  One respondent explicitly asked for clarification on why DO-178B only allows two levels of requirements. Several other respondents referred to this issue indirectly.

- How do I know if I have the right number of requirements of each type?

  Four respondents asked for guidance on how to tell if they had the right number of requirements of each type. One respondent stated that they tend to see a much higher ratio of low-level requirements to high-level requirements on small projects. One respondent stated that they see a much different ratio on a level A project than on a level D project. One respondent cited the difference in “enforcing” such ratios by different certification authorities.

- How should data flow and control flow requirements be specified?

  One respondent asked for clarification on how data flow and control flow requirements should be specified.

4.3.3 What is the Relationship of Software and Hardware Requirements?

Only three respondents raised concerns about the relationship between hardware and software requirements. Other issues emanating from this question are discussed below.

- How should the boundary between software and hardware be defined?

  Two respondents requested more guidance on identifying the boundary between software and hardware and distinguishing software requirements from hardware requirements. This issue is addressed by several of the REM methodologies discussed previously, particularly SCR and CoRE (section 2.3).

- How do hardware requirements affect software requirements?

  One respondent asked for guidance on how hardware requirements might “interfere” with software requirements.

- How should hardware and software requirements be managed in concurrent development?

  One respondent asked for guidance on how hardware requirements should be managed in concurrent development.

4.3.4 Should Requirements be Specified Differently in Different Domains?

One respondent asked for guidance on how to make “the requirements specification technique fit the item to be specified.” While different problem domains can often be more clearly specified by domain-specific techniques, this raises the important question of whether there are techniques that can cover most problem domains. If so, this would reduce the training and tool demands on both the developers and the certification authorities.

4.3.5 How Should Traceability and Change of Requirements be Managed?

As might be expected, there were several comments related to how requirements should be traced and how changes to requirements should be managed.

- What relationships between requirements for system, safety, high-level software, low-level software, derived, and hardware electronics requirements, as well as code and tests should be captured and traced?

Surprisingly, no respondents asked for guidance on what sorts of relationships should be captured between the various types of requirements, source code, and tests. The establishment of a link seems to suffice. However, answering the remaining issues will require consideration of what relationships are useful and why.

- How should changes to requirements be managed during a product's life-cycle?

  The question of how changes to requirements should be managed were raised in several comments, ranging from "how changes are planned, documented, implemented, controlled, monitored," to "how much regression testing is needed, to how consistency and completeness can be maintained during change."

- When should requirements be placed under configuration control?

  One respondent stated the opinion that requirements were being placed under configuration control too soon, and that this had a deleterious effect on their quality by discouraging necessary change.

- How can level 1 configuration control be applied to requirements in a modeling or database tool?

  Another respondent felt there was a conflict between Control Category 1 configuration control and requirements in a modeling tool or a database.

4.3.6 How Should Timing Requirements be Specified?

Only one respondent mentioned concerns with relationship to timing requirements, and expressed concern that MBD does not usually address the real-time aspects of embedded real-time systems. However, it seems that this should be a significant concern, particularly with respect to IMA. The CoRE methodology discussed in section 2.3 provides an explicit mechanism for recording timing requirements.

4.3.7 What Should the Certification Authorities do Differently?

Several respondents raised concerns about the certification authorities' review of requirements. These tended to fall into the following issues.

- Inconsistent administration of requirements review and acceptance

  Four respondents stated that the review and acceptance of requirements is not administered uniformly. Interpretations appear to vary widely between the different Aircraft Certification Offices.

- The certification authorities' acceptance of requirements in any format other than a document

Two respondents expressed concerns on whether the certification authorities would accept requirements in any form other than a document, for example, in the native format of a database or a modeling tool.

4.3.8 Should Civil Standards for REM be Harmonized With Military Standards?

One respondent asked whether an attempt should be made to harmonize the REM processes used for civil and military programs.

4.4 MODEL-BASED DEVELOPMENT ISSUES.

Many of the same issues that were mentioned as certification issues in the survey resurfaced as concerns for MBD. As a result, many of these issues are listed again, but with an emphasis on the specific concerns related to MBD. Since models are proposed as a way to represent system and software requirements, the use of MBD appears to be very tightly intertwined with REM. Figure 8 shows the number of survey comments related to each major MBD issue. The main concerns seem to be regarding the problems posed by the use of tools, the validation of the models, the relationship of software requirements to software design in MBD, and issues of traceability and configuration management in MBD. The issue of specifying the intent of requirements was raised by a few respondents in relationship to MBD. However, there were issues raised in the literature search of how to specify environmental assumption and requirements for human factors that were not raised by the respondents. All MBD issues are discussed in the following sections.

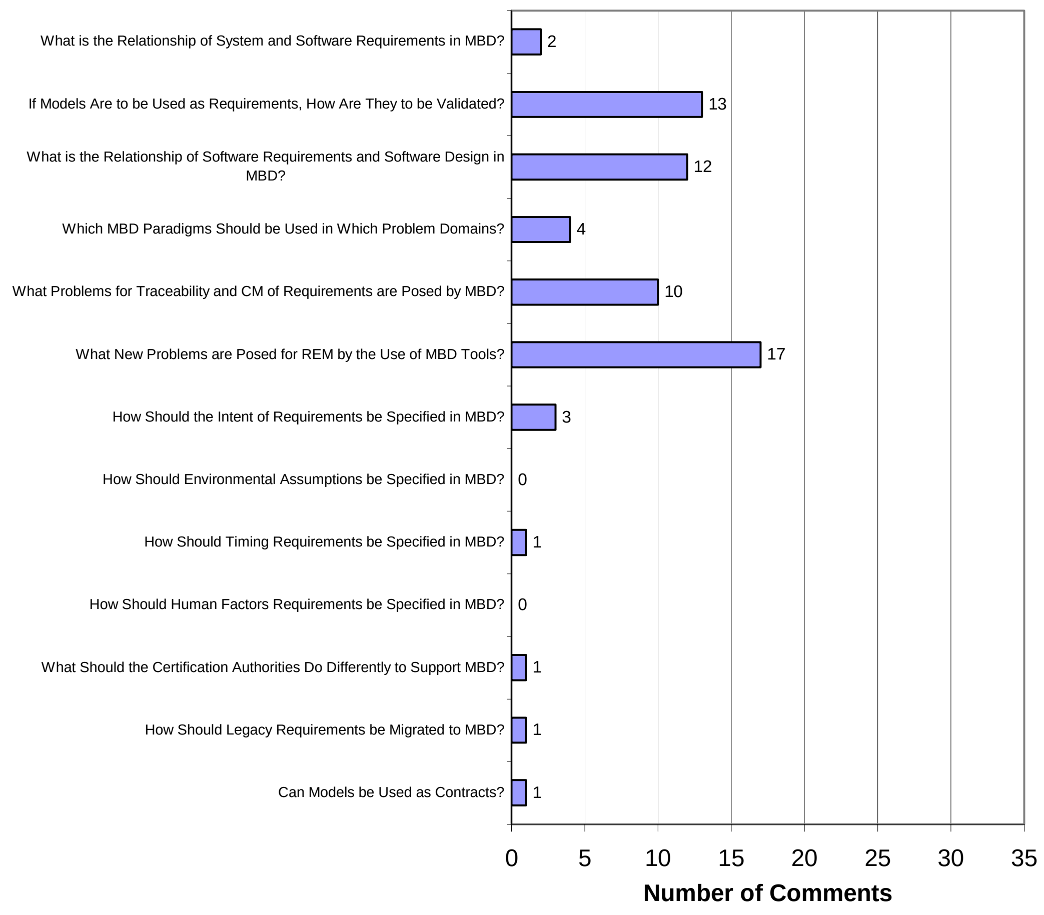

Figure 8. Survey Comments on MBD Issues

4.4.1 What is the Relationship of Systems and Software Requirements in MBD?

The relationship of systems and software requirements appears to become even more of a concern in MBD, largely because systems engineers will often produce models of sufficient detail that code can automatically be generated from them. This leads to the following issues.

- Does MBD eliminate an important review process by moving software engineering into systems engineering?

  One respondent was concerned that MBD takes and shifts the creation of the low-level software requirements from software engineering into systems engineering, where the attention to detail that is engendered by DO-178B is lacking, which could be a realistic concern. Systems engineers typically have the entire DO-178B process between them and the actual implementation. If that is eliminated, it could also eliminate an important review for correctness.

- If the systems requirements are provided as a model, can it be used as the high-level and low-level requirements?

  If the system model is of such detail that it can be used to auto-generate code, does this mean that the system requirements, high-level requirements, and low-level requirements are same thing? If so, how are the objectives of tables A-2 through A-7 of DO-178B to be met?

4.4.2 If Models Will be Used as Requirements, How are They to be Validated?

Closely related to the relationship of systems and software requirements is the issue of model validation. Despite the large number of comments related to this topic, this issue seems to break down into only three more detailed issues.

- If the same model is used for the system, high-level, and low-level requirements, how is it to be validated?

  Several respondents questioned how to develop trust that the model is right, particularly if it also serves as part of the system requirements.

- Can simulation be used for model validation?

  Several respondents requested guidance on how simulation (testing of the model) could be used to validate the model.

- Are coverage metrics over models needed to measure the adequacy of simulation?

  This immediately leads to the question of how the adequacy of the simulation can be measured. One respondent asked if coverage criteria are needed for the model as currently is provided for source code.

4.4.3 What is the Relationship of Software Requirements and Software Design in MBD?

Many of the same issues regarding the relationship of software requirements and software design in the general case were revisited in the context of MBD.

- Clarification of terms used in DO-178B. What are high-level requirements? What are low-level requirements? What is the software architecture in DO-178B? What are derived requirements? What is source code?

  When asked, "In your opinion, what level(s) of requirements do the models represent in MBD?" the respondents provided the answers indicated in table 8.

Table 8. Levels of Requirements Represented by Model-Based Development

Level of Requirements	Number of Respondents Agreeing
System requirements	20
Data interconnect	16
High-level software requirements	33
Low-level software requirements	25
Hardware requirements	9
The model does not represent requirements	7
Did not answer	4

These responses seem to indicate a great deal of confusion regarding how to apply DO-178B to MBD.

- Should the low-level requirements model distinguish between the platform independent model and the platform specific model?

  While not indicated in the survey responses, this issue has been raised by the RTCA SC-205/WG-71 subgroup on MBD. Splitting the low-level requirements model into a platform-independent model (i.e., one that does not take into account the limitations of the platform onto which it is implemented) and a platform-specific model that addresses the platform-specific constraints would assist in reuse and help to separate the true functional requirements from constraints imposed by the implementation.

- When can source code annotations be used on a model used to specify requirements?

  Another issue raised by the RTCA SC-205/WG-71 committee addresses the concern of how much implementation detail should be included in a requirements model. Some modeling languages allow state transitions to be annotated with source code in an imperative language, such as C. In an extreme example, the model could consist of a single state with a single transition annotated with the C source code of an entire program, as in figure 9(a).

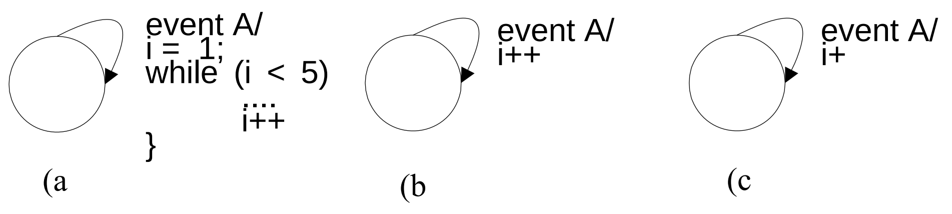

Figure 9. Source Code Annotations on Requirements Models

Figure 9(a) is clearly an overspecification of the low-level requirements. On the other hand, in figure 9(b), the transition is annotated with a single C statement. Does this also represent an overspecification? What about figure 9(c) in which the transition is annotated with a single line of pseudocode? Precisely where is the line?

4.4.4 Which MBD Modeling Paradigms Should be Used for Which Problem Domains?

Four respondents raised concerns about which modeling paradigms should be used in MBD for which problem domains.

- What is the role of object-oriented approaches in MBD?

  Two responses focused on the role of object-oriented design techniques relative to data-flow paradigms used in tools such as Esterel Technologies SCADE™ and MATLAB Simulink.

- How should the requirements of graphical objects (symbology) be specified?

  One respondent raised the question of how requirements for symbology should be captured. Can the requirements be stated using a tool, such as VAPS™, instead of in a document? If so, how is the model to be validated and its implementation verified?

4.4.5 What Problems for Traceability and Change Management of Requirements are Posed by MBD?

Several respondents cited issues related to traceability and change management when MBD is used.

- If some requirements are represented in a model, how are other requirements and test cases linked to the model?

  This concern arises when a model does not consist of finite set of “things” that can be linked to other “things.” Instead, a model consists of several components that interact with each other in rather complicated ways and a test case, or a higher-level requirement, may trace to several of these components. This clouds how test cases and more traditional requirements should be traced to the model.

- How can models (as opposed to documents) be placed under configuration control?

  Some respondents indicated that configuration management organizations were unwilling to place models or databases under configuration control. Others expressed concern about whether a model could be controlled at the right level of granularity.

- How are versions of the models compared?

  One respondent asked how different versions of a model could be compared and their differences identified.

- How are requirements traced across models in several different tools?

  One respondent raised concerns about tracing requirements across models and requirements stored in several different tools.

4.4.6 What Problems are Posed for REM by the Use of MBD Tools?

There were several comments in the survey related to tool issues. Since much of MBD's power comes from its extensive use of automated tools, this is not surprising. A few respondents made positive comments, stating that MBD helped to reduce requirements volatility and made the requirements easier to review. However, the majority of comments expressed concerns summarized in the following issues.

- Can tools be qualified at reasonable cost?

  Several respondents raised concerns about whether the MBD tools will need to be qualified and whether this can be done at reasonable cost. If the tools cannot be qualified, a few respondents questioned whether it will be feasible to verify the output of the tools.

- How are requirements to be reviewed if they are stored in a model?

  A few respondents asked how the requirements will be reviewed if they are represented in a model. Will the reviewer need access to the tool?

- Does the cost of acquiring and learning to use tools outweigh the benefits?

  A few respondents were skeptical about whether the cost of acquiring the tools and learning to use them would outweigh their benefits. Some respondents cautioned against believing that finding the right tool would magically solve the problem of capturing requirements.

- Can tools be integrated to support the traceability of requirements across several modeling tools?

  One respondent pointed out the difficulty of tracing and maintaining configuration control of requirements when the requirements and models are stored in several different tools.

- How will the obsolescence of tools and archival formats be handled?

  Surprisingly, none of the respondents raised the issue of tool obsolescence. However, this will be an increasingly important issue, as different programs use more tools (and different versions of tools) on their products that have to be maintained for decades.

- What if the semantics of the modeling tool are ambiguous?

  None of the respondents raised the issue of dealing with tools with ambiguous, poorly defined semantics. Problems with such tools often arise during code generation and verification when the implementation does not match the designer's intent.

4.4.7 How Should Intent of Requirements be Specified in MBD?

Three respondents alluded to the issue of how intent should be specified in MBD. One respondent was concerned that if system engineers are allowed to delve directly into detailed models, the intent may not be captured. Another was concerned that the use of models would cause the actual requirements to be lost in the detail. Another felt that models are actually design, not requirements.

4.4.8 How Should Environmental Assumptions be Specified in MBD?

None of the respondents raised the issue of how environmental assumptions should be specified in MBD. Due to its impact on system safety, this issue is raised to be addressed in the context of MBD.

4.4.9 How Should Timing Requirements be Specified in MBD?

One respondent pointed out that MBD does not usually capture real-time timing requirements.

4.4.10 How Should Human Factors Requirements be Specified in MBD?

None of the respondents raised the issue of how human factors requirements should be specified in MBD. Due to its impact on system safety, this issue is raised to be addressed in the context of MBD.

4.4.11 What Should the Certification Authorities do Differently to Support MBD?

One respondent raised concerns about how the certification authorities are supporting the use of MBD.

- Will the certification authorities make the transition from document-based to model-based requirements?

  The respondent questioned whether the certification authorities were willing to approve data that is not in a document form.

- Will the certification authorities acquire the tools necessary to view models?

  The respondent questioned how the certification authorities would acquire the tools necessary to review models in their native format.

4.4.12 How Should Legacy Requirements be Migrated to Support MBD?

One respondent raised the issue of how legacy requirements should be migrated to be used in and support MBD.

4.4.13 Can Models be Used as Contracts?

One respondent commented on how models can be used as contracts, since when a model is developed jointly between the customer and the supplier, it is unclear who owns the model. This actually raises two similar issues:

- How can models be used as a contract between the customer and the supplier?
- How can ownership of a model be divided among the customer and the supplier?

4.5 INTEGRATED MODULAR AVIONICS ISSUES.

Figure 10 shows the issues raised relative to IMA. While several issues were raised, none stand out in particular. Respondents were concerned with: (1) an increase risk of common mode and cascading failures, (2) the specification of application and platform services requirements, (3) the support of incremental acceptance for certification (which was identified as a key concern in the literature search), (4) methods to specify and meet system performance requirements, (5) methods to specify and meet dependencies between applications, and (6) issues of traceability and configuration management.

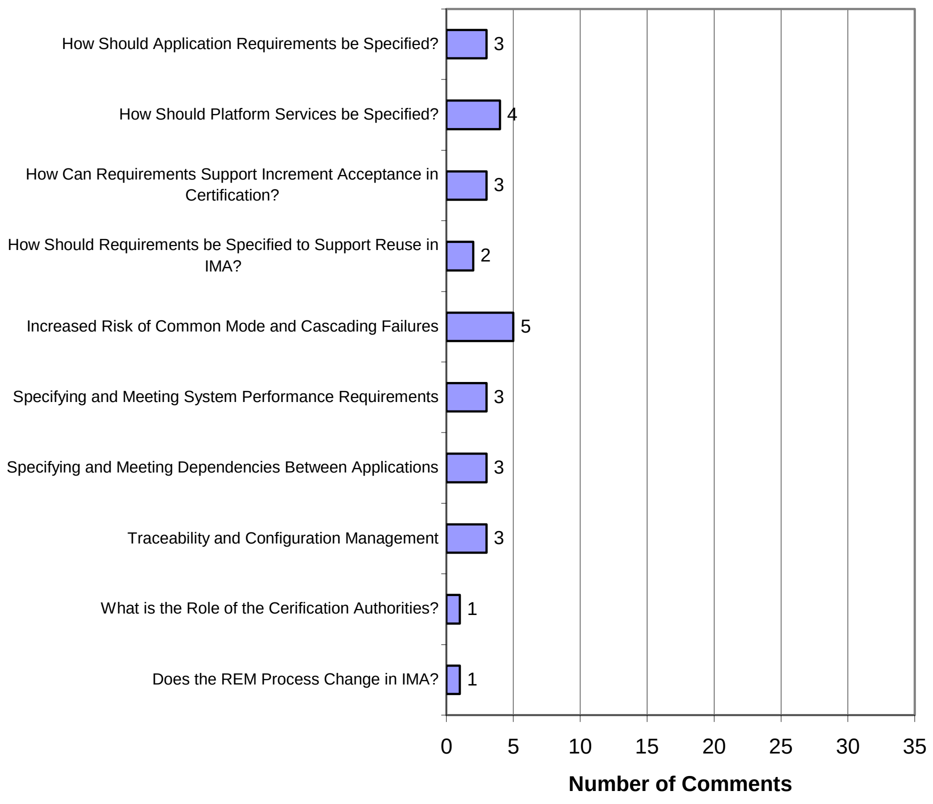

Figure 10. Survey Comments on IMA Issues

4.5.1 How Should Application Requirements be Specified?

All three respondents expressed concern over issues of the integration of applications. Only one respondent directly raised the issue of specifying application requirements, but only to point out that requirements need to be organized around the platform requirements and the application-level requirements. One was concerned about issues of change impact analysis, data and control coupling, and interfacing integrated functions. Another felt that it would be more difficult to verify traceability of the software/software integration. Another respondent agreed that establishing integration requirements would be a concern, as would defining overall system performance and meeting hard real-time deadlines.

These concerns could be mitigated with better specification of application interfaces and behavior. This also emphasizes that in addition to the usual specification of inputs, outputs, functional behavior, and environmental assumptions. The application's requirements for shared resources, such as memory and processor cycles, have to be specified to support the integration of the applications on the IMA platform. These issues are as follows:

- How should the application boundary be specified?

  What inputs does the application need, and what information should be specified about them? What outputs does the application produce, and what information should be specified about them? Should the inputs and outputs be specified directly in terms of the platform services that provide or accept them, or should a level of abstraction be introduced similar to that in the four-variable model of SCR?

- How should the application behavior be specified?

  How should the functional behavior of the application be specified? How are the outputs related to the inputs? Should acceptable variances of output values be specified? How should environmental assumptions be specified? Should the application's behavior be specified if environmental assumptions are violated? How should timing information, such as when outputs must be produced, be specified?

- How should an application's requirements for shared services be specified?

  What shared resources are needed by the application, and how is this information to be specified. For example, how much memory does the application require? How much processor resource?

- How should failures of platform services be handled?

  If a failure of a platform service is detected, what does the application do?

- What does the platform do in response to partition violations?

  If an application exceeds its time or memory constraints, does the platform simply prevent the error? Does it shut down the application? Is this specified in the application's or the platform's requirements?

4.5.2 How Should Platform Services be Specified?

Two respondents expressed concern with not having the same level of visibility into the hardware and the lower levels of implementation. There is a natural tension here, since one objective of IMA is to isolate applications from the details of the underlying platform implementation. At the same time, application developers need to fully understand how the platform services can affect their application, particularly when trying to integrate applications. One way to address this tension is to provide developers with more complete specifications of the platform services. Such information would consist of the service's inputs, outputs, and function behavior, as well as timing behavior and failure modes. These specifications would need to address the following issues.

- How should the inputs for a platform service be specified?

  What inputs are needed and how are they provided to the service? Are there combinations of inputs (preconditions or environmental assumptions) that must be met for the service to function correctly?

- How should the behavior of the platform service be specified?

  What outputs are produced by the service? What are their values in relation to the inputs? Are there side effects, and if so, how are they specified? If the preconditions are not met, should the behavior of the service be specified? How should timing information, such as when outputs will be produced, be specified?

- How are failures of the platform service indicated?

  If the service can fail, how is this indicated to the application?

- How are subsets of common standards specified?

  Adhering to standard interfaces, such as TCP (Transmission Control Protocol) can be very helpful in making applications and platform services more portable, but it may not always be necessary or efficient to implement the entire standard. If only a subset of the standard is implemented, how is this made clear to developers?

4.5.3 How Should Requirements be Specified to Support Incremental Acceptance in Certification?

One primary objective of IMA is to support incremental acceptance of IMA systems in the certification process. This is closely related to supporting change and reuse in IMA systems. For example, how can the implementation of a platform service be changed while minimizing or eliminating the need to recertify the rest of the platform or the applications? If an application module is changed, perhaps to replace an obsolete piece of hardware, can certification be limited to just that module? How does one assess the impact of change in an IMA system? How does one determine what parts of the platform need to be retested?

The way in which requirements for platform services are specified clearly plays a role in this. For example, if the implementation of a platform service is changed without altering the specification of that service, do the applications using the service need to be recertified? If the specification of the service is altered, can the requirements for the different applications be specified in a way that helps determine which applications are impacted? If the behavior of an application is changed, can the way in which the requirements are specified assist in determining the impact on other applications and determining which ones need to be recertified?

4.5.4 How Should Requirements be Specified to Support Reuse in IMA?

Closely related to supporting incremental certification is the issue of supporting reuse, another objective of IMA. Clearly, the specification of an application or of a service's requirements plays a key role in achieving this objective. The application's or service's boundary (inputs and outputs) can be designed to maximize opportunities for reuse. The weaker the preconditions (environmental assumptions) the application or service makes about its environment, the more likely the function can be used in a variety of settings. Specification of precisely what the application or function does is essential for a developer to determine if it can be reused.

4.5.5 Increased Risk of Common Mode and Cascading Failures.

Several respondents expressed concerns about the possibility of common mode failures and cascading errors.

- Meeting Availability Requirements of Critical Functions

  Even though an IMA system should be sufficiently redundant that availability requirements of critical functions are met, three respondents expressed concern that failure of the platform would be more likely to cause loss of critical functions than in a federated system.

- Specifying Requirements for Coping With Cascading Errors

  Two respondents raised the issue of cascading errors. In a highly integrated system, a failure in one system may result in a cascade of errors as multiple systems fail. Requirements need to be developed for diagnosing what the real problem is and providing the flight crew with meaningful information.

- Dealing with Increased Functional Integration

  One advantage of IMA is that it makes it simpler to integrate functions that were previously implemented as stand-alone functions, to off-load from the flight crew some integration tasks that they perform during flight. However, this increased integration increases the possibility of a failure in one application affecting another application.

4.5.6 Specifying and Meeting System Performance Requirements.

It may be more challenging to ensure that the end-to-end performance and timing requirements for a function are met in an IMA environment simply because the developers have less visibility into the lower levels of the implementation. Also, scheduling and latencies may change as applications are added or removed from the platform. One respondent stated that, “IMA means that the engineer has to understand the big picture (integrated), and most don’t.”

This issue was discussed in sections 4.5.1 and 4.5.2 on how the requirements for the application and the platform services should be specified.

4.5.7 Specifying and Meeting Dependencies Between Applications.

Dependencies between applications need to be specified, including data flow, control flow, and relationships that need to be maintained between their states. Such dependencies may be more difficult to meet in an IMA environment, since the scheduling and latencies may change as applications are added or removed from the platform.

4.5.8 Traceability and Configuration Management.

Three respondents felt that traceability and configuration management would be more difficult in an IMA environment.

4.5.9 What is the Role of the Certification Authorities?

One respondent stated that applications on a single IMA platform might have different type certificates (e.g., an engine type certificate and an aircraft type certificate) and that they were unsure of the roles of the FAA directorates in this situation.

4.5.10 Does the REM Process Change in IMA?

As application developers tend to have greater control over the platform resources in a federated environment than in IMA, the potential exists for greater and more frequent negotiation for resources between the application developers and the platform integrator. Does this change the process for managing requirements in significant ways?

4.6 OBJECT-ORIENTED DEVELOPMENT ISSUES.

As with MBD, many issues that were raised as certification issues in the survey were also raised as concerns for OOD. However, many of the comments were more closely related to implementation issues than to requirements management. Figure 11 shows the number of survey comments related to each major OOD issue. The main concerns were with problems of traceability and configuration management and the relationship of software requirements to software design in OOD. Important issues were identified in the literature search that were not raised by the respondents. These included how to specify the intent of requirements, environmental assumptions, and how to specify requirements for human factors. Each of these major issues is discussed in the following sections.

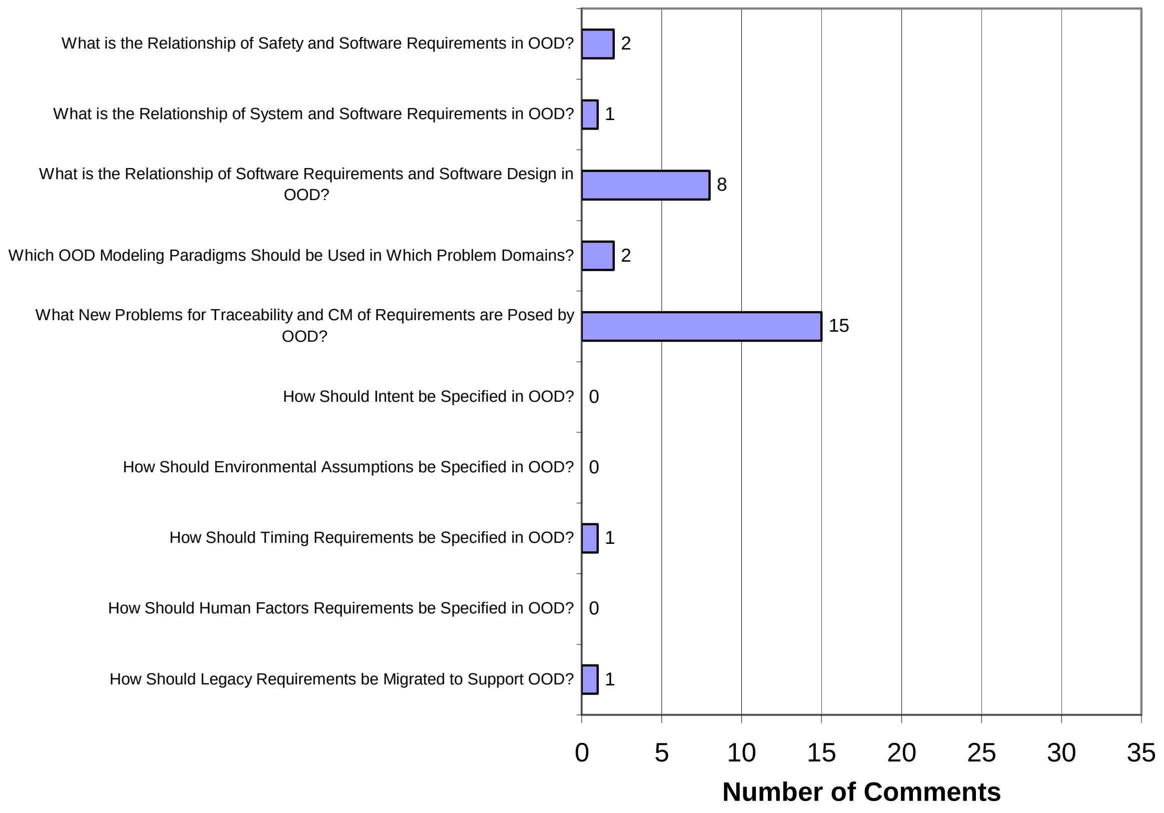

Figure 11. Survey Comments on OOD Issues

4.6.1 What is the Relationship of Safety and Software Requirements in OOD?

Two comments were related directly to safety issues in OOD, both stating that the safety aspects of OOD are unclear. One respondent commented that there is no clear industry consensus of the safe constructs of OOD. Many of the same general safety issues raised earlier were raised again, particularly, what should be done differently due to safety requirements, and which software development techniques can be used to improve system safety. These would apply equally well to OOD.

4.6.2 What is the Relationship of System and Software Requirements in OOD?

One respondent raised systems engineering issues in relationship to OOD. These included the difficulty of getting systems engineers to shift from a structured design paradigm to an object-oriented paradigm and the cost to retrain the systems engineers to make the transition.

4.6.3 What is the Relationship of Software Requirements and Software Design in OOD?

Several respondents asked for more guidance on how to map the terminology used in DO-178B to OOD. Two respondents directed us to the Object-Oriented Technology in Aviation (OOTiA) Handbook for more details.

- Clarification of terms used in DO-178B. What are high-level requirements? What are low-level requirements? What are derived requirements?

  Respondents were particularly interested in what would be considered a low-level requirement in OOD.

- How can reusable objects be created in OOD?

  One respondent asked for more guidance on how reusable objects (from requirements to system tests) could be created and used in new developments.

4.6.4 Which OOD Modeling Paradigms Should be Used in Which Problem Domains?

Two respondents raised questions related to use of the different OOD modeling paradigms.

- What is the role of model-based approaches in OOD?

  One respondent asked for more guidance on how to use UML in model-based development.

- How should the different OOD modeling paradigms be used in specifying requirements?

  Another respondent asked if low-level requirements should be specified using use cases or state diagrams. This raises the larger question of how the many different object-oriented modeling techniques could be used for the specification of requirements.

4.6.5 What Problems for Traceability and Change Management of Requirements are Posed by OOD?

Many respondents raised concerns about unused functionality introduced by reuse of objects, obtaining test coverage of object-oriented code, dealing with nondeterminism introduced by OOD, and tracing from requirements to code and to test cases. While these could be raised as major issues in their own right, it seems they also manifest themselves as problems in traceability and change management, and are therefore grouped into this section.

- How do we trace high-level requirements to low-level requirements?

  Two respondents felt that it was more difficult to trace high-level requirements to and from low-level requirements in OOD. One respondent felt that this was because OOD does not lend itself well to functional decomposition.

- How do we trace low-level requirements to code?

  One respondent felt that there were also significant issues in tracing code back to the requirements in OOD due to the introduction of abstraction.

- How should unused functionality be handled?

  Several respondents raised the problem of unused functionality in OOD. One main benefit of OOD is its inherent support for reuse. However, this may introduce code that is not used in a specific implementation and would have to be shown to be deactivated or removed.

- How are requirements traced to test cases?

  Several respondents also felt it was more difficult to trace requirements to test cases and achieve full structural coverage of code with requirement-based test cases in OOD.

- How is nondeterminism handled?

  Three respondents addressed the issue of dealing with nondeterminism introduced by the use of OOD, particularly through the use of dynamic dispatching.

4.6.6 How Should Intent be Specified in OOD?

None of the respondents raised the issue of how intent should be specified in OOD. Due to its impact on system safety, this is raised as an issue to be addressed in the context of OOD.

4.6.7 How Should Environmental Assumptions be Specified in OOD?

None of the respondents raised the issue of how environmental assumptions should be specified in OOD. Due to its impact on system safety, this is raised as an issue to be addressed in the context of OOD.

4.6.8 How Should Timing Requirements be Specified in OOD?

One respondent stated that they watch real-time aspects closely when using OOD.

4.6.9 How Should Human Factors Requirements be Specified in OOD?

None of the respondents raised the issue of how requirements for human factors should be specified in OOD. Due to its impact on system safety, this is raised as an issue to be addressed in the context of OOD.

4.6.10 How Should Legacy Requirements be Migrated to Support OOD?

One respondent raised the issue of how legacy requirements could be migrated to be used in and support OOD.

5. PRIORITIZING THE SAFETY AND CERTIFICATION ISSUES.

Sections 2 through 4 describe the activities conducted in Phase 1 of the project to identify the safety and certification issues in REM. In Phase 2, this information was used to guide the development of recommended practices in REM. Due to the large number of issues identified in Phase 1, the first step in this process was to refine the list of issues into a more manageable set. This section describes the final set of safety and certifications issues and the process followed in their selection.

5.1 SUMMARIZING THE INDUSTRY SURVEY ISSUES.

Many of the survey questions asked the respondents to identify issues with respect to specific technologies, such as MBD, OOD, or IMA. In the Phase I analysis, the issues were associated with these categories (MBD, OOD, or IMA) or with categories of safety, certification, and general (where general issues spanned all of the other categories). As the survey responses were studied further, it became clear that many of the same issues were being raised across multiple categories. To get a better picture of the respondents concerns, related issues were grouped together and the number of responses compiled for each issue, regardless of the technology, cited in the original survey question. A summary of the issues commented on at least ten times in the survey is shown in figure 12. These are discussed in more detail in the following sections.

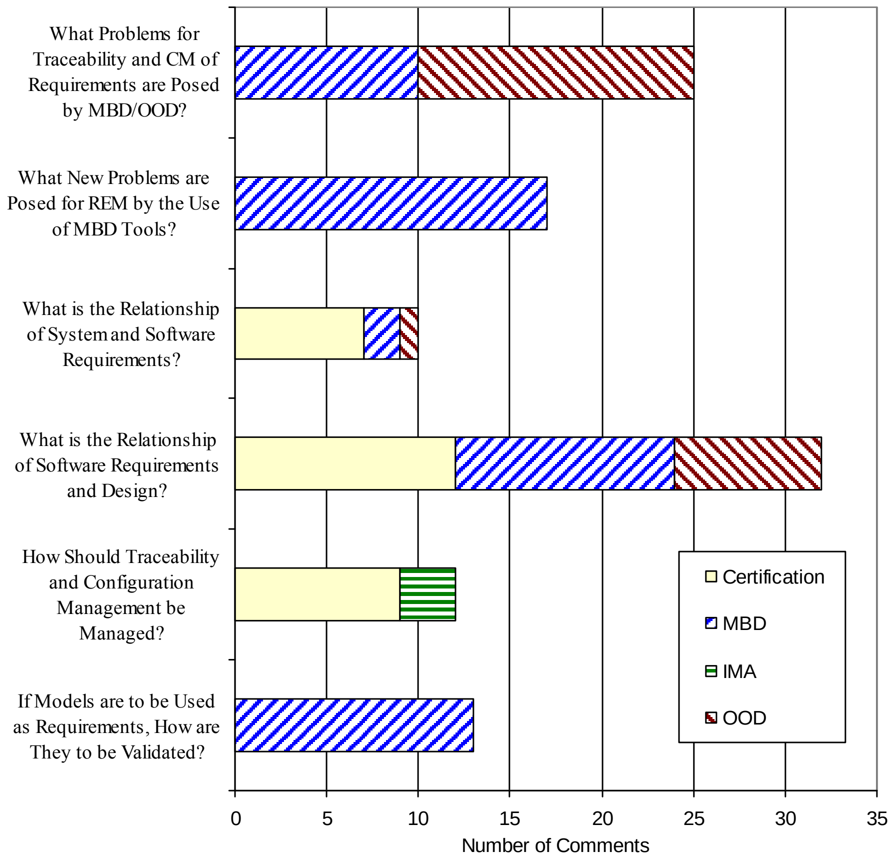

Figure 12. Most Frequently Cited Issues in Survey

5.1.1 What is the Relationship of Software Requirements to Software Design?

As shown in figure 12, one of the most frequently cited issues was confusion over the relationship of software requirements to software design. Thirty-two survey comments mentioned this issue in some form. Many comments requested clarification of the terms used in DO-178B. For example, what are high-level, low-level, and derived requirements? Why does DO-178B only identify two levels of requirements? How does one know if they have the right number of requirements of each type?

5.1.2 What Problems for Traceability and Configuration Management are Posed by MBD/OOD?

Traceability and configuration management of requirements when using MBD or OOD was mentioned in 25 survey comments. In MBD, concerns were raised on how to link requirements and test cases to a model, how models can be placed under configuration control, how versions of models can be compared, and how requirements can be traced to models in several different tools. In OOD, questions were raised on how to trace high-level requirements to low-level requirements, how to trace low-level requirements to code, how to deal with unused functionality, how to trace requirements to test cases, and how to deal with nondeterminism.

5.1.3 What New Problems are Posed for REM by the Use of MBD Tools?

A number of questions were raised in the survey related to the use of tools in MBD. Seventeen survey comments touched on this topic. Concerns included whether tools can be qualified at reasonable cost, how requirements will be reviewed if they are stored in a model, if the cost of acquiring and learning to use tools outweigh the benefits, and how tools can be integrated to support traceability of requirements across several modeling tools.

5.1.4 If Models are Used as Requirements, How are They to be Validated?

Another frequently cited concern (13 survey comments) was how requirements stated as models were to be validated. The major concern here centered on the situation where a single model was used as the system, high-level software, and low-level software requirements. Concerns were also raised about how simulation could be used for model validation, and if structural coverage of the model could be used in place of structural coverage of the source code.

5.1.5 How Should Traceability and Configuration Management be Managed?

Twelve survey comments raised concerns about traceability and configuration management related to DO-178B and IMA, but the concerns were quite different from those mentioned regarding traceability and configuration management in MBD and OOD (section 5.1.2). As a result, this was identified as a separate issue. Most certification concerns were related to what relationships needed to be traced between system requirements, safety requirements, high-level software requirements, low-level software requirements, derived requirements, hardware requirements, code, and test cases. In fact, the concerns were very similar to those raised regarding the relationship of software requirements to software design (section 5.1.1) and the relationship of system and software requirements (section 5.1.6). Several comments also mentioned concerns about how changes should be managed during a product's life cycle. Regarding IMA, three of the survey comments stated that traceability and configuration management would be more difficult in an IMA environment.

5.1.6 What is the Relationship of System and Software Requirements?

Ten survey comments expressed confusion over the relationship of system and software requirements. The respondents expressed frustration that system requirements and software requirements are frequently very similar and asked for better guidance on what distinguishes them. This issue was also raised by two respondents in regard to MBD, where the models created by the system engineers can often be used to autogenerate code.

5.2 SUMMARIZING THE LITERATURE SEARCH ISSUES.

The search of literature related to REM also identified several practices (or lack of a practice) that could cause significant problems in REM. Several of these were not raised in the survey comments. These are discussed in the following sections.

5.2.1 Defining the System Boundary.

As discussed in section 2, the SCR [10-13] CoRE [14-16], RSML [17 and 18], and SpecTRM [7-9, 19, and 20] methodologies for REM all emphasize the importance of clearly identifying the system boundary. This provides a definition of what lies within the system to be built and what lies within the larger world. Without this, it is exceptionally easy to write requirements that duplicate or conflict with those being defined at a higher level, or to miss requirements because they are assumed to be provided by the environment. In particular, this seems to be a concern with early object-oriented approaches to REM that failed to distinguish which objects were parts of the system to be developed and which were parts of the environment.

While this issue was not directly raised in the survey, several of the respondents touched upon it in asking for clarification on the relationship of system and software requirements. Identification of the system boundary is an important aspect in understanding the relationship between system requirements and software requirements.

5.2.2 Documenting the Environmental Assumptions.

The SCR [10-13], CoRE [14-16], RSML [17 and 18], and SpecTRM [7-9, 19, and 20] methodologies for REM also emphasize the need for identifying all environmental assumptions, i.e., clearly identifying all behaviors and properties of the environment on which the system depends. Hooks and Farry [59] cite incorrect facts or assumptions as the most common form of requirements errors. Failure to identify and document the environmental assumptions has also been the cause of several dramatic system failures such as the Ariane 5 disaster [60]. In some cases, the failure occurred because a subsystem developed by one team did not meet the assumptions made by another team.

When combined with the requirements for the system, the environmental assumptions effectively form a "contract" that allows components to be developed independently. The environmental assumptions define the obligations that must be met by the system's environment. If these are met, then the system is required to satisfy its requirements. Thus, identification of environmental assumptions is essential for the reuse of a component. Since the use of common components is central to IMA, identification of the environmental assumptions is essential in specifying the requirements of IMA components.

Surprisingly, this issue was not directly raised by any survey respondents. A few comments related to "Identifying and Meeting Dependencies Between Applications" (section 4.5.7) raised concerns about how to demonstrate that an integrated system meets its requirements, but did not raise the concern about how to define requirements of a component on other components or its environment.

5.2.3 Providing Intent and Rationale.

Several referenced studies emphasized the need to go beyond simply specifying what the system is to do and include information about why the requirements exist at all. As discussed in section 2.5, SpecTRM [7-9, 19, and 20] requirements models are embedded within a larger intent specification that "provides a seamless flow of rationale and reasoning from the highest level goals of the system, down through the SpecTRM-RL model all the way down to the implementation and operator training materials." This provides essential information as new members are brought onto the engineering team and helps to prevent important decisions, particularly those related to safety [5, 7, and 8], from being undone during development and maintenance. This can also be extremely helpful when distributed design teams work on a single project by providing important context to each individual team [9].

Hooks and Farry [59] also debated that rationale should be provided for every requirement. While requirements state what the system is to do, rationale explains why the requirement exists and why particular values are specified. Rationale can: (1) reduce the amount of time required to understand a requirement, (2) help the readers to avoid making incorrect assumptions, (3) decrease the cost of maintenance by documenting why certain choices or values are specified, (4) make the development of variations of the same product cheaper by providing insight into the impact of changing the requirements and assumptions, and (5) decrease the cost of educating new employees by providing them with information about why the system behaves the way it does.

Providing rationale can also improve the quality and reduce the cost of creating the requirements. Providing the rationale for a bad requirement or assumption can be difficult. Forcing the specifier to think about why the requirement is necessary or why the assumption is being made will often improve the quality of the requirement or lead the writer to remove implementation detail. This eliminates unnecessary constraints on the developers.

Only four survey comments touched on providing intent or rationale for requirements, and three of these were raised in the context of MBD. The comments related to MBD all raised concerns that the use of models as requirements was obscuring the true requirements and their intent.

5.2.4 Validation of Requirements.

Validation of the requirements, i.e., determining if the requirements specify the right system, is one of the central issues in REM. The SCR [10-13], CoRE [14-16], RSML [17 and 18], and SpecTRM [7-9, 19, and 20] methodologies address validation through the use of tabular and graphical presentations that make it easier for all stakeholders to review and understand a complex mathematical specification. SCR [12 and 13], RSML [17 and 18], and SpecTRM [7-9, 19, and 20] also address validation by providing interpreters that allow the user to simulate the requirements specification with inputs and observe how its internal state and outputs change. Often, the inputs and outputs can be connected to graphical depictions of the operator interface so the stakeholders can review the behavior of the system almost as if it were the completed system.

As useful as simulation of the requirements is in requirements validation, it needs a requirements specification that is largely completed. An approach to drive requirements validation even earlier into the life cycle is to define operational concepts as one of the first steps in the REM process. Operational concepts are simply scripts, or scenarios, that describe how the system will be used [59]. Ideally, they are written in a natural, intuitive style that all stakeholders can understand. This helps build a consensus between the stakeholders and identify which functions the system must provide. Since they focus on how the operators and other systems interact with the system, they will often uncover operator interface issues.

Use cases are a popular way to identify and document interactions between a system, its operators, and other systems [23-25]. While conventions for the format of use cases are similar, there are numerous styles that introduce varying degrees of rigor and formality. One of their attractions is that they can be used relatively informally to elicit a better understanding of the system functions the operators need.

Very few survey comments discussed the issue of requirements validation. Most of those that did were concerned with validation when the requirements were stated as models (see section 4.4.2).

5.2.5 Verification of Requirements.

Verification usually refers to whether a design or implementation satisfies its specification and is not usually as large a concern for requirements as validation. However, there are a number of checks for consistency and completeness of requirements that can be viewed as a verification activity. The SCR [10-13], CoRE [14-16], RSML [17 and 18], and SpecTRM [7-9, 19, and 20] languages all provide facilities to assist in writing requirements that are complete and consistent. Here, completeness ensures that a value is specified for each output for every possible combination of system state and input, while consistency ensures that only one value is specified for each output for every possible combination of system state and input. Note that completeness as used here does not refer to ensuring that all needed functions are specified in the requirements—that notion of completeness is more correctly treated as an issue in requirements validation.

5.2.6 Managing Change and Complexity.

The requirements for any large and complex system are also likely to be large and complex. For such systems, it is extremely difficult to fully specify the requirements correctly at the start of the project. Even if the initial set of requirements is correct, the requirements usually change throughout project development, sometimes dramatically. Any methodology or notation for REM must be able to deal with both complexity and frequent change if it is to be useful.

Both CoRE [14-16] and RSML introduce specific constructs to deal with change and complexity. CoRE incorporates object-oriented concepts to group together requirements that are related and likely to change together. It also introduces intermediate variables into the specification that represent quantities from the problem domain that are stable and unlikely to change (for example, the pitch and roll of an aircraft). These intermediate variables are used as firewalls in the requirements specification to keep change from rippling through the specification. While RSML [17 and 18] does not emphasize the management of change as strongly as CoRE does, it does provide a macro facility that allows complex and related specifications to be associated with a single name.

5.2.7 Development by Multiple Entities.

Increasingly, airborne systems are being implemented as IMA architectures in which several functions are implemented on a single, fault-tolerant computing platform [55 and 56]. Such systems are attractive in that they offer the promise of reducing space, power, weight, and other costs. However, the higher physical integration and distribution of system functions across organizational boundaries in IMA heightens concern about how their requirements are specified and managed.

One promise of IMA is that applications can be purchased, developed incrementally, or reused from other programs, where individual applications and the IMA platform may be developed by different companies. The overall functions for the aircraft are accomplished through the combined interactions of these application functions. Thus, the overall safety of the aircraft is an emergent property of the system as a whole, and cannot be ensured by analyzing the applications in isolation. However, the physical distribution of the developers (not to mention their corporate and legal separation) requires a means of allocating the overall safety requirements among them in such a way that the overall safety requirements can be demonstrated to be met.

One step toward achieving this is to provide a complete, precise, and unambiguous specification of the requirements to each product team. The SCR [10-13], CoRE [14-16], RSML [17 and 18], and SpecTRM [7-9, 19, and 20] methodologies all describe ways to do this.

In addition, it is also important to define the dependencies that each system makes of its environment and of the other systems with which it interacts. This was discussed in section 5.2.2 with regard to identifying each system's environmental assumptions.

Finally, it is also seems essential to convey the rationale, or the why, for the requirements of each application. This is particularly true for the safety-related requirements. This helps the application developer to understand the emergent safety requirements at the system level and how they apply to their application. Intent specifications address this by providing a format to provide all design teams with the system design decisions and rationale relevant to their application [9].

Most of the survey comments related to IMA touched on these topics. However, none of the major IMA issues were cited more than five times and, thus, were not included in the main survey issues discussed in section 5.1.

5.2.8 Specifying Requirements for the Human Machine Interface.

In references 5 and 19, Leveson argues persuasively that incorrect specification and design of the HMI is one of the most common sources of safety-related errors in avionics systems. The higher levels of integration, automation, and complexity of these systems places greater cognitive demands on the flight crew [61]. To fly commercial flights today, pilots must master several complex dynamically interacting systems, often operating at different levels of automation, that have evolved over a number of years. To provide maximum flexibility and to accommodate the variety of situations that occur during flight, these systems typically have many different modes of operation, with different responses to crew actions and the other systems in each mode.

Similar arguments are presented in references 62 through 68, which discuss the phenomenon of mode confusion, perhaps more appropriately referred to as automation surprises. Mode confusion occurs when the flight crew believes they are in a mode different than the one they are actually in and consequently make inappropriate requests or responses to the automation. Mode confusion can also occur when the flight crew does not fully understand the behavior of the automation in certain modes, i.e., when the crew has a poor “mental model” of the automation.

Mode confusion is often the result of automation that is not designed with a focus on the human operator [64 and 68] such that automation does not match the natural work flow of the operator, places unnecessary and unevenly distributed demands on the operator, behaves inconsistently in different modes, or provides poor feedback on the status and behavior of the automation. To address these concerns, the process for developing an Intent Specification [8 and 19] elevates the development of the requirements for the HMI to a separate process that proceeds in parallel with the development of the system itself and the safety process.

Despite its importance in the literature, only one survey comment mentioned HMI requirements as a concern.

5.3 PRIORITIZING THE ISSUES.

As discussed in section 5.1, the survey comments were distilled into the list of issues shown in table 9. To prioritize the issues, they are ranked by the number of survey comments associated with each of them. While this provides an admittedly imperfect indication of their importance, it is sufficient to serve as the starting point for the development of the recommended practices in section 6.

Table 9. Prioritized Industry Survey Issues

Rank	Issue
1	Clarifying the relationship of software requirements to software design
2	Traceability and configuration management in MBD and OOD
3	Problems posed by the use of MBD tools
4	Validation of requirements stated as models
5	Traceability and configuration management of requirements
6	Clarifying the relationship of system requirements and software requirements

In similar fashion, the list of issues identified in the literature search (section 5.2) is shown in table 10. Since no objective measure exists to prioritize these issues, they are ranked in the order of importance assigned by the authors.

Table 10. Prioritized Literature Search Issues

Rank	Issue
1	Defining the system boundary
2	Documenting the environmental assumptions
3	Development by multiple entities
4	Providing intent and rationale
5	Validation of requirements
6	Managing change and complexity
7	Specification of HMI requirements
8	Verification of requirements

6. DEVELOPMENT OF THE RECOMMENDED PRACTICES.

This section discusses the process followed in developing the Requirements Engineering Management Handbook [1], which was the primary activity in Phase 2. Section 6.1 discusses the process followed in development and validation of the recommended practices. Section 6.2 describes the recommended practices themselves. Finally, section 6.3 maps the recommended practices to the safety and certification issues discussed in section 5.

6.1 PROCESS FOLLOWED.

At the end of Phase 1, several approaches were considered for development of the recommended practices. One approach addressed each issue in order of importance and developed a set of practices to address just that issue. Another approach emphasized the integration of several best practices identified in the literature that would address the identified safety and certification issues. The second approach was selected, largely because of the information found in the industry survey.

The industry survey indicated that system developers needed better guidance of what information should be included in a requirements specification, how that information should be collected, and how it should be presented. For example, when asked in what format requirements are being captured (section 3.2.2.1), the overwhelming majority of the survey respondents indicated that requirements are being captured as English text, shall statements, or as tables and diagrams. A few respondents reported using UML or data flow diagrams, while a slightly larger number claimed to be using executable models such as MATLAB Simulink or Esterel Technologies SCADE™. In similar fashion, when asked what tools are used for requirements capture (section 3.2.2.2), the overwhelming majority of the respondents stated that they were using the Telelogic DOORS tool or a word processor such as Microsoft® Word®. A similar response was obtained to the question of what tools are used for requirements traceability (section 3.2.2.3). None of the respondents reported use of any of the best practices identified in the literature such as SCR [10-13], CoRE [14-16], RSML [17 and 18], and SpecTRM [7-9, 19, and 20]. Even for widely known approaches, such as structured analysis or use cases, there were only a few respondents that reported using these practices. These impressions were reinforced by the free-text comments provided by the survey respondents.

For these reasons, the recommendation made at the end of Phase 1 was to develop an incremental approach for improved REM that would allow practitioners to progress from their current state of practice by gradually introducing more of the fundamental concepts identified in the best practices. Central to this would be the development of a conceptual model that organizes the requirements into a structure, identifies what information needs to be captured about each type of requirement, defines the relationships between requirements, and supports the development of requirements by multiple entities.

This was done in two steps. The first was to develop a set of recommended practices for the development of requirements by a single team. This established the basic set of recommended practices for any development. The next step extended these practices with a set of recommended practices for the development of requirements by multiple entities. One interesting consequence of this was that the recommended practices for development by multiple entities largely stressed doing a better job of the recommended practices for a single team. In hindsight, this is understandable since the essential purpose of REM is to allow all stakeholders to ensure that the correct system is eventually built.

To validate these practices, they were used to create two detailed examples. These examples were also used in the Handbook to illustrate the recommended practices. The first, a thermostat for an isolette used in a neonatal care unit, provides a small, easily understood example that illustrates application of the recommended practices on a single system. The second, a very simple Flight Control System, FGS, and Autopilot, illustrates how requirements could be allocated from a system to its subsystems to facilitate development by multiple entities. To further validate the recommended practices, the Handbook, and the examples were reviewed on a regular basis during their development by experts in REM, system safety analysis, and software certification.

Even though several survey respondents reported using executable modeling tools such as MATLAB Simulink [26 and 27] or Esterel Technologies SCADE™ [28] for requirements specification, and many issues raised in the survey comments were directly related to MBD, there was not enough time to directly address the MBD issues and develop the best practices described above. However, subgroup 4 of the RTCA/EUROCAE/WG-71 committee is currently developing recommendations for the use of MBD in software development. While no recommendations had been made at the time of this work, the subgroup is leaning toward the position that a model used in software development must be verified against a higher-level set of requirements, where those requirements will probably be in a textual form. If this position is taken, the primary REM issue in MBD will be how to develop this higher-level set of requirements. This is precisely the question addressed by the recommended practices.

Another issue that was not addressed completely was the development of a set of best practices for developing the requirements for the HMI. Despite their importance in system safety as identified in the literature survey, there was not sufficient time to address such a large topic, particularly since it was not directly called out in the statement of work and only one survey respondent mentioned it as a concern. However, requirements for HMI are addressed at a high level in the best practice of developing the operational concepts (section 6.2).

6.2 THE RECOMMENDED PRACTICES.

The Requirements Engineering Management Handbook [1] describes 11 main-level recommended practices that can be introduced into any development effort. For each main-level practice, one of the two examples (the neonatal isolette or the FGS) is used in the Handbook to illustrate the practice and make the ideas concrete.

Within each main-level practice, several more sublevel recommended practices are identified in the Handbook. These are more sensitive to how the main-level practice is implemented and, thus, more dependent on the example used to illustrate the main-level practice. The sublevel recommended practices are also more likely to have to be modified as the main-level practice is tailored to fit an organization’s existing methods and tools.

The 11 main-level practices are briefly summarized in the following sections. A more detailed discussion and examples are presented in the Requirement Engineering Management Handbook [1].

6.2.1 Develop the System Overview.

The system overview serves as the introduction to the system requirements. Though it will evolve continuously as the requirements are defined, it is usually one of the first artifacts created. Its purpose is to orient the reader quickly to the system and the environment in which it will operate. It should not attempt to completely describe the system. Rather, its intent is to provide the reader with a high-level view of what the system does, how it interacts with its environment, and why the system is needed. At a minimum, it should include a short synopsis of the system, one or more context descriptions, a brief description of each external entity in the context diagrams, and a set of high-level system goals, objectives, and constraints. It may also include other relevant information, such as historical background, but this needs to be balanced against the need to orient a new reader quickly.

6.2.2 Identify the System Boundary.

One of the most important activities in requirements engineering is to clearly define the boundary between the system and its environment. This provides a sound understanding of what lies within the system to be built and what lies within the larger world. Every system is embedded within the environment in which it operates, and this environment is often a larger collection of systems. Without a clear definition of the system boundary, it is exceptionally easy to write requirements that duplicate or conflict with those defined at a higher level, or to miss requirements because they are assumed to be provided by the environment. This is particularly important when a system is being developed by multiple entities.

One way to define the system boundary is to view the system as a component that interacts with its environment through a set of monitored and controlled variables (figure 13) [1].

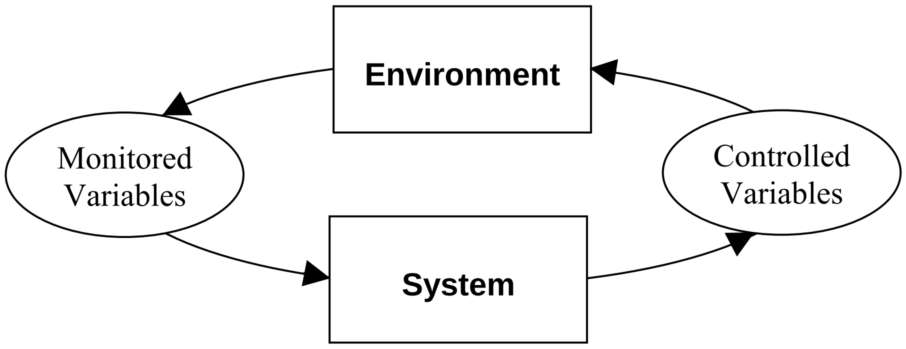

Figure 13. The System and its Environment

The monitored variables represent quantities in the environment to which the system responds, and the controlled variables represent quantities in the environment that the system is to affect. For example, monitored values might be the actual altitude of an aircraft and its airspeed, and controlled variables might be the position of a control surface such as an aileron or the displayed value of the altitude on the primary flight display. Conceptually, the monitored and controlled variables exist in the environment outside of the system and would continue to exist even if the system were eliminated.[2] The purpose of the system is to maintain a relationship between the monitored and controlled variables that achieves the system goals. The definition of the monitored and controlled variables and their attributes define the boundary of the system.

Eventually, the system boundary should be extended into a complete definition of the physical interfaces to the system. This should identify all discrete inputs and outputs, all messages, all fields in a message, and the protocols used to receive and send messages. If the interface adheres to a standard interface or protocol, the standard can be cited.

[1] This is the approach taken in the SCR [10 and 11] and CoRE methodologies [14 and 16]. A similar notion is used in RSML [17], where they are referred to as the manipulated and controlled variables.

[2] While monitored and controlled variables appear similar to inputs and outputs, the terms inputs and outputs are used in this document to refer to the values provided and generated by the software in the system.

6.2.3 Develop the Operational Concepts.

Operational concepts are scripts or scenarios that describe how the system will be used [59]. They are a useful step in the progression from the system overview to the detailed requirements. Operational concepts view the system as a Black Box and describe how it interacts with its operators and other systems in its environment. They help to identify which functions the operators expect the system to perform, the values the operators can provide as inputs, and the information the operators need as feedback. They also help identify the sequence in which the operators can invoke system functions during normal operation and the response of the operators when the system behaves differently than expected.

Ideally, operational concepts are written in a natural, intuitive style that all stakeholders can understand. This helps build a consensus between the different stakeholders and identify system functions that may have been overlooked. Since the operational concepts focus on how the operators and other systems interact with the system, they will often uncover operator interface issues, particularly in terms of what information is needed from the operator, when functions can be invoked by the operators, and what information the operators need from the system.

Use cases are a popular way to identify and document interactions between a system, its operators, and other systems. The literature on use cases is large and ranged from high-level requirements down to detailed design. Some of the best known references include [23, 24, and 25]. While conventions for the format of use cases are similar, there are numerous styles that introduce varying degrees of rigor and formality. However, use cases seem to be especially appropriate for use early in the requirements engineering process in helping to understand and describe how operators and other systems will interact with the system. One of their attractions is that they can be used relatively informally to elicit a better understanding of the system functions the operators need.

Use cases are an excellent way to obtain early validation of the requirements and to document how the operators and other systems (the actors) interact with the system being developed. Their use can help identify inconsistencies and oversights and provide deeper insight into how the system is to be used. They also help to identify overlooked external entities, as well as monitored and controlled variables.

While creating the use cases, a preliminary list of system functions should be assembled. These will be used as input to the recommended practice, Develop the Functional Architecture, described in section 6.2.5.

6.2.4 Identify the Environmental Assumptions.

Environmental assumptions are the assumptions about the environment on which a system depends for its correct operation. These can be specified as a mathematical relationship between the controlled and the monitored variables.[3] This relationship can be as simple as the types, ranges, and units of the monitored and controlled variables. For example, the system assumes the altitude will never be less than 100 feet below sea level or more than 50,000 feet above sea level. Or, the relationship can be as complex as a complete mapping from the controlled variables to the monitored variables that describes the full behavior of the environment.

Identifying the assumptions a system makes about its environment is as important a part of REM as specifying the required behavior of the system. Hooks and Farry cite incorrect facts or assumptions as the most common form of requirements errors [59]. Failure to identify and document the environmental assumptions has also been the cause of several dramatic system failures. In some cases, the failure occurred because a subsystem developed by one team did not meet the assumptions made by another team. An essential first step to preventing such failures is to identify and document these assumptions.

While it is desirable that a system depends on as few environmental assumptions as possible, it is not possible to design a system that does not make some environmental assumptions. No system can accept an infinite range of inputs, and at some point, at least the types and ranges of the inputs and outputs must be selected. Often, there are more complex assumptions that also need to be documented. Other common sources of environmental assumptions relate the monitored variables to the controlled variables (effectively a partial model of the plant being controlled) or limit the rate at which monitored values can change.

When combined with the requirements for the system, the environmental assumptions effectively form a "contract" that allows components to be developed independently. The environmental assumptions define the obligations that must be met by the environment of the system, including other systems with which it interacts. If these are met, then the system under development is required to satisfy its requirements. In software development, environmental assumptions are often referred to as "preconditions" and the requirements as "postconditions".

Identifying the environmental assumptions is essential for reuse of a component. In several cases, dramatic failures have occurred because an existing system was reused in a different environment, and the developers were unaware of the assumptions made by the original developers.

6.2.5 Develop the Functional Architecture.

For small examples, the entire requirements specification can be written in a few pages. For real systems, this is not the case. For example, when the CoRE methodology was applied to the avionics of the C-130J aircraft, there were over 1600 monitored and controlled variables [15].

[3] In the SCR [10 and 11] and CoRE [14 and 16] methodologies, this is called the NAT (natural) relationship.

To be usable, a requirements specification for a system of any size must be organized in some manner.

The best organization may depend on how the requirements will be used. One structure may enhance the readability of the requirements, while another may make it easier to define a family of products. These goals may conflict with each other—organizing to support a product family may produce a specification that is harder to understand than one tailored for a single member of the family. Automated tools can help in this respect, producing different views of the same underlying specification as needed.

The functional architecture is developed through a process that recursively identifies the functions to be provided by the system, grouping together the functions that are logically related and likely to change together. Ideally, this structure would be used to organize the detailed system requirements so that they are readable and robust in the face of change.[4] In actuality, other factors, such as dealing with the safety requirements of the system or with implementation constraints, usually require that this structure be modified. These issues are discussed in section 6.2.6, which describes an iterative approach to modifying the functional architecture to deal with safety requirements and implementation constraints.

Another important aspect of the functional architecture is that it provides the traceability of requirements within the specification. Since the requirements are grouped recursively by function, the structure of the document automatically traces the lower-level requirements to the higher-level requirements.

This process starts with examining the preliminary set of system functions that were identified during the development of the operational concepts and grouping the closely related functions into the main functions of the system. The decomposition of each major function into the next level of functions proceeds naturally as the requirements are refined.

A simple way of depicting the system functions and their dependencies on each other is through a dependency diagram. Figure 14 shows a dependency diagram for a simple Thermostat.

Figure 14 reveals the main system functions of the Thermostat and the dependencies between them. The monitored variables are shown as arrows that do not originate from a function (Current Temperature and Operator Settings). Controlled variables are shown as arrows that do not terminate in a function (Heat Control and Operator Feedback), and data dependencies between functions are shown as arrows that both originate and terminate at functions (Mode, Desired Range, and Thermostat On/Off).

[4] This is very similar to the principles used to organize requirements in the CoRE methodology [14 and 16].

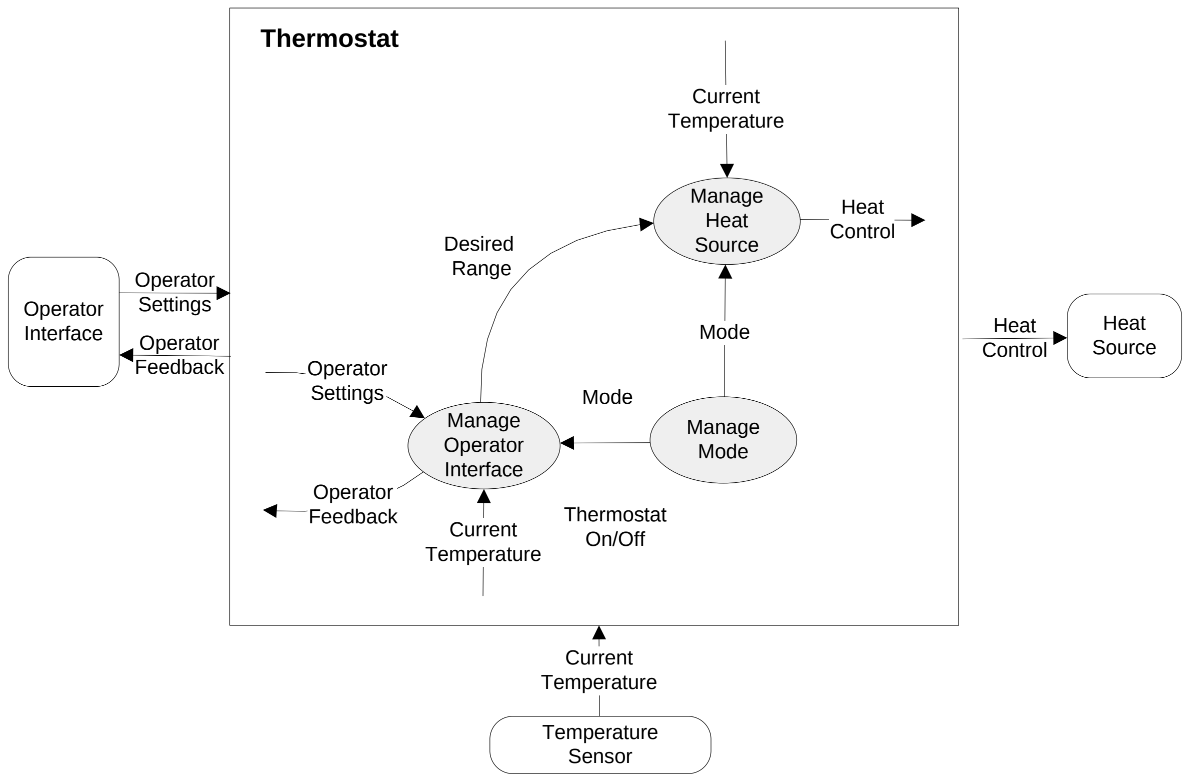

Figure 14. Thermostat Dependency Diagram

In reality, the dependencies between functions run in the opposite direction from that shown in figure 14. For example, the specification of the Manage Heat Source Function depends on the definition of the Desired Range internal variable specified in the Manage Operator Interface Function and the arrow should point from the Manage Heat Source Function to the Manage Operator Interface Function. However, drawing the dependency diagrams with the arrows pointing in the direction of dependency is nonintuitive and conflicts with the widely accepted use of data flow diagrams in which the arrows indicate the direction of data flow. In practice, drawing the arrows as shown is much more readily accepted and the practical impact on the specification is minor, even though technically, no data moves in a requirements specification.

The arrows flowing into a function indicate all the definitions needed by that function, and the arrows flowing out of a function show all the values defined by that function. To make the requirements specification robust in the face of change, the label on each arrow should represent a stable, high-level concept from the problem domain that is unlikely to change. Dependencies that are likely to change are hidden within functions. This creates firewalls that help prevent changes from rippling throughout the specification. Ideally, the more volatile a dependency, the further it should be pushed down in the function hierarchy.

Organizing the requirements this way can also facilitate the reuse of requirements, since the areas most likely to change between products are more likely to be localized. For a more systematic, planned approach to reuse, a commonality and variability analysis can be introduced into the overall process to try to identify which parts of the requirements are stable and which parts are likely to change [69]. The functionality analysis can then be conducted with that additional information as an input to the process.

As systems become larger, a single level of system functions may not be adequate. To deal with this, functions can be nested within functions, allowing arbitrarily large specifications to be created. Recursively decomposing functions into subfunctions, or child functions, in this way provides a natural framework to organize the requirements wherein logically related requirements are grouped together and the dependencies between groups of requirements are encapsulated into the internal variables.

The dependency diagrams and the definitions of the internal variables implicitly define the high-level system requirements for each function that is not at the lowest level (i.e., each nonterminal function). If desired, explicit high-level requirements can also be defined for each nonterminal function. However, care should be taken to not pull up details from the lower-level functions into the high-level functions in an effort to be thorough and precise. Doing this negates the entire purpose of developing the functional architecture, which is to provide a framework on which a complex specification is organized.

Figure 15 shows an example that provides a set of high-level requirements for the Thermostat Function. These simply state that the Thermostat will set the value of the Heat Control and the Operator Feedback controlled variables, but it does not define how they will be set. At this level of abstraction, all that is known is that their values are set by the Thermostat Function and range of those values (since their allowed values are defined as part of the environmental assumptions).

REQ-1 The Thermostat Function shall set the value of the Heat Control.

Rationale A primary function of the Thermostat is to turn the Heat Control on and off so as to maintain the Current Temperature in the Isolette within the Desired Temperature Range.

REQ-2 The Thermostat Function shall set the value of the operator feedback.

Rationale The Thermostat Function provides the Operator Feedback to the Operator Interface, which will use it to report the overall status of the Thermostat to the operator.

Figure 15. High-Level Requirements for the Thermostat Function

The precise details of how those variables will be set is provided in the lower levels of the function hierarchy. Attempts to provide more information at the highest level either introduce ambiguity or duplicate information defined at the lower levels. For example, on the surface, it might seem that REQ-1 could be better stated as

REQ-1	The Thermostat shall set the value of the Heat Control so as to maintain the Current Temperature in the Isolette within the Desired Temperature Range.

There are two problems with this. First, it is only true when the Thermostat is in its NORMAL mode of operation. During the INIT or FAILED mode, the Heat Control is to be set to the value of Off. Unfortunately, the system modes are not defined until the next lower level of refinement, so specifying precisely how the Heat Control is to be set requires using terms and definitions that are not available at this level.

Second, it confuses "what" the system is to do with "why" it is to do it; i.e., it is mixing rationale with the requirement itself (see section 6.2.11). The complete specification of how Heat Control is to be set is a surprisingly complex algorithm of its current mode, the Current Temperature, the Desired Temperature Range, and its previous state [1]. Maintaining the Current Temperature in the Isolette within the Desired Temperature Range is really an explanation of why this requirement exists, i.e., its rationale. Note that in figure 15 this information is provided to help the reader to understand why the requirement is included, but it is provided as rationale, not as the requirement.

6.2.6 Revise the Architecture to Meet Implementation Constraints.

Ideally, the functional architecture developed in the previous section could be used as the structure of the requirements specification, with the detailed behavior and performance requirements defined for each function. Unfortunately, components do fail with consequences for safety, legacy systems do exist that new systems must integrate with, and implementation constraints do affect the system architecture. Many of these concerns are addressed in the design of the system architecture, with the consequence that the final system architecture may not map directly to the logical architecture developed during functional analysis. Rather than trying to continuously map between the ideal functional architecture and the final architecture of the system, it is more practical to revise the functional architecture to take the most important of these constraints into account.

On the other hand, the original functional architecture was developed through an analysis of the problem domain without consideration of implementation constraints. Generally, the problem domain is less likely to change than are constraints imposed by the implementation. For this reason, the closer the final architecture can be kept to the original functional architecture, the more stable it is likely to be. It is also desirable to minimize as much as possible the differences between the ideal functional architecture and the final architecture.

This recommended practice describes an iterative process that starts from the ideal functional architecture developed in the preceding section and leads to a functional architecture that describes the final architecture of the system. This final architecture can then be used as the framework for organizing the detailed requirements.

For safety-critical systems, the process of modifying the functional architecture to accommodate implementation constraints is often driven by the need to achieve the very high levels of reliability dictated by the system safety process. For commercial avionics systems, this process is described in ARP 4761 [33]. The reader is directed to Handbook of Requirements Engineering Management [1] for an example of this recommended practice.

6.2.7 Identify the System Modes.

Leveson defines modes as distinct behaviors of the system [66]. For example, a system may respond to a stimuli, such as a button pressed one way during system power up, another way during a self test, and yet another way during normal operation. These behaviors are three modes of the system. The system modes can be very simple or very complex. They may be relevant to the entire system or to only part of the system. However, modes are always introduced to deal with discontinuities in the system behavior.

A system mode may or may not be explicitly displayed to a user of a system, but it is, by definition, visible, since the system will respond differently to stimuli while in different modes. In this sense, the modes are externally visible parts of the system behavior that need to be specified in the system requirements. Identifying the system modes also simplifies the specification of the system requirements by allowing the relationship between the monitored and controlled variables to be broken down into smaller pieces for each system mode. For these reasons, it is helpful to identify the major operating modes of the system before starting to write the detailed functional requirements.

While state transition diagrams are a popular way to specify the system modes [34], overly complex diagrams may be an indication that design decisions are being included that should be omitted from the requirements specification. Modes should be defined only to identify the externally visible discontinuities in the system behavior. This allows requirements to be written in the form, "If the system is in the initialization mode, the controlled variable shall be set to …" or "If the system is in failed mode, the controlled variable shall be set to …." This also makes it easier to determine if the requirements are complete and consistent.

6.2.8 Develop the Detailed Behavior and Performance Requirements.

The preceding recommended practices discussed how to create the system overview, define the system boundary, develop the operational concepts, define the environmental assumptions, begin the functional analysis, modify the functional analysis to accommodate implementation constraints, and identify the main system modes. With these activities partially or fully completed, the detailed behavior and performance requirements can finally be specified.

The detailed requirements define the behavior the system imposes on its environment. These can be specified as a mathematical relationship between the monitored and the controlled variables.[5] This relationship defines how the controlled variables will change in response to changes in the monitored variables. Due to the size and complexity of most systems, this is done by first

[5] In the SCR [10 and 11] and CoRE [14 and 16] methodologies, this is called the REQ (requirement) relationship.

defining at the lowest level of the functional architecture what the value of each controlled and internal variable would be for a perfect system. In CoRE [16], this is referred to as the ideal value function for that variable. The ideal value function is a complete chain of assigments that define how the controlled variables will change (in the ideal case) in response to changes in the monitored variables. However, this is usually too restrictive. For most systems, a range of values describing the ideal value are acceptable. To capture this, a tolerance describing the ideal value is specified for each controlled variable. This tolerance can be a simple constant or an arbitrarily complex function of the system state. Finally, the performance aspects of the system need to be specified. This is done by specifying a latency, i.e., a period of time, by which each controlled variable must complete its response. Again, the latency can be a simple constant or an aribtrarily complex function of the system state.[6]

To specify the ideal value function, the value of each controlled variable and each internal variable is specified at the lowest level of the functional architecture. To ensure this defines a "complete" chain of assignments from the montiored to the controlled variables, this is done for all possible combinations of system modes and conditions under which the assignment holds. To make sure the requirements are consistent, they must also be checked to ensure they do not overlap, i.e., that they specify two values for a variable for the same mode and condition. This can be made simpler if the requirements are specified in a tabular format, as in SCR [10 and 11] and CoRE [14 and 16].

Once the ideal value function is specified for each controlled variable and each internal variable, the requirements are completed by specifying the tolerance and latency for each controlled variable. Latency specifies the maximum lag between the time one or more monitored variables change value and an affected controlled variable must change its value.

When a controlled variable represents a numerical value (as opposed to a Boolean or enumerated value), the acceptable tolerance from the ideal value should also be specifed. For example, a controlled value that displays the altitude of the aircraft should include the delta above and below the actual altitude of the aircraft that can be tolerated. For Boolean and enumerated values, the tolerance is not usually meaningful and should not be stated.

6.2.9 Define the Software Requirements.

At some point, the system requirements must be allocated to hardware and software. As the use of general purpose processors has grown, more system requirements are allocated to software. In fact, the detailed system requirements and the software requirements often look remarkably alike. However, there are usually enough differences that the system requirements and the software requirements are treated as two separate specifications. This is largely because the inputs and outputs to the software do not exactly match the monitored and controlled variables that form the basis of the system requirements. This section describes an approach that provides a seamless transition from the system requirements to the software requirements in which the software requirements are created by extending the system requirements.

[6] Specification of the requirements as an ideal value function, tolerance, and latency is described in the CoRE methodology [14 and 16].

The basis for this approach is the four-variable model developed by Parnas and Madey [11] as part of the SCR methodology. The four-variable model clarifies the relationship between system and software functional requirements so that the software requirements can be specified as an addition to the system requirements. An overview of the four-variable model is shown in figure 16.

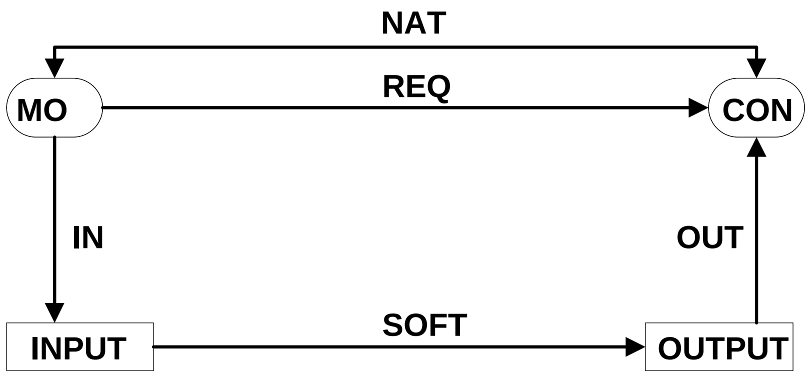

Figure 16. The Four-Variable Model

The monitored and controlled variables, the system environmental assumptions, and the system requirements have been discussed extensively already in section 23. The monitored variables (MON) consist of the quantities in the environment the system is to monitor and the controlled variables (CON) are the quantities in the environment the system will control. The environmental assumptions (NAT) are the relationships maintained by the environment on which the system depends, while the system requirements (REQ) define the relationship the system will maintain between the monitored and controlled variables.

The monitored and controlled variables cannot be sensed directly or controlled by the software program. Instead, the monitored variables must be provided by the system to the software program as input variables (INPUT) the program can read. In addition, the system must translate the output variables (OUTPUT) written by the software program into changes upon the environmentally controlled variables. The monitored and controlled variables are typically at a higher level of abstraction than the input and output variables. For example, while a monitored variable might be the altitude of an aircraft reported by a radar altimeter that is an integer between -20 and 2500 feet, the corresponding input variable might be an ARINC 429 bus word in which the altitude is encoded as a 17-bit 2's complement integer located in bits 13 to 28 of the bus word with the sign in bit 29 and the decimal point located between bits 15 and 16 representing a value between -8192.0 and +8191.875.

The IN relation defines the relationship of each input variable to one or more monitored variables. This consists of the definition of the input variable's ideal value function, its latency, and its accuracy. The ideal value function defines the ideal value of the input variable as a function of one or more monitored variables. The latency specifies the maximum time from when a monitored variable (that the input variable depends on) is changed until that change is indicated by the input variable. The accuracy specifies the amount by which the input variable's actual value may deviate from its ideal value.

In similar fashion, the OUT relationship defines the relationship of each controlled variable to one or more input variables. This includes a definition of the controlled variable's ideal value function, its latency, and its accuracy. The ideal value function defines the controlled variable's ideal value as a function of one or more output variables. The latency specifies the maximum time from when an output variable (that the controlled variable depends on) is changed until that change is indicated by the controlled variable. The accuracy specifies the amount by which the controlled variable's actual value may deviate from its ideal value.

Specification of the NAT, REQ, IN, and OUT relations implicitly bounds the allowed behavior of the software, shown in figure 16, as the SOFT relation, without specifying the design of the software. While the SOFT relation defines the true requirements for the software, the mapping between the system requirements specified in REQ and the software requirements in SOFT is not obvious. The system requirements and the software requirements are at different levels of abstraction and have different domains and ranges. One way to address these concerns is to extend the software requirements SOFT into three parts, IN', REQ', and OUT' [41], as shown in figure 17.

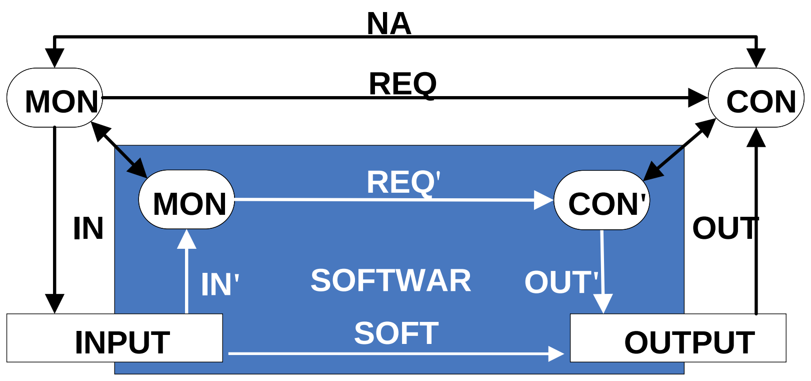

Figure 17. "Extended" Software Requirements

In figure 17, IN' is the inverse of IN and defines how to recreate an image of the monitored variables (MON') in software from the INPUT variables. Similarly, OUT' is the inverse of OUT and defines how the OUTPUT variables will change as an image of the controlled variables (CON') is changed in software. Just as REQ describes the relationship that must be maintained by the system between the monitored and controlled variables, REQ' describes the relationship that must be maintained by the software between the images of the monitored and controlled variables. The SOFT relation specifying the software requirements is replaced by IN', REQ', and OUT'.

The primary advantage of this extension is that it makes the relationship between the system requirements and the software requirements direct and straightforward. The ideal value function defined in the system requirements REQ maps directly onto an identical function in the software requirement REQ'. As a result, the software requirements actually consist of an addition to the system requirements defining the INPUT and OUTPUT variables and the IN, OUT, IN', and OUT' relations. Another advantage is that since IN' and OUT' are implemented in software (it may be helpful to think of them as the specification for hardware drivers), they provide a useful separation of concerns in the software requirements; REQ' will change as the system requirements change, while IN' and OUT' will change as the underlying hardware changes. This helps insulate the software implementing the system requirements from changes in the hardware platform.

It should be noted that that MON' and CON' are not the same as the system level variables represented by MON and CON. They are images of the monitored and controlled variables recreated in the software. Small differences in value are introduced by the hardware and software, and latencies are introduced between the images and the actual quantities in the environment. For example, the value of an aircraft's altitude recreated in software is always going to lag behind and differ somewhat from the aircraft's true altitude. These differences must be taken into account to ensure that the allowed latencies and tolerances specified for the controlled variables are met.

To do this, the software developer needs to confirm that the overall latency and accuracy specified for each controlled variable can be met given the latency and accuracy of the input and output variables and the computation time of the software.

While organizing the software requirements in this way makes them a straightforward extension of the system requirements, DO-178B [29] specifies that the software requirements should be organized into high- and low-level software requirements. Also, the high-level software requirements should be shown to comply with the system requirements (objective 1 of Table A-3) and the low-level software requirements should be shown to comply with the high-level software requirements (objective 1 of Table A-4). These are defined in DO-178B as:

- High-level requirements—Software requirements developed from analysis of system requirements, safety-related requirements, and system architecture.

- Low-level requirements—Software requirements derived from high-level requirements, derived requirements, and design constraints from which source code can be directly implemented without further information.

Which requirements discussed so far are high-level software requirements, which are low-level software requirements, and how can compliance of the software requirements with the system requirements be shown? These relationships are clarified in figure 18.

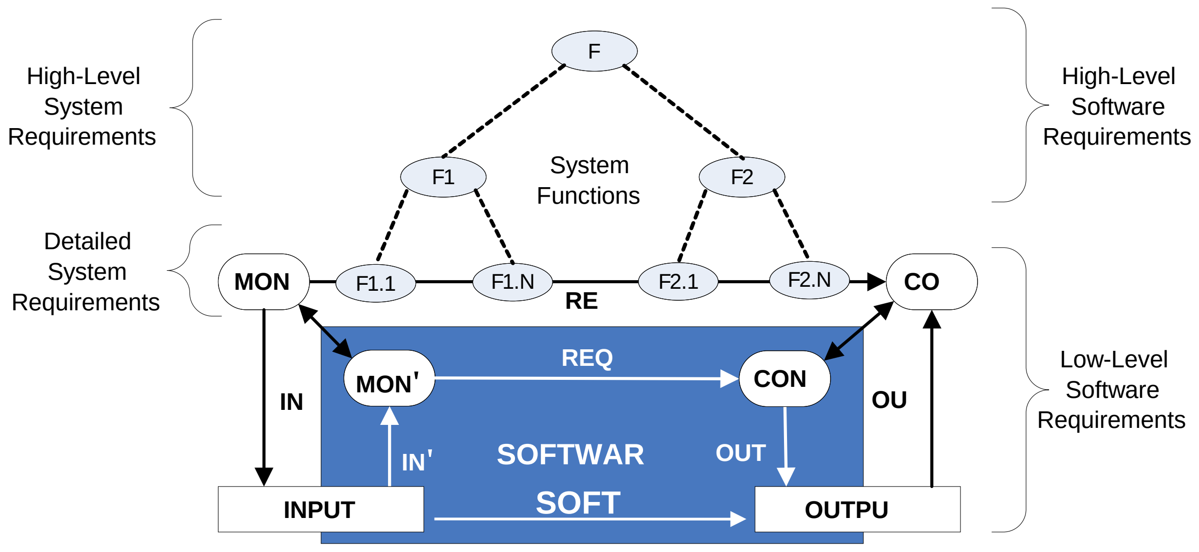
High-Level System Requirements
Detailed System Requirements
F
F1
F2
System Functions
High-Level Software Requirements
MON
F1.1
F1.N
F2.1
F2.N
CO
RE
MON'
REQ
CON
IN
OU
IN'
SOFTWAR
OUT
INPUT
SOFT
OUTPU
Low-Level Software Requirements

Figure 18. High- and Low-Level Software Requirements

In figure 18, the structure of the nested system functions are depicted above the four-variable model. The lowest-level system functions (F1.1 through F2.N) define the detailed system behavioral [7] requirements as discussed in section 6.2.8. In other words, each low-level system function defines its part of the REQ relation between the monitored and controlled variables. The higher-level system functions (F, F1, and F2) define the high-level system requirements, as discussed in section 6.2.6.

The low-level software requirements, i.e., those from which source code can be directly implemented without further information, include IN', REQ' (whose behavior portion is identical to REQ), and OUT'. In other words, the low-level software requirements consist of the detailed system requirements along with the definitions of IN' and OUT'.

Since the behavioral software requirements are identical to the behavioral system requirements, the obvious choice for the high-level software requirements are the system requirements that are not also low-level software requirements, i.e., the dependency diagrams and the definitions of the internal variables.

6.2.10 Allocating System Requirements to Subsystems.

To deal with the complexity of large systems, it is standard practice to divide the system into subsystems that can be developed independently and allocate the system requirements to these subsystems [5 and 54]. This section discusses how this can be done within the framework described here.

[7] The latency and tolerance defined for each controlled variable are end-to-end requirements and are more appropriately viewed as high-level requirements.

Consider the situation shown in figure 19. Here, the contractor specifying the requirements for system 1 (contractor 1) has identified the monitored and controlled variables and has completed the functional decomposition down through functions F1.1 through F1.N, F2, and F3.1 through F3.N. At this point (or perhaps even sooner), the contractor decides to spin off functions F1 and F3 as separate subsystems for development by independent subcontractors.

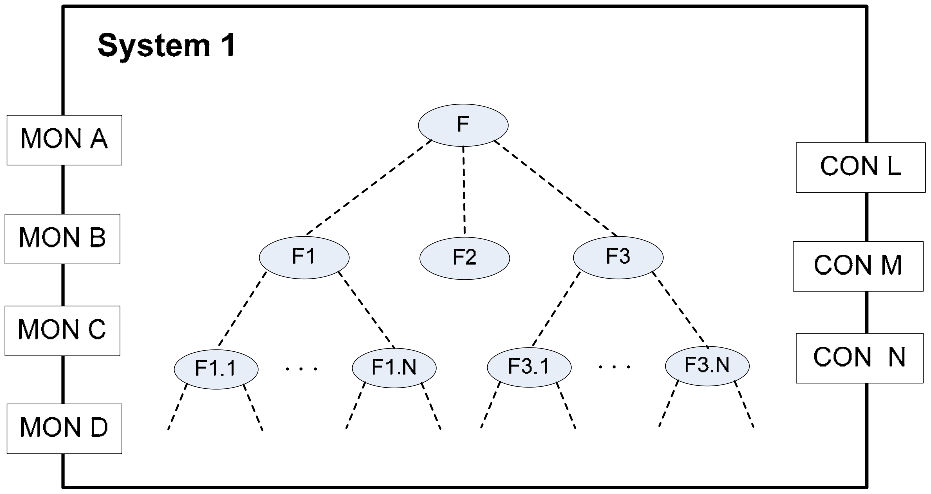

Figure 19. Functional Decomposition of System 1

Figure 20 shows an overview of the approach for doing this. Function F1 and its child functions have been moved to the specification for system 1.1, and function F3 and its child functions have been moved to the specification for system 1.3. The objective is to create a requirements specification of the entire system 1 for use by contractor 1, a requirements specification for system 1.1 for use by both contractor 1 and subcontractor 1.1, and a requirements specification for system 1.3 for use by both contractor 1 and subcontractor 1.3

The requirements for system 1 will be completed and maintained by contractor 1. It contains a specification for the entire system that the contractor may not wish to share with subcontractor 1.1 and subcontractor 1.3. However, the specification for system 1 will not contain all the detailed requirements for functions F1 and F3.[8] Instead, the detailed requirements for function F1 will be developed and refined cooperatively by contractor 1 and subcontractor 1.1. In a similar manner, the detailed requirements for function F3 will be developed cooperatively by contractor 1 and subcontractor 1.3.

The division of labor between the contractor the subcontractor in developing the requirements for either subsystem will likely depend on the confidence the contractor has in the subcontractor. If the contractor has high confidence in the subcontractor, the contractor may provide only the initial subsystem requirements specification and allow the subcontractor to do most of the work of refining it into a complete requirements specification. If the contractor does not have high confidence in the subcontractor, the contractor may elect to do all of the refinement and provide the subcontractor with a complete requirements specification. Typically, the subsystem

[8] It may contain the detailed requirements for function F2 that were not allocated to a separate subsystem.

specification will not be completed when the actual legal contract is signed, but the allocation of responsibilities for completing the subsystem requirements specification will be specified in the contract.

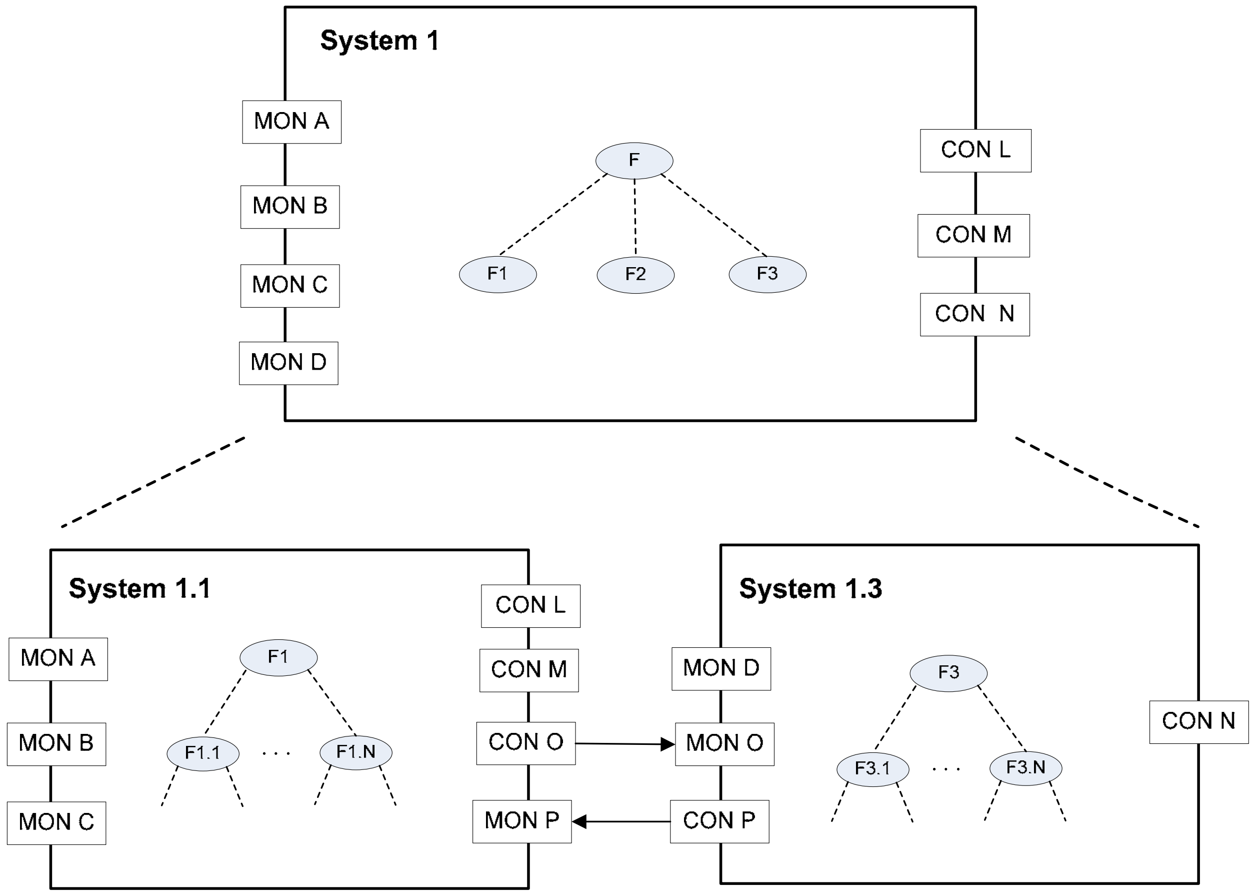

Figure 20. Decomposition of System 1 Into Subsystems

The purpose of the subsystem requirements specification is to provide a common specification that the contractor and the subcontractor share. However, it may not be necessary for the subcontractor to share all this information with the contactor. There may be details, such as algorithms or even some detailed requirements, that the subcontractor considers proprietary. This may result in a situation in which the subcontractor maintains some portion of the subsystem specification separately.

What information should be in the initial requirements specification for the subsystem? As shown in figure 20, function F1 and its child functions have been moved into subsystem 1.1, and function F3 and its child functions have been moved into subsystem 1.3. This implies that the function hierarchy for F1 and F3, and whatever requirements have been developed for that hierarchy, are now part of the specification of subsystem 1.1 and subsystem 1.3. This ensures that no work done by the contractor is discarded. If the decomposition into subsystems was planned, the contractor may not have developed any child functions for F1 and F3.

In addition, figure 20 also shows that a copy of functions F1 and F3 (but not their child functions) is retained along with their high-level requirements in the specification for system 1. Duplication of the high-level requirements for F1 and F3 in the specification of system 1 is done deliberately to provide traceability and continuity between system 1 and subsystem 1.1—each high-level requirement of subsystem 1.1 and subsystem 1.3 traces directly to an identical requirement in system 1.

At this point, the requirements specification for subsystem 1.1 and subsystem 1.3 consist of little more than the functions copied from system 1 and their high-level requirements. As discussed previously, a complete requirements specification consists of much more than just a list of shall statements. The next step is to fill in the information needed to make the subsystem specifications a complete requirements specification in its own right. This consists of adding to each subsystem specification a system overview, monitored and controlled variables, operational concepts, and environmental assumptions.

Of course, the subsystem requirements specification is not yet complete. At this point, the contractor and subcontractor are ready to continue the development of the subsystem requirements until the detailed behavior and performance requirements are specified. As discussed earlier, this task is performed by both the contractor and the subcontractor, though the division of labor between them will probably be based on the contractor's confidence in the subcontractor.

The first task in this process is completion of the functional decomposition of the subsystem, as discussed in section 6.2.5. Once the major functions of the subsystem have been identified, the preliminary system safety assessment should be resumed for the subsystem (section 6.2.6) to identify any derived system safety requirements. The functional architecture of the subsystem should be modified to take into account these safety requirements or other implementation constraints. The major subsystem modes should be identified (section 6.2.7), the detailed behavior and performance requirements should be developed (section 6.2.8), and if the subsystem is to be implemented in software, the software requirements should be developed (section 6.2.9).

Finally, developing the detailed behavior and performance requirements for each subsystem will include specifying the latency and tolerance for each controlled variable in the subsystem. These should also be reviewed to ensure they are consistent with the system specification. The sum of the latencies for the controlled variables in the subsystems should be at least less than the end-to-end latency specified for the corresponding controlled variable in parent system.[9]

6.2.11 Provide Rationale.

Requirements document what a system will do. Design documents how the system will do it. Rationale documents why a requirement exists or is written the way it is. Rationale should be included whenever there is something in the requirement that may not be obvious to the reader or

[9] If physical architecture implementing the system introduces additional latencies (for example, if a bus delay is introduced between two subsystems), these latencies need to included in this check.

when it is necessary to help the reader to understand why the requirement exists. Rationale should be added during all phases of requirements definition. Rationale can consist of free text, pointers to trade studies, other documents, papers, or book citations. The system goals (section 6.2.1) are often a useful source of rationale.

Hooks and Farry claim that providing rationale is the single most effective way of reducing the cost and improving the quality of requirements [59]. Rationale can reduce the amount of time required to understand a requirement by providing background information on why the requirement exists. Since most requirements specifications are read multiple times, investing the effort to provide the rationale should save time and money over the long run. Rationale can also help the readers avoid making incorrect assumptions, the cost of which grows rapidly as development proceeds towards implementation. Rationale can also decrease the cost of maintenance by documenting why certain choices or values were selected. It can also make the development of variations of the same product cheaper by providing insight into the impact of changing the requirements and assumptions. Finally, rationale can decrease the cost of educating new employees by providing them with background information about why the system behaves the way it does.

Providing rationale can also improve the quality and reduce the cost of creating the requirements. Coming up with the rationale for a bad requirement or assumption can be difficult. Forcing the specifier to think about why the requirement is necessary, or why the assumption is being made, will often improve the quality of the requirement. It may even lead to the elimination of the requirement as the writer realizes it is unnecessary or is actually an implementation detail. This eliminates an additional constraint on the developers and the cost of verifying an unnecessary requirement. Rationale also makes it easier to keep the requirements focused on what the system will do by providing a place for background information.

Rationale should not document what the system must do; i.e., rationale should not include requirements or be used as an excuse for writing poor requirements. Rationale is not contractually binding and does not need to be implemented. If anything included in the rationale is essential to the required behavior of the system, it must be specified as a requirement.

Rationale should be provided for

- every requirement for which the reason for why the requirement exists is not obvious.
- every environmental assumption for which the reason why the assumption is included is not obvious.
- every value or range in a requirement or assumption explaining why that particular value or range was specified.

The best rationale is short and to the point. Rather than copying a trade study justifying a particular requirement, summarize the relationship of the trade study to the requirement and cite the trade study in the rationale.

Rationale should be collected along with the development of the requirement or assumption it is explaining. This ensures that the justification is captured while the author is thinking about it. It also helps to ensure that there is time for others to review and question the rationale.

6.3 MAPPING THE RECOMMENDED PRACTICES TO THE ISSUES.

This section summarizes how the recommended practices discussed in section 6.2 address the key safety and certification issues identified in section 5.3. This information is summarized in table 11, which shows which recommended practices address which issues. Practices that directly address an issue, i.e., those that are essential or play a major role in addressing the issue, are indicated by a solid circle (●). Practices that indirectly address an issue, i.e., those that are helpful but that do not play a major role, are indicated by an open circle (○).

For example, the recommended practice of "Develop the Operational Concepts" directly addresses (●) the issue of "Validation of Requirements," since it provides early identification of the functions to be provided by the system and how those functions will be invoked by the operators and other systems. In contrast, the recommended practice of "Develop the System Overview" indirectly addresses (○) the issue of "Validation of Requirements," since it provides an overview and context of the system that is helpful, but is not essential, for validating the system requirements.

This mapping of recommended practices to the issues is admittedly subjective, and different individuals would probably arrive at different mappings. However, it does provide useful information in several ways. First, it provides at least a rough indication of how well different issues are addressed by the recommended practices. For example, the issue of "Development by Multiple Entities" is addressed very strongly by the recommended practices. This is because the recommended practices become even more critical when several entities are involved.

On the other hand, some issues are not addressed particularly well. For example, the issue of "Problems Posed by MBD Tools" is not addressed at all. This reflects the decision made at the start of Phase 2 to avoid a focus on specific technologies, such as MBD and OOD (section 6.1). In contrast, although the issue of "Validation of Requirements" stated as models is a technology-specific issue, it is addressed quite strongly. This is because development of a traditional set of requirements provides a means to validate the model. In similar fashion, although the issue of "Specification of the HMI Requirements" was ruled out of scope in section 6.1, the recommended practice of "Develop the Operational Concepts" directly (if not completely) addresses this issue by forcing the developers to explore how the system will be used by its operators.

Providing a full explanation of how each recommended practice addresses each issue would essentially repeat (several times) the discussion found in section 6.2 and the Requirements Engineering Management Handbook [1]. A very brief explanation of how each recommended practice addresses an issue is presented in tables 12 through 22. For a more detailed explanation, the reader is directed to the discussion of the specific recommended practice in section 6.2 or the Requirements Engineering Management Handbook [1].

Table 11. Benefits of the Recommended Practices

Recommended Practices	Develop the System Overview	Identify the System Boundary	Develop the Operational Concepts	Identify the Environmental Assumptions	Develop the Functional Architecture	Revise Architecture to Meet Constraints	Identify the System Modes	Develop Detailed System Requirements	Develop the Software Requirements	Allocate System Functions to Subsystems	Provide Rationale
Issues From the Industry Survey											
Clarify the Relationship of SW Requirements to SW Design					●	●			●		
Traceability and Configuration Management in MBD and OOD		○									
Problems Posed by the Use of MBD Tools											
Validation of Requirements Stated as Models	○	●	●	●	○	○	●	●			
Traceability and Configuration Management of Requirements		○			●	●			●	●	
Clarify the Relationship of System and Software Requirements		●		○	●	●	●	●	●		
Issues From the Literature Search											
Defining the System Boundary	○	●		●		○					
Documenting the Environmental Assumptions	○	○	○	●		○					●
Development by Multiple Entities	●	●	●	●	●	●				●	●
Providing Intent and Rationale				○							●
Validation of Requirements	○		●				○	○			●
Managing Change and Complexity		○	○		●	●			●	○	●
Specification of HMI Requirements			●	○							●
Verification of Requirements							●	●	●		

SW = Software ● – Directly Addresses ○ – Indirectly Addresses

Table 12. Benefits—Develop System Overview

Issues From the Industry Survey		
Clarify the Relationship of SW Requirements to SW Design		
Traceability and Configuration Management in MBD and OOD		
Problems Posed by the Use of MBD Tools		
Validation of Requirements Stated as Models	○	Provides a description of the context of the model.
Traceability and Configuration Management of Requirements		
Clarify the Relationship of System and Software Requirements		
Issues From the Literature Search		
Defining the System Boundary	○	Provides the initial definition of the system boundary.
Documenting the Environmental Assumptions	○	Provides the initial identification the external entities and their interactions with the system.
Development by Multiple Entities	●	Provides overview and context for each developer.
Providing Intent and Rationale		
Validation of Requirements	○	Provides overview and context of the system.
Managing Change and Complexity		
Specification of HMI Requirements		
Verification of Requirements		

SW = Software ● – Directly Addresses ○ – Indirectly Addresses

Table 13. Benefits—Identify System Boundary

Issues From the Industry Survey		
Clarify the Relationship of SW Requirements to SW Design		
Traceability and Configuration Management in MBD and OOD	○	Defines what is inside and outside the system.
Problems Posed by the Use of MBD Tools		
Validation of Requirements Stated as Models	●	Monitored and controlled variables correspond to the model inputs and outputs
Traceability and Configuration Management of Requirements	○	Defines what is inside and outside the system.
Clarify the Relationship of System and Software Requirements	●	Provides foundation for the four-variable model relating system to software.
Issues From the Literature Search		
Defining the System Boundary	●	Primary purpose of this recommended practice.
Documenting the Environmental Assumptions	○	Defines what is inside and outside the system.
Development by Multiple Entities	●	Provides clear definition of each (sub)system boundary.
Providing Intent and Rationale		
Validation of Requirements		
Managing Change and Complexity	○	Defines what is inside and outside the system.
Specification of HMI Requirements		
Verification of Requirements		

SW = Software ● – Directly Addresses ○ – Indirectly Addresses

Table 14. Benefits—Develop the Operational Concepts

Issues From the Industry Survey		
Clarify the Relationship of SW Requirements to SW Design		
Traceability and Configuration Management in MBD and OOD		
Problems Posed by the Use of MBD Tools		
Validation of Requirements Stated as Models	●	Provides early identification of functions to be provided by model.
Traceability and Configuration Management of Requirements		
Clarify the Relationship of System and Software Requirements		
Issues From the Literature Search		
Defining the System Boundary		
Documenting the Environmental Assumptions	○	Operational concepts will often reveal environmental dependencies.
Development by Multiple Entities	●	Operational concepts describe how each system will interact with other systems.
Providing Intent and Rationale		
Validation of Requirements	●	Provides early identification of functions to be provided by system.
Managing Change and Complexity	○	Operational concepts help to identify overlooked behavior early, minimizing later change.
Specification of HMI Requirements	●	Operational concepts define how system will be used by operators.
Verification of Requirements		

SW = Software ● – Directly Addresses ○ – Indirectly Addresses

Table 15. Benefits—Identify the Environmental Assumptions

Issues From the Industry Survey		
Clarify the Relationship of SW Requirements to SW Design		
Traceability and Configuration Management in MBD and OOD		
Problems Posed by the Use of MBD Tools		
Validation of Requirements Stated as Models	●	Identifies constraints on the inputs the model depends on for correct execution.
Traceability and Configuration Management of Requirements		
Clarify the Relationship of System and Software Requirements	○	Identifies constraints the inputs the software depends on for correct execution.
Issues From the Literature Search		
Defining the System Boundary	●	Extends the definition of the system boundary by providing type, range, and units of the monitored and controlled variables.
Documenting the Environmental Assumptions	●	This is the primary purpose of this recommended practice.
Development by Multiple Entities	●	Essential to defining the contract between each (sub)system and its environment.
Providing Intent and Rationale	○	Identification of environmental assumptions encourages providing rationale for why the assumption exits.
Validation of Requirements		
Managing Change and Complexity		
Specification of HMI Requirements	○	Identification of environmental assumptions may reveal dependencies on operator behavior.
Verification of Requirements		

SW = Software ● – Directly Addresses ○ – Indirectly Addresses

Table 16. Benefits—Develop the Functional Architecture

Issues From the Industry Survey		
Clarify the Relationship of SW Requirements to SW Design	●	The final system functional architecture maps directly to the software architecture as part of the extended four-variable model.
Traceability and Configuration Management in MBD and OOD		
Problems Posed by the Use of MBD Tools		
Validation of Requirements Stated as Models	○	The final system functional architecture organizes the requirements against which the model will be checked.
Traceability and Configuration Management of Requirements	●	The final system functional architecture provides the traceability of the requirements.
Clarify the Relationship of System and Software Requirements	●	The final system functional architecture maps directly to the software architecture as part of the extended four-variable model.
Issues From the Literature Search		
Defining the System Boundary		
Documenting the Environmental Assumptions		
Development by Multiple Entities	●	The final system functional architecture provides the basis for allocation into subsystems.
Providing Intent and Rationale		
Validation of Requirements		
Managing Change and Complexity	●	Organizing requirements into logically related groups with low coupling is the primary mechanism for managing change and complexity.
Specification of HMI Requirements		
Verification of Requirements		

SW = Software ● – Directly Addresses ○ – Indirectly Addresses

Table 17. Benefits—Revise the Functional Architecture

Issues From the Industry Survey		
Clarify the Relationship of SW Requirements to SW Design	●	The final system functional architecture maps directly to the software architecture as part of the extended four-variable model.
Traceability and Configuration Management in MBD and OOD		
Problems Posed by the Use of MBD Tools		
Validation of Requirements Stated as Models	○	The final system functional architecture organizes the requirements against which the model will be checked.
Traceability and Configuration Management of Requirements	●	The final system functional architecture provides the traceability of the requirements.
Clarify the Relationship of System and Software Requirements	●	The final system functional architecture maps directly to the software architecture as part of the extended four-variable model.
Issues From the Literature Search		
Defining the System Boundary	○	Revising the functional architecture to deal with implementation constraints may result in changes to the system boundary.
Documenting the Environmental Assumptions	○	Revising the functional architecture to deal with implementation constraints may introduce new environmental assumptions.
Development by Multiple Entities	●	The final system functional architecture provides the basis for allocation into subsystems.
Providing Intent and Rationale		
Validation of Requirements		
Managing Change and Complexity	●	Organizing requirements into logically related groups with low coupling is the primary mechanism for managing change and complexity.
Specification of HMI Requirements		
Verification of Requirements		

SW = Software ● – Directly Addresses ○ – Indirectly Addresses

Table 18. Benefits—Identify the System Modes

Issues From the Industry Survey		
Clarify the Relationship of SW Requirements to SW Design		
Traceability and Configuration Management in MBD and OOD		
Problems Posed by the Use of MBD Tools		
Validation of Requirements Stated as Models	●	System modes are essential to defining the detailed system requirements against which the model will be validated.
Traceability and Configuration Management of Requirements		
Clarify the Relationship of System and Software Requirements	●	System modes map directly to the software modes as part of the extended four-variable model.
Issues From the Literature Search		
Defining the System Boundary		
Documenting the Environmental Assumptions		
Development by Multiple Entities		
Providing Intent and Rationale		
Validation of Requirements	○	Using the system modes to partition the detailed system requirements makes them easier to understand and validate.
Managing Change and Complexity		
Specification of HMI Requirements		
Verification of Requirements	●	System modes are used in checking that the requirements are complete and consistent.

SW = Software   ● – Directly Addresses   ○ – Indirectly Addresses

Table 19. Benefits—Develop Detailed System Requirements

Issues From the Industry Survey		
Clarify the Relationship of SW Requirements to SW Design		
Traceability and Configuration Management in MBD and OOD		
Problems Posed by the Use of MBD Tools		
Validation of Requirements Stated as Models	●	Model is validated against the detailed system requirements.
Traceability and Configuration Management of Requirements		
Clarify the Relationship of System and Software Requirements	●	System requirements map directly to software requirements as part of the extended four-variable model.
Issues From the Literature Search		
Defining the System Boundary		
Documenting the Environmental Assumptions		
Development by Multiple Entities		
Providing Intent and Rationale		
Validation of Requirements	○	Specifying the detailed system requirements using tabular formats makes validation easier.
Managing Change and Complexity		
Specification of HMI Requirements		
Verification of Requirements	●	Detailed requirements are stated in a format that makes them easy to check for completeness and consistency.

SW = Software   ● – Directly Addresses   ○ – Indirectly Addresses

Table 20. Benefits—Develop the Software Requirements

Issues From the Industry Survey		
Clarify the Relationship of SW Requirements to SW Design	●	The final system functional architecture maps directly to the software design as part of the extended four-variable model.
Traceability and Configuration Management in MBD and OOD		
Problems Posed by the Use of MBD Tools		
Validation of Requirements Stated as Models		
Traceability and Configuration Management of Requirements	●	The system requirements map directly to the software requirements as part of the extended four-variable model.
Clarify the Relationship of System and Software Requirements	●	The system requirements map directly to the software requirements as part of the extended four-variable model.
Issues From the Literature Search		
Defining the System Boundary		
Documenting the Environmental Assumptions		
Development by Multiple Entities		
Providing Intent and Rationale		
Validation of Requirements		
Managing Change and Complexity	●	The extended four-variable model organizes the software requirements into platform-independent and platform-dependent pieces.
Specification of HMI Requirements		
Verification of Requirements	●	The extended four-variable model makes much of the verification of the software requirements against the system requirements immediate.

SW = Software ● – Directly Addresses ○ – Indirectly Addresses

Table 21. Benefits—Allocate System Functions to Subsystems

Issues From the Industry Survey		
Clarify the Relationship of SW Requirements to SW Design		
Traceability and Configuration Management in MBD and OOD		
Problems Posed by the Use of MBD Tools		
Validation of Requirements Stated as Models		
Traceability and Configuration Management of Requirements	●	This practice outlines a process for allocating requirements to subsystems and maintaining traceability between them.
Clarify the Relationship of System and Software Requirements		
Issues From the Literature Search		
Defining the System Boundary		
Documenting the Environmental Assumptions		
Development by Multiple Entities	●	This is the primary purpose of this recommended practice.
Providing Intent and Rationale		
Validation of Requirements		
Managing Change and Complexity	○	This practice defines a process for dividing a complex system into subsystems based on the functional architecture.
Specification of HMI Requirements		
Verification of Requirements		

SW = Software ● – Directly Addresses ○ – Indirectly Addresses

Table 22. Benefits—Provide Rationale

Issues From the Industry Survey		
Clarify the Relationship of SW Requirements to SW Design		
Traceability and Configuration Management in MBD and OOD		
Problems Posed by the Use of MBD Tools		
Validation of Requirements Stated as Models		
Traceability and Configuration Management of Requirements		
Clarify the Relationship of System and Software Requirements		
Issues From the Literature Search		
Defining the System Boundary		
Documenting the Environmental Assumptions	●	Encourages including no only what the environmental assumption is, but also why it is important, and why specific values are specified.
Development by Multiple Entities	●	Provides important context for each entity, explaining why a requirement, assumption, or value was specified.
Providing Intent and Rationale	●	This is the primary purpose of this recommended practice.
Validation of Requirements	●	Directly supports requirements validation by explaining why the requirement, assumption, or value was specified.
Managing Change and Complexity	●	Provides important context for the design or maintenance team, explaining why a requirement, assumption, or value was specified.
Specification of HMI Requirements	●	Provides important context for the design team, explaining why an HMI-related requirement, assumption, or value was specified.
Verification of Requirements		

SW = Software ● – Directly Addresses ○ – Indirectly Addresses

7. FINDINGS AND RECOMMENDATIONS.

The main findings from the literature search are presented in section 7.1. The main findings from the industry survey are presented in section 7.2. The recommendations documented in the Handbook for Requirements Engineering Management [1] are presented in section 7.3. Finally, recommendations for future work are presented in section 7.4.

7.1 FINDINGS FROM THE LITERATURE SEARCH.

Several best practices specifically developed for REM were discovered through the literature search (section 2). Each of these has its own strengths and weaknesses. For example, SCR [10-13] and CoRE [14-16] provide a strong conceptual model for relating system and software requirements, use tables for ease of presentation and validation, and encourage the specification of environmental assumptions, an often overlooked aspect of requirements. RSML [17 and 18] introduces additional tabular formats and graphical notations that make requirements specifications accessible to a still wider audience. SpecTRM [7-9, 19, and 20] adds to RSML techniques for capturing the intent, or rationale, of requirements, and for developing requirements for human factors.

The use of UML [21 and 22] for REM was also explored. While UML does not appear to provide the same level of rigor as the approaches described above, it is widely known, is supported by many commercial tools, and has its own useful notations for the elicitation and presentation of requirements. In particular, use cases [23-25] appear to be a very effective technique for the early validation of requirements and for specifying the operational concepts that define how operators and other systems will interact with the system.

A more recent advance is the use of modeling tools, such as MATLAB Simulink [26 and 27] and Esterel Technologies SCADE Suite [28], for the specification of requirements. Like the UML tools, these have strong support from their vendors and are being used increasingly for specifying system and software requirements. However, critics of these tools assert that they are actually being used to specify detailed design, or even source code, not requirements.

The literature search also included a brief review of regulatory documents for software development and examined how current regulations impact the practice of REM. In particular, a careful reading of DO-178B identified a lack of consistency in defining whether the requirements for a mid-level software component should be treated as high-level or low-level requirements (section 2.1).

Several issues were identified in the literature search that an REM process should address (section 5.2). These include clearly defining the system boundary, documenting all environmental assumptions, supporting development by multiple entities, providing intent and rationale, supporting the validation of requirements, managing change and complexity, specifying HMI requirements, and supporting the verification of requirements.

7.2 FINDINGS FROM THE INDUSTRY SURVEY.

The industry survey (section 3) found that virtually none of the best practices identified in the literature search are widely practiced. Even the object-oriented approaches, probably the most widely known of the best practices, were mentioned by only a few respondents. The overwhelming majority of respondents appear to be capturing requirements in English text, tables, and diagrams using either Telelogic DOORS or a word processor. Modeling tools, such as MATLAB Simulink and Esterel Technologies SCADE, are starting to be used for the modeling of requirements, but there does not seem to be a consensus on what level of requirements these models represent.

The single greatest concern of the respondents was how to interpret and follow the guidance provided in DO-178B related to REM. Twenty-nine comments, some of them quite detailed, expressed concerns with how to manage requirements within a DO-178B process. Most of these seemed to focus on the relationship of software requirements to software design, how to manage traceability and configuration management of requirements, and the relationship of system requirements to software requirements.

Several respondents provided comments related to MBD. While models are increasingly being used to represent requirements, there does not appear to be a consensus on what level of requirements (system, high-level software, low-level software, or hardware) these models represent, or even if they represent requirements. The main concerns appear to be with problems posed by the use of tools, validation of the models, the relationship of software requirements to software design in MBD, and issues of traceability and configuration management in MBD.

Concerns raised with regard to OOD were similar to many of those raised for MBD and certification issues, in general. These included questions the handling of traceability and configuration management and the relationship of software requirements to design. However, a closer reading of the comments reveals that many concerns are really questions of how to implement the requirements, rather than issues in REM, per se.

Surprisingly few survey comments directly cited safety concerns, though the case could be made that virtually all the comments have implications for system safety. The main safety-related need appears to be for better guidance on what should be done differently for safety requirements, as opposed to other requirements, and for better integration between ARP 4754 [32] and DO-178B [29].

Only five comments were raised with regard to what the certification authorities could do differently. Four respondents stated that the review and acceptance of requirements is not administered uniformly by the certification authorities. Two respondents expressed concerns about how the certification authorities would view or approve requirements provided in something other than the typical document format, e.g., a model that that requires a commercial modeling tool to be effectively viewed and navigated.

While 50 major and 60 minor safety and certification issues were identified in the industry survey, these were ultimately distilled down into 6 main issues (section 5.1). These included clarifying the relationship of software requirements and software design, traceability and configuration management in MBD and OOD, problems posed by the use of MBD tools, validation of requirements stated as models, traceability and configuration management of requirements, and clarifying the relationship of system and software requirements.

7.3 RECOMMENDATIONS DOCUMENTED IN THE REM HANDBOOK.

As a result of the findings, the recommendation made at the end of Phase 1 was to develop an incremental approach for improved REM that would allow practitioners to progress from their current state of practice by gradually introducing more of the best practices identified in the literature search. This approach is documented in the Requirements Engineering Management Handbook [1] developed in Phase 2 of the project.

The Requirements Engineering Handbook identifies the following 11 main-level recommended practices and 90 sublevel recommended practices (the numbers shown in parentheses are the number of sublevel recommended practices) gleaned from the literature how to collect, write, and organize requirements:

1. Develop the System Overview (7)
2. Identify the System Boundary (7)
3. Develop the Operational Concepts (14)
4. Identify the Environmental Assumptions (5)
5. Develop the Functional Architecture (7)
6. Revise the Architecture to Deal with Implementation Constraints (10)
7. Identify the System Modes (3)
8. Develop the Detailed Behavioral and Performance Requirements (9)
9. Define the Software Requirements (11)
10. Allocate the System Requirements to Subsystems (10)
11. Provide Rationale (7)

Introducing any of these practices into an REM process that does not already include them should improve the quality of the process. The recommended practices were mapped against the issues identified in the industry survey and the literature search and found to significantly address almost all issues raised. Issues not completely covered are identified in section 7.4.

To validate and illustrate the recommended practices in the Handbook, two running examples were developed. While the examples might appear to suggest a specific style and format, their true purpose will illustrate and clarify the recommended practices and their benefits. There are many different formats that could be used to satisfy the same objectives. In applying the Handbook, the emphasis should be on incorporating the main-level recommended practices into an existing REM process, not on following the particular style of specification chosen for the examples.

7.4 RECOMMENDATIONS FOR FUTURE WORK.

There are several areas for further work in REM. One of the most surprising findings from the industry survey is that 14 years after the publication of DO-178B, there still seem to be many questions on how to specify and manage requirements within a DO-178B process. Most of these concerns seem to relate to questions regarding what are high-level requirements, what are low-level requirements, and how do they relate to the software design.

An approach for addressing these issues, the extended four-variable model, was discussed in the recommended practice of section 6.2.9 that overlaps the upper architecture of the software requirements with the architecture of the system requirements. In this approach, the externally visible behavior of the system is not completely specified until the lowest level of the system requirements is completed. As a consequence, the externally visible behavior of the software is also not completely specified until the corresponding level of refinement is reached in the software requirements. However, some survey comments suggest that some companies and certification authorities are requiring that the externally visible behavior of the software be completely specified at the top-most level of software requirements. The intent of this may be (quite reasonably) to ensure that the software developers are not introducing low-level requirements that affect the externally visible behavior of the system unless they are identified as derived requirements. However, this makes no sense if the architecture of the system and software requirements overlap, as discussed in section 6.2.9. Some details of the externally visible behavior will be specified at the intermediate levels of the software requirements.

One recommendation is that additional guidance beyond DO-178B be developed on the relationship of high-level requirements, low-level requirements, and their relationship to the software architecture. Hopefully, this guidance would consider the recommended practices described in the Handbook of Requirements Engineering [1]. Due to the apparent difficulty in clarifying this issue, it is strongly recommended that any guidance be validated and illustrated by applying it to at least one example similar to those used in the Handbook.

REM issues related to specific technologies, such as MBD or OOD, were not addressed directly in this study to focus on developing a broader set of recommended practices. An obvious direction for further work would be to address the REM issues raised in these technologies more directly. Based on the survey responses, the greatest need appears to be in relationship to MBD. Any work in this area should be coordinated with that being done by RTCA SC-205/WG-71 committee.

Another topic that was only partially addressed in the study is the specification of HMI requirements. There is considerable evidence (see section 5.2.8) that automation surprises, or mode confusion, play a key role in system safety, and that the cause of automation surprises goes much deeper than just the physical interface presented to the operator. Common sources of mode confusion include logic that:

- does not match the natural workflow of the operator,
- places unnecessary and unevenly distributed demands on the operator,
- behaves inconsistently in difference modes, or

- provides poor feedback on the status and behavior of automation.

Work on identifying how HMI requirements should be developed, specified, validated, and managed is a recommended direction for future work.

As systems continue to grow in complexity, it will eventually be necessary to replace the informal notations used to describe the system architecture today with more rigorous models that allow the system architects to specify precisely what components exist in a system, how they are connected to each other, and how they are mapped to the physical platform. An example of such a notation is the Architecture Analysis and Design Language (AADL) [70]. A formal architectural model provides a natural framework in which to embed the requirements for each subsystem and allows the subsystem developers to better understand the relationships between the subsystems. If the requirements were specified formally, the nominal behavior of the overall system could be verified through simulation or analysis. Such an architectural model could also be annotated with performance and failure rates to support the automated analysis of performance, safety, reliability, and security characteristics of the system.

8. REFERENCES.

1. Lempia, D. and Miller, S., "Requirements Engineering Management Handbook," FAA report DOT/FAA/AR-08/32, 2009.

2. Brooks, F., "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer*, April 1987.

3. Boehm, B., *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

4. Davis, A., *Software Requirements (Revised): Object, Functions, and States*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

5. Leveson, N., *Safeware: System Safety and Computers*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1995.

6. Lutz, R., "Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems," *IEEE International Symposium on Requirements Engineering*, San Diego, CA, January 1993.

7. Howard, J. and Anderson, P., "The Safety Risk of Requirements Incompleteness," *Proceedings of the 20th International System Safety Conference (ISSC 2002)*, Denver, CO, August 2002.

8. Lee, G., Howard, J., and Anderson, P., "Safety-Critical Requirements Specification Using SpecTRM," *Proceedings of the 2nd Meeting of the U.S. Software System Safety Working Group*, February 2002.

9. Howard, J., "Preserving System Safety Across the Boundary Between System Integrator and Software Contractor," *Proceedings of the SAE World Congress 2004*, Detroit, MI, March 2004.

10. Van Schouwen, A., "The A-7 Requirements Model: Re-examination for Real-Time Systems and an Application to Monitoring Systems," Technical Report 90-276, Queens University, Hamilton, Ontario, 1990.

11. Parnas, D., and Madey, J., "Functional Documentation for Computer Systems Engineering (Volume 2)," Technical Report CRL 237, McMaster University, Hamilton, Ontario, September 1991.

12. Heitmeyer, C., Labaw, B., and Kiskis, D., "Consistency Checking of SCR-Style Requirements Specification," *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, March 1995, pp. 56-65.

13. Heitmeyer, C., Kirby, J., and Labaw, B., "Automated Consistency Checking of Requirements Specification," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Vol. 5, No. 3, July 1996, pp. 231-261.

14. Faulk, S., Brackett, J., Ward, P., and Kirby, Jr., J., "The CoRE Method for Real-Time Requirements," *IEEE Software*, Vol. 9, No. 5, September 1992, pp. 22-33.

15. Faulk, S., Finneran, L., Kirby, J., Shah, S., and Sutton, J., "Experience Applying the CoRE Method to the Lockheed C-130J Software Requirements," *Proceedings of the Ninth Annual Conference on Computer Assurance*, Gaithersburg, MD, June 1994, pp. 3-8.

16. Faulk, S., Finneran, L., Kirby, J., and Moini, A., "Consortium Requirements Engineering Guidebook," Technical Report SPC-92060-CMS, Software Productivity Consortium, Herndon, VA, December 1993.

17. Leveson, N., Heimdahl, M., Hildreth, H., and Reese, J., "Requirements Specifications for Process-Control Systems," *IEEE Transactions on Software Engineering*, Vol. 20, No. 9, September 1994, pp. 684-707.

18. Leveson, N., Heimdahl, M., Hildreth, H., and Reese, J., "TCAS II Collision Avoidance System (CAS) System Requirements Specification Change 6.00," Federal Aviation Administration, U.S. Department of Transportation, March 1993.

19. Leveson, N., "Intent Specifications: An Approach to Building Human-Centered Specifications," *IEEE Transactions on Software Engineering*, Vol. 26, No. 1, September 2000, pp. 15-35.

20. Leveson, N., Reese, J., and Heimdahl, M., "SpecTRM: A CAD System for Digital Automation," *Proceedings of 17th Digital Avionics System Conference (DASC98)*, Seattle, November 1998.

21. Booch, G., Rumbaugh, J., and Jacobson, I., *The Unified Modeling Language User Guide*, Addison Wesley, Reading, MA, 1999.

22. Fowler, M., *UML Distilled, A Brief Guide to the Standard Object Modeling Language, Third Edition*, Addison Wesley, Reading MA, September 2003.

23. Cockburn, A., *Writing Effective Use Cases*, Addison-Wesley, Boston, MA, 2001.

24. Alexander, I. and Zink, T., "An Introduction to Systems Engineering With Use Cases," *IEEE Computer and Control Engineering*, December 2002.

25. Leffingwell, D. and Widrig, D., *Managing Software Requirements*, Addison-Wesley, Boston, MA, 2003.

26. Dabney, J., and Harmon, T., *Mastering Simulink*, Pearson Prentice Hall, Upper Saddle River, NJ, 2004.

27. The Mathworks, http://www.mathworks.com

28. Esterel Technologies, http://www.esterel-technologies.com

29. "Software Considerations in Airborne Systems and Equipment Certification," DO-178B, RTCA, Washington, DC, December 1992.

30. "Software Considerations in Airborne Systems and Equipment Certification," DO-248B, RTCA, Washington, DC, October 2001.

31. "Guidelines for Communication, Navigation, Surveillance, and Air Traffic Management (CNS/ATM) Systems Software Integrity Assurance," DO-278, RTCA, Washington, DC, March 2002.

32. "Certification Considerations for Highly-Integrated or Complex Aircraft Systems," ARP 4754, SAE International, November 1996.

33. "Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment," ARP 4761, SAE International, December 1996.

34. Harel, D., "Statecharts: A Visual Formalism for Complex Systems," *Science of Computer Programming*, Vol. 8, 1987, pp. 231.

35. Harel, D., et. al., "STATEMATE: A Working Environment for the Development of Complex Reactive Systems," *IEEE Transactions on Software Engineering*, Vol. 16, No. 4, April 1990, pp. 403-414.

36. Harel, D. and Naamad, A., "The STATEMATE Semantics of Statecharts," *ACM Transactions on Software Engineering and Methodology*, Vol. 5, No. 4, October 1996, pp. 293-333.

37. Telelogic StateMate, Product Description, http://www.telelogic.com/products/statemate

38. Telelogic Rhapsody, Product Description, http://www.telelogic.com/products/rhapsody

39. Berry, G. and Gonthier, G., “The Synchronous Programming Language Esterel: Design, Semantics, and Implementation,” *Science of Computer Programming*, Vol. 19, 1992, pp. 87-152.

40. Miller, S. and Tribble, A., “Extending the Four-Variable Model to Bridge the System-Software Gap,” *20th Digital Avionics Systems Conference (DASC01)*, Daytona Beach, Florida, October 14-18, 2001.

41. Miller, S. and Hoech, K., “Specifying the Mode Logic of a Flight Guidance System in CoRE,” Technical Report WP97-2011, Rockwell Collins, Cedar Rapids, IA, August 1997.

42. Miller, S., “Specifying the Mode Logic of a Flight Guidance System in CoRE and SCR,” *Second Workshop on Formal Methods in Software Practice (FMSP98)*, Clearwater Beach, Florida, March 4-5, 1998.

43. Ratan, V., Partridge, K., Reese, J., and Leveson, N., “Safety Analysis Tools for Requirements Specifications,” http://www.safeware-eng.com/index.php/publications.

44. Leveson, N., Heimdahl, M., and Reese, J., “Designing Specification Languages for Process Control Systems: Lessons Learned and Steps to the Future,” *Proceedings Seventh ACM SIGSOFT Symposium on the Foundations on Software Engineering,* LNCS 1687, September 1999, pp. 127-145.

45. Thompson, J., Heimdahl, M., and Miller, S, “Specification-Based Prototyping for Embedded Systems,” *Proceedings Seventh ACM SIGSOFT Symposium on the Foundations on Software Engineering*, LNCS 1687, September 1999, pp. 163-179.

46. Miller, S., Tribble, A., Carlson, T., and Danielson, E., “Flight Guidance System Requirements Specification,” Technical report CR-2003-212426, NASA Langley Research Center, June 2003, available at http://techreports.larc.nasa.gov/ltrs/refer/2003/cr/NASA-2003-cr212426.refer.html.

47. Cavada, R., Cimatti, A., Jochim, C., Kreighren, G., Olivetti, E., Pistore, M., Roveri, M., and Tchaltsev, A., “The NuSMV 2.4 User Manual,” 2005, available at http://nusmv.irst.itc.it.

48. Owre, S., Rushby, J., Shankar, N., and Henke, F., “Formal Verification for Fault-Tolerant Architectures: Prolegomena to the Design of PVS,” *IEEE Transactions on Software Engineering*, Vol. 21, No. 2, February 1995, pp. 107-125.

49. Johsi, A., Miller, S., and Heimdahl, M., “Mode Confusion Analysis of a Flight Guidance System Using Formal Methods,” *Proceedings of the 22nd Digital Avionics Systems Conference (DASC’03)*, Indianapolis, IN, October 12-16, 2003.

50. Tribble, A., Lempia, D., and Miller, S., "Software Safety Analysis of a Flight Guidance System," *Proceedings of the 21st Digital Avionics Systems Conference (DASC'02)*, Irvine, CA, October 27-31, 2002.

51. Tribble, A., Miller, S., and Lempia, D., "Software Safety Analysis of a Flight Guidance System," NASA Contractor Report CR-2004-213004, March 2004. Available at http://techreports.larc.nasa.gov/ltrs/dublincore/2004/cr/NASA-2004-cr213004.html.

52. Tribble, A. and Miller, S., "Safety Analysis of Software Intensive Systems," *IEEE Aerospace and Electronic Systems*, Vol. 19, No. 10, October 2004, pp. 21-26.

53. Miller, S., Heimdahl, M., and Tribble, A., "Proving the Shalls," *Proceedings of FM 2003: the 12th International FME Symposium,* Pisa, Italy, September 8-14, 2003.

54. Stevens, R., Jackson, B., and Arnold, S., *Systems Engineering: Coping With Complexity*," Prentice-Hall, London, 1998.

55. "Integrated Modular Avionics (IMA) Development, Guidance, and Certification Considerations," DO-297, RTCA, Washington, DC, November 8, 1992.

56. "Design Guidance for Integrated Modular Avionics," ARINC 651-1, November 1997.

57. Rushby, J., "Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance," Technical Report, SRI International, Menlo Park, CA, March 1999.

58. Conmy, P., Nicholson, M., McDermid, J., "Safety Assurance Contracts for Integrated Modular Avionics," *Australian Computer Society, Inc.*, 2003.

59. Hooks, I. and Farry, K., "Customer Centered Products: Creating Successful Products through Smart Requirements Management," *AMACOM American Management Association*, New York, New York, 2001.

60. Lions, J., "Ariane 5 Flight 501 Failure: Report by the Inquiry Board," Paris, France, July 19, 1996, available at http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html.

61. "Flight Guidance System Approval," Joint Advisory Circular AC/ACJ 25.1329, February 14, 2002.

62. Sarter, N. and Woods, D., "How in the World did I Ever get Into That Mode? Mode Error and Awareness in Supervisory Control," *Human Factors*, Vol. 37, No. 1, 1995, pp. 5-19.

63. Hughes, D. and Dornheim, M., "Automated Cockpits Special Report, Parts I & II," *Aviation Week & Space Technology*, January 30-February 6, 1995.

64. Billings, C., *Aviation Automation: the Search for a Human Centered Approach*, Lawrence Erlbaum Associates, Inc., Mahwah, NJ, 1997.

65. Commercial Aviation Safety Team, Final Report of the Loss of Control JSAT: Results and Analysis, P. Russell and J. Pardee, Co-chairs, December 15, 2000.

66. Leveson, N., et al., "Analyzing Software Specifications for Mode Confusion Potential," *Proceedings of a Workshop on Human Error and System Development*, Glasgow, Scotland, March 1997, pp. 132-146.

67. Leveson N., "Designing Automation to Reduce Operator Errors," *Proceedings of the IEEE Systems, Man, and Cybernetics Conference*, October 1997.

68. Sarter, N., Woods, D., and Billings, C., in *Handbook of Human Factors/Ergonomics*, 2nd Ed., "Automation Surprises," G. Salvendy, ed., John Wiley & Sons, New York, NY, 1997.

69. Coplien, J., Hoffman, D., and Weiss, D., "Commonality and Variability in Software Engineering," *IEEE Software*, Vol. 20, No. 6, November 1998.

70. "Architecture Analysis and Design Language (AADL)," SAE AS5506, November 2006, available at http://www.aadl.info.

9. GLOSSARY OF THE FOUR-VARIABLE MODEL.

MON—Monitored variables, which are the values in the environment that the system monitors and responds to.

CON—Controlled variables, which are the values in the environment that the system controls.

INPUT—Input variables, which are the values in hardware through which the software senses the monitored variables (MON).

OUTPUT—Output variables, which are the values in hardware through which the software changes the controlled variables (CON).

NAT—A mathematical relationship between the controlled variables (CON) and the monitored variables (MON) specifying the "natural" system constraints imposed by the environment (the environmental assumptions).

REQ—A mathematical relationship between the monitored variables (MON) and controlled variables (CON) specifying the system requirements defining how the controlled variables should respond to changes in the monitored variables.

IN—A mathematical relationship between the monitored variables (MON) and the input variables (INPUT) specifying how the input variables respond to changes in the monitored variables.

OUT—A mathematical relationship between the output variables (OUT) and the controlled variables (CON) specifying how the controlled variables respond to changes in the output variables.

SOFT—A mathematical relationship between input variables (INPUT) and the output variables (OUTPUT) explicitly specifying the software requirements.

MON'—An image of the monitored variables (MON) created in software.

CON'—An image of the controlled variables (CON) created in software.

IN'—A mathematical relationship between the input variables (INPUT) and the software image of the monitored variables (MON') specifying the portion of the extended software requirements defining how the image of the monitored variables should be recreated in software.

REQ'—A mathematical relationship between the image of the monitored variables created in software (MON') and the image of the controlled variables created in software (CON') specifying the portion of the extended software requirements corresponding to the overall system requirements (REQ).

APPENDIX A—ADDITIONAL REFERENCES

Abbott, R. and Moorhead, D., “Software Requirements and Specifications: A Survey of Needs and Languages,” *The Journal of Systems and Software*, Vol. 2, 1981, pp. 297-316.

Alford, M., “Strengthening the Systems Engineering Process,” *Engineering Management Journal*, Vol. 4, No. 1, 1992, pp. 7-14.

Antón, A., “Goal-Based Requirements Analysis,” 2*nd IEEE International Conference on Requirements Engineering,* 1996.

Bailin, S., “An Object-Oriented Requirements Specification Method,” *Communications of the ACM*, Vol. 32, No.5, 1989, pp. 608-623.

Balzer, R. and Goldman, N., “Principles of Good Software Specification and Their Implications for Specification Languages,” *National Computer Conference*, 1981, pp. 393-400.

Barnard, J. and Price, A., “Managing Code Inspection Information,” *IEEE Software*, Vol. 11, No. 2, 1994, pp. 59-69.

Basili, V. and Perricone, B., “Software Errors and Complexity: An Empirical Investigation,” *Communications of the ACM*, Vol. 27, No. 1, 1984, pp. 42-52.

Basili, V. and Rombach, D., “The TAME Project: Towards Improvement-Oriented Software Environments,” *IEEE Transactions on Software Engineering*, Vol. SE-14, No. 6, 1988, pp. 758-773.

Basili, V. and Weiss, D., “Evaluation of a Software Requirements Document by Analysis of Change Data,” *Proceedings of the 5th International Conference on Software Engineering*, 1981, pp. 314-321.

Berry, D., “The Importance of Ignorance in Requirements Engineering,” *The Journal of Systems and Software*, Vol. 28, No. 2, 1995, pp. 179-184.

Boehm, B. and In, H., “Identifying Quality-Requirements Conflicts,” *IEEE Software*, Vol. 13, No. 2, 1996, pp. 25-35.

Boehm, B. and Ross, R., “Theory-W Software Project Management: Principles and Examples,” *IEEE Transactions on Software Engineering*, Vol. 15, No. 7, 1989, pp. 902-916.

Boehm, B., Gray, T., and Seewaldt, T., “Prototyping Versus Specifying: A Multiproject Experiment,” *IEEE Transactions on Software Engineering*, Vol. 10, No. 3, 1984, pp. 290-302.

Boehm, B., Bose, P., Horowitz, E., and Lee, M., “Software Requirements as Negotiated Win Conditions,” *1st IEEE International Conference on Requirements Engineering*, 1994, pp. 74-83.

Boehm, B., "Verifying and Validating Software Requirements and Design Specifications," *IEEE Software*, Vol. 1, No. 1, 1984, pp. 75-88.

Boehm, B., "Industrial Software Metrics Top 10 List," *IEEE Software*, Vol. 4, No. 5, 1987, pp. 84-85.

Bowen, J. and Hinchey, M., "Seven More Myths of Formal Methods," Technical report PRG-TR-7-94, Oxford University Computing Laboratory, Oxford, England, 1994.

Brackett, J., "Software Requirements," Technical Report SEI-CM-19-1.2, Software Engineering Institute, 1990.

Brooks, Jr., F., "No Silver Bullet—Essence and Accidents of Software Engineering," *IEEE Computer*, Vol. 20, No. 4, 1987, pp. 10-19.

Bubenko, Jr., J., "Challenges in Requirements Engineering," *Second IEEE International Symposium on Requirements Engineering*, March 1995, pp. 160-162.

Byrne, J., and Barlow, T., "Structured Brainstorming: A Method for Collecting User Requirements," *Human Factors and Ergonomics Society 37th Annual Meeting*, 1993, pp. 427-432.

Carlshamre, P. and Karlsson, J., "A Usability-Oriented Approach to Requirements Engineering," *2nd IEEE International Conference on Requirements Engineering*, 1996, pp. 145-152.

Ceri, S., "Requirements Collection and Analysis in Information Systems Design," *Proceedings of the IFIP 10th World Computer Congress*, 1986, pp. 205-214.

Christel, M. and Kang, K.,"Issues in Requirements Elicitation," Technical Report CMU/SEI-92-TR-12, Software Engineering Institute, Carnegie Mellon University, 1992.

Chung, L., "Representation and Utilization of Non-Functional Requirements for Information System Design," *Proceedings of the third international conference on Advanced information systems engineering*, 1991, pp. 5-30.

Conmy, P. and McDermid, J., "High Level Failure Analysis for Integrated Modular Avionics," *6th Australian Workshop on Safety Critical Systems and Software*, Vol. 3, 2001.

Conmy, P., Nicholson, M., and McDermid, J., "Safety Assurance Contracts for Integrated Modular Avionics," *8th Australian Workshop on Safety Critical Systems and Software*, Vol. 33, 2003.

Cordes, D. and Carver, D., "Knowledge Base Applications With Software Engineering: A Tool for Requirements Specifications," *Proceedings of the First International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, June 2-3, 1988.

Cordes, D. and Carver, D., "Evaluation Method for User Requirements Documents," *Information Software and Technology*, Vol. 31, No. 4, 1989, pp. 181-188.

Cordes, D. and Carver, D., "An Object-Based Requirements Modeling Method," *Journal of the American Society for Information Science*, Vol. 43, No. 1, 1992, pp. 62-71.

Costello, R. and Liu, D., "Metrics for Requirements Engineering," *The Journal of Systems and Software*, Vol. 29, No. 1, 1995, pp. 39-63.

Curtis, B., Krasner, H., and Iscoe, N., "A Field Study of the Software Design Process for Large Systems," *Communications of the ACM*, Vol. 31, No. 11, 1988, pp. 1268-1287.

Dalal, S., Horgan, J., and Kettenring, J., "Reliable Software and Communication: Software Quality, Reliability, and Safety," *15th International Conference on Software Engineering*, IEEE Computer Society Press, 1993, pp. 425-435.

Dasarathy, B., "Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods of Validating Them," *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 1, 1985, pp. 80-86.

Davis, A., "A Taxonomy for the Early Stages of the Software Development Life Cycle," *The Journal of Systems and Software*, Vol. 8, No. 4, 1988.

Davis, A., Overmyer, S., Jordan, K., Caruso, J., Dandashi, F., Dinh, A., Kincaid, G., Ledeboer, G., Reynolds, P., Sitaram, P., and Theofanos, M., "Identifying and Measuring Quality in a Software Requirements Specification," *1st International Symposium on Software Metrics*, 1993, pp. 141-152.

Davis, A., "A Comparison of Techniques for the Specification of External System Behavior," *Communications of the ACM*, Vol. 31, No. 9, 1988, pp. 1098-1115.

Davis, A., "Operational Prototyping: A New Development Approach," *IEEE Software*, Vol. 9, No. 5, 1992, pp. 70-78.

Davis, G., "Strategies for Information Requirements Determination," *IBM Systems Journal*, Vol. 21, No. 1, 1982, pp. 4-30.

de Boer, R., "Computer Aided Requirements—Vehicle for Software System Development," Vakgroep Informatica, Faculteit der Economische Wetenschappen, Erasmus Universiteit Rotterdam, 1993.

Dion, R., "Process Improvement and the Corporate Balance Sheet," *IEEE Software*, Vol. 10, No. 4, 1993, pp. 28-35.

Dobson, J., Blyth, A., Chudge, J., and Strens, M., "The ORDIT Approach to Requirements Identification," *International Computer Software and Applications Conference*, 1992, pp. 356-361.

Doolan, E., “Experience With Fagan's Inspection Method,” *Software-Practice and Experience*, Vol. 22, No. 2, 1992, pp. 173-182.

Dutertre, B. and Sorea, M., “Modeling and Verification of a Fault-Tolerant Real-Time Startup Protocol Using Calendar Automata,” *FORMATS/FTRTFT’04*, Grenoble, France, 2004

Easterbrook, S. and Nuseibeh, B., “Managing Inconsistencies in an Evolving Specification,” *Second IEEE International Symposium on Requirements Engineering*, 1995, pp. 48-55.

El Emam, K. and Madhavji, N., “A Field Study of Requirements Engineering Practices in Information Systems Development,” *Second IEEE International Symposium on Requirements Engineering*, 1995, pp 68-80.

El Emam, K. and Madhavji, N., “Measuring the Success of Requirements Engineering Processes,” In *Second IEEE International Symposium on Requirements Engineering*, 1995, pp. 204-211.

Fagan, M., “Design and Code Inspections to Reduce Errors in Program Development,” *IBM Systems Journal*, Vol. 12, No. 3, 1976, pp. 182-211.

Fagan, M., “Advances in Software Inspections,” *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 7, 1986, pp. 744-751.

Fickas, S. and Nagarajan, P., “Critiquing Software Specifications,” *IEEE Software*, Vol. 5, No. 6, 1988, pp. 37-47.

Finkelstein, A., “Re-Use of Formatted Requirements Specifications,” *Software Engineering Journal*, Vol. 3, No. 5, 1988, pp. 186-197.

Finkelstein, A., Easterbrook, S., Kramer, J., and Nuseibeh, B., “Requirements Engineering Through Viewpoints,” Technical report, Imperial College, London, England, 1992.

Finkelstein, A., Gryce, C., Lewis-Bowen, J., ”Relating Requirements and Architectures: A Study of Data-Grids,” *Journal of Grid Computing*, 2004.

Fraser, M., Kumar, K., and Vaishnavi, V., ”Informal and Formal Requirements Specification Languages: Bridging the Gap,” *IEEE Transactions on Software Engineering*, Vol. 17, No. 5, 1991, pp. 454-466.

Gause, D. and Weinberg, G., *Exploring Requirements: Quality Before Design*, Dorset House Publishing, New York, NY, 1989.

Giannakopoulou, D. and Penix, J., “Component Verification and Certification in NASA Missions,” *4th lCSE Workshop on Component-Based Software Engineering,* 2004.

Goldsack, S. and Finkelstein, A., “Requirements Engineering for Real-Time Systems,” *Software Engineering Journal*, Vol. 6, No. 3, 1991, pp. 101-115.

Gomaa, H., “The Impact of Prototyping on Software System Engineering,” *System and Software Requirements Engineering*, 1990, pp. 543-552.

Gomaa, H. and Scott, D., “Prototyping as a Tool in the Specification of User Requirements,” *Proceeding of the 5th International Conference on Software Engineering*, 1981, pp. 333-339.

Goodrich, V. and Olfman, L., “An Experimental Evaluation of Task and Methodology Variables for Requirements Definition Phase Success,” *23rd Annual Hawaii International Conference on Systems Sciences*, 1990, pp. 201-209.

Grady, R. and Van Slack, T., “Key Lessons in Achieving Widespread Inspection Use,” *IEEE Software*, Vol. 11, No. 4, 1994, pp. 46-57.

Greenwell, W. and Knight, J., “Framing Analysis of Software Failure With Safety Cases,” 2006.

Haag, S., Raja, M., and Schkade, L., “Quality Function Deployment: Usage in Software Development,” *Communications of the ACM*, Vol. 39, No. 1, 1996, pp. 41-49.

Hall, A., “Seven Myths of Formal Methods,” *IEEE Software*, Vol. 7, No. 5, 1990, pp. 11-19.

Hammer, M., “Reengineering Work: Don't Automate, Obliterate,” *Harvard Business Review*, July-August, 1990, pp. 104-112.

Harker, S. and Eason, K., “The Change and Evolution of Requirements as a Challenge to the Practice of Software Engineering,” *IEEE International Symposium on Requirements Engineering*, 1993, pp. 266-272.

Hauser, J. and Clausing, D., “The House of Quality,” *Harvard Business Review*, May-June issue, 1988, pp. 63-73.

Herman, J., “Requirements Engineering and Analysis, Workshop Proceeding,” Technical report, CMU/SEI-91-TR-30, Carnegie Mellon University, Pittsburgh, PA, 1991.

Hissam, S., “Predictible Assembly From Certifiable Components,” *CrossTalk—The Journal of Defense Software Engineering*, June 2004.

Holbrook, III, H., “A Scenario-Based Methodology for Conducting Requirements Elicitation,” *ACM SIGSOFT Software Engineering Notes*, Vol. 15, No. 1, 1990.

Hooks, I. and Farry, K., “*Customer Centered Products: Creating Successful Products Through Smart Requirements Management*,” AMACOM American Management Association, New York, NY, 2001.

Hsia, P., Davis, A., and Kung, D., “Status Report: Requirements Engineering,” *IEEE Software,* Vol. 10, No. 6, November/December 1993, pp. 75-59.

Hughes, J., O'Brien, J., Rodden, T., Rouncefiled, M., and Sommerville, I., “Presenting Ethnography in the Requirements Process,” *Second IEEE International Symposium on Requirements Engineering*, 1995, pp. 27-34.

Hunter, M. and Van Landingham, R., “Listening to the Customer Using QFD,” *Quality Progress*, 1994, pp. 55-59.

Ibanez, M. and Rempp, H., “European User Survey Analysis,” Report USV_EUR, version 2.1, January 1996.

Jackson, M., *Software Requirements & Specifications*, Addison-Wesley Publishing Company, Boston, MA, 1995.

Jacobson, I., Christersson, M., Jonsson, P., and Overgaard, G., *Object-Oriented Software Engineering—A Use Case Driven Approach*, Addison-Wesley Publishing Company, Boston, MA, 1992.

Jacobson, I., Ericsson, M., and Jacobson, A., *The Object Advantage-Business Process Reengineering With Object Technology*, Addison-Wesley Publishing Company, Boston, MA, 1995.

Jarke, M. and Pohl, K., “Requirements Engineering in 2001: (Virtually) Managing a Changing Reality,” *Software Engineering Journal*, November 1994, pp. 257-266.

Jeffrey, H. and Putman, A., “Relationship Definition and Management: Tools for Requirements Analysis,” *The Journal of Systems and Software*, 1994, pp. 277-294.

Johnson, R., “Trade-off Analysis of Consumer Values,” *Journal of Marketing Research*, 1974, pp. 121-127.

Jones, C., *Systematic Software Development Using VDM*, Prentice-Hall International, London, 1986.

Kabodian, A., “*The Customer Is Always Right!*” McGraw-Hill, Inc., Seäven, 1996.

Kaindl, H., “The Missing Link in Requirements Engineering,” *ACM SIGSOFT*, 1993, pp. 30-39.

Karlsson, J. and Lindvall, M., “Experiences With Requirements Specification Evolution,” Memo 95-02, ASLAB, Linköping University, Sweden, 1994.

Karlsson, J., “Towards a Strategy for Software Requirements Selection,” Licentiate Thesis 513, Department of Computer and Information Science, Linköping University, Sweden, 1995.

Karlsson, J., “Software Requirements Prioritizing,” *2nd IEEE International Conference on Requirements Engineering*, 1996, pp. 100-116.

Karlsson, J. and Ryan, K., “Supporting the Selection of Software Requirements,” *8th International Workshop on Software Specification and Design*, 1996, pp. 146-149.

Karlsson, J., Olsson, S., and Ryan, K., "Improved Practical Support for Large-Scale Requirements Prioritizing," *Requirements Engineering Journal*, 1997.

Karlsson, J. and Ryan, K., "Prioritizing Requirements Using a Cost-Value Approach," *IEEE Software*, September/October issue, 1997, pp. 67-74.

Karlsson, J., "Managing Software Requirements Using Quality Function Deployment," *Software Quality Journal*, Vol. 6, No. 4, December 1997, pp. 311-326.

Karlsson, J., Wohlin, C., and Regnell, B., "An Evaluation of Methods for Prioritizing Software Requirements," *Journal of Information and Software Technology*, Vol. 39, No. 14-15, 1998, pp. 939-947.

Keller, S., Kahn, L., and Panara, R., *Specifying Software Quality Requirements With Metrics*, IEEE Computer Society Press, 1990.

King, B., *Better Designs in Half the Time: Implementing QFD in America*, Goal QPC, Inc., Salem, NH, 1987.

Kovitz, B., *Practical Software Requirements*, Manning Publications, 1998.

Kotonya, G., and Sommerville, I., "Viewpoints for Requirements Definition," *Software Engineering Journal*, Vol. 7, No. 6, 1992, pp. 375-387.

Kramer, J., Potts, V., and Whitehead, K., "Tool Support for Requirements Analysis," *Software Engineering Journal*, Vol. 3, No. 3, 1988, pp. 86-96.

Kuwana, E., "Representing Knowledge in Requirements Engineering: An Empirical Study of What Software Engineers Need to Know," *IEEE International Symposium on Requirements Engineering*, San Diego, CA, January 1993, pp. 273-276.

Lano, R., *A Structured Approach for Operational Concept Formulation*, IEEE Computer Society Press, Washington, DC, 1990.

LaBudde, E., "Why is Requirements Engineering Underused?" *IEEE Software*, Vol. 11, No. 2, March/April 1994.

Lee, M., "Foundations of the WinWin Requirements Negotiation System," Technical report USC/CSE-96-501, Center for Software Engineering, University of Southern California, Los Angeles, CA, 1996.

Lee, M. and Boehm, B., "The WinWin Requirements Negotiation System: A Model-Driven Approach," Technical report USC/CSE-96-502, Center for Software Engineering, University of Southern California, Los Angeles, CA, 1996.

Lee, Y., Rachlin, E., and Scandura, P., Jr., "Handbook for Ethernet-Based Aviation Databuses: Certification and Design Considerations," FAA report DOT/FAA/AR-05/54, November 2005.

Leite, J., “A Survey on Requirements Analysis,” Technical report RTP-071, Department of Information and Computer Science, University of California at Irvine, Irvine, CA, 1987.

Leveson, N. and Weiss, K., “Making Embedded Software Reuse Practical and Safe,” *AMC SIGSOFT Software Engineering Notes*, Vol. 29, No. 6, November 2004.

Lindstrom, D., “Five Ways to Destroy a Development Project,” *IEEE Software*, Vol. 10, No. 5, September/October 1993, pp. 55-58.

Lindvall, M., “A Study of Traceability in Object-Oriented Systems Development,” Licentiate Thesis 462, Department of Computer and Information Science, Linköping University, Sweden, 1994.

Loucopoulos, P. and Champion, R., “Knowledge-Based Support for Requirements Engineering,” *Information and Software Technology*, Vol. 31, No. 3, April 1989, pp. 124-135.

Loucopoulos, P. and Champion, R., “Concept Acquisition and Analysis for Requirements Specification,” *Software Engineering Journal*, Vol. 5, No. 2, March 1990, pp. 116-124.

Loucopoulos, P. and Karakostas, V., *System Requirements Engineering*, McGraw-Hill, 1995.

Lougee, H., “Reuse and DO-178B Certified Software: Beginning with Reuse Basics,” *CrossTalk—The Journal of Defense Software Engineering*, December 2004, pp. 23-28.

Lubars, M., Potts, C., and Richter, C., “A Review of the State of the Practice in Requirements Modeling,” *IEEE International Symposium on Requirements Engineering*, January 1993, pp. 2-14.

Lutz, R., “Analyzing Software Requirements Errors in Safety-Critical, Embedded Systems,” *IEEE International Symposium on Requirements Engineering*, January 1993, pp. 126-133.

Löwgren, J., *Human-Computer Interaction: What Every System Developer Should Know*, Studentlitteratur, Lund, Sweden, 1992.

Macaulay, L., *Requirements Engineering*, Springer, London, 1996.

MacLean, A., Young, R., and Moran, T., “Design Rationale: The Argument Behind the Artifact,” *CHI '89 Proceedings*, 1989, pp. 247-252.

Maiden, N. and Rugg, G., “ACRE: Selecting Methods for Requirements Acquisition,” *Software Engineering Journal*, Vol. 11, No. 3, 1996, pp. 183-192.

Maiden, N., Mistry, P., and Sutcliffe, A., “How People Categorize Requirements for Reuse: A Natural Approach,” *Second IEEE International Symposium on Requirements Engineering*, 1995, pp. 148-155.

Martin, J. and Tsai, W., “N-Fold Inspection: A Requirements Analysis Technique,” *Communications of the ACM*, Vol. 33, No. 2, 1990, pp. 225-232.

McDermid, J., "*Software Engineer's Reference Book*," Butterworth-Heinemann, Ltd., Oxford, England, 1991.

Meyer, B., "On Formalism in Specifications," *IEEE Software*, Vol. 2, No. 1, 1985, pp. 6-26.

Mizuno, Y., "Software Quality Improvement," *IEEE Computer*, Vol. 16, No. 3, 1983, pp. 66-72.

Mullery, G., "CORE—A Method for Controlled Requirements Specification," *Proceeding of the Fourth International Conference on Software Engineering*, 1979, pp. 126-135.

Mylopoulos, J., Chung, L., and Nixon, B., "Representing and Using Nonfunctional Requirements: A Process-Oriented Approach," *IEEE Transactions on Software Engineering*, Vol. 18, No. 6, 1992, pp. 483-497.

Nelsen, E., *Systems Engineering and Requirement Allocation*, IEEE Computer Society Press, 1990.

Ohnishi, A. and Agusa, K., *CARD: A Software Requirement Definition Environment*, IEEE Computer Society Press, 1993.

Palmer, J. and Fields, N., "An Integrated Environment for Requirements Engineering," *IEEE Software*, Vol. 9, No. 3, 1992, pp. 80-85.

Parnas, D., "The Use of Precise Specifications in the Development of Software," *IFIP Congress 77*, 1977, pp. 861-867.

Pohl, K., *"The Three Dimensions of Requirements Engineering*," Springer-Verlag, 1993.

Porter, A. and Votta, L., *An Experiment to Assess Different Defect Detection Methods for Software Requirements Inspections*, IEEE Computer Society Press, 1994.

Poston, R., "Preventing Most-Probable Errors in Requirements," *IEEE Software*, Vol. 4, No. 5, 1987, pp. 81-83.

Potts, C., "Seven (Plus or Minus Two) Challenges for Requirements Research," *Proceedings of the 6th International Workshop on Software Specification and Design*, 1991, pp. 256-259.

Potts, C. and Bruns, G., "Recording the Reasons for Design Decisions," *Proceedings of the 10th International Conference on Software Engineering*, 1988, pp. 418-426.

Potts, C., Takahashi, K., and Antón, A., "Inquiry-Based Scenario Analysis of System Requirements," *Proc. International Conference on Requirements Engineering*, Georgia Institute of Technology, 1994.

Prehn, S. and Toetenel, W., "*VDM'91: Formal Software Development Methods*," Springer-Verlag, New York, 1991.

Ramamoorthy, C., Usuda, Y., Prakash, A., and Tsai, W., "The Evolution Support Environment System," *IEEE Transactions on Software Engineering*, Vol. 16, No. 11, 1990, pp. 1225-1234.

Ramesh, B. and Dhar, V., "Supporting Systems Development by Capturing Deliberations During Requirements Engineering," *IEEE Transactions on Software Engineering*, Vol. 18, No. 6, 1992, pp. 498-510.

Ramesh, B. and Luqi, "Process Knowledge Based Rapid Prototyping for Requirements Engineering," *IEEE International Symposium on Requirements Engineering*, 1993, pp. 248-255.

Regnell, B., Kimbler, K., and Wesslén, A., "Improving the Use Case Driven Approach to Requirements Engineering," *2nd IEEE International Symposium on Requirements Engineering*, 1995, pp. 40-47.

Robertson, J. and Robertson, S., *Mastering the Requirement Process*, Addison-Wesley, Boston, MA, 1999.

Robinson, W., "Negotiation Behavior During Requirements Specification," *12th International Conference on Software Engineering*, 1990, pp. 268-276.

Roman, G., "A Taxonomy of Current Issues in Requirements Engineering," *IEEE Computer*, Vol. 18, No. 4, 1985, pp. 14-21.

Ross, D. and Schoman, K., Jr., "Structured Analysis for Requirements Definition," *IEEE Transactions on Software Engineering*, Vol. 3, No. 1, 1977, pp. 6-15.

Royce, W., "Managing the Development of Large Software Systems: Concepts and Techniques," *Proceedings IEEE WESCON*, 1970, pp. 1-9.

Royce, W., *Software Requirements Analysis: Sizing and Costing*, Addison-Wesley Publishing Company, Boston, MA, 1975.

Rumbaugh, J., "Getting Started: Using Use Cases to Capture Requirements," *Journal of Object-Oriented Programming*, September 1994, pp. 8-12,.

Russell, G., "Experience With Inspection in Ultralarge-Scale Developments," *IEEE Software*, Vol. 8, No. 1, 1991, pp. 25-31.

Ryan, K., "The Role of Natural Language in Requirements Engineering," *IEEE International Symposium on Requirements Engineering*, 1993, pp. 240-242.

Ryan, K. and Karlsson, J., "Prioritizing Software Requirements in an Industrial Setting," *International Conference on Software Engineering*, 1997.

Rzepka, W. and Ohno, Y., "Requirements Engineering Environments: Software Tools for Modeling User Needs," *IEEE Computer*, Vol. 18, No. 4, 1985, pp. 9-12.

Sacha, K., "Transnet Approach to Requirements Specification and Prototyping," *IEEE*, 1992.

Saeki, M., Horai, H., and Enomoto, H., “Software Development Process From Natural Language Specification,” *11th International Conference on Software Engineering*, 1989.

Sagaspe, L., Bel, G., Bieber, P., Boniol, F., and Castel, C., “Safe Allocation of Avionics Shared Resources,” *Ninth IEEE International Symposiumn on High-Assurance Systems Engineering*, 2005, pp. 25-33.

Scharer, L., *Pinpointing Requirement*, IEEE Computer Society Press, 1990.

Schneider, G., Martin, J., and Tsai, W., “An Experimental Study of Fault Detection in User Requirements Documents,” *ACM Transactions on Software Engineering and Methodology*, Vol. 1, No. 2, 1992, pp. 188-204.

Schneider, G. and Winters, J., *Applying Use Cases*, Addison-Wesley, Boston, MA, 1998.

Schulmeyer, G., *Zero Defect Software*, McGraw-Hill, 1990.

Sibley, E., Wexelblat, R., Michael, J., Tanner, M., and Littman, D., “The Role of Policy in Requirements Definition,” *IEEE International Symposium on Requirements Engineering*, 1993, pp. 277-280.

Senge, P., *The Fifth Discipline: The Art & Practice of the Learning Organization*, Doubleday, New York, NY, 1990.

Shumate, K. and Keller, M., *Software Specification and Design*, John Wiley & Sons, Ltd., 1992.

Sidiqi, J., “Challenging Universal Truths of Requirements Engineering,” *IEEE Software*, Vol. 11, No. 2, 1994, pp. 18-19.

Sidiqi, J. and Shekaran, M., “Requirements Engineering: The Emerging Wisdom,” *IEEE Software*, Vol. 13, No. 2, pp. 15-19, 1996.

Singer, C., “A Requirements Tutorial. Quality Systems and Software Requirements,” Special Report SR-NWT-002159, Bellcore, 1992.

Sommerville, I. and Kotonya, G., *Requirement Engineering: Processes and Techniques*, Wiley, 1998.

Sommerville, I., *Software Engineering*, Addison-Wesley Publishing Company, Boston, MA, 1996.

Spence, I. and Carey, B., “Customers do not Want Frozen Specifications,” *Software Engineering Journal*, Vol. 6, No. 4, 1991, pp. 175-180.

Spivey, J., *The Z Notation: A Reference Manual*, Prentice-Hall International, Inc., New York, NY, 1992.

Strauss, S. and Ebenau, R., *Software Inspection Process*, McGraw-Hill, Inc., 1994.

Sullivan, L., "Quality Function Deployment," *Quality Progress*, June 1986, pp. 39-50.

Swartout, W. and Balzer, R., "On the Inevitable Intertwining of Specification and Implementation," *Communications of the ACM*, Vol. 25, No. 7, 1982, pp. 438-440.

Tamai, T. and Itou, A., "Requirements and Design Change in Large-Scale Software Development: Analysis From the Viewpoint of Process Backtracking," *15th International Conference on Software Engineering*, 1993, pp. 167-176.

Thayer, R., Pyster, A., and Wood, R., "The Challenge of Software Engineering Management," *IEEE Computer*, Vol. 13, No. 8, 1980, pp. 51-59.

van Vliet, H., *Software Engineering: Principles and Practice*, John Wiley & Sons, Ltd., 1993.

Wang, W., Hufnagel, S., Hsia, P., and Yang, S., "Scenario Driven Requirements Analysis Method," *2nd International Conference on Systems Integration*, 1992, pp. 127-136.

Wangler, B., "Contributions to Functional Requirements Modeling," PhD thesis, Stockholm University, 1993.

Wardle, P., "Methodology and Tools for Requirements Capture, Traceability and Verification," *International Conference on Software Engineering for Real-Time Systems*, 1991, pp. 46-50.

Watkins, R. and Neal, M., "Why and How of Requirements Tracing," *IEEE Software*, Vol. 11, No. 4, 1994, pp. 104-106.

Weigers, K., *Software Requirements*, Microsoft Press, 1999.

Weiss, K., Ong, E., and Leveson, N., "Reusable Specification Components for Model-Driven Development," *Proceedings of the International Conference on System Engineering*, *INCOSE*, 2003.

Whiston, P. and Goodchild, P., *SHIMA-Small Aircraft/Helicopter Integrated Modular Avionics*, Smiths Industries Aerospace, 2000.

Wiktorin, L., "Programvarusystem. Kortade Ledtider And Ökad Produktivitet. [Software Systems. Decreased Lead Times and Increased Productivity]," Technical report V040016, Sveriges Verkstadsindustrier, Stockholm, Sweden, 1994.

Wing, J., "A Specifier's Introduction to Formal Methods," *IEEE Computer*, Vol. 23, No. 9, 1990, pp. 8-24.

WM-data AB, Ericsson Radio Systems AB, Delta Method Handbook,
http://www.deltamethod.net.

Wright, S., "Requirements Traceability—What? Why? and How?" *Colloquium on Tools and Techniques for Maintaining Traceability During Design*, 1991.

Yadav, S., Bravoco, R., Chatfield, A., and Rajkumar, T., "Comparison of Analysis Techniques for Information Requirement Determination," *Communications of the ACM*, Vol. 31, No. 9, 1988, pp. 1090-1097.

Yeh, R. and Ng, P., "Software Requirements—A Management Perspective," *System and Software Requirements Engineering*, 1990, pp. 450-461.

Yeh, R. and Zave, P., "Specifying Software Requirements," *Proceedings of the IEEE*, Vol. 68, No. 9, 1980, pp. 1077-1085.

Yourdon, E., *Modern Structured Analysis*, Prentice-Hall International, Inc., Englewood Cliffs, New Jersey, 1989.

Yue, K., "What Does it Mean to Say That a Specification is Complete?" *4th International Workshop on Software Specification and Design*, 1987, pp. 42-49.

Zave, P., "A Comprehensive Approach to Requirements Problems," *International Computer Software and Applications Conference*, Chicago, IL, 1979, pp. 117-122.

Zave, P., "Classification of Research Efforts in Requirements Engineering," *Second IEEE International Symposium on Requirements Engineering*, 1995, pp. 214-216.

Zultner, R. "Objectives: The Missing Piece of the Requirements Puzzle," in *Structured Development Forum XI*, 1990, pp. 1-11.

APPPENDIX B—SAMPLE INDUSTRY SURVEY FORM

Does your current (or most recent) software development project comply with:

__ DO-178B
__ ED-12B
__ None of the above

How many employees are involved in the development of DO-178B/ED-12B compliant software on your current (or most recent) project? *(Check only one)*

__ 0
__ 1-9
__ 10-99
__ 100+

Select your job function on your current (or most recent) DO-17B/ED-12B compliant project. *(Check all that apply)*

__ Product Development/Engineering
__ Process/Tools Support
__ Research
__ Regulatory (DER, FAA liaison, etc.)
__ Marketing/Sales
__ Management
__ Other (Please specify) ____

What software levels of DO-178B/ED-12B apply to your current (or most recent) DO-178B/ED-12B compliant project? *(Check all that apply)*

__ A
__ B
__ C
__ D
__ E

What role does your company assume in your current (or most recent) DO-178B/ED-12B compliant project? *(Check all that apply)*

__ Airframe Manufacturer
__ System Integrator
__ Subsystem Developer
__ Other (Please specify) ____

Who is the primary source of information for requirements on your current (or most recent) DO-178B/ED-12B compliant project?
(Check all that apply)

__ End Customer (airlines, pilots, maintenance, etc.)
__ Airframe Manufacturer
__ System Integrator
__ Other (Please specify) ____________________________________

In what form are requirements captured on your current (or most recent) DO-178B/ED-12B compliant project?

Enter an "x" in every row/column cell that applies	System Requirements	Data Interconnect {ICD}	High-Level Software Requirements	Low-Level Software Requirements	Hardware Requirements
English Text or Shall Statements					
Tables and Diagrams					
UML Use Cases					
UML Sequence Diagrams					
UML State Diagrams					
Executable Models (e.g., Simulink®, SCADE Suite™					
Data Flow Diagrams (e.g., Yourdon)					
Other (Specify)					
Other (Specify)					

What tools are used for requirements capture on your current (or most recent) DO-178B/ED-12B compliant project?

Enter an "x" in every row/column cell that applies	System Requirements	Data Interconnect {ICD}	High-Level Software Requirements	Low-Level Software Requirements	Hardware Requirements
Database (e.g., Microsoft Access)					
DOOR					
Rational ROSE®					
RDD-100®					
Requisite Pro®					
Rhapsody					
SCADE Suite™					
Simulink					
Slat					
Spreadsheet (e.g., Microsoft Excel)					
Statemat					
Word Processor (e.g., Microsoft Word)					
VAPS					
Designer's					
Other (Specify)					
Other (Specify)					
Other (Specify)					

What tools are used for requirements traceability/compliance on your current (or most recent) DO-178B/ED-12B compliant project?

Enter an "x" in every row/column cell that applies	System Requirements to High-Level Software Requirements	High-Level Software Requirements to Low-Level Software Requirements	Low-Level Software Requirements to Software Design	System Requirements to Hardware
Database (e.g., Microsoft Access)				
DOORS®				
Rational ROSE®				
Requisite Pro®				
Rhapsody				
Slat				
Spreadsheet (e.g., Microsoft Excel)				
Word Processor (e.g., Microsoft Word)				
Other (Specify)				
Other (Specify)				
Other (Specify)				

What techniques are used to ensure that the requirements are accurate, consistent, and verifiable on your current (or most recent) DO-178B/ED-12B compliant project?

Enter an "x" in every row/column cell that applies	System Requirements	Data Interconnect {ICD}	High-Level Software Requirements	Low-Level Software Requirements	Hardware Requirements
Reviews/Inspections					
Creation of Test Cases					
Rapid Prototyping					
Automated Analysis					
Flight Test					
Simulation					
Other (Specify)					
Other (Specify)					

What techniques are used to show that the high-level and low-level software requirements are compatible with the target computer on your current (or most recent) DO-17B/ED-12B compliant project?

What techniques are used to show that the high-level and low-level software requirements conform to standards on your current (or most recent) DO-17B/ED-12B compliant project?

In your opinion, what new issues related to Requirements Engineering Management are introduced with new development paradigms/approaches such as:

Model-Based Development (the use of executable modeling tools such as Simulink®, SCADE Suite™, Statemate, etc. and their associated code generators.)?

Integrated Modular Avionics (IMA)?

Object Oriented Design?

Other (Please specify)?

In your opinion, what level(s) of requirements do the models represent in Model-Based Development?
(Check all that apply).

__ System Requirements
__ Data Interconnect (ICD)
__ High-Level Software Requirements
__ Low-Level Software Requirements
__ Hardware Requirements
__ The Model Does Not Represent Requirements

What are your favorite references on Requirements Engineering?

With regard to Requirements Engineering Management, how is guidance provided by DO-178B/ED-12B, DO-254 and/or ARP 4754 inadequate or unclear? What additional guidance would you like to see provided?

Please provide any additional comments on Requirements Engineering Management you feel are relevant to this survey.

APPENDIX C—INDUSTRY SURVEY COVER LETTER

Dear Members of the Aviation Software Community:

The FAA is supporting research in the area of requirements engineering management. We have contracted Rockwell Collins, Inc. to help us with the effort. This mailing is intended for individuals who have experience writing and working with requirements.

To accurately scope the effort Rockwell Collins, Inc. is conducting an informal survey of the industry. The final result of this effort will be a Best Practices Handbook on Requirements Engineering Management. If you could spare a few moments to reply to this mail (see the survey part below), it would be a great help to the effort.

Replying to the attached e-mail will provide information to the Rockwell Collins, Inc. team (attention of Mr. David Statezni, e-mail: destatez@rockwellcollins.com). Alternatively, if you would like your contact information to remain confidential, then please forward this survey form to the attention of Mr. Charles Kilgore (e-mail: Charles.Kilgore@faa.gov) (FAX: (609) 485-4005 & Phone: (609) 485-6235). Charles will strip all company information from the survey and forward it on to Rockwell Collins, Inc.

Any information you provide is strictly confidential and unless you explicitly allow us to do so, your name and the organization will not be disclosed in the report. A copy of the full survey results will be sent to those who participate in this survey. Prior to dissemination of these results, the FAA will review them to ensure confidentiality is maintained.

We would ask that you reply by March 10, 2005 to allow time for incorporation of this survey data into the final report. Thanks in advance for any help you can provide in this effort!

Sincerely,

Barbara Lingberg
FAA, AIR-120
Sponsor, Software and Digital Systems Safety

APPENDIX D—INDUSTRY SURVEY BACKGROUND INFORMATION RESULTS

Does your current (or most recent) software development project comply with:

40	__DO-178B
7	__ED-12B
2	__ None of the above
0	__ Did not answer

User comments - (development of the SCADE KCG 4.2 code gen qualified at level A)

How many employees are involved in the development of DO-178B/ED-12B compliant software on your current (or most recent) project?
(Check only one)

1	__ 0
17	__ 1-9
21	__ 10-99
3	__ 100+
0	__ Did not answer

Select your job function on your current (or most recent) DO-17B/ED-12B compliant project.
(Check all that apply)

19	__ Product Development/Engineering
8	__ Process/Tools Support
2	__ Research
14	__ Regulatory (DER, FAA liaison, etc.)
1	__ Marketing/Sales
13	__ Management
4	__ Other (Please specify) ___

2-Quality Assurance – 1-SQA – 1-SQA/DER

What software levels of DO-178B/ED-12B apply to your current (or most recent) DO-178B/ED-12B compliant project?
(Check all that apply)

29	__ A
16	__ B
18	__ C
14	__ D
8	__ E

What role does your company assume in your current (or most recent) DO-178B/ED-12B compliant project?
(Check all that apply)

6	__ Airframe Manufacturer
10	__ System Integrator
31	__ Subsystem Developer
10	__ Other (Please specify)

FAA Software development for RADAR systems, Supplier, Avionics manufacturer, Software development, Software V&V, Supplier of TSO'ed avionics equipment, Take on portions of subsystems software development project. Engine, Development of reusable software component for subsystem developer, Tool developer – SCADE

Who is the primary source of information for requirements on your current (or most recent) DO-178B/ED-12B compliant project?
(Check all that apply)

9	__ End Customer (airlines, pilots, maintenance, etc.)
26	__ Airframe Manufacturer
20	__ System Integrator
10	__ Other (Please specify)

Existing System requirements, US Gov't, FAA, Company designed products, Flight test pilots, Advanced design specialists, Internal SYS group, Internal, OEM + internal systems engineer, Industry MOPS & Internal systems, subsystem supplier

Industry Survey Methods and Tools RESULTS
In what form are requirements captured on your current (or most recent) DO-178B/ED-12B compliant project?

APPENDIX E—INDUSTRY SURVEY METHODS AND TOOLS RESULTS

Enter an "x" in every row/column cell that applies	System Requirements	Data Interconnect {ICD}	High-Level Software Requirements	Low-Level Software Requirements	Hardware Requirements
English Text or Shall Statements	39	27	36	32	29
Tables and Diagrams	31	30	30	19	18
UML Use Cases	1		2	4	
UML Sequence Diagrams			3	6	
UML State Diagrams			1	7	
Executable Models (e.g., Simulink®, SCADE Suite™, etc.)	7	1	8	8	1
Data Flow Diagrams (e.g., Yourdon)	4		6	9	
Other (Specify) - Proprietary Database, DOORS objects	1	4	2	2	1
Other (Specify)XML		1			
Operational models or prototypes	1	1			1
UML			1	1	

What tools are used for requirements capture on your current (or most recent) DO-178B/ED-12B compliant project?

Enter an "x" in every row/column cell that applies	System Requirements	Data Interconnect {ICD}	High-Level Software Requirements	Low-Level Software Requirements	Hardware Requirements
Database (e.g., Microsoft Access)	3	4	3	3	
DOORS	23	13	22	18	12
Rational ROSE®			1	3	
RDD-100®					
Requisite Pro®	5	3	5	4	4
Rhapsody	1				
SCADE Suite™	2		3	1	
Simulink®	5	1	5	3	1
Slate	1		1	1	
Spreadsheet (e.g. Microsoft Excel)	5	4	5	4	3
Statemate					
Word Processor (e.g., Microsoft Word)	19	20	18	17	16
VAPS™		1	3	3	
Designer's Workbench™			1	1	
Proprietary Database, SCADE like pic tool		1	1		
Interleaf	1	1	1	1	1
BEACON	1	1	1	1	
CaliberRM	1	1	1	1	1
XM:		1			
Wiring diagram		1			1
Schematic capture	1				1
RTM	1		1	1	
Artisan			1	1	
Home-grown Autocoder			1		
Tau			1	1	

What tools are used for requirements traceability/compliance on your current (or most recent) DO-178B/ED-12B compliant project?

Enter an "x" in every row/column cell that applies	System Requirements to High-Level Software Requirements	High-Level Software Requirements to Low-Level Software Requirements	Low-Level Software Requirements to Software Design	System Requirements to Hardware
Database (e.g., Microsoft Access)	4	3	3	2
DOORS®	24	24	19	14
Rational ROSE®				
Requisite Pro®	5	4	5	4
Rhapsody				
Slate				
Spreadsheet (e.g., Microsoft Excel)	9	9	7	7
Word Processor (e.g., Microsoft Word)	10	7	7	8
Vbay DCM		1	1	1
Interleaf	1	1	1	1
Internally developed tool	1	1	1	1
RTM	1	1	1	1
Python Scripts		1	1	

What techniques are used to ensure that the requirements are accurate, consistent, and verifiable on your current (or most recent) DO-178B/ED-12B compliant project?

Enter an "x" in every row/column cell that applies	System Requirements	Data Interconnect {ICD}	High-Level Software Requirements	Low-Level Software Requirements	Hardware Requirements
Reviews/Inspections	34	30	33	32	24
Creation of Test Cases	23	17	34	33	20
Rapid Prototyping	9	5	12	10	7
Automated Analysis	6	4	5	6	5
Flight Test	15	6	6	5	5
Simulation	11	4	9	6	4
Aircraft Integration		1			
Other (Specify) Iron-bird Testing	1	1			1

APPENDIX F—INDUSTRY SURVEY COMMENTS

This section documents the comments provided by the respondents to free-text questions in the survey. The comments are provided as they appeared on the returned surveys; i.e., no attempt was made to correct grammar or spelling.

F.1 Ensuring Compatibility With the Target Computer.

What techniques are used to show that the high-level and low-level software requirements are compatible with the target computer on your current (or most recent) DO17B/ED-12B compliant project?

Survey Comment

We prove that the software meets its requirements on the target hardware by performing a full suite of tests which trace to the software requirements.

Survey Comment

Prototypes

Survey Comment

Painful testing and unhappy requirements discovery.

Survey Comment

SW / SW integration done on target hardware

Survey Comment

This begins with the process of choosing the target computer based on the system and software requirements and the system architecture. The software is then designed within the constraints of the target computer environment. Software profiling as well as memory margin analysis are used as a measure of how well the software component meets requirements in the target computer environment. Peer reviews, inspections, and analyses are also used. Finally, robustness testing is used to show that the software will function safely under abnormal operating conditions.

Survey Comment

Compatibility of high-level requirements to target computer is proven by means of the testing. Compatibility of low-level requirements to target computer is proven by means of testing. Review of low level requirements and source code is performed to verify against the known limitation of the target computer (precision, accuracy, scheduling, execution time, …)

Survey Comment

Analysis, inspection and development of test cases

Survey Comment

Integration Test on HW / SW or System level

Survey Comment

Review/Inspection, Prototype Software

Survey Comment

Reviews using checklists
Requirements Based Tests on the target (both 'Black Box' and 'white box' tests)
Hardware/software integration and test
Data/control flow analysis of low level I/O and control
Timing analyses

Survey Comment

Rapid Prototyping
Peer reviews

Survey Comment

Testing is done on the target computer and/or target CPU in an offtarget platform. Requirements inspections and reviews are also performed. In some cases, reviews and inspections are all that are performed.

Survey Comment

Reviews and inspections. Development of Test cases.

Survey Comment

Projects at the LRU or subsystem level determine this. May vary between product.

Survey Comment

Review and Analysis

Survey Comment

Assuring that system level tests (integration testing) covers as many as possible test cases.

Survey Comment

Traceability reviews

Survey Comment

Run tests for score on the target, expect to get expected results.
Peer reviews of code & HW.
Integration testing.

Survey Comment

Operation on the emulator connected to the target hardware platform
Software operation on the target microprocessor in the hardware platform.

Survey Comment

HW/SW integration based test.

Survey Comment

Reviews
Test on target
Test on emulated target

Survey Comment

Test cases/Test procedures

Survey Comment

Software is developed, tested and verified on the target hardware.

Survey Comment

Test

Survey Comment

Peer review process utilizing check lists
Preliminary software design
Compilation and execution on target

Survey Comment

Traceability Analysis
Requirements document reviews

Survey Comment

Requirements based testing performed directly on the target computer.

Survey Comment

Requirements peer reviews
Periodic throughput and memory analysis results are published at the end of each project so that subsystem engineers are aware of any target computer limitations that may impose design constraints they must observe in defining the requirements for the subsequent project.

Survey Comment

Primarily Service History.
We have been using the same computer architecture for years.
We also start using the target hardware very early in the program so that the software runs on the target hardware for a very long time.

Survey Comment

Compatibility of requirements to target computer is inherent if the verification tests all run and pass since we are verifying requirements and executing the tests on the target hardware.
If we need an early look at some specific requirements we can rapid prototype.

Survey Comment

Memory, Throughput and I/O throughput budgets
RMS Scheduling and schedulability analysis

Survey Comment

Development and execution of test cases based on the requirements.

Survey Comment

Reviews/inspections of high-level and low-level requirements using a DO-178B based checklist
Execution of high-level and low-level software requirements based test procedures on target hardware
Rapid prototyping of high-level and low-level requirements and executing some integration tests on the target hardware

Survey Comment

Compatibility with the target computer is performed on a sophisticated closed loop bench with the engine control in the loop controlling to an engine model. We also test the compatibility in the engine test cell.
We do use an language emulator to assist in predicting structural type testing expected answers.

Survey Comment

Analysis, simulation, test cases.

Survey Comment

Reviews/inspections and testing.

Survey Comment

Analysis and testing (note that this is very specific since we are developing a tool that runs on a workstation)

F.2 Requirements Conform to Standards.

What techniques are used to show that the high-level and low-level software requirements conform to standards on your current (or most recent) DO17B/ED-12B compliant project?

Survey Comment

We write a software development plan which identifies the software requirements standard which will be used on the project, then we peer-review the software requirements to ensure they meet the standards.

Survey Comment

Inspections / reviews

Survey Comment

Painful testing and requirements discovery.

Survey Comment

Review

Survey Comment

Peer reviews, inspections, and analyses are used to determine that the software requirements conform to standards.

Survey Comment

Conformance of high-level requirements and low-level requirements to standards is performed by means of reviews of the requirements towards the rules defined in the requirements standard. Appropriate checklists are used as support during the review process.

Survey Comment

Inspection

Survey Comment

Reviews

Survey Comment

Review/Inspection

Survey Comment

Reviews using checklists
Use of MATLAB for control laws
Use of DOORS for syntax checking and balancing
Use of Cdadl for design (low level requirements) standards

Survey Comment

A detailed review is conducted with the customer for all of the low level requirements. This review results in agreement with the customer on technical content of the requirements and compliance with standards.

Survey Comment

Peer reviews
Template utilization

Survey Comment

Requirements Reviews and inspections as well as audits by SQA

Survey Comment

Reviews and inspections.

Survey Comment

Don't know

Survey Comment

Review and Analysis

Survey Comment

Reviews

Survey Comment

Maybe some guidelines in the Software Development Plan.
Requirements are reviewed to make sure they make sense for the
Project and they are testable.
Doors & Simulink dictate the format of the requirements.

Survey Comment

Design and Code Inspections (Gilb inspection model)

Survey Comment

Reviews
"Canned" style templates

Survey Comment

Reviews/Inspections

Survey Comment

Design and code reviews are conducted to ensure requirements, design, and coding guidelines are followed

Survey Comment

Development using standards. Review against standards

Survey Comment

Peer review process utilizing check lists

Survey Comment

Requirements document inspection

Survey Comment

Formal Peer Reviews

Survey Comment

Project standards are published as part of the Manual for FMS Subsystem Engineers and Certification Engineers, and the FMS Developer's Manual
Requirements peer reviews are held and one of the aspects examined is conformance to published project standards that pertain to requirements.

Survey Comment

This is done both in the reviews (requirements review, design review, coding peer review, etc.) and the SQA development audit.

Survey Comment

Peer Review

Survey Comment

Enforcement of a Software Code of Practice
Review process

Survey Comment

Peer review.

Survey Comment

We use checklists for the defined standards. We have peer reviews of the artifacts to ensure the artifacts meet the standards.

Survey Comment

Reviews/inspections of high-level and low-level requirements using a DO-178B based checklist

Survey Comment

We have standard work standards for all phases of the software lifecycle. (reqs, design, code, test). All reviews and checklists include Standard Work verification.
We feel our autocode generator permits us to make an argument that the code generated is standard.

Survey Comment

Engineering peer review, SW Quality Assurance review.

Survey Comment

Reviews/inspections

Survey Comment

Manual inspections.

F.3 New Issues Introduced by Model-Based Development.

In your opinion, what new issues related to Requirements Engineering Management are introduced with new development paradigms/approaches such as:

1. **Model-Based Development (the use of executable modeling tools such as Simulink®, SCADE Suite™, Statemate, etc. and their associated code generators.)?**

Survey Comment

If Models will replace high-level and low-level software requirements (i.e., we write the requirements using model-language instead of English-language). We must ensure the systems and subsystems team members are fluent in model-language so that their peer reviews are meaningful. Also, requirements decomposition is a form of complexity management, i.e., the system is broken into smaller and smaller pieces, each being manageable steps, until the end result is achievable. Models may represent a too large of a step that the mind can comfortably take at once. Also, Models, being computer programs, are often less forgiving than English text which can accommodate a large amount of ambiguity be resolved in order to define a valid system. As a example, in English we might say "sign here" whereas a model language may require us to say "using a ballpoint pen with a .7mm tip containing black ink in a cursive format with a font size of no less that 10 point and no more than 20 point, etc. … We may lose the actual requirement in the details.

Survey Comment

Test verification or qualification.

Survey Comment

Verification and validation of the code generating tools.

Survey Comment

In concept, it would be easy to fit the simulation models into the Low Level Requirements category, and the resulting generated C code into the Source Code category. This raises the following issues:
Is the generated code traceable to the models?
Are the models traceable to the generated code?
Does the generated code need to be verified against the models?
Does the code generator need to be qualified?
How much certification credit (if any) can be taken for verification via simulations?
Another strategy might be to claim that the model is actually both the Low Level Requirements and the Source Code. This eliminates an entire set of DO-178B processes relating to Source Code. However, this brings up the issue of Tool Qualification. The code generator would have to be qualified as a development tool under DO-178B. However, the development tool qualification could require as much or more effort than required to take the traditional approach.

Survey Comment

Requirements defined in models are
not as explicit as textual requirements
more formal language
more error prone because of less requirements granularity and broader functionality in one and the same model
Tools delivered for this model-based development allow easier requirement validation
Accuracy and correctness of requirements in the model are more difficult to prove on model level

Survey Comment

These are “development” tools. Their results need to either be manually and tediously inspected (generated code can be sloppy, hard to follow), or the tools need to be strenuously qualified. This qualification can be very difficult, requires cooperation of tool vendor to go smoothly. Tool source code hard to come by. Need to understand how to apply service history, or some other type of alternative means of qualification for these complex development tools in order to realize their full potential.
Acceptable means of CM on the data maintained in these tools.

Survey Comment

Many tools follow a simple function orientied approach (SCADE, Matlab). Software Engineering Methodology already in the seventies found out, that a simple function orientied way of analysing and designing is not appropriate for managing complexity (c.p. D. L. Parnas). That’s why a modular approach was chosen in the eighties to overcome the software crisis.

Survey Comment

Configuration control of models. Models, toolboxes, utilities are created as needed, and have no integrated revision control for individual blocks or functions. All items must be controlled manually by the user archiving the baselines.

Survey Comment

Difficulty in demonstrating independence since the code is generated by the same tool as the model that is used for test
Need to establish and use additional coding standards (e.g., naming conventions, insertion of test points)
Added process requirement for tool qualification
Generally Model Based Development does not address the real time aspects of embedded real time systems, especially concerning redundancy management, cross channel I/O and synchronization and timing, failure management including reconfiguration and fault handling

Survey Comment

The requirement is virtually the design, so any error automatically propagates into design.

Survey Comment

The use of MDB methodologies or tools should not be dictated. They are project-dependent. The use of model-based tools & methodologies must be assessed on a case-by-case basis by the developer. What works for one development activity may not work for another. Also, there can be an overemphasis on tooling in lieu of lack of overall system experience; tools do not write good requirements, engineers do. A tool is no better than the quality of data put into it.

Therefore, requirements regarding MDB tools or methodologies are impractical to specify.

Also, it is difficult to use DO-178B processes with Model Based Development. DO-178B does not have precise and widely agreed upon definitions for System Requirements, HL Software Requirements, LL Software Requirements and HW Requirements. Nor should it. This is project dependent.

Model based development introduces a new level of abstraction, and translation. MBD should not be extrapolated to everything as this results in an abuse of the methodology.

Comment: In general, models can provide some clarity to text-based requirements. They provide a mechanism to help understand the text-based requirements but they are not a substitute.

MBD tools have a great deal of capability and flexibility in choosing what you want models to represent. Therefore, based upon what is represented in the models, as well as what is represented in artifacts outside of the models, results in all possibilities being potentially suitable uses for models.

Survey Comment

How to validate the model. Evaluating the completeness of the model. Applicability of the modeling approach to solving the problem at hand. Development tool qualification.

Survey Comment

Tool verification effort can get to cost more than the value added by the tool.

Survey Comment

How to make changes, compare versions and document results

Survey Comment

Less requirements volatility – lets the customer see the requirements in action much earlier in the project. This is a major benefit.
Maybe issues on who owns the models? The customer or us? Or both?
We are learning as we go through it.

Survey Comment

Configuration Management reluctance to control databases (not documents)
Regulators inability to review data in a tool's native form (they don't have tools)
Regulators reluctance to approve data not in a document form.

Survey Comment

Model-based development (MBD) simply sounds like rapid prototyping via simulation (rather than via rapid HW & SW development). The net result can therefore be overly-detailed SYS requirements with no rationale and no documentation of the true "high-level" intent or development philosophy. Indeed, it seems to move "SW rqmt & design" step to the SYS arena. But what SYS creates are not SYS requirements so much as they are low-level SW requirements – except without the process "care" that DO-178B intended.

Survey Comment

SCADE Suite^{TM}

Survey Comment

1. Overuse of tools. Often, the effort to purchase, learn, and use a tool outweighs its benefits.
2. Tool qualification issues.

Survey Comment

Integration of Tools. Training. Process Improvement

Survey Comment

How is documentation of the requirements to be produced (e.g., can it be automated as a function of the tool)?
Will MBD be accepted by systems engineers who are accustomed to generating primarily textual requirements? Paradigm shift challenge.
If requirements are contained in the model (as opposed to, say, a database), how will linking of low-level requirements to high-level requirements be accomplished? Software to requirements? Test cases to requirements?
How can legacy requirements be imported into the modeling tool? If they cannot be readily imported, won't it be expensive to populate the modeling tool with all the requirements for a large project, such as FMS with over 300,000 LOC?

Survey Comment

How are the requirements reviewed? Do they violate safety assumptions? Where are the interfaces different?

Survey Comment

Simulink and proprietary tools for Logic requirements prototyping are used extensively with major benefits

Survey Comment

Fidelity of the model to the actual requirements

Survey Comment

Where are the "shalls" in a model. How are the requirements traced to higher and lower-level requirements and / or design?

Survey Comment

It is difficult to identify or tag what an actual requirement is within the model.
It is difficult to generate traceability to portions of the model.

Survey Comment

The biggest issue for us is model validation. We want to be able to validate the robustness of the model to enable us to use it as a vehicle for certification purposes. We have a physics-based model approach whereby the model is validated and confirmed with actual running data.

Survey Comment

First of all the verification of the output of the tool. This is addressed by tool qualification, but represents a big challenge.
The second part is the representation of the requirements. If they are only available and represented in the modeling tool, the question arises how do you verify the requirements? By taking a look at the requirements in the tool? Is there a need for tool qualification, even if no code is generated?

F.4 New Issues Introduced by Integrated Modular Avionics.

In your opinion, what new issues related to Requirements Engineering Management are introduced with new development paradigms/approaches such as:

2. Integrated Modular Avionics (IMA)?

Survey Comment

FAR part 25 may have adequately addressed failures assuming an essentially federated system in which single points of failure have a limited system effect. Highly integrated systems may exhibit faults in systems which, in the pilot's mind, are completely unrelated. Aircraft certification requirements may have to be altered to address cascade failure effects.

Survey Comment

Too many eggs in one basket. Potential to scramble the eggs. Potential to break all the eggs at once.

Survey Comment

Test coverage.

Survey Comment

Physical proximity. Example is one capacitor exploded, took out both channels of an engine control (which was not IMA), and shut engine down. So faults in one system have increased 'sneak paths' to impact another – track burn, power supply pull down or noise, dripping solder, explosion debris, excessive local EMI effects under fault conditions, connector wear and tear, common mode issues if connectors not mated properly.

Survey Comment

More difficult to check, verify and guarantee traceability following aspects:
HW compatibility (on target processor)
HW/SW interfacing
SW/SW interfacing
Integration issues

Survey Comment

Incremental certification.
lack of control over and visibility into the lower level implementation of the real-time environment – may have unexpected implications on the real time performance, especially under failure conditions
no control over changes/enhancements – would need to demonstrate no effect of the change to the application, also need to determine scope for regression test when there is a change that does affect the application

Survey Comment

Allocation of requirements to partitions is necessary to allow proper overall design to occur.

Survey Comment

Requirements need to be defined around two sources: IMA Platform system requirements & Application-level requirements.
DO-178B does not seem to address the many levels of integration that can and do occur on

typical IMA software projects. Also not addressed that may be of particular concern are:
- Change Impact Analysis at the system level
- Data and control coupling at the integration level
- Independence of development/verification for commonly used items across IMA
- Common failure modes and cascading effects
- Interface requirements between the integrated functions
- How to prove that partitioning works, in particular what must be done to prove that time partitioning is working and that budgets are set correctly

IMA means that the engineer has to understand the big picture (integrated), and most don't.

Survey Comment

Establishing Integration requirements for the end user. Defining system performance requirements. Meeting hard real-time deadlines. Partitioning.

Survey Comment

Can result in massive CM problems where there are multiple configurations spawned from a core design. Test CM also more of a challenge since distributed functionality means more of the IMA HW and SW elements must be "configured" for any for-score testing.

Survey Comment

Sufficient work up front to devise workable reuse strategies.

Survey Comment

IMA is fully applied in one (ongoing) project.
Current opinion is that a federated System incorporating concepts of
High functional cohesion
Open SW architecture and Abstraction Layers
Open interface connectivity

Would provide the maximum benefits.

Survey Comment

For us specifically, we do not have an IMA system. With that said, however our plans in the future would be to have partial engine control SW on aircraft and partial on engine. With the engine type certificate being different than the aircraft type cert, we are uncertain about the roles of the FAA directorates.

One reason for our decision to look towards IMA systems is to help the obsolescence problem whereby we can design it somewhat modular for those items that do have a short shelf life. We would like to see obsolescence handle by modular upgrade and as such modular cert is of large concern to us.
Since the verification is the bigger part of the development, the major challenge will be to have a composable system that allows independent and reusable verification. The constraints for such a composable systems are stringent and not very well understood.

F.5 New Issues Introduced by Object-Oriented Design.

In your opinion, what new issues related to Requirements Engineering Management are introduced with new development paradigms/approaches such as:

3. **Object Oriented Design (OOD)?**

Survey Comment

Heap management. Hidden functionality. Nondeterminism

Survey Comment

Applicability of the reuse.

Survey Comment

Low level requirements definition. What is considered as low level requirements in OOD? Use Cases and State diagrams?
Traceability low level requirements <-> high-level requirements is not straightforward in OOD.
low level requirements testing : What is the meaning of full requirements coverage and structural coverage in OOD

Survey Comment

Generation of unnecessary requirements/functionality. Need to verify or disable these extra features

Survey Comment

It's not covered by the DO-178B.
It requires a restriction on a subset of OO techniques (i.e., avoid inheritence and dynamic creation of objects).

Survey Comment

Please refer to the OOTiA handbook for details

Survey Comment

OOD does not lend itself to functional decomposition – hard to trace requirements (many to many), hard to demonstrate compliance, requires additional analysis, requires extra re-qual after changes
Prone to 'generic code' which leads to worst case 'dead code', best case 'deactivated code' – extra work to demonstrate compliance
Safety aspects and implications are not intuitive.

Survey Comment

There are some technical barriers to things like Object Oriented Design in that they can induce processing overhead and in some circumstances induce runtime indeterminism. We cannot, for example, employ dynamic dispatching (key to OOD) without detailed analysis or it could lead to all kinds of uncertain call chains and problems with verification. The whole concept of a class and its descendent child classes makes for difficulties in verification of class-wide operations. (If I test a subroutine that deals with a class-wide parameter, do I know it will work with *any* children of that class without testing it with all possible children?)

Technical barriers exist to using OOD, but they are not insurmountable. The techniques would have to be evaluated to determine if there were any potential runtime issues and some parts of OOD might need to be avoided due to indeterminism. However, many of the techniques could be valuable if studied and employed properly. The resistance is largely institutional and psychological.

Survey Comment

Considerations of how effectively OO design can be implemented to achieve DO-178B certification.

OO design has worked well for us in breaking the software problem set down to its smallest components, but it doesn't necessarily convey all the integration issues between hardware and software.

Survey Comment

OOD is not necessarily appropriate for all programs. Experience has shown that projects that use OOD simply for the sake of using OOD fall behind schedule and have a low rate of success. Making SW satisfy some object oriented philosophies can result in very complicated, bad SW.

Programs that took a rational approach and applied OOD as appropriate were very successful. It is not OOD, it is the deployment of it.

OOD can be a valuable tool when used appropriately and deployed correctly with appropriate standards, but is not appropriate for all cases.

Survey Comment

How to test all aspects of OO implementation. Traceability of object code to source code. What does structural coverage mean?

Survey Comment

Some. We watch the real time aspects.

Survey Comment

No ability to show determinism as required in the high levels of DO-178B.
No clear industry discussion of safe constructs or a safe subset of the method.

Survey Comment

One of the benefits of this paradigm is the ability to create more easily reused components. Anytime you reuse a component, it's likely a general purpose component contains functionality unneeded for this project. Likewise, when extending an object to meet new requirements, the base object may contain unneeded functionality. Either way, you have to deal with requirements traceability or dead code issues.

Survey Comment

Understanding and applying the new FAA guidance on OOD.

Survey Comment

Language, terminology. But current RM techniques need not change to accommodate new design techniques. OOD provides increased possibility for CASE tool integration.

Survey Comment

How much time/effort (and therefore, how much money) will need to be invested to retrain systems engineers who are accustomed to generating primarily textual requirements, so that they know how to capture OO requirements using a new form of notation?
Paradigm shift challenge – lack of acceptance by systems engineers who think procedurally and tend to anticipate a structured design in the way they organize their requirements.
How can legacy requirements be efficiently transformed to support the OO model?

Survey Comment

Yes, OOD is the key to cohesion and layering.

Survey Comment

Traceability back to requirements is a big issue with object oriented design. This is due to the fact that, if done right, there is significant abstraction which makes it hard to look at portions of the code and tie them back to any requirements (derived requirements can help somewhat with this issue).

Survey Comment

OOD by itself is not a challenge. The usage of C++ or generally speaking of more sophisticated compilers/linkers and runtime environments is a challenge. How is such a complex translation and runtime environment fully verified?

Survey Comment

This could apply. Main issues are in testing and coverage analysis.

F.6 New Issues Introduced by Other Development Approaches.

In your opinion, what new issues related to Requirements Engineering Management are introduced with new development paradigms/approaches such as:

4. Other (Please specify)?

Survey Comment

VAPS models:
- Definition of symbology:
  - What is considered as requirement?
  - Can this replace a requirements document?
- Requirements traceability is not straightforward.
- Verification:
  - What needs to be tested of that symbology? What can be done by simulation?
  - Requirements coverage: When is it complete?

Survey Comment

Model based development (e.g., UML models, in contrast to function based models like Matlab)

It's not covered by the DO-178B. The role of the model respectively the role of its elements in the requirements process is not clear resp. not addressed by the DO-178B: Is it optional, just for understanding the system? Many people would deny using it, because it – superficially – adds cost, although it adds value, because people have the systems under control, which they are going to build. This in turn speeds up development and finally saves cost.

Survey Comment

Process requirements for Commercial Derivative/Military Aviation software and dual use software – since the approach to Requirement Management for Military Programs has a different focus than that of Civil Certified Programs

Survey Comment

By far, the biggest issue is requirements. IMA must merge both ARP4754 with DO-178B. The first question to answer is "where do system requirements stop and s/w requirements begin?"

Survey Comment

What issues are there in convincing shared services, like Software Control Library, DERs and SQE specialists, to accept outputs of the MBD or OOD tools and be able to assess their quality and archive them? We haven't even been able to convince these supporting services to accept release of a DOORS database instead of a requirements document, which seems like a pretty mild shift, compared to some of the newer technologies. In other words, Engineering can't just decide on its own to change this paradigm. Everyone involved in the design, development, release, and certification process must be onboard

Survey Comment

Human-Machine Interface prototyping and review with the Customer

Survey Comment

One difficult issue for reqs management is the use of different tools and integration of those tools for effective traceability and configuration control

F.7 What Level of Requirements Do Models Represent?

In your opinion, what level(s) of requirements do the models represent in Model-Based Development? (*Check all that apply*)

20	__ System Requirements
16	__ Data Interconnect (ICD)
33	__ High-Level Software Requirements
25	__ Low-Level Software Requirements
9	__ Hardware Requirements
7	__ The Model Does Not Represent Requirements
4	__ Did not answer

F.8 What Are Your Favorite References on REM?

What are your favorite references on Requirements Engineering?

Survey Comment

System Engineering & Analysis-Blanchard & Fabrycey

Survey Comment

I am still looking for a good book to use within our organization to transform our culture into one that is more in tune with modern structured methods of development. Most references are either too cumbersome, lack clarity or dive off into the very deep waters of esoterica.

Survey Comment

Books on OOA/D like "Developing Software with UML", Bernd Oestereich

Survey Comment

Rapid Development by Steve McConnell (1996 – Microsoft Press)

The Capability Maturity Model: Guidelines for Improving the Software Process (1994 – Software Engineering Institute)

Survey Comment

SEI Capability Maturity Model

Various papers presented at the annual Software Engineering Process Group conference in the mid-1990's when there was a real focus on implementation of requirements management concepts.

Survey Comment

"Requirements Engineering – A good practice guide"
"Requirements Management" by Fastrak Training Inc.
Managing Software Requirements: A Use Case Approach, Second Edition by Dean Leffingwell, Don Widrig
Managing Software Requirements: A Unified Approach (The AddisonWesley Object Technology Series) by Dean Leffingwell, Don Widrig
Software Requirements, Second Edition
by Karl E. Wiegers
Mastering the Requirements Process
by Suzanne Robertson, James Robertson
RealTime UML: Developing Efficient Objects for Embedded Systems (2nd Edition)
by Bruce Powel Douglass

Survey Comment

SEI publications.

Survey Comment

DO-178B and years of painful (at times) experience. The painful part is explaining it to someone that has never written to DO-178B.

Survey Comment

Material by Ivy Hooks

Survey Comment
TCP – Technical Consistent Process

Survey Comment
INCOSE
Telelogic Writing Better Requirements
Johns Hopkins SE Masters Degree Material
System Engineering Overview (Internal Westinghouse Document)
Personal Models

Survey Comment
Technical Consistent Process

Survey Comment
Artisan, Simulink

Survey Comment
Requirements Engineering Hull

F.9 How is Existing Guidance Inadequate or Unclear?

With regard to Requirements Engineering Management, how is guidance provided by DO-178B/ED-12B, DO-254 and/or ARP 4754 inadequate or unclear? What additional guidance would you like to see provided?

Survey Comment
Not inadequate

Survey Comment
The aerospace industry has evolved into a large number of sub-tier suppliers who do not have the staff or the sophistication to develop software, manage its configuration, or control its quality. The OEMs are badly understaffed and immature in their appreciation for discipline in dealing with the establishment, analysis, allocation and ultimately the verification and validation processes needed to manage engineering requirements. The resulting chaos is inefficient and ineffective - but manages to work remarkably well!

Survey Comment
There is no standard formatting, the process is arbitrary and not uniformly administered by the various FAA ACOs.

Survey Comment
A better SAE ARP4754. This only partially makes the relationship between hardware reliability and software quality level clear. Also it's Validation standards are totally unrealistic in relation to what industry does.

Survey Comment

After having written several requirements documents as well as critiquing several others, and having to write test cases based on poorly written requirements, I had at one time come to the conclusion that generating good requirements is more of an art form that a science, something that is learned over time and with experience. However, if this can be learned over time, then it should be possible to record that knowledge and that others should be able to learn it without having to learn from experience. I believe that a set of rules including, language, syntax, and clear definitions, could be devised that would greatly improve the quality of the requirements.

Survey Comment

Concept of safety-related requirements
What to do with these safety-related requirements during the development process is not clearly defined.
What consequences and impact on testing for these safety related requirements
Concept of derived requirements
Currently identification of derived requirements is required, but what is the added value and for what is it useful?
DO-178B doesn't talk about the importance of the rationale or design decision leading to these derived requirements.
Different requirement levels (granularity of requirements)
DO-178B talks about different levels in requirements from which the highest level is called the high-level requirements and the lowest level is called low level requirements. For these 2 levels the required objectives and activities are well defined.
However one may have additional requirement levels in between. For these it is not very clear what activities need to be performed.
Correlation to HW requirements
Where do HW requirements interfere with SW requirements
What activities need to be performed when performing concurrent development of SW and HW and its interfaces.
Requirements and relation to SW level
In practice we see that the abstraction level (i.e., level of detail) of SW requirements for a level A project is not the same as for a level D project. However this relation between the SW level and the SW requirements is not clearly defined.

Survey Comment

Acceptable means of applying level 1 configuration control to requirements and traceability data within a modeling tool or database (such as DOORS) tool

Survey Comment

See previous answers and reference to OOTiA Survey Comment

Survey Comment

The definitions of High Level vs Low Level Requirements leave a lot to personal opinion. I have heard statements that one has directly coded the whole design from high-level requirements, and that no low level requirements were needed. Whereas I interpret that as meaning that they defined the implementation into the high-level requirements. So in effect their high-level requirements are really low level ones and have not documented any high-level requirements. Some guidelines or standardization on how many high-level requirements vs low level requirements vs lines of code are typical for a project. Statements such as “Testing high-level requirements should not require white box testing as the way low level requirements do” may help clarify if one is dealing with a low level requirement or not.

Survey Comment

More guidance for lower-level requirements in terms of documentation, trace, test

DO-254 – not clear if (and how much) is in force

More guidance for how changes need to be handled during maintenance (i.e., post certification) documentation, regression test, etc., especially for changes to address obsolescence

Survey Comment

The emphasis on verification of requirements has lead to a number of futile dodges that a) don't buy much additional safety and b) cost a lot of money to accomplish. When people start labeling their design diagrams "Requirements" and then go out to verify that the code generated matches the design diagrams, this is not right. Design diagrams are NOT "Requirements" they are "DESIGN". In the case of auto code generation, they are "IMPLEMENTATION". How is a Simulink (or similar tool) drawing different than expressing an implementation in Ada, Fortran or C? Do you verify that the output of an Ada compiler matches the source code statements in any way other than through normal testing to see if you get the desired result? There is extremely little value in checking that a compiler takes "Y := M * X + B;" and translates it into "Load A,X; Mult A,M; Add A,B; Store A,Y;". None of that tells you if "Y := M * X + B;" was the right thing to do in the first place. What if the operation should have been "" instead of "+"?

What we need is better guidance on what constitutes a valid, testable requirement and what levels of gyrations are needed to verify it.

Survey Comment

As mentioned above, DO-178B does not provide good definitions that allow for an unambiguous interpretation of “system requirement”, “software requirement”, and “hardware requirement”. Where are the boundaries? We have rigorous processes around SW, but very loose processes relative to system requirements. Also, there are many that claim that DO-178B A31 means that the requirements have to trace to the aircraft level requirements from the airframe manufacturer.

The biggest problem with these documents and cert. authority interpretation is the abstraction between all the different levels of requirements including Systems Requirements.

Survey Comment

Projects live and die by the completeness and accuracy of traceability information. The ease with which requirements can be captured, traced, elaborated, and evolved should be emphasized at the beginning of the project in deciding on tools and methods.
What constitutes a low level requirement has much latitude for interpretation.

Survey Comment

The documents lack in clarity in some places. Plus the FAA individuals seem to have specific interpretations that may not agree between everyone. Experience and mentoring. seems to be the best teacher.

Survey Comment

I feel that the requirements should precede the actual code development, but not in such a rigidly structured format as is required by DO-178B. I feel that the act of documenting the requirements is good, but it shouldn't be required to be in a formally 'released' state, and require tons of paperwork to change once the actual development is underway. Most people need to 'play' with their code before it is really where they want it to be, and with DO-178B to stray from the first iteration of code is cost prohibitive in terms of the documentation required to change any of the requirements. Hence, the product ends up with what may not be the 'best' solution for the given task, or as is typically the case, the process gets short cut. I think it would be better to loosen things up to allow the developers to optimize their solutions before rigid controls are placed on the design documents.

Survey Comment

The project software development plan

Survey Comment

Some of our customers ask for additional steps to be performed beyond DO-178B.

These should be incorporated into DO-178B.

Survey Comment

Not so disappointed with the areas in DO-178B/254 in the requirements area, my heartburn has to do with the assumption that you can plan a project to great detail at the start, run it for 2 years and not have a myriad of changes at the end.

Survey Comment

DO-178B was SW oriented. "Requirements Engineering Management" demands a SYS oriented document.

Survey Comment

Requirements engineering is critical to the cost effective implementation and verification of systems. As such, the standards to a poor job providing guidance as to what is to be accomplished by the various levels.

Survey Comment

DO-178B is redundantly ambiguous. If I wrote a software specification that was like DO-178B I'd be fired in a heart beat. The writing committee should repeat the sacred mantra 'Unambiguous, Clear and Concise' 1000 times before writing each word.

Survey Comment

- Incorporate or reference to FAA Action Notice.

Survey Comment

Think there's too much "guidance". Too many versions of the same idea

Survey Comment

DO-178B is, by design, very high-level and sometimes quite vague.
I would love to see clearer guidance on detailed topics in requirements development and management, to answer the hard questions that engineers have as they get down to the business of creating requirements for a specific project. This guidance might include the following topics, for example.
Making the distinction between requirements and design
The role of commentary in clarifying requirements (what should be part of a requirements vs. what kinds of material to put into commentary)
How design constraints at one level influence requirements at the next level
Making the requirements specification technique fit the item to be specified
How requirements will be managed over the life of a product (I'm alluding to the difference between developing requirements in the first place, and subsequently modifying requirements, tracking changes to requirements, etc., while maintaining the consistency and completeness of the requirements suite)
How to write requirements at the desired level of detail and avoid mixing levels
Examples, examples, examples to support the above

Survey Comment

Maybe I've just become used to DO-178B the way it currently is but I think it is clear enough (vague enough?) at least regarding requirements management.

Survey Comment

General: When a TSO does not exist, there may be cases where the overall system functions are achieved by the combination of the equipment and the A/C.
How to manage this shared responsibility is not addressed by the standards.

DO-178B:
Embedded Database Management (DO200A is fine for data preparation systems,
But not adequate for embedded applications)

Test Data reduction techniques (classes of equivalence, etc.)

Defensive Programming (it is not encouraged and may cause code coverage issues, while is a very powerful technique to ensure robustness)

DO-254:
It is proposed to encourage an analysis of the safety aspects related to independent functions on the same PLD, and to apply the rules in accordance to each function's safety levels (provided decoupling is ensured)

Survey Comment

DO-178B needs to address the issue of model-based development related to validation of the model and ensuring that the model accurately represents the system requirements.

Survey Comment

We would like to see more guidance on data flow, data coupling and control flow as well as high and low level requirements layering. Also, guidance as to when requirements stop and design starts. Some DO-178B compliance “enforcers”, such as certain FAA personnel and Airbus engineers are of the opinion that the detailed requirements should be at such a low level of detail that all programmers would generate basically identical code. Is this what we want?

Survey Comment

It has been our experience that companies developing avionics software struggle greatly with the low-level requirements issue. In these cases the low-level requirements are typically captured as “shall” statements. Smaller systems (typically less than 20,000 SLOC) often develop a significant number of low level requirements which requires a great deal of work in regard to testing and maintenance. When counting the number of high-level and low-level requirements for these smaller systems we typically see a much lower ratio of SLOC/rqmt compared to larger systems. Sometimes as low as 5 SLOC/rqmt. We’ve seen this push their overall cost/SLOC up.

Large systems (ranging up to 100s of thousands of SLOC) typically focus mainly on high-level requirements. Their ratios are typically more on the order of 2040 SLOC/rqmt and sometimes as high as 80 SLOC/rqmt depending on the type of system. However, if these larger system developers detailed lowlevel requirements to the same level that some of the smaller systems do, their costs would be unwieldy.

There is no silver bullet to solving this problem. However, there may be additional guidance that could be added, or examples provided, that address the level of detail issue.

Survey Comment

I think we all already know that requirements management is not adequate in DO-178B and the handoff to ARP4754 is not well defined. I think DO-254 vs. DO-178B is OK as DO-254 is clearly DO-178B based and is complete.
What is unclear about the guidance is the following.
effective linkages from the SSA to the Software implementation (e.g., there is no guidance for how to efficiently test the requirements imposed on SW from the SSA.
The software is considered, from a SSA viewpoint, typically a computing element and if the computing element fails, the system may have redundancy or some other failure accommodation scheme. One item missing is that there are many methods and tools that can be applied in Software to improve the safety of the system by improving, monitoring, etc. the reliability of the computing element. I feel we need guidance specific to software that permits improved safety opportunities for the system. (example: In many discrete HW systems memory with parity checking is standard. Right now we in the industry are using PowerPCs that have on-chip memory elements that do not have parity. Should a software parity scheme be devised?)

If we do use models as a basis for requirements, how do we validate the pedigree of these models?

Survey Comment

A definition of Requirements Engineering Management.

Survey Comment

There is almost no guidance for requirements engineering, which is OK.

Survey Comment

New guidance is needed when Model-Based Development is used to define HLR and/or LLR. Topics that should be addressed are related to simulation (or testing) of requirements and coverage analysis at model level.

F.10 Additional Comments.

Please provide any additional comments on Requirements Engineering Management you feel are relevant to this survey.

Survey Comment

The aerospace industry has evolved into a large number of sub-tier suppliers who do not have the staff or the sophistication to develop software, manage its configuration, or control its quality. The OEMs are badly understaffed and immature in their appreciation for discipline in dealing with the establishment, analysis, allocation and ultimately the verification and validation processes needed to manage engineering requirements. The resulting chaos is inefficient and ineffective but manages to work remarkably well!

Survey Comment

* What is the guidance on the use of models as requirements?
* To what extent can simulation be used for requirements validation?

Survey Comment

Use of such tools can lead to quicker development cycles, reduced cost, less integration time, more completely defined systems up front… advantages are numerous. Quality of tools and qualification of the tools by some other means than the standard compliance to DO-178B used for airborne software needs to be addressed in order to realize the capabilities of such tools. Service history, and/or incentive to vendors to cooperate with users seems the most logical means.

Survey Comment

Other issues to be considered:
management of changes (i.e., how are changes planned, documented, implemented, controlled, monitored)
handling and documentation of regression test after changes, especially the determination of major vs. minor changes
capture and management of customer requirements and how to keep them current (especially when customers are reluctant to incorporate and release changes to their own specifications)
the use of requirements metrics (e.g., Degree of Technical Compliance, productivity)

Survey Comment

The most significant issue we see with Requirements Engineering Management is the lack of priority it receives relative to the overall project. As long as the requirements "weigh" enough, then they are declared "good." But when your primary task is to develop test cases based on the requirements, "weight" does not help – accuracy, completeness, and detail does.

Survey Comment

Due to the natural means of architectural design and decomposition, there is the confusing predicament over definitions in DO-178B whenever it mentions:

- "the software" (What scale and scope does this refer to; the individual configurable item or the entire system? How does this relate to products that are designed as point products vs. software that's designed to be integrated in IMA?)
- "the system" (same issues as above)
- A lack of clear definitions of terms associated with system requirements, software requirements, and hardware requirements. This then creates confusion in many areas, in particular, some potential overlap of DO-178B with ARP4754.

DO-178B should remain a s/w process document and not try to be all encompassing. Systems requirements management guidance does not belong in DO-178B. FAA software specialists do not necessarily have the skills to provide guidance or make assessments of System requirements.

Survey Comment

We are still trying to figure out how to do model based development

Survey Comment

Our biggest problem with requirements is that the customer specs far too much detail – so the intent of those details is sometimes difficult to ascertain. And the testing of high-level requirements turns out to be at too-detailed a level because the customer has given us that level of detail. I wonder if MBD will aggravate this problem.

Survey Comment

<u>Robust Requirements Analysis: The Key to Lower Cost DO-178B Certified Software</u>
The real culprit behind many certifiable-software development woes is the requirements analysis process.

Refer to this white paper: "Robust Requirements Analysis: The Key to Low Cost DO-178B Certified Software, Hoyt Lougee, Foilage Industries, January 18, 2005."

Survey Comment

Sometimes I get the feeling that improvements in requirements engineering management is regarded as simply finding the perfect tool. My preference would be to give more emphasis to training engineers on the criteria and thinking process required to produce high quality, testable requirements. Toolsets would be brought in to assist the engineers in achieving clarity and completeness in their requirements.

Survey Comment

Some experiments have been conducted on Object-oriented Requirement Engineering in order to obtain complete reusable Objects (from Requirements through to System Test).
This topic should be further explored.

Survey Comment

I believe as we develop more integrated systems, I believe we will need to either assure we have clear partitions where functions cannot conflict. Effective requirements management for these IMA systems is necessary to validate that either the a. the partitions are so strong that the functions are not at all coupled or b. that the requirements management process can accommodate and flag potential requirement conflicts.

APPENDIX G—INDUSTRY SURVEY COMMENTS ORGANIZED BY ISSUE

G.1 General Issues.

G.1.1 What is Requirements Engineering Management?

Survey Comment

Other issues to be considered:
management of changes (i.e., how are changes planned, documented, implemented, controlled, monitored)
handling and documentation of regression test after changes, especially the determination of major vs. minor changes
capture and management of customer requirements and how to keep them current (especially when customers are reluctant to incorporate and release changes to their own specifications)
the use of requirements metrics (e.g., Degree of Technical Compliance, productivity)

Survey Comment

Sometimes I get the feeling that improvements in requirements engineering management is regarded as simply finding the perfect tool. My preference would be to give more emphasis to training engineers on the criteria and thinking process required to produce high quality, testable requirements. Toolsets would be brought in to assist the engineers in achieving clarity and completeness in their requirements.

Survey Comment

A definition of Requirements Engineering Management.

G.1.2 Clarification of DO-178B Terms and Guidance.

Survey Comment

No ability to show determinism as required in the high levels of DO178B.
No clear industry discussion of safe constructs or a safe subset of the method.

Survey Comment

Low level requirements definition. What is considered as low level requirements in OOD? Use Cases and State diagrams?
Traceability low level requirements <-> high-level requirements is not straightforward in OOD.
low level requirements testing : What is the meaning of full requirements coverage and structural coverage in OOD

Survey Comment

It’s not coverd by the DO178-B.
It requires a restriction on a subset of OO techniques (i.e., avoid inheritence and dynamic creation of objects).

Survey Comment

Considerations of how effectively OO design can be implemented to achieve DO-178B certification.

OO design has worked well for us in breaking the software problem set down to its smallest components, but it doesn't necessarily convey all the integration issues between hardware and software.

Survey Comment

New guidance is needed when Model-Based Development is used to define HLR and/or LLR. Topics that should be addressed are related to simulation (or testing) of requirements and coverage analysis at model level.

Survey Comment

Model-based development (MBD) simply sounds like rapid prototyping via simulation (rather than via rapid HW & SW development). The net result can therefore be overly-detailed SYS requirements -- with no rationale and no documentation of the true "high-level" intent or development philosophy. Indeed, it seems to move "SW rqmt & design" step to the SYS arena. But what SYS creates are not SYS requirements so much as they are low-level SW requirements - except without the process "care" that DO178B intended.

Survey Comment

There is no standard formatting, the process is arbitrary and not uniformly administered by the various FAA ACOs.

Survey Comment

Think there's too much "guidance". Too many versions of the same idea

Survey Comment

More guidance for lower-level requirements in terms of documentation, trace, test

DO-254 - not clear if (and how much) is in force

More guidance for how changes need to be handled during maintenance (i.e., post certification) - documentation, regression test, etc., especially for changes to address obsolescence

Survey Comment

Projects live and die by the completeness and accuracy of traceability information. The ease with which requirements can be captured, traced, elaborated, and evolved should be emphasized at the beginning of the project in deciding on tools and methods.

What constitutes a low level requirement has much latitude for interpretation.

Survey Comment

It has been our experience that companies developing avionics software struggle greatly with the low-level requirements issue. In these cases the low-level requirements are typically captured as "shall" statements. Smaller systems (typically less than 20,000 SLOC) often develop a significant number of low level requirements which requires a great deal of work in regard to testing and maintenance. When counting the number of high-level and low-level requirements for these smaller systems we typically see a much lower ratio of SLOC/rqmt compared to larger systems. Sometimes as low as 5 SLOC/rqmt. We've seen this push their overall cost/SLOC up.

Large systems (ranging up to 100s of thousands of SLOC) typically focus mainly on high-level requirements. Their ratios are typically more on the order of 20-40 SLOC/rqmt and sometimes as high as 80 SLOC/rqmt depending on the type of system. However, if these larger system developers detailed low-level requirements to the same level that some of the smaller systems do, their costs would be unwieldy.
There is no silver bullet to solving this problem. However, there may be additional guidance that could be added, or examples provided, that address the level of detail issue.

Survey Comment

Requirements engineering is critical to the cost effective implementation and verification of systems. As such, the standards to a poor job providing guidance as to what is to be accomplished by the various levels.

Survey Comment

The definitions of High Level vs Low Level Requirements leave a lot to personal opinion. I have heard statements that one has directly coded the whole design from high-level requirements, and that no low level requirements were needed. Whereas I interpret that as meaning that they defined the implementation into the high-level requirements. So in effect their high-level requirements are really low level ones and have not documented any high-level requirements. Some guidelines or standardization on how many high-level requirements vs low level requirements vs lines of code are typical for a project. Statements such as “Testing high-level requirements should not require white box testing as the way low level requirements do” may help clarify if one is dealing with a low level requirement or not.

Survey Comment

The emphasis on verification of requirements has lead to a number of futile dodges that a) don't buy much additional safety and b) cost a lot of money to accomplish. When people start labeling their design diagrams "Requirements" and then go out to verify that the code generated matches the design diagrams, this is not right. Design diagrams are NOT "Requirements" - they are "DESIGN". In the case of auto code generation, they are "IMPLEMENTATION". How is a Simulink (or similar tool) drawing different than expressing an implementation in Ada, Fortran or C? Do you verify that the output of an Ada compiler matches the source code statements in any way other than through normal testing to see if you get the desired result? There is extremely little value in checking that a compiler takes "Y := M * X + B;" and translates it into "Load A,X; Mult A,M; Add A,B; Store A,Y;". None of that tells you if "Y := M * X + B;" was the right thing to do in the first place. What if the operation should have been "-" instead of "+"? What we need is better guidance on what constitutes a valid, testable requirement and what levels of gyrations are needed to verify it.

Survey Comment

The documents lack in clarity in some places. Plus the FAA individuals seem to have specific interpretations that may not agree between everyone. Experience and mentoring seems to be the best teacher.

Survey Comment

DO-178B is, by design, very high-level and sometimes quite vague.
I would love to see clearer guidance on detailed topics in requirements development and management, to answer the hard questions that engineers have as they get down to the business of creating requirements for a specific project. This guidance might include the following topics, for example.
Making the distinction between requirements and design
The role of commentary in clarifying requirements (what should be part of a requirements vs. what kinds of material to put into commentary)
How design constraints at one level influence requirements at the next level
Making the requirements specification technique fit the item to be specified
How requirements will be managed over the life of a product (I'm alluding to the difference between developing requirements in the first place, and subsequently modifying requirements, tracking changes to requirements, etc., while maintaining the consistency and completeness of the requirements suite)
How to write requirements at the desired level of detail and avoid mixing levels
Examples, examples, examples to support the above

Survey Comment

We would like to see more guidance on data flow, data coupling and control flow as well as high and low level requirements layering. Also, guidance as to when requirements stop and design starts. Some DO-178B compliance "enforcers", such as certain FAA personnel and Airbus engineers are of the opinion that the detailed requirements should be at such a low level of detail that all programmers would generate basically identical code. Is this what we want?

Survey Comment

As mentioned above, DO-178B does not provide good definitions that allow for an unambiguous interpretation of "system requirement", "software requirement", and "hardware requirement". Where are the boundaries? We have rigorous processes around SW, but very loose processes relative to system requirements. Also, there are many that claim that DO-178B A3-1 means that the requirements have to trace to the aircraft level requirements from the airframe manufacturer. The biggest problem with these documents and cert. authority interpretation is the abstraction between all the different levels of requirements including Systems Requirements.

Survey Comment

By far, the biggest issue is requirements. IMA must merge both ARP4754 with DO178B. The first question to answer is "where do system requirements stop and s/w requirements begin?"

Survey Comment

Due to the natural means of architectural design and decomposition, there is the confusing predicament over definitions in DO-178B whenever it mentions:
"the software" (What scale and scope does this refer to; the individual configurable item or the entire system? How does this relate to products that are designed as point products vs. software that's designed to be integrated in IMA?)
"the system" (same issues as above)
A lack of clear definitions of terms associated with system requirements, software requirements, and hardware requirements. This then creates confusion in many areas, in particular, some potential overlap of DO-178B with ARP-4754.

DO-178B should remain a s/w process document and not try to be all encompassing. Systems requirements management guidance does not belong in DO-178B. FAA software specialists do not necessarily have the skills to provide guidance or make assessments of System requirements.

Survey Comment

I think we all already know that requirements management is not adequate in DO-178B and the hand-off to ARP4754 is not well defined. I think DO-254 vs. DO-178B is OK as DO-254 is clearly DO-178B based and is complete.
What is unclear about the guidance is the following.
effective linkages from the SSA to the Software implementation (e.g., there is no guidance for how to efficiently test the requirements imposed on SW from the SSA.
The software is considered, from a SSA viewpoint, typically a computing element and if the computing element fails, the system may have redundancy or some other failure accommodation scheme. One item missing is that there are many methods and tools that can be applied in Software to improve the safety of the system by improving, monitoring, etc. the reliability of the computing element. I feel we need guidance specific to software that permits improved safety opportunities for the system. (example: In many discrete HW systems memory with parity checking is standard. Right now we in the industry are using PowerPCs that have on-chip memory elements that do not have parity. Should a software parity scheme be devised?)
If we do use models as a basis for requirements, how do we validate the pedigree of these models?

Survey Comment

Concept of safety-related requirements
What to do with these safety-related requirements during the development process is not clearly defined.
What consequences and impact on testing for these safety related requirements
Concept of derived requirements
Currently identification of derived requirements is required, but what is the added value and for what is it useful?
DO-178B doesn't talk about the importance of the rationale or design decision leading to these derived requirements.
Different requirement levels (granularity of requirements)
DO-178B talks about different levels in requirements from which the highest level is called the high-level requirements and the lowest level is called low level requirements. For these 2 levels the required objectives and activities are well defined.
However one may have additional requirement levels in between. For these it is not very clear what activities need to be performed.
Correlation to HW requirements
Where do HW requirements interfere with SW requirements
What activities need to be performed when performing concurrent development of SW and HW and its interfaces.
Requirements and relation to SW level
In practice we see that the abstraction level (i.e., level of detail) of SW requirements for a level A project is not the same as for a level D project. However this relation between the SW level and the SW requirements is not clearly defined.

Survey Comment

DO-178B is redundantly ambiguous. If I wrote a software specification that was like DO-178B I'd be fired in a heart beat. The writing committee should repeat the sacred mantra 'Unambiguous, Clear and Concise' 1000 times before writing each word.

Survey Comment

What is the guidance on the use of models as requirements?
To what extent can simulation be used for requirements validation?

Survey Comment

Where are the "shalls" in a model. How are the requirements traced to higher and lower-level requirements and / or design?

Survey Comment

The use of MDB methodologies or tools should not be dictated. They are project-dependent. The use of model-based tools & methodologies must be assessed on a case-by-case basis by the developer. What works for one development activity may not work for another. Also, there can be an overemphasis on tooling in lieu of lack of overall system experience; tools do not write good requirements, engineers do. A tool is no better than the quality of data put into it. Therefore, requirements regarding MDB tools or methodologies are impractical to specify.
Also, it is difficult to use DO-178B processes with Model Based Development. DO-178B does not have precise and widely agreed upon definitions for System Requirements, HL Software Requirements, LL Software Requirements and HW Requirements. Nor should it. This is project dependent.
Model based development introduces a new level of abstraction, and translation. MBD should not be extrapolated to everything as this results in an abuse of the methodology.
Comment: In general, models can provide some clarity to text-based requirements. They provide a mechanism to help understand the text-based requirements but they are not a substitute.
MBD tools have a great deal of capability and flexibility in choosing what you want models to represent. Therefore, based upon what is represented in the models, as well as what is represented in artifacts outside of the models, results in all possibilities being potentially suitable uses for models.

Survey Comment

In concept, it would be easy to fit the simulation models into the Low Level Requirements category, and the resulting generated C code into the Source Code category. This raises the following issues:

Is the generated code traceable to the models?
Are the models traceable to the generated code?
Does the generated code need to be verified against the models?
Does the code generator need to be qualified?
How much certification credit (if any) can be taken for verification via simulations?

Another strategy might be to claim that the model is actually both the Low Level Requirements and the Source Code. This eliminates an entire set of DO-178B processes relating to Source Code. However, this brings up the issue of Tool Qualification. The code generator would have to be qualified as a development tool under DO-178B. However, the development tool qualification could require as much or more effort than required to take the traditional approach.

Survey Comment

Model based development (e.g., UML models, in contrast to function based models like Matlab) It's not coverd by the DO178-B. The role of the model respectively the role of its elements in the requirements process is not clear resp. not addressed by the DO178-B: Is it optional, just for understanding the system? Many people would deny using it, because it - superficially - adds cost, although it adds value, because people have the systems under control, which they are going to build. This in turn speeds up development and finally saves cost.

Survey Comment

General: When a TSO does not exist, there may be cases where the overall system functions are achieved by the combination of the equipment and the A/C. How to manage this shared responsibility is not addressed by the standards.

DO-178B:
Embedded Database Management (DO200A is fine for data preparation systems, but not adequate for embedded applications)
Test Data reduction techniques (classes of equivalence, etc.)
Defensive Programming (it is not encouraged and may cause code coverage issues, while is a very powerful technique to ensure robustness)

DO-254:
It is proposed to encourage an analysis of the safety aspects related to independent functions on the same PLD, and to apply the rules in accordance to each function's safety levels (provided decoupling is ensured)

DO-178B needs to address the issue of model-based development related to validation of the model and ensuring that the model accurately represents the system requirements.

G.1.3 Do We Really Need More Guidance on Requirements Engineering Management?

Survey Comment

Not inadequate

Survey Comment

Maybe I've just become used to DO-178B the way it currently is but I think it is clear enough (vague enough?) at least regarding requirements management.

Survey Comment

There is almost no guidance for requirements engineering, which is OK.

Survey Comment

The aerospace industry has evolved into a large number of sub-tier suppliers who do not have the staff or the sophistication to develop software, manage its configuration, or control its quality. The OEMs are badly understaffed and immature in their appreciation for discipline in dealing with the establishment, analysis, allocation and ultimately the verification and validation processes needed to manage engineering requirements. The resulting chaos is inefficient and ineffective- but manages to work remarkably well!

G.2 Safety Issues.

G.2.1 What Should be Done Differently During Development Due to Safety Requirements?

Survey Comment

I think we all already know that requirements management is not adequate in DO-178B and the hand-off to ARP4754 is not well defined. I think DO-254 vs. DO-178B is OK as DO-254 is clearly DO-178B based and is complete.
What is unclear about the guidance is the following.
effective linkages from the SSA to the Software implementation (e.g., there is no guidance for how to efficiently test the requirements imposed on SW from the SSA.
The software is considered, from a SSA viewpoint, typically a computing element and if the computing element fails, the system may have redundancy or some other failure accommodation scheme. One item missing is that there are many methods and tools that can be applied in Software to improve the safety of the system by improving, monitoring, etc. the reliability of the computing element. I feel we need guidance specific to software that permits improved safety opportunities for the system. (example: In many discrete HW systems memory with parity checking is standard. Right now we in the industry are using PowerPCs that have on-chip memory elements that do not have parity. Should a software parity scheme be devised?)
If we do use models as a basis for requirements, how do we validate the pedigree of these models?

Survey Comment

How are the requirements reviewed? Do they violate safety assumptions? Where are the interfaces different?

Survey Comment

Concept of safety-related requirements
What to do with these safety-related requirements during the development process is not clearly defined.
What consequences and impact on testing for these safety related requirements
Concept of derived requirements
Currently identification of derived requirements is required, but what is the added value and for what is it useful?
DO-178B doesn't talk about the importance of the rationale or design decision leading to these derived requirements.
Different requirement levels (granularity of requirements)
DO-178B talks about different levels in requirements from which the highest level is called the high-level requirements and the lowest level is called low level requirements. For these 2 levels the required objectives and activities are well defined.
However one may have additional requirement levels in between. For these it is not very clear what activities need to be performed.
Correlation to HW requirements
Where do HW requirements interfere with SW requirements

What activities need to be performed when performing concurrent development of SW and HW and its interfaces.
Requirements and relation to SW level

In practice we see that the abstraction level (i.e., level of detail) of SW requirements for a level A project is not the same as for a level D project. However this relation between the SW level and the SW requirements is not clearly defined.

G.2.2 Can ARP 4754 and DO-178B be Better Integrated?

Survey Comment

I think we all already know that requirements management is not adequate in DO-178B and the hand-off to ARP4754 is not well defined. I think DO-254 vs. DO-178B is OK as DO-254 is clearly DO-178B based and is complete.
What is unclear about the guidance is the following.
effective linkages from the SSA to the Software implementation (e.g., there is no guidance for how to efficiently test the requirements imposed on SW from the SSA.
The software is considered, from a SSA viewpoint, typically a computing element and if the computing element fails, the system may have redundancy or some other failure accommodation scheme. One item missing is that there are many methods and tools that can be applied in Software to improve the safety of the system by improving, monitoring, etc. the reliability of the computing element. I feel we need guidance specific to software that permits improved safety opportunities for the system. (example: In many discrete HW systems memory with parity checking is standard. Right now we in the industry are using PowerPCs that have on-chip memory elements that do not have parity. Should a software parity scheme be devised?)
If we do use models as a basis for requirements, how do we validate the pedigree of these models?

G.2.3 What Techniques Can be Used to Improve Software Safety?

Survey Comment

I think we all already know that requirements management is not adequate in DO-178B and the hand-off to ARP4754 is not well defined. I think DO-254 vs. DO-178B is OK as DO-254 is clearly DO-178B based and is complete.
What is unclear about the guidance is the following.
effective linkages from the SSA to the Software implementation (e.g., there is no guidance for how to efficiently test the requirements imposed on SW from the SSA.
The software is considered, from a SSA viewpoint, typically a computing element and if the computing element fails, the system may have redundancy or some other failure accommodation scheme. One item missing is that there are many methods and tools that can be applied in Software to improve the safety of the system by improving, monitoring, etc. the reliability of the computing element. I feel we need guidance specific to software that permits improved safety opportunities for the system. (example: In many discrete HW systems memory with parity checking is standard. Right now we in the industry are using PowerPCs that have on-chip memory elements that do not have parity. Should a software parity scheme be devised?)

If we do use models as a basis for requirements, how do we validate the pedigree of these models?

Survey Comment

General: When a TSO does not exist, there may be cases where the overall system functions are achieved by the combination of the equipment and the A/C. How to manage this shared responsibility is not addressed by the standards.

DO-178B:
Embedded Database Management (DO200A is fine for data preparation systems, but not adequate for embedded applications)
Test Data reduction techniques (classes of equivalence, etc.)
Defensive Programming (it is not encouraged and may cause code coverage issues, while is a very powerful technique to ensure robustness)

DO-254:
It is proposed to encourage an analysis of the safety aspects related to independent functions on the same PLD, and to apply the rules in accordance to each function's safety levels (provided decoupling is ensured)

DO-178B needs to address the issue of model-based development related to validation of the model and ensuring that the model accurately represents the system requirements.

G.2.4 How Should the Intent of Requirements be Specified?

Survey Comment

The emphasis on verification of requirements has lead to a number of futile dodges that a) don't buy much additional safety and b) cost a lot of money to accomplish. When people start labeling their design diagrams "Requirements" and then go out to verify that the code generated matches the design diagrams, this is not right. Design diagrams are NOT "Requirements" - they are "DESIGN". In the case of auto code generation, they are "IMPLEMENTATION". How is a Simulink (or similar tool) drawing different than expressing an implementation in Ada, Fortran or C? Do you verify that the output of an Ada compiler matches the source code statements in any way other than through normal testing to see if you get the desired result? There is extremely little value in checking that a compiler takes "Y := M * X + B;" and translates it into "Load A,X; Mult A,M; Add A,B; Store A,Y;". None of that tells you if "Y := M * X + B;" was the right thing to do in the first place. What if the operation should have been "-" instead of "+"?

What we need is better guidance on what constitutes a valid, testable requirement and what levels of gyrations are needed to verify it.

Survey Comment

DO-178B is, by design, very high-level and sometimes quite vague.
I would love to see clearer guidance on detailed topics in requirements development and management, to answer the hard questions that engineers have as they get down to the business of creating requirements for a specific project. This guidance might include the following topics, for example.

Making the distinction between requirements and design
The role of commentary in clarifying requirements (what should be part of a requirements vs. what kinds of material to put into commentary)
How design constraints at one level influence requirements at the next level
Making the requirements specification technique fit the item to be specified
How requirements will be managed over the life of a product (I'm alluding to the difference between developing requirements in the first place, and subsequently modifying requirements, tracking changes to requirements, etc., while maintaining the consistency and completeness of the requirements suite)
How to write requirements at the desired level of detail and avoid mixing levels
Examples, examples, examples to support the above

Survey Comment

If Models will replace high-level and low-level software requirements (i.e., we write the requirements using model-language instead of English-language). We must ensure the systems and subsystems team members are fluent in model-language so that their peer reviews are meaningful. Also, requirements decomposition is a form of complexity management, i.e., the system is broken into smaller and smaller pieces, each being manageable steps, until the end result is achievable. Models may represent a too large of a step that the mind can comfortably take at once. Also, Models, being computer programs, are often less forgiving than English text which can accommodate a large amount of ambiguity be resolved in order to define a valid system. As a example, in English we might say "sign here" whereas a model language may require us to say "using a ball-point pen with a .7mm tip containing black ink in a cursive format with a font size of no less that 10 point and no more than 20 point, etc. … We may lose the actual requirement in the details.

Survey Comment

Model-based development (MBD) simply sounds like rapid prototyping via simulation (rather than via rapid HW & SW development). The net result can therefore be overly-detailed SYS requirements -- with no rationale and no documentation of the true "high-level" intent or development philosophy. Indeed, it seems to move "SW rqmt & design" step to the SYS arena. But what SYS creates are not SYS requirements so much as they are low-level SW requirements - except without the process "care" that DO178B intended.

G.2.5 How Should Environmental Assumptions be Specified?

No Comments

G.2.6 How Should Human Factors Requirements be Specified?

Survey Comment

Human-Machine Interface prototyping and review with the Customer

G.3 Certification Issues.

G.3.1 What is the Relationship of System and Software Requirements?

Survey Comment

As mentioned above, DO-178B does not provide good definitions that allow for an unambiguous interpretation of “system requirement”, “software requirement”, and “hardware requirement”. Where are the boundaries? We have rigorous processes around SW, but very loose processes relative to system requirements. Also, there are many that claim that DO-178B A3-1 means that the requirements have to trace to the aircraft level requirements from the airframe manufacturer. The biggest problem with these documents and cert. authority interpretation is the abstraction between all the different levels of requirements including Systems Requirements.

Survey Comment

By far, the biggest issue is requirements. IMA must merge both ARP4754 with DO178B. The first question to answer is “where do system requirements stop and s/w requirements begin?”

Survey Comment

Our biggest problem with requirements is that the customer specs far too much detail - so the intent of those details is sometimes difficult to ascertain. And the testing of high-level requirements turns out to be at too-detailed a level because the customer has given us that level of detail. I wonder if MBD will aggravate this problem.

Survey Comment

Due to the natural means of architectural design and decomposition, there is the confusing predicament over definitions in DO-178B whenever it mentions:
“the software” (What scale and scope does this refer to; the individual configurable item or the entire system? How does this relate to products that are designed as point products vs. software that’s designed to be integrated in IMA?)
“the system” (same issues as above)
A lack of clear definitions of terms associated with system requirements, software requirements, and hardware requirements. This then creates confusion in many areas, in particular, some potential overlap of DO-178B with ARP-4754.
DO-178B should remain a s/w process document and not try to be all encompassing. Systems requirements management guidance does not belong in DO-178B. FAA software specialists do not necessarily have the skills to provide guidance or make assessments of System requirements.

Survey Comment

DO-178B was SW oriented. “Requirements Engineering Management” demands a SYS oriented document.

Survey Comment

General: When a TSO does not exist, there may be cases where the overall system functions are achieved by the combination of the equipment and the A/C. How to manage this shared responsibility is not addressed by the standards.

DO-178B:
Embedded Database Management (DO200A is fine for data preparation systems, but not adequate for embedded applications)

Test Data reduction techniques (classes of equivalence, etc.)
Defensive Programming (it is not encouraged and may cause code coverage issues, while is a very powerful technique to ensure robustness)

DO-254:
It is proposed to encourage an analysis of the safety aspects related to independent functions on the same PLD, and to apply the rules in accordance to each function's safety levels (provided decoupling is ensured)

DO-178B needs to address the issue of model-based development related to validation of the model and ensuring that the model accurately represents the system requirements.

Survey Comment

A better SAE ARP4754. This only partially makes the relationship between hardware reliability and software quality level clear. Also it's Validation standards are totally unrealistic in relation to what industry does.

G.3.2 What is the Relationship of Software Requirements and Software Design?

Survey Comment

The most significant issue we see with Requirements Engineering Management is the lack of priority it receives relative to the overall project. As long as the requirements "weigh" enough, then they are declared "good." But when your primary task is to develop test cases based on the requirements, "weight" does not help - accuracy, completeness, and detail does.

Survey Comment

It is difficult to identify or tag what an actual requirement is within the model.
It is difficult to generate traceability to portions of the model.

Survey Comment

After having written several requirements documents as well as critiquing several others, and having to write test cases based on poorly written requirements, I had at one time come to the conclusion that generating good requirements is more of an art form that a science, something that is learned over time and with experience. However, if this can be learned over time, then it should be possible to record that knowledge and that others should be able to learn it without having to learn from experience. I believe that a set of rules including, language, syntax, and clear definitions, could be devised that would greatly improve the quality of the requirements.

Survey Comment

Projects live and die by the completeness and accuracy of traceability information. The ease with which requirements can be captured, traced, elaborated, and evolved should be emphasized at the beginning of the project in deciding on tools and methods.

What constitutes a low level requirement has much latitude for interpretation.

Survey Comment

It has been our experience that companies developing avionics software struggle greatly with the low-level requirements issue. In these cases the low-level requirements are typically captured as

"shall" statements. Smaller systems (typically less than 20,000 SLOC) often develop a significant number of low level requirements which requires a great deal of work in regard to testing and maintenance. When counting the number of high-level and low-level requirements for these smaller systems we typically see a much lower ratio of SLOC/rqmt compared to larger systems. Sometimes as low as 5 SLOC/rqmt. We've seen this push their overall cost/SLOC up.
Large systems (ranging up to 100s of thousands of SLOC) typically focus mainly on high-level requirements. Their ratios are typically more on the order of 20-40 SLOC/rqmt and sometimes as high as 80 SLOC/rqmt depending on the type of system. However, if these larger system developers detailed low-level requirements to the same level that some of the smaller systems do, their costs would be unwieldy.
There is no silver bullet to solving this problem. However, there may be additional guidance that could be added, or examples provided, that address the level of detail issue.

Survey Comment

Requirements engineering is critical to the cost effective implementation and verification of systems. As such, the standards to a poor job providing guidance as to what is to be accomplished by the various levels.

Survey Comment

Concept of safety-related requirements
What to do with these safety-related requirements during the development process is not clearly defined.
What consequences and impact on testing for these safety related requirements
Concept of derived requirements
Currently identification of derived requirements is required, but what is the added value and for what is it useful?
DO-178B doesn't talk about the importance of the rationale or design decision leading to these derived requirements.
Different requirement levels (granularity of requirements)
DO-178B talks about different levels in requirements from which the highest level is called the high-level requirements and the lowest level is called low level requirements. For these 2 levels the required objectives and activities are well defined.
However one may have additional requirement levels in between. For these it is not very clear what activities need to be performed.
Correlation to HW requirements
Where do HW requirements interfere with SW requirements
What activities need to be performed when performing concurrent development of SW and HW and its interfaces.
Requirements and relation to SW level

In practice we see that the abstraction level (i.e., level of detail) of SW requirements for a level A project is not the same as for a level D project. However this relation between the SW level and the SW requirements is not clearly defined.

Survey Comment

The definitions of High Level vs Low Level Requirements leave a lot to personal opinion. I have heard statements that one has directly coded the whole design from high-level requirements, and that no low level requirements were needed. Whereas I interpret that as meaning that they defined the implementation into the high-level requirements. So in effect their high-level requirements are really low level ones and have not documented any high-level requirements. Some guidelines or standardization on how many high-level requirements vs low level requirements vs lines of code are typical for a project. Statements such as "Testing high-level requirements should not require white box testing as the way low level requirements do" may help clarify if one is dealing with a low level requirement or not.

Survey Comment

The emphasis on verification of requirements has lead to a number of futile dodges that a) don't buy much additional safety and b) cost a lot of money to accomplish. When people start labeling their design diagrams "Requirements" and then go out to verify that the code generated matches the design diagrams, this is not right. Design diagrams are NOT "Requirements" - they are "DESIGN". In the case of auto code generation, they are "IMPLEMENTATION". How is a Simulink (or similar tool) drawing different than expressing an implementation in Ada, Fortran or C? Do you verify that the output of an Ada compiler matches the source code statements in any way other than through normal testing to see if you get the desired result? There is extremely little value in checking that a compiler takes "Y := M * X + B;" and translates it into "Load A,X; Mult A,M; Add A,B; Store A,Y;". None of that tells you if "Y := M * X + B;" was the right thing to do in the first place. What if the operation should have been "-" instead of "+"? What we need is better guidance on what constitutes a valid, testable requirement and what levels of gyrations are needed to verify it.

Survey Comment

The documents lack in clarity in some places. Plus the FAA individuals seem to have specific interpretations that may not agree between everyone. Experience and mentoring. seems to be the best teacher.

Survey Comment

DO-178B is, by design, very high-level and sometimes quite vague.
I would love to see clearer guidance on detailed topics in requirements development and management, to answer the hard questions that engineers have as they get down to the business of creating requirements for a specific project. This guidance might include the following topics, for example.
Making the distinction between requirements and design
The role of commentary in clarifying requirements (what should be part of a requirements vs. what kinds of material to put into commentary)
How design constraints at one level influence requirements at the next level
Making the requirements specification technique fit the item to be specified
How requirements will be managed over the life of a product (I'm alluding to the difference between developing requirements in the first place, and subsequently modifying requirements, tracking changes to requirements, etc., while maintaining the consistency and completeness of the requirements suite)
How to write requirements at the desired level of detail and avoid mixing levels
Examples, examples, examples to support the above

Survey Comment

We would like to see more guidance on data flow, data coupling and control flow as well as high and low level requirements layering. Also, guidance as to when requirements stop and design starts. Some DO-178B compliance "enforcers", such as certain FAA personnel and Airbus engineers are of the opinion that the detailed requirements should be at such a low level of detail that all programmers would generate basically identical code. Is this what we want?

G.3.3 What is the Relationship of Software and Hardware Requirements?

Survey Comment

As mentioned above, DO-178B does not provide good definitions that allow for an unambiguous interpretation of "system requirement", "software requirement", and "hardware requirement". Where are the boundaries? We have rigorous processes around SW, but very loose processes relative to system requirements. Also, there are many that claim that DO-178B A3-1 means that the requirements have to trace to the aircraft level requirements from the airframe manufacturer. The biggest problem with these documents and cert. authority interpretation is the abstraction between all the different levels of requirements including Systems Requirements.

Survey Comment

Concept of safety-related requirements
What to do with these safety-related requirements during the development process is not clearly defined.
What consequences and impact on testing for these safety related requirements
Concept of derived requirements
Currently identification of derived requirements is required, but what is the added value and for what is it useful?
DO-178B doesn't talk about the importance of the rationale or design decision leading to these derived requirements.
Different requirement levels (granularity of requirements)
DO-178B talks about different levels in requirements from which the highest level is called the high-level requirements and the lowest level is called low level requirements. For these 2 levels the required objectives and activities are well defined.
However one may have additional requirement levels in between. For these it is not very clear what activities need to be performed.
Correlation to HW requirements
Where do HW requirements interfere with SW requirements
What activities need to be performed when performing concurrent development of SW and HW and its interfaces.
Requirements and relation to SW level

In practice we see that the abstraction level (i.e., level of detail) of SW requirements for a level A project is not the same as for a level D project. However this relation between the SW level and the SW requirements is not clearly defined.

Survey Comment

Due to the natural means of architectural design and decomposition, there is the confusing predicament over definitions in DO-178B whenever it mentions:
"the software" (What scale and scope does this refer to; the individual configurable item or the entire system? How does this relate to products that are designed as point products vs. software that's designed to be integrated in IMA?)

"the system" (same issues as above)
A lack of clear definitions of terms associated with system requirements, software requirements, and hardware requirements. This then creates confusion in many areas, in particular, some potential overlap of DO-178B with ARP-4754.

DO-178B should remain a s/w process document and not try to be all encompassing. Systems requirements management guidance does not belong in DO-178B. FAA software specialists do not necessarily have the skills to provide guidance or make assessments of System requirements.

G.3.4 Should Requirements be Specified Differently in Different Domains?

Survey Comment
DO-178B is, by design, very high-level and sometimes quite vague.
I would love to see clearer guidance on detailed topics in requirements development and management, to answer the hard questions that engineers have as they get down to the business of creating requirements for a specific project. This guidance might include the following topics, for example.
Making the distinction between requirements and design
The role of commentary in clarifying requirements (what should be part of a requirements vs. what kinds of material to put into commentary)
How design constraints at one level influence requirements at the next level
Making the requirements specification technique fit the item to be specified
How requirements will be managed over the life of a product (I'm alluding to the difference between developing requirements in the first place, and subsequently modifying requirements, tracking changes to requirements, etc., while maintaining the consistency and completeness of the requirements suite)
How to write requirements at the desired level of detail and avoid mixing levels
Examples, examples, examples to support the above

G.3.5 How Should Traceability and Change of Requirements be Managed?

Survey Comment
Other issues to be considered:
management of changes (i.e., how are changes planned, documented, implemented, controlled, monitored)
handling and documentation of regression test after changes, especially the determination of major vs. minor changes
capture and management of customer requirements and how to keep them current (especially when customers are reluctant to incorporate and release changes to their own specifications)
the use of requirements metrics (e.g., Degree of Technical Compliance, productivity)

Survey Comment
One difficult issue for reqs management is the use of different tools and integration of those tools for effective traceability and configuration control

Survey Comment
As mentioned above, DO-178B does not provide good definitions that allow for an unambiguous interpretation of "system requirement", "software requirement", and "hardware requirement".

Where are the boundaries? We have rigorous processes around SW, but very loose processes relative to system requirements. Also, there are many that claim that DO-178B A3-1 means that the requirements have to trace to the aircraft level requirements from the airframe manufacturer. The biggest problem with these documents and cert. authority interpretation is the abstraction between all the different levels of requirements including Systems Requirements.

Survey Comment

Acceptable means of applying level 1 configuration control to requirements and traceability data within a modeling tool or database (such as DOORS) tool

Survey Comment

More guidance for lower-level requirements in terms of documentation, trace, test
DO-254 - not clear if (and how much) is in force
More guidance for how changes need to be handled during maintenance (i.e., post certification) - documentation, regression test, etc., especially for changes to address obsolescence

Survey Comment

I feel that the requirements should precede the actual code development, but not in such a rigidly structured format as is required by DO-178B. I feel that the act of documenting the requirements is good, but it shouldn't be required to be in a formally 'released' state, and require tons of paperwork to change once the actual development is underway. Most people need to 'play' with their code before it is really where they want it to be, and with DO-178B to stray from the first iteration of code is cost prohibitive in terms of the documentation required to change any of the requirements. Hence, the product ends up with what may not be the 'best' solution for the given task, or as is typically the case, the process gets short cut. I think it would be better to loosen things up to allow the developers to optimize their solutions before rigid controls are placed on the design documents.

Survey Comment

Not so disappointed with the areas in Do178B/254 in the requirements area, my heartburn has to do with the assumption that you can plan a project to great detail at the start, run it for 2 years and not have a myriad of changes at the end.

Survey Comment

DO-178B is, by design, very high-level and sometimes quite vague.
I would love to see clearer guidance on detailed topics in requirements development and management, to answer the hard questions that engineers have as they get down to the business of creating requirements for a specific project. This guidance might include the following topics, for example.
Making the distinction between requirements and design
The role of commentary in clarifying requirements (what should be part of a requirements vs. what kinds of material to put into commentary)
How design constraints at one level influence requirements at the next level
Making the requirements specification technique fit the item to be specified
How requirements will be managed over the life of a product (I'm alluding to the difference between developing requirements in the first place, and subsequently modifying requirements, tracking changes to requirements, etc., while maintaining the consistency and completeness of the requirements suite)
How to write requirements at the desired level of detail and avoid mixing levels

Examples, examples, examples to support the above

Survey Comment

Projects live and die by the completeness and accuracy of traceability information. The ease with which requirements can be captured, traced, elaborated, and evolved should be emphasized at the beginning of the project in deciding on tools and methods.

What constitutes a low level requirement has much latitude for interpretation.

G.3.6 How Should Timing Requirements be Specified?

Survey Comment

Difficulty in demonstrating independence since the code is generated by the same tool as the model that is used for test
Need to establish and use additional coding standards (e.g., naming conventions, insertion of test points)
Added process requirement for tool qualification

Generally Model Based Development does not address the real time aspects of embedded real time systems, especially concerning redundancy management, cross channel I/O and synchronization and timing, failure management including reconfiguration and fault handling

G.3.7 What Should the Certification Authorities do Differently?

Survey Comment

Configuration Management reluctance to control databases (not documents)
Regulators inability to review data in a tool's native form (they don't have tools)
Regulators reluctance to approve data not in a document form.

Survey Comment

There is no standard formatting, the process is arbitrary and not uniformly administered by the various FAA ACOs.

Survey Comment

The documents lack in clarity in some places. Plus the FAA individuals seem to have specific interpretations that may not agree between everyone. Experience and mentoring seems to be the best teacher.

Survey Comment

Some of our customers ask for additional steps to be performed beyond DO-178B.
These should be incorporated into DO-178B.

Survey Comment

Think there's too much "guidance". Too many versions of the same idea

G.3.8 Should Civil Standards for REM be Harmonized With Military Standards?

Survey Comment

Process requirements for Commercial Derivative/Military Aviation software and dual use software - since the approach to Requirement Management for Military Programs has a different focus than that of Civil Certified Programs

G.4 Model-Based Development (MBD) Issues.

G.4.1 What is the Relationship of System and Software Requirements in MBD?

Survey Comment

Model-based development (MBD) simply sounds like rapid prototyping via simulation (rather than via rapid HW & SW development). The net result can therefore be overly-detailed SYS requirements -- with no rationale and no documentation of the true "high-level" intent or development philosophy. Indeed, it seems to move "SW rqmt & design" step to the SYS arena. But what SYS creates are not SYS requirements so much as they are low-level SW requirements - except without the process "care" that DO178B intended.

Survey Comment

How is documentation of the requirements to be produced (e.g., can it be automated as a function of the tool)?

Will MBD be accepted by systems engineers who are accustomed to generating primarily textual requirements? Paradigm shift challenge.

If requirements are contained in the model (as opposed to, say, a database), how will linking of low-level requirements to high-level requirements be accomplished? Software to requirements? Test cases to requirements?

How can legacy requirements be imported into the modeling tool? If they cannot be readily imported, won't it be expensive to populate the modeling tool with all the requirements for a large project, such as FMS with over 300,000 LOC?

G.4.2 If Models will be Used as Requirements, How are They to be Validated?

Survey Comment

Requirements defined in models are

not as explicit as textual requirements

more formal language

more error prone because of less requirements granularity and broader functionality in one and the same model

Tools delivered for this model-based development allow easier requirement validation

Accuracy and correctness of requirements in the model are more difficult to prove on model level

Survey Comment

What is the guidance on the use of models as requirements?

To what extent can simulation be used for requirements validation?

Survey Comment

New guidance is needed when Model-Based Development is used to define HLR and/or LLR. Topics that should be addressed are related to simulation (or testing) of requirements and coverage analysis at model level.

Survey Comment

I think we all already know that requirements management is not adequate in DO-178B and the hand-off to ARP4754 is not well defined. I think DO-254 vs. DO-178B is OK as DO-254 is clearly DO-178B based and is complete.
What is unclear about the guidance is the following.
effective linkages from the SSA to the Software implementation (e.g., there is no guidance for how to efficiently test the requirements imposed on SW from the SSA.
The software is considered, from a SSA viewpoint, typically a computing element and if the computing element fails, the system may have redundancy or some other failure accommodation scheme. One item missing is that there are many methods and tools that can be applied in Software to improve the safety of the system by improving, monitoring, etc. the reliability of the computing element. I feel we need guidance specific to software that permits improved safety opportunities for the system. (example: In many discrete HW systems memory with parity checking is standard. Right now we in the industry are using PowerPCs that have on-chip memory elements that do not have parity. Should a software parity scheme be devised?)
If we do use models as a basis for requirements, how do we validate the pedigree of these models?

Survey Comment

The biggest issue for us is model validation. We want to be able to validate the robustness of the model to enable us to use it as a vehicle for certification purposes. We have a physics-based model approach whereby the model is validated and confirmed with actual running data.

Survey Comment

How to validate the model. Evaluating the completeness of the model. Applicability of the modeling approach to solving the problem at hand. Development tool qualification.

Survey Comment

Difficulty in demonstrating independence since the code is generated by the same tool as the model that is used for test
Need to establish and use additional coding standards (e.g., naming conventions, insertion of test points)
Added process requirement for tool qualification
Generally Model Based Development does not address the real time aspects of embedded real time systems, especially concerning redundancy management, cross channel I/O and synchronization and timing, failure management including reconfiguration and fault handling

Survey Comment

The emphasis on verification of requirements has lead to a number of futile dodges that a) don't buy much additional safety and b) cost a lot of money to accomplish. When people start labeling their design diagrams "Requirements" and then go out to verify that the code generated matches the design diagrams, this is not right. Design diagrams are NOT "Requirements" - they are "DESIGN". In the case of auto code generation, they are "IMPLEMENTATION". How is a Simulink (or similar tool) drawing different than expressing an implementation in Ada, Fortran or C? Do you verify that the output of an Ada compiler matches the source code statements in any way other than through normal testing to see if you get the desired result? There is extremely little value in checking that a compiler takes "Y := M * X + B;" and translates it into "Load A,X; Mult A,M; Add A,B; Store A,Y;". None of that tells you if "Y := M * X + B;" was the right thing to do in the first place. What if the operation should have been "-" instead of "+"?

What we need is better guidance on what constitutes a valid, testable requirement and what levels of gyrations are needed to verify it.

Survey Comment

Fidelity of the model to the actual requirements

Survey Comment

Less requirements volatility - lets the customer see the requirements in action much earlier in the project. This is a major benefit.

Maybe issues on who owns the models? The customer or us? Or both?
We are learning as we go through it.

Survey Comment

How are the requirements reviewed? Do they violate safety assumptions? Where are the interfaces different?

Survey Comment

First of all the verification of the output of the tool. This is addressed by tool qualification, but represents a big challenge.

The second part is the representation of the requirements. If they are only available and represented in the modeling tool, the question arises how do you verify the requirements? By taking a look at the requirements in the tool? Is there a need for tool qualification, even if no code is generated?

Survey Comment

General: When a TSO does not exist, there may be cases where the overall system functions are achieved by the combination of the equipment and the A/C. How to manage this shared responsibility is not addressed by the standards.

DO-178B:
Embedded Database Management (DO200A is fine for data preparation systems, but not adequate for embedded applications)
Test Data reduction techniques (classes of equivalence, etc.)
Defensive Programming (it is not encouraged and may cause code coverage issues, while is a very powerful technique to ensure robustness)

DO-254:
It is proposed to encourage an analysis of the safety aspects related to independent functions on the same PLD, and to apply the rules in accordance to each function's safety levels (provided decoupling is ensured)

DO-178B needs to address the issue of model-based development related to validation of the model and ensuring that the model accurately represents the system requirements.

G.4.3 What is the Relationship of Software Requirements and Software Design in MBD?

Survey Comment

The emphasis on verification of requirements has lead to a number of futile dodges that a) don't buy much additional safety and b) cost a lot of money to accomplish. When people start labeling their design diagrams "Requirements" and then go out to verify that the code generated matches the design diagrams, this is not right. Design diagrams are NOT "Requirements" - they are "DESIGN". In the case of auto code generation, they are "IMPLEMENTATION". How is a Simulink (or similar tool) drawing different than expressing an implementation in Ada, Fortran or C? Do you verify that the output of an Ada compiler matches the source code statements in any way other than through normal testing to see if you get the desired result? There is extremely little value in checking that a compiler takes "Y := M * X + B;" and translates it into "Load A,X; Mult A,M; Add A,B; Store A,Y;". None of that tells you if "Y := M * X + B;" was the right thing to do in the first place. What if the operation should have been "-" instead of "+"?

What we need is better guidance on what constitutes a valid, testable requirement and what levels of gyrations are needed to verify it.

Survey Comment

We are still trying to figure out how to do model based development

Survey Comment

What is the guidance on the use of models as requirements?
To what extent can simulation be used for requirements validation?

Survey Comment

New guidance is needed when Model-Based Development is used to define HLR and/or LLR. Topics that should be addressed are related to simulation (or testing) of requirements and coverage analysis at model level.

Survey Comment

Where are the "shalls" in a model. How are the requirements traced to higher and lower-level requirements and / or design?

Survey Comment

The use of MDB methodologies or tools should not be dictated. They are project-dependent. The use of model-based tools & methodologies must be assessed on a case-by-case basis by the developer. What works for one development activity may not work for another. Also, there can be an overemphasis on tooling in lieu of lack of overall system experience; tools do not write good requirements, engineers do. A tool is no better than the quality of data put into it. Therefore, requirements regarding MDB tools or methodologies are impractical to specify.

Also, it is difficult to use DO-178B processes with Model Based Development. DO-178B does not have precise and widely agreed upon definitions for System Requirements, HL Software Requirements, LL Software Requirements and HW Requirements. Nor should it. This is project dependent.

Model based development introduces a new level of abstraction, and translation. MBD should not be extrapolated to everything as this results in an abuse of the methodology.

Comment: In general, models can provide some clarity to text-based requirements. They provide a mechanism to help understand the text-based requirements but they are not a substitute.

MBD tools have a great deal of capability and flexibility in choosing what you want models to represent. Therefore, based upon what is represented in the models, as well as what is represented in artifacts outside of the models, results in all possibilities being potentially suitable uses for models.

Survey Comment

If Models will replace high-level and low-level software requirements (i.e., we write the requirements using model-language instead of English-language). We must ensure the systems and subsystems team members are fluent in model-language so that their peer reviews are meaningful. Also, requirements decomposition is a form of complexity management, i.e., the system is broken into smaller and smaller pieces, each being manageable steps, until the end result is achievable. Models may represent a too large of a step that the mind can comfortably take at once. Also, Models, being computer programs, are often less forgiving than English text which can accommodate a large amount of ambiguity be resolved in order to define a valid system. As a example, in English we might say “sign here” whereas a model language may require us to say “using a ball-point pen with a .7mm tip containing black ink in a cursive format with a font size of no less that 10 point and no more than 20 point, etc. … We may lose the actual requirement in the details.

Survey Comment

In concept, it would be easy to fit the simulation models into the Low Level Requirements category, and the resulting generated C code into the Source Code category. This raises the following issues:

Is the generated code traceable to the models?
Are the models traceable to the generated code?
Does the generated code need to be verified against the models?
Does the code generator need to be qualified?
How much certification credit (if any) can be taken for verification via simulations?

Another strategy might be to claim that the model is actually both the Low Level Requirements and the Source Code. This eliminates an entire set of DO-178B processes relating to Source Code. However, this brings up the issue of Tool Qualification. The code generator would have to be qualified as a development tool under DO-178B. However, the development tool qualification could require as much or more effort than required to take the traditional approach.

Survey Comment

The requirement is virtually the design, so any error automatically propagates into design.

Survey Comment

In your opinion, what level(s) of requirements do the models represent in Model-Based Development? *(Check all that apply)*

20__ System Requirements
16__ Data Interconnect (ICD)
33__ High-Level Software Requirements
25__ Low-Level Software Requirements
9__ Hardware Requirements
7__ The Model Does Not Represent Requirements

Survey Comment

Requirements defined in models are
not as explicit as textual requirements
more formal language
more error prone because of less requirements granularity and broader functionality in one and the same model
Tools delivered for this model-based development allow easier requirement validation
Accuracy and correctness of requirements in the model are more difficult to prove on model level

Survey Comment

Model based development (e.g., UML models, in contrast to function based models like Matlab) It's not covered by the DO178-B. The role of the model respectively the role of its elements in the requirements process is not clear resp. not addressed by the DO178-B: Is it optional, just for understanding the system? Many people would deny using it, because it - superficially - adds cost, although it adds value, because people have the systems under control, which they are going to build. This in turn speeds up development and finally saves cost.

G.4.4 Which MBD Modeling Paradigms Should be Used in Which Problem Domains?

Survey Comment

How to validate the model. Evaluating the completeness of the model. Applicability of the modeling approach to solving the problem at hand. Development tool qualification.

Survey Comment

VAPS models:
- Definition of symbology:
  - What is considered as requirement?
  - Can this replace a requirements document?
- Requirements traceability is not straightforward.
- Verification:
  - What needs to be tested of that symbology? What can be done by simulation?

Requirements coverage: When is it complete?

Survey Comment

Many tools follow a simple function oriented approach (SCADE, Matlab). Software Engineering Methodology already in the seventies found out, that a simple function oriented way of analyzing and designing is not appropriate for managing complexity (c.p. D.L. Parnas). That's why a modular approach was chosen in the eighties to overcome the software crisis.

Survey Comment

Model based development (e.g., UML models, in contrast to function based models like Matlab)

It's not covered by the DO178-B. The role of the model respectively the role of its elements in the requirements process is not clear resp. not addressed by the DO178-B: Is it optional, just for understanding the system? Many people would deny using it, because it - superficially - adds cost, although it adds value, because people have the systems under control, which they are going to build. This in turn speeds up development and finally saves cost.

G.4.5 What New Problems for Traceability and CM of Requirements are Posed by MBD?

Survey Comment

These are "development" tools. Their results need to either be manually and tediously inspected (generated code can be sloppy, hard to follow), or the tools need to be strenuously qualified. This qualification can be very difficult, requires cooperation of tool vendor to go smoothly. Tool source code hard to come by. Need to understand how to apply service history, or some other type of alternative means of qualification for these complex development tools in order to realize their full potential.
Acceptable means of CM on the data maintained in these tools.

Survey Comment

Where are the "shalls" in a model. How are the requirements traced to higher and lower-level requirements and / or design?

Survey Comment

What issues are there in convincing shared services, like Software Control Library, DERs and SQE specialists, to accept outputs of the MBD or OOD tools and be able to assess their quality and archive them? We haven't even been able to convince these supporting services to accept release of a DOORS database instead of a requirements document, which seems like a pretty mild shift, compared to some of the newer technologies. In other words, Engineering can't just decide on its own to change this paradigm. Everyone involved in the design, development, release, and certification process must be on-board

Survey Comment

How to make changes, compare versions and document results

Survey Comment

Configuration control of models. Models, toolboxes, utilities are created as needed, and have no integrated revision control for individual blocks or functions. All items must be controlled manually by the user archiving the baselines.

Survey Comment

One difficult issue for reqs management is the use of different tools and integration of those tools for effective traceability and configuration control

Survey Comment

Requirements defined in models are
not as explicit as textual requirements
more formal language
more error prone because of less requirements granularity and broader functionality in one and the same model

Tools delivered for this model-based development allow easier requirement validation
Accuracy and correctness of requirements in the model are more difficult to prove on model level

Survey Comment

Configuration Management reluctance to control databases (not documents)
Regulators inability to review data in a tool's native form(they don't have tools)
Regulators reluctance to approve data not in a document form.

Survey Comment

How is documentation of the requirements to be produced (e.g., can it be automated as a function of the tool)?
Will MBD be accepted by systems engineers who are accustomed to generating primarily textual requirements? Paradigm shift challenge.
If requirements are contained in the model (as opposed to, say, a database), how will linking of low-level requirements to high-level requirements be accomplished? Software to requirements? Test cases to requirements?
How can legacy requirements be imported into the modeling tool? If they cannot be readily imported, won't it be expensive to populate the modeling tool with all the requirements for a large project, such as FMS with over 300,000 LOC?

Survey Comment

It is difficult to identify or tag what an actual requirement is within the model.
It is difficult to generate traceability to portions of the model.

G.4.6 What New Problems are Posed for REM by the Use of MBD Tools?

Survey Comment

Difficulty in demonstrating independence since the code is generated by the same tool as the model that is used for test
Need to establish and use additional coding standards (e.g., naming conventions, insertion of test points)
Added process requirement for tool qualification
Generally Model Based Development does not address the real time aspects of embedded real time systems, especially concerning redundancy management, cross channel I/O and synchronization and timing, failure management including reconfiguration and fault handling

Survey Comment

The use of MDB methodologies or tools should not be dictated. They are project-dependent. The use of model-based tools & methodologies must be assessed on a case-by-case basis by the developer. What works for one development activity may not work for another. Also, there can be an overemphasis on tooling in lieu of lack of overall system experience; tools do not write good requirements, engineers do. A tool is no better than the quality of data put into it. Therefore, requirements regarding MDB tools or methodologies are impractical to specify.
Also, it is difficult to use DO-178B processes with Model Based Development. DO-178B does not have precise and widely agreed upon definitions for System Requirements, HL Software Requirements, LL Software Requirements and HW Requirements. Nor should it. This is project dependent.
Model based development introduces a new level of abstraction, and translation. MBD should not be extrapolated to everything as this results in an abuse of the methodology.
Comment: In general, models can provide some clarity to text-based requirements. They provide a mechanism to help understand the text-based requirements but they are not a substitute.
MBD tools have a great deal of capability and flexibility in choosing what you want models to represent. Therefore, based upon what is represented in the models, as well as what is represented in artifacts outside of the models, results in all possibilities being potentially suitable uses for models.

Survey Comment

One difficult issue for reqs management is the use of different tools and integration of those tools for effective traceability and configuration control

Survey Comment

Use of such tools can lead to quicker development cycles, reduced cost, less integration time, more completely defined systems up front… advantages are numerous. Quality of tools and qualification of the tools by some other means than the standard compliance to DO-178B used for airborne software needs to be addressed in order to realize the capabilities of such tools. Service history, and/or incentive to vendors to cooperate with users seems the most logical means.

Survey Comment

Integration of Tools. Training. Process Improvement

Survey Comment

Overuse of tools. Often, the effort to purchase, learn, and use a tool outweighs its benefits.
Tool qualification issues.

Survey Comment

Tool verification effort can get to cost more than the value added by the tool.

Survey Comment

How to validate the model. Evaluating the completeness of the model. Applicability of the modeling approach to solving the problem at hand. Development tool qualification.

Survey Comment

These are “development” tools. Their results need to either be manually and tediously inspected (generated code can be sloppy, hard to follow), or the tools need to be strenuously qualified.

This qualification can be very difficult, requires cooperation of tool vendor to go smoothly. Tool source code hard to come by. Need to understand how to apply service history, or some other type of alternative means of qualification for these complex development tools in order to realize their full potential.
Acceptable means of CM on the data maintained in these tools.

Survey Comment

In concept, it would be easy to fit the simulation models into the Low Level Requirements category, and the resulting generated C code into the Source Code category. This raises the following issues:
Is the generated code traceable to the models?
Are the models traceable to the generated code?
Does the generated code need to be verified against the models?
Does the code generator need to be qualified?
How much certification credit (if any) can be taken for verification via simulations?

Another strategy might be to claim that the model is actually both the Low Level Requirements and the Source Code. This eliminates an entire set of DO-178B processes relating to Source Code. However, this brings up the issue of Tool Qualification. The code generator would have to be qualified as a development tool under DO-178B. However, the development tool qualification could require as much or more effort than required to take the traditional approach.

Survey Comment

Test verification or qualification.

Survey Comment

Verification and validation of the code generating tools.

Survey Comment

First of all the verification of the output of the tool. This is addressed by tool qualification, but represents a big challenge.

The second part is the representation of the requirements. If they are only available and represented in the modeling tool, the question arises how do you verify the requirements? By taking a look at the requirements in the tool? Is there a need for tool qualification, even if no code is generated?

Survey Comment

How is documentation of the requirements to be produced (e.g., can it be automated as a function of the tool)?
Will MBD be accepted by systems engineers who are accustomed to generating primarily textual requirements? Paradigm shift challenge.
If requirements are contained in the model (as opposed to, say, a database), how will linking of low-level requirements to high-level requirements be accomplished? Software to requirements? Test cases to requirements?
How can legacy requirements be imported into the modeling tool? If they cannot be readily imported, won't it be expensive to populate the modeling tool with all the requirements for a large project, such as FMS with over 300,000 LOC?

Survey Comment

SCADE Suite™

Survey Comment

Simulink and proprietary tools for Logic requirements prototyping are used extensively with major benefits

Survey Comment

Requirements defined in models are
not as explicit as textual requirements
more formal language
more error prone because of less requirements granularity and broader functionality in one and the same model

Tools delivered for this model-based development allow easier requirement validation
Accuracy and correctness of requirements in the model are more difficult to prove on model level

G.4.7 How Should the Intent of Requirements be Specified in MBD?

Survey Comment

Model-based development (MBD) simply sounds like rapid prototyping via simulation (rather than via rapid HW & SW development). The net result can therefore be overly-detailed SYS requirements -- with no rationale and no documentation of the true "high-level" intent or development philosophy. Indeed, it seems to move "SW rqmt & design" step to the SYS arena. But what SYS creates are not SYS requirements so much as they are low-level SW requirements - except without the process "care" that DO178B intended.

Survey Comment

If Models will replace high-level and low-level software requirements (i.e., we write the requirements using model-language instead of English-language). We must ensure the systems and subsystems team members are fluent in model-language so that their peer reviews are meaningful. Also, requirements decomposition is a form of complexity management, i.e., the system is broken into smaller and smaller pieces, each being manageable steps, until the end result is achievable. Models may represent a too large of a step that the mind can comfortably take at once. Also, Models, being computer programs, are often less forgiving than English text which can accommodate a large amount of ambiguity be resolved in order to define a valid system. As a example, in English we might say "sign here" whereas a model language may require us to say "using a ball-point pen with a .7mm tip containing black ink in a cursive format with a font size of no less that 10 point and no more than 20 point, etc. … We may lose the actual requirement in the details.

Survey Comment

The emphasis on verification of requirements has lead to a number of futile dodges that a) don't buy much additional safety and b) cost a lot of money to accomplish. When people start labeling their design diagrams "Requirements" and then go out to verify that the code generated matches the design diagrams, this is not right. Design diagrams are NOT "Requirements" - they are "DESIGN". In the case of auto code generation, they are "IMPLEMENTATION". How is a

Simulink (or similar tool) drawing different than expressing an implementation in Ada, Fortran or C? Do you verify that the output of an Ada compiler matches the source code statements in any way other than through normal testing to see if you get the desired result? There is extremely little value in checking that a compiler takes "Y := M * X + B;" and translates it into "Load A,X; Mult A,M; Add A,B; Store A,Y;". None of that tells you if "Y := M * X + B;" was the right thing to do in the first place. What if the operation should have been "-" instead of "+"?

What we need is better guidance on what constitutes a valid, testable requirement and what levels of gyrations are needed to verify it.

G.4.8 How Should Environmental Assumptions be Specified in MBD?

No Comments

G.4.9 How Should Timing Requirements be Specified in MBD?

Survey Comment
Difficulty in demonstrating independence since the code is generated by the same tool as the model that is used for test Need to establish and use additional coding standards (e.g., naming conventions, insertion of test points) Added process requirement for tool qualification Generally Model Based Development does not address the real time aspects of embedded real time systems, especially concerning redundancy management, cross channel I/O and synchronization and timing, failure management including reconfiguration and fault handling

G.4.10 How Should Human Factors Requirements be Specified in MBD?

No Comments

G.4.11 How Should Legacy Requirements be Migrated to Support MBD?

Survey Comment
How is documentation of the requirements to be produced (e.g., can it be automated as a function of the tool)? Will MBD be accepted by systems engineers who are accustomed to generating primarily textual requirements? Paradigm shift challenge. If requirements are contained in the model (as opposed to, say, a database), how will linking of low-level requirements to high-level requirements be accomplished? Software to requirements? Test cases to requirements? How can legacy requirements be imported into the modeling tool? If they cannot be readily imported, won't it be expensive to populate the modeling tool with all the requirements for a large project, such as FMS with over 300,000 LOC?

G.4.12 What Should the Certification Authorities Do Differently to Support MBD?

Survey Comment

Configuration Management reluctance to control databases (not documents)
Regulators inability to review data in a tool's native form (they don't have tools)
Regulators reluctance to approve data not in a document form.

G.4.13 Can Models be Used as Contracts?

Survey Comment

Less requirements volatility - lets the customer see the requirements in action much earlier in the project. This is a major benefit.

Maybe issues on who owns the models? The customer or us? Or both?
We are learning as we go through it.

G.5 Integrated Modular Avionics (IMA) Issues.

G.5.1 How Should Application Requirements be Specified?

Survey Comment

Requirements need to be defined around two sources: IMA Platform system requirements & Application-level requirements.
DO-178B does not seem to address the many levels of integration that can and do occur on typical IMA software projects. Also not addressed that may be of particular concern are:
Change Impact Analysis at the system level
Data and control coupling at the integration level
Independence of development/verification for commonly used items across IMA
Common failure modes and cascading effects
Interface requirements between the integrated functions
How to prove that partitioning works, in particular what must be done to prove that time partitioning is working and that budgets are set correctly
IMA means that the engineer has to understand the big picture (integrated), and most don't.

Survey Comment

Establishing Integration requirements for the end user. Defining system performance requirements. Meeting hard real-time deadlines. Partitioning.

Survey Comment

More difficult to check, verify and guarantee traceability following aspects:
HW compatibility (on target processor)
HW/SW interfacing
SW/SW interfacing
Integration issues

G.5.2 How Should Platform Services be Specified?

Survey Comment

More difficult to check, verify and guarantee traceability following aspects:
HW compatibility (on target processor)
HW/SW interfacing
SW/SW interfacing
Integration issues

Survey Comment

Incremental certification.
lack of control over and visibility into the lower level implementation of the real-time environment - may have unexpected implications on the real time performance, especially under failure conditions
no control over changes/enhancements - would need to demonstrate no effect of the change to the application, also need to determine scope for regression test when there is a change that does affect the application

Survey Comment

Establishing Integration requirements for the end user. Defining system performance requirements. Meeting hard real-time deadlines. Partitioning.

Survey Comment

Requirements need to be defined around two sources: IMA Platform system requirements & Application-level requirements.
DO-178B does not seem to address the many levels of integration that can and do occur on typical IMA software projects. Also not addressed that may be of particular concern are:
Change Impact Analysis at the system level
Data and control coupling at the integration level
Independence of development/verification for commonly used items across IMA
Common failure modes and cascading effects
Interface requirements between the integrated functions
How to prove that partitioning works, in particular what must be done to prove that time partitioning is working and that budgets are set correctly
IMA means that the engineer has to understand the big picture (integrated), and most don't.

G.5.3 How Should Requirements be Specified to Support Incremental Acceptance in Certification?

Survey Comment

Incremental certification.
lack of control over and visibility into the lower level implementation of the real-time environment - may have unexpected implications on the real time performance, especially under failure conditions
no control over changes/enhancements - would need to demonstrate no effect of the change to the application, also need to determine scope for regression test when there is a change that does affect the application

Survey Comment

Requirements need to be defined around two sources: IMA Platform system requirements & Application-level requirements.
DO-178B does not seem to address the many levels of integration that can and do occur on typical IMA software projects. Also not addressed that may be of particular concern are:
Change Impact Analysis at the system level
Data and control coupling at the integration level
Independence of development/verification for commonly used items across IMA
Common failure modes and cascading effects
Interface requirements between the integrated functions
How to prove that partitioning works, in particular what must be done to prove that time partitioning is working and that budgets are set correctly
IMA means that the engineer has to understand the big picture (integrated), and most don't.

Survey Comment

For us specifically, we do not have an IMA system. With that said, however our plans in the future would be to have partial engine control SW on aircraft and partial on engine. With the engine type certificate being different than the aircraft type cert, we are uncertain about the roles of the FAA directorates.
One reason for our decision to look towards IMA systems is to help the obsolescence problem whereby we can design it somewhat modular for those items that do have a short shelf life. We would like to see obsolescence handle by modular upgrade and as such modular cert is of large concern to us.

Since the verification is the bigger part of the development, the major challenge will be to have a composable system that allows independent and reusable verification. The constraints for such a composable systems are stringent and not very well understood.

G.5.4 How Should Requirements be Specified to Support Reuse in IMA?

Survey Comment

Requirements need to be defined around two sources: IMA Platform system requirements & Application-level requirements.
DO-178B does not seem to address the many levels of integration that can and do occur on typical IMA software projects. Also not addressed that may be of particular concern are:
Change Impact Analysis at the system level
Data and control coupling at the integration level
Independence of development/verification for commonly used items across IMA
Common failure modes and cascading effects
Interface requirements between the integrated functions
How to prove that partitioning works, in particular what must be done to prove that time partitioning is working and that budgets are set correctly
IMA means that the engineer has to understand the big picture (integrated), and most don't.

Survey Comment

Sufficient work up front to devise workable reuse strategies.

G.5.5 Increased Risk of Common Mode and Cascading Failures.

Survey Comment

FAR part 25 may have adequately addressed failures assuming an essentially federated system in which single points of failure have a limited system effect. Highly integrated systems may exhibit faults in systems which, in the pilot's mind, are completely unrelated. Aircraft certification requirements may have to be altered to address cascade failure effects.

Survey Comment

Too many eggs in one basket. Potential to scramble the eggs. Potential to break all the eggs at once.

Survey Comment

Physical proximity. Example is one capacitor exploded, took out both channels of an engine control (which was not IMA), and shut engine down. So faults in one system have increased 'sneak paths' to impact another - track burn, power supply pull down or noise, dripping solder, explosion debris, excessive local EMI effects under fault conditions, connector wear and tear, common mode issues if connectors not mated properly.

Survey Comment

Requirements need to be defined around two sources: IMA Platform system requirements & Application-level requirements.
DO-178B does not seem to address the many levels of integration that can and do occur on typical IMA software projects. Also not addressed that may be of particular concern are:
Change Impact Analysis at the system level
Data and control coupling at the integration level
Independence of development/verification for commonly used items across IMA
Common failure modes and cascading effects
Interface requirements between the integrated functions
How to prove that partitioning works, in particular what must be done to prove that time partitioning is working and that budgets are set correctly
IMA means that the engineer has to understand the big picture (integrated), and most don't.

Survey Comment

IMA is fully applied in one (ongoing) project.
Current opinion is that a federated System incorporating concepts of
High functional cohesion
Open SW architecture and Abstraction Layers
Open interface connectivity
Would provide the maximum benefits.

G.5.6 Specifying and Meeting System Performance Requirements.

Survey Comment

Incremental certification.
lack of control over and visibility into the lower level implementation of the real-time environment - may have unexpected implications on the real time performance, especially under failure conditions

no control over changes/enhancements - would need to demonstrate no effect of the change to the application, also need to determine scope for regression test when there is a change that does affect the application

Survey Comment

Establishing Integration requirements for the end user. Defining system performance requirements. Meeting hard real-time deadlines. Partitioning.

Survey Comment

Requirements need to be defined around two sources: IMA Platform system requirements & Application-level requirements.
DO-178B does not seem to address the many levels of integration that can and do occur on typical IMA software projects. Also not addressed that may be of particular concern are:
Change Impact Analysis at the system level
Data and control coupling at the integration level
Independence of development/verification for commonly used items across IMA
Common failure modes and cascading effects
Interface requirements between the integrated functions
How to prove that partitioning works, in particular what must be done to prove that time partitioning is working and that budgets are set correctly
IMA means that the engineer has to understand the big picture (integrated), and most don't.

Survey Comment

Allocation of requirements to partitions is necessary to allow proper overall design to occur.

G.5.7 Specifying and Meeting Dependencies Between Applications.

Survey Comment

More difficult to check, verify and guarantee traceability following aspects:
HW compatibility (on target processor)
HW/SW interfacing
SW/SW interfacing
Integration issues

Survey Comment

Requirements need to be defined around two sources: IMA Platform system requirements & Application-level requirements.
DO-178B does not seem to address the many levels of integration that can and do occur on typical IMA software projects. Also not addressed that may be of particular concern are:
Change Impact Analysis at the system level
Data and control coupling at the integration level
Independence of development/verification for commonly used items across IMA
Common failure modes and cascading effects
Interface requirements between the integrated functions
How to prove that partitioning works, in particular what must be done to prove that time partitioning is working and that budgets are set correctly
IMA means that the engineer has to understand the big picture (integrated), and most don't.

Survey Comment

Establishing Integration requirements for the end user. Defining system performance requirements. Meeting hard real-time deadlines. Partitioning.

G.5.8 Traceability and Configuration Management.

Survey Comment

More difficult to check, verify and guarantee traceability following aspects:
HW compatibility (on target processor)
HW/SW interfacing
SW/SW interfacing
Integration issues

Survey Comment

Can result in massive CM problems where there are multiple configurations spawned from a core design. Test CM also more of a challenge since distributed functionality means more of the IMA HW and SW elements must be "configured" for any for-score testing.

Survey Comment

Allocation of requirements to partitions is necessary to allow proper overall design to occur.

G.5.9 What is the Role of the Certification Authorities?

Survey Comment

For us specifically, we do not have an IMA system. With that said, however our plans in the future would be to have partial engine control SW on aircraft and partial on engine. With the engine type certificate being different than the aircraft type cert, we are uncertain about the roles of the FAA directorates.
One reason for our decision to look towards IMA systems is to help the obsolescence problem whereby we can design it somewhat modular for those items that do have a short shelf life. We would like to see obsolescence handle by modular upgrade and as such modular cert is of large concern to us.

Since the verification is the bigger part of the development, the major challenge will be to have a composable system that allows independent and reusable verification. The constraints for such a composable systems are stringent and not very well understood.

G.5.10 Does the REM Process Change in IMA?

Survey Comment

Requirements need to be defined around two sources: IMA Platform system requirements & Application-level requirements.
DO-178B does not seem to address the many levels of integration that can and do occur on typical IMA software projects. Also not addressed that may be of particular concern are:
Change Impact Analysis at the system level
Data and control coupling at the integration level
Independence of development/verification for commonly used items across IMA
Common failure modes and cascading effects

Interface requirements between the integrated functions
How to prove that partitioning works, in particular what must be done to prove that time partitioning is working and that budgets are set correctly
IMA means that the engineer has to understand the big picture (integrated), and most don't.

G.5.11 Object-Oriented Development (OOD) Issues.

G.5.12 What is the Relationship of Safety and Software Requirements in OOD?

Survey Comment

OOD does not lend itself to functional decomposition - hard to trace requirements (many to many), hard to demonstrate compliance, requires additional analysis, requires extra re-qual after changes
Prone to 'generic code' which leads to worst case 'dead code', best case 'deactivated code' - extra work to demonstrate compliance
Safety aspects and implications are not intuitive

Survey Comment

No ability to show determinism as required in the high levels of DO178B.
No clear industry discussion of safe constructs or a safe subset of the method.

G.5.13 What is the Relationship of System and Software Requirements in OOD?

Survey Comment

How much time/effort (and therefore, how much money) will need to be invested to retrain systems engineers who are accustomed to generating primarily textual requirements, so that they know how to capture OO requirements using a new form of notation?
Paradigm shift challenge - lack of acceptance by systems engineers who think procedurally and tend to anticipate a structured design in the way they organize their requirements.
How can legacy requirements be efficiently transformed to support the OO model?

G.5.14 What is the Relationship of Software Requirements and Software Design in OOD?

Survey Comment

Language, terminolgy. But current RM techniques need not change to accommodate new design techniques. OOD provides increased possiblity for CASE tool integration.

Survey Comment

Understanding and applying the new FAA guidance on OOD.

Survey Comment

Considerations of how effectively OO design can be implemented to achieve DO-178B certification.
OO design has worked well for us in breaking the software problem set down to its smallest components, but it doesn't necessarily convey all the integration issues between hardware and software.

Survey Comment

It's not coverd by the DO178-B.

It requires a restriction on a subset of OO techniques (i.e., avoid inheritence and dynamic creation of objects).

Survey Comment

Low level requirements definition. What is considered as low level requirements in OOD? Use Cases and State diagrams?
Traceability low level requirements <-> high-level requirements is not straightforward in OOD.
low level requirements testing : What is the meaning of full requirements coverage and structural coverage in OOD

Survey Comment

Please refer to the OOTiA handbook for details

Survey Comment

See previous answers and reference to OOTiA

Survey Comment

Some experiments have been conducted on Object-oriented Requirement Engineering in order to obtain complete reusable Objects (from Requirements through to System Test).
This topic should be further explored.

G.5.15 Which OOD Modeling Paradigms Should be Used in Which Problem Domains?

Survey Comment

Low level requirements definition. What is considered as low level requirements in OOD? Use Cases and State diagrams?
Traceability low level requirements <-> high-level requirements is not straightforward in OOD.
low level requirements testing : What is the meaning of full requirements coverage and structural coverage in OOD

Survey Comment

Model based development (e.g., UML models, in contrast to function based models like Matlab)

It's not coverd by the DO178-B. The role of the model respectively the role of its elements in the requirements process is not clear resp. not addressed by the DO178-B: Is it optional, just for understanding the system? Many people would deny using it, because it - superficially - adds cost, although it adds value, because people have the systems under control, which they are going to build. This in turn speeds up development and finally saves cost.

G.5.16 What New Problems for Traceability and CM of Requirements are Posed by OOD?

Survey Comment

Yes, OOD is the key to cohesion and layering.

Survey Comment

OOD is not necessarily appropriate for all programs. Experience has shown that projects that use OOD simply for the sake of using OOD fall behind schedule and have a low rate of success. Making SW satisfy some object oriented philosophies can result in very complicated, bad SW.

Programs that took a rational approach and applied OOD as appropriate were very successful. It is not OOD, it is the deployment of it.
OOD can be a valuable tool when used appropriately and deployed correctly with appropriate standards, but is not appropriate for all cases.

Survey Comment

No ability to show determinism as required in the high levels of DO178B.
No clear industry discussion of safe constructs or a safe subset of the method.

Survey Comment

How to test all aspects of OO implementation. Traceability of object code to source code. What does structural coverage mean?

Survey Comment

There are some technical barriers to things like Object Oriented Design in that they can induce processing overhead and in some circumstances induce runtime indeterminism. We cannot, for example, employ dynamic dispatching (key to OOD) without detailed analysis or it could lead to all kinds of uncertain call chains and problems with verification. The whole concept of a class and its descendent child classes makes for difficulties in verification of class-wide operations. (If I test a subroutine that deals with a class-wide parameter, do I know it will work with *any* children of that class without testing it with all possible children?)
Technical barriers exist to using OOD, but they are not insurmountable. The techniques would have to be evaluated to determine if there were any potential runtime issues and some parts of OOD might need to be avoided due to indeterminism. However, many of the techniques could be valuable if studied and employed properly. The resistance is largely institutional and psychological.

Survey Comment

This could apply. Main issues are in testing and coverage analysis.

Survey Comment

One of the benefits of this paradigm is the ability to create more easily reused components. Anytime you reuse a component, it’s likely a general purpose component contains functionality unneeded for this project. Likewise, when extending an object to meet new requirements, the base object may contain unneeded functionality. Either way, you have to deal with requirements traceability or dead code issues.

Survey Comment

Generation of unnecessary requirements/functionality. Need to verify or disable these extra features

Survey Comment

Applicability of the reuse.

Survey Comment

Heap management. Hidden functionality. Nondeterminism

Survey Comment

Traceability back to requirements is a big issue with object oriented design. This is due to the fact that, if done right, there is significant abstraction which makes it hard to look at portions of the code and tie them back to any requirements (derived requirements can help somewhat with this issue).

Survey Comment

OOD does not lend itself to functional decomposition - hard to trace requirements (many to many), hard to demonstrate compliance, requires additional analysis, requires extra re-qual after changes
Prone to 'generic code' which leads to worst case 'dead code', best case 'deactivated code' - extra work to demonstrate compliance
Safety aspects and implications are not intuitive

Survey Comment

OOD by itself is not a challenge. The usage of C++ or generally speaking of more sophisticated compilers/linkers and run-time environments is a challenge. How is such a complex translation and run-time environment fully verified?

Survey Comment

Low level requirements definition. What is considered as low level requirements in OOD? Use Cases and State diagrams?
Traceability low level requirements <-> high-level requirements is not straightforward in OOD.
low level requirements testing : What is the meaning of full requirements coverage and structural coverage in OOD

Survey Comment

It's not coverd by the DO178-B.
It requires a restriction on a subset of OO techniques (i.e., avoid inheritence and dynamic creation of objects).

G.5.17 How Should Intent be Specified in OOD?

No Comments

G.5.18 How Should Environmental Assumptions be Specified in OOD?

No Comments

G.5.19 How Should Timing Requirements be Specified in OOD?

Survey Comment

Some. We watch the real time aspects.

G.5.20 How Should Human Factors Requirements be Specified in OOD?

No Comments

G.5.21 How Should Legacy Requirements be Migrated to Support OOD?

Survey Comment

How much time/effort (and therefore, how much money) will need to be invested to retrain systems engineers who are accustomed to generating primarily textual requirements, so that they know how to capture OO requirements using a new form of notation?

Paradigm shift challenge - lack of acceptance by systems engineers who think procedurally and tend to anticipate a structured design in the way they organize their requirements.
How can legacy requirements be efficiently transformed to support the OO model?