Safety Analysis of Requirements for a Product Family

Robyn R. Lutz *
Iowa State University and Jet Propulsion Laboratory
rlutz@cs.iastate.edu
Guy G. Helmer & Michelle M. Moseman
Iowa State University
ghelmer, mmoseman@cs.iastate.edu
David E. Statezni & Stephen R. Tockey
Rockwell Avionics and Communication
srtockey, destatez@collins.rockwell.com

Abstract

A safety analysis was performed on the software requirements for a family of flight instrumentation displays of commercial aircraft. First, an existing Safety Checklist was extended to apply to four-variable models and used to analyze the requirements models for representative members of the product family. The results were evaluated against an initial specification of the product family's required commonalities and variabilities. The Safety Checklist was found to be effective at analyzing the completeness of the product family requirements and at identifying additional variabilities and commonalities. Secondly, a forward and backward search for hazards was performed on representative members of the product family. Additional safety requirements for enhanced fault tolerance were derived from these searches. The safety analysis techniques used here appear to have applicability for enhancing the completeness and robustness of a product family's safety-related software requirements.

1 Introduction

With increased reuse of software components comes a growing awareness of the difficulty of specifying the requirements for a family of software products in such a way that the possible variations among the family members are adequately represented [22]. From a safety perspective, ensuring completeness in the software requirements is an essential part of the reuse process. The safety analysis helps verify that all implicit required commonalities and all permitted variations among the family members are accurately and completely specified.

In the work described here, a safety analysis was performed on the software requirements for a family of flight instrumentation displays of commercial aircraft. An overview of the safety analysis process is shown in Fig. 1.

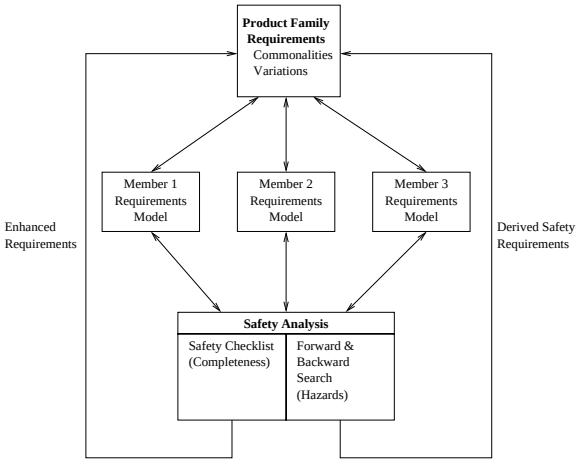

Figure 1: Overview of Safety Analysis Process

First, a set of completeness criteria for analyzing requirements, derived from [8, 10, 11], was tailored for a four-variable model [17] and models were created of parts of three representive members of the product family. The relevant completeness criteria, formatted as a Safety Checklist (a set of questions to guide analysis), were then used to analyze the requirements models for these three representative members of the product family. The results of this safety analysis were evaluated against the initial specification of the product family's required commonalities and variabilities

 This research was supported in part by Rockwell Collins, Inc. Research Grant 155-735.

provided by the project for whom the safety analysis was performed. The Safety Checklist was found to be effective at analyzing the completeness of the product family requirements and at identifying additional variabilities and commonalities.

Secondly, a forward and backward search for hazards was performed on representative members of the product family. These searches suggested additional safety requirements for robustness that can be explicitly included in the product family's software requirements specification.

The major contribution of this work is to describe a way in which existing and well-documented safety analysis techniques (safety checklist, forward and backward search) can be combined to evaluate the completeness of a product family's requirements specification and to identify additional derived safety requirements to enhance the product family's robustness.

1.1 Background

The safety analysis work described here supported a project working to develop reusable software and hardware components. For the software portion of this project, the Software Productivity Consortium's (SPC) description of the process of producing reusable software [21] was used as a baseline. Our work formed part of their "Domain Verification Activity," which SPC defines as, "Verify the correctness, consistency, and completeness of DE (Domain Engineering) work products" (Fig. 2).

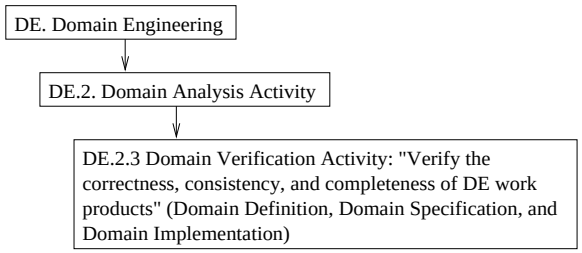

Figure 2: Excerpt from Reuse-Driven Software Process [21]

The product family that we worked with was a flight instrumentation display. Up to twenty-four items belonging to the software components analyzed in this study can be displayed to the pilot on a cockpit digital instrumentation display as graphical or textual icons. Examples of the features displayed include aircraft orientation, autopilot status, and failure indicators.

The SPC reuse-driven software process distinguishes a "Commonality", i.e., an assumption about a characteristic that is common to all the members of a product family, from a "Variability", i.e., an assumption that distinguishes among the members of a product family. An example of a commonality in this domain is the requirement that a graphical icon be displayed in every flight instrumentation display showing the status of the autopilot. An example of a variability is that the requirements differ among the flight instrumentation displays as to what shape that icon takes. In addition, as part of the domain specification activities, the project produced a decision model. In the SPC reuse-driven process, a decision model is the set of requirements and engineering decisions needed to produce a member of the product family.

The decision model, as well as the domain assumptions, the domain glossary, and the requirements specifications as they became available, served as input to the safety analysis process. The requirements specification documents of three legacy members of the product family that was being developed also were an input to the safety analysis. At the beginning of our work, the project anticipated specifying the product family requirements in either CoRE [3] or in SCR* [6], which drove our interest in tailoring our safety analysis techniques to the four-variable model, upon which both CoRE and SCR* are based (see also [16]). The requirements for the initial prototype were later developed using SCR*, allowing a clean fit with the four-variable version of the Safety Checklist.

From a safety perspective, our primary concern was the completeness of the commonalities and variabilities that the reuse process identified and the requirements for robustness. Thus, the detailed requirements specification of three existing representatives of the product family were the focus of our initial work. By modeling and analyzing the individual members we were able to determine whether variabilities existed that were masked in the individual requirements specifications but needed to be called out in the product family's requirements. We also checked for unstated commonalities that needed to be made explicit in the product family description.

It is worth noting that no design or implementation errors were found in existing aircraft by our safety analysis. The issues found involved the experimental prototypes of a future system currently in the requirements stage of development. Our focus was on validating the emerging product family requirements from a safety perspective.

Following Leveson [10], software safety is defined to be freedom from undesired and unplanned events that result in a specified level of loss. Software safety analy-

sis techniques focus on how software can contribute to conditions that result in loss or failure. In the avionics domain, flight software must often be demonstrably safe. For example, the entire set of display outputs produced by the software analyzed here would need to be certified to DO-178B Level A (the highest level of criticality) since the software's anomalous behavior could cause or contribute to a catastrophic failure condition for the aircraft [20].

1.2 Related Work

The safety analysis techniques used here are well defined in the literature. The completeness criteria, originally developed by Jaffe, Leveson, Heimdahl, and Melhart as a set of predicates that must hold on a finite-state model of the requirements [8], were translated by Lutz into an English-language checklist and used to target root causes of safety-related software errors found during spacecraft integration and system testing. In one study this Safety Checklist was found to address the causes of software safety errors discovered during integration and system testing in 77% of the cases [11]. The checklist was further extended to handle human-computer interactions in Leveson [10].

Forward and backward searches for hazards and their contributing causes are widely used to evaluate the safety aspects of both hardware and software [4, 5, 13, 14, 18]. Previous work indicates that Software Failure Modes and Effects Analysis (forward search for hazards) [15, 19, 23], in conjunction with Software Fault Tree Analysis (backward search for feasible combinations of enabling circumstances), [1, 2, 10] is effective at identifying unsafe situations that can sometimes be alleviated by derived software safety requirements [12]. One goal of our work was to enable the transfer of these safety analysis techniques to the reuse project as it scaled up the dimensions of the product family it was supporting. The safety analysis techniques chosen—Safety Checklist, SFMEA, and SFTA— support this goal in that checklists, FMEAs, and FTAs are techniques familiar to both software and hardware engineers, are widely used and well documented, and can be taught.

2 Safety Analyis Techniques

2.1 Safety Checklist

The checklists in [8, 10, 11] were combined into a single checklist that partitioned the criteria into those appropriate for the components of the four-variable model [7]. The four-variable model documents a system by describing its function in terms of the operations on the input variables, monitored variables, controlled variables, and output variables [17]. Twelve of the forty-six items were allocated to IN, the relationship between monitored and input variables (e.g., "Input received before startup must be acknowledged or ignored"); five checklist items to OUT, the relationship between output and control variables (e.g., "Obsolete data should not be used to generate outputs"); and thirty-four checklist items to REQ, the relationship between monitored and controlled variables (e.g., "Every monitored variable must be used"). The fourth relation, NAT, which defines the environmental context of the system, is not directly addressed by the checklist.

The Safety Checklist was applied to two key components of the display software, the flight director icons and the flight control icons. The process that was followed was:

- Step 1: Informal modeling of the software. Both object-oriented models (akin to UML's class diagrams or OMT's object diagrams in that they identified the object classes, their attributes, and their operations) and state diagrams were created, both to confirm our understanding of the domain and to facilitate the application of the checklist (which assumes a requirements model exists). Since the safety analysis began simultaneously with the project's process of defining the product family requirements, the product family specifications did not yet exist. Thus, the modeling was complicated by the need to capture in a single requirements model the variety among the three family members for which we had the specifications. In general this was accomplished by choosing the best documented family member as the paradigm, modeling it, and describing all variations in textual footnotes. We found that methods to diagramatically capture the variations are an area in which improvement is needed.

- Step 2: Application of the Safety Checklist. A sample of twenty-eight criteria from the forty-six items in the Safety Checklist was applied to the requirements model for one feature component (which can contain one or two composite icons), and five key criteria were applied to the requirements for another, more complicated component (which can contain up to fifteen icons) . The first feature's requirements model contained eight states; the second feature's model contained seventy-four. Criteria from the checklist were selected on the basis of relevancy to this application at this stage of development (the requirements models contained primarily REQ items). Fig. 3

contains examples of the checklist criteria used to verify the components.

1. Each monitored variable must be used.
2. Every mapping from controlled to monitored variables must have a behavior defined for every possible value of monitored variables.
3. Every state must handle timeouts (when input has not arrived within a specified interval or by a specified time).
4. Transitions out of a mode must be deterministic based on the values of monitored variables.
5. Acceptable ranges of values must be specified for each monitored variable used to define the value or duration for some controlled variable.
6. When a monitored variable indicates the input arrival rate exceeds input capacity, abnormal action (such as graceful degradation) may be required.
7. A hysteresis delay action must be specified for human-computer interface data to allow time for meaningful human interpretation. Requirements may also be needed that state what to do if data should have been changed during the hysteresis period.
8. If hazardous state cannot reach a safe state, all paths from the hazardous state must lead to minimum risk states.

Figure 3: Examples of the Safety Checklist Items

- Step 3: Issues found were documented and passed to the project for review. Their status is reported below.

Thirty-four issues were found by means of the requirements modeling and the application of the Safety Checklist items. (This count excludes typographical errors and small, internal inconsistencies that did not affect the product family requirements.) Of these thirty-four items:

- Eleven items were resolved by updated requirements or domain specifications. Updates to the domain products (e.g., by adding a variability to the requirements) resolved several of these issues; an updated requirements document for a member of the product family resolved the others.
- Nine items were open issues (i.e., what precisely a particular software requirement for the product family should be). An example is that there is an unstated requirement for redisplay of an icon when an error flag becomes false. These open issues were referred to the development team for a decision or proposed as changes for subsequent updates to the product family documents.
- Eight items were resolved by reference either to the requirements for other software that interfaced with these components or to design details.
- Six items were due to analyst misunderstanding or error.

2.2 Forward and Backward Search

Forward and backward searches were performed on two components (flight director and flight control) in three representative members of the product family (i.e., three different aircraft) to try to identify unforeseen hazards and their contributing causes. The forward and backward search also allowed further exploration of some of the open issues resulting from the safety checklist analysis. Fig. 4 shows the process that we used.

The forward search was performed first, using the Software Failure Modes and Effects technique (SFMEA) to explore in a structured way the effects of unexpected data or behavior that the software for each feature might experience. In a message-passing model of a distributed system, two kinds of failures are generally represented: communication failures and process failures [9]. In accordance with this model, two kinds of failures are analyzed in a SFMEA for each software component: communication failures (needed to analyze data dependencies and interface errors) and process failures (needed to analyze the effects of software failing to function correctly). To assist in the analysis of any possible failures of the software, two types of tables, Data Tables and Event Tables, are constructed. (A more detailed description of forward and backward searches appears in [12]).

The SFMEA was organized so as to facilitate consideration of the product family requirements. The effect of each category of anomalous data or behavior (e.g., "Timing of Data Wrong," "Abnormal Termination of Process") was considered separately for each of the three representative members of the product family. The effects were documented jointly for all members of the family but were keyed (via superscripts) to

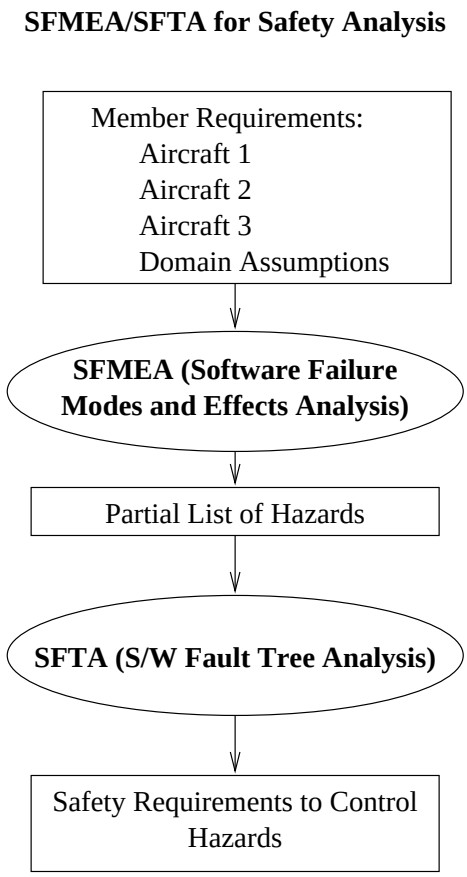

Figure 4: Overview of Forward and Backward Search

Data Label	Data Item	Data Fault Type	Description	Effect
999	MODE WD 1	Absent Data or Timing of Data Wrong	No Data (Flight Control Failure) or Obsolete Data (Flight Control Failure should be declared)	ACTIVE ALTITUDE TYPE ^{X,Y}, ALTITUDE ARM/ALTITUDE ABORT: VERTICAL ARM MODE (Altitude Arm, Vertical Approach); ACTIVE ALTITUDE TYPE, MANY OTHER ELEMENTS: VERTICAL CAPTURE MODE/Vertical Degradation removed 5 sec after Flight Control Failure
999	MODE WD 1	Incorrect Data	Wrong Data (or Flight Control Failure incorrectly declared)	ACTIVE ALTITUDE TYPE ^{X,Y}, ALTITUDE ARM/ALTITUDE ABORT: VERTICAL ARM MODE (Altitude Arm, Vertical Approach); ACTIVE ALTITUDE TYPE, MANY OTHER ELEMENTS: VERTICAL CAPTURE MODE/Vertical Degradation may be missing or inadvertently displayed

Figure 5: Excerpt from SFMEA Data Tables

Event	Event Fault Type	Description	Effect
AP Engaged → Clear	Halt/Abnormal Termination	Processing Stops	"AP Engaged" remains
AP Engaged → Clear	Omission	Jump in Processing	"AP Engaged" remains
AP Engaged → Clear	Incorrect Logic/Event	AUTOPILOT ENGAGE VALID = FALSE or DISCRETE WD SSM != NORM or AUTOPILOT ENGAGE = FALSE computed incorrectly or branch incorrect	"AP Engaged" may remain or other annunciations may be erroneously displayed
AP Engaged → Clear	Timing/Order	Obsolete Data	FLIGHT CONTROL FAILURE should be TRUE, so will clear anyway

Figure 6: Excerpt from SFMEA Event Tables

the individual variations. Fig. 5 shows an example excerpted from an SFMEA Data Table, and Fig. 6 shows an example excerpted from an SFMEA Event Table. The superscripts X and Y relate the failure effect to the specific family members (aircraft types). (In retrospect, we decided that it would be clearer to divide the row in the SFMEA documenting the effects of a single fault into sub-rows, with each sub-row describing the effect of a separate variation.)

This format means that commonalities and variations among the failure effects for the members can be readily extracted from the SFMEA documentation and reviewed. In general, we found that since there were few differences among the members with respect to their input data and their key functionality, that there were few significant differences among the SFMEAs for the representative members.

Nine possible hazardous situations (failure modes) were identified by the SFMEAs. Three of these were of particular concern from a safety perspective. These were: "Autopilot engaged when display shows not engaged," "Fault flag displayed erroneously," and "Cue (aircraft icon) displayed when should be removed."

These three failure modes then served as initial root nodes for the backward search. The backward search used the Software Fault Tree Analysis (SFTA) technique [10] to work backward in time considering the possible combinations of events that could have led to the failure indicated by the root node. For example, Fig. 7 shows the first two levels of the analysis for the hazard, "Cue displayed when should be removed." The possible causes for the second levels were then broken out in the third level, and so on, until the combinations of root causes for this hazard were adequately understood. The SFTA identified enabling circumstances for each of the hazards and allowed an evaluation of the credibility of the postulated hazard.

A simple format, called "WHAT IF?/HANDLED BY" pairs, was developed to facilitate review of the

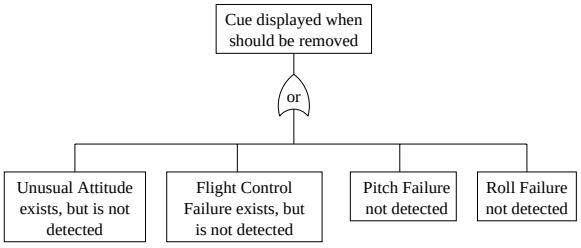

Figure 7: Excerpt from Software Fault Tree Analysis

safety results by the domain experts on the project. The safety analyst listed the possible hazards by describing a hazardous situation as the first element of the pair (the "WHAT IF?"). The safety analyst then described the response (often involving hardware or software beyond the scope of the study) as the second element of the pair (the "HANDLED BY"). For example, ("WHAT IF: arrival rate of input data exceeds software capacity?; HANDLED BY: excess data is ignored"). The domain expert was then able to quickly sort through the list, either confirming the accuracy of the postulated response or correcting it. In those cases where the current requirements for the response to the hazardous circumstances might be strengthened, the "HANDLED BY" element provided a possible derived requirement for subsequent inclusion in the product family requirements specification.

The identification of circumstances that contributed to the possibility of a hazardous situation allowed us to derive some software safety requirements that were recommended to the project as additions to the product family requirements. These included requirements for persistence checks (to ignore transient failures), reasonableness checks on the output (to disallow inconsistent displays), additional checks of validity bits, and references to a single-point failure policy (that no single failure shall jeopardize the system) in the requirements.

3 Lessons Learned

The Safety Checklist was effective at identifying additional variations (instances in which the product family requirements were incomplete) and at identifying additional commonalities (hidden assumptions that were required to be true). It is of interest that most of the identified gaps in requirements completeness were readily resolved by updating the domain assumptions for the product family, but might have been difficult to fix once a design had been chosen. This confirms the common understanding that fixing requirements errors at the requirements stage is often easy, but that fixing requirements errors once they are erroneously implemented is difficult. The explicit and accurate documentation of variabilities in the domain assumptions appears to be key to building safe members of a product family.

The criteria in the Safety Checklist can often be declared satisfied only by knowledge of design details that do not exist at the requirements stage (e.g., "Are there any paths to hazardous states?"). Thus, applying the Safety Checklist is not a one-shot analysis technique but instead serves as a set of liens on the developing software. Some of these concerns can be discharged at the requirements stage, but others remain in effect until further design decisions or details are known.

The need to iterate the Safety Checklist as the requirements mature and the design evolves appears to be especially true with regard to product family software requirements. For several criteria in the checklist, whether or not the criterion was satisfied depended on which variability was selected by the customer. For example, if the hardware detects no activity in a buffer that provides an interface to the software under consideration, then whether or not an error indication is displayed is a customer-driven variability. Such checklist criteria can serve as useful input to the development process since they allow prediction of possible safety concerns even before decisions are made. (Thanks to S. Easterbrook for this insight.)

The commonalities in the product family members could usually be verified by the checklist at the requirements level. However, for some variations, evaluating the criteria in the checklist had to be deferred until design decisions were made and design details were known. In these cases, the documentation of the reason why this particular criteria in the checklist could not be declared satisfied at the requirements stage serves as a derived, domain-dependent checklist, tailored to the particular application. The checklist thus serves a useful role in providing continuity in the safety analysis and in preserving knowledge about the safety-related issues, even for variable requirements.

Some criteria in the checklist were inherently satisfied by the representation of the requirements in the four-variable model. Of the forty-six criteria, fifteen were satisfied by the CoRE and SCR methodologies. CoRE/SCR satisfied eight of the twelve IN criteria and two of the five OUT criteria, but only five of the thirty-four REQ criteria. The safety analysis thus focused on verifying the REQ criteria.

The four-variable model was a good match with the Safety Checklist in that the items in the checklist were mapped to elements in the model. An added advan-

tage was that when items in the checklist were applicable to more than one element of the model (e.g., some apply to both IN and REQ), the separate application of the checklist items to the separate elements provided a clean decomposition in the safety analysis.

All the safety analysis techniques chosen required a substantial amount of domain knowledge. This is typical of safety analyses since errors tend to occur at the interfaces between components, or between software and hardware. The three individuals who performed the safety analyses were not familiar with digital instrumentation displays when they began this work, but had the advantage of extensive documentation of mature systems and the ready support of domain experts. The process that quickly evolved and that worked well was for the safety analysts to apply the safety techniques in a structured manner and rapidly pass the initial list of postulated issues to the domain experts for an informal review. The project provided strong technical support to the safety analysis, including timely access to project updates, ongoing email discussions, regular telecons, and rapid feedback on technical questions.

The forward search was useful in identifying some hazards that might be prevented or handled by additional software requirements for the product family. By considering the effect of faulty data and anomalous software behavior, the SFMEA allowed an initial list of possible hazards to be assembled for further analysis. The SFMEAs provided an organized way to consider the effect of the variations in the product family for each postulated fault in the safety analysis. By adding additional rows to the SFMEA tables, the different members of the family could be considered together (same fault, possibly variable effects).

We are wary of generalizing from our experience since combinations of subtle variations in the requirements may, in other applications, be able to produce significant variations in the effects documented by the SFMEA. It does, however, seem likely that once a baseline SFMEA has been performed on a product family's requirements, that subsequent SFMEAs for new instances of that family can be produced fairly rapidly. The SFMEAs for new members can reuse much of the baseline SFMEA and concentrate on the effects of the specific combination of variations relevant to that particular product member. This reuse could reduce the size, the effort, and the tedium of subsequent SFMEAs by focusing resources on what is unique about a particular instance of the product family. Further work to pursue this line of inquiry is merited.

The backward search was less effective in this domain. This was primarily due to the fact that the backward search from a display to the source of its data quickly broadens the scope of the system that must be considered. Since our safety analysis was restricted to the display subsystem, we were unable to effectively evaluate the software that fed data to the display subsystem or the hardware that provided these interfaces. Despite these limitations of scope, the backward search did provide insight into additional software requirements that we could recommend to enhance safety. An example is a requirement for added checks that the display elements output are consistent with each other.

4 Conclusion

In developing the software requirements for a product family, safety concerns focus primarily on the completeness and consistency of the requirements and on the requirements for robustness. The safety analysis techniques described here were useful in detecting some incomplete software requirements for the flight instrumentation display family. Application of the portion of the safety checklist that contains criteria for REQ (the expected behavior of the system) identified some implicit assumptions that needed to be captured as additional commonalities as well as some previously undocumented variabilities in the requirements.

In order to enhance the robustness of the product family software, a forward search for hazards (SFMEA) and a backward search for enabling circumstances (SFTA) were performed. Derived safety requirements to provide enhanced handling of transient failures, of inconsistent displays, and of invalid data were recommended for inclusion in the product family requirements. In addition, the forward and backward search provided a preliminary hazards list as a baseline for future safety analysis.

The safety analysis reported here assisted in the development of complete and robust software requirements for a product family from legacy family members. Further work is needed to better understand how safety analysis techniques such as the ones applied here (evaluation of requirements against key completeness criteria, forward search, and backward search) can, once performed on the requirements for a product family, be exploited for rapid reuse on future instances of the product family.

Acknowledgments

The authors thank Roger W. Shultz, Steven P. Miller, Karl F. Hoech, Gary W. Daugherty, and James N. Potts for valuable discussions and suggestions, and

Stuart R. Faulk for answering some questions about the four-variable model.

References

[1] Cha, S. S., N. G. Leveson, and T. J. Shimeall (1991), "Safety Verification of Ada Programs Using Fault Tree Analysis," In *IEEE Software*, 8, 4, 48–59.

[2] de Lemos, R., A. Saeed, and T. Anderson (1995), "Analyzing Safety Requirements for Process-Control Systems," *IEEE Software* 12, 3, 42–53.

[3] Faulk, S., J. Brackett, P. Ward, and J. Kirby, Jr. (1992), "The CoRE Method for Real-Time Requirements," In *IEEE Software*, September 1992, 22-33.

[4] Fencott, C. and B. Hebbron (1995), "The Application of HAZOP Studies to Integrated Requirements Models for Control Systems," *ISA Transactions* 34, 297–308.

[5] Heimdahl, M. P. E. and N. G. Leveson (1996), "Completeness and Consistency in Hierarchical State-Based Requirements," *IEEE Transactions on Software Engineering* 22, 6, 363–377.

[6] Heitmeyer, C., A. Bull, C. Gasarch, and B. Labaw (1995), "SCR: A Toolset for Specifying and Analyzing Requirements," In *Proceedings of the 10th Annual Conference on Computer Assurance*, IEEE, Gaithersburg, MD, pp. 109–122.

[7] Helmer, Guy G. (1997), "Safety Checklist for Four-Variable Requirements Methods," TR98-01, Iowa State University Department of Computer Science.

[8] Jaffe, M. S., N. G. Leveson, M. P.E. Heimdahl, and B. E. Melhart (1991), "Software Requirements Analysis for Real-Time Process-Control Systems," *IEEE Transactions on Software Engineering*, 17, 3, 241-257.

[9] Lamport, L. and N. Lynch (1990), "Distributed Computing Models and Methods," In *Handbook of Theoretical Computer Science, vol. B, Formal Models and Semantics*, J. van Leeuwen, Ed. Cambridge/Amsterdam: MIT Press/Elsevier, pp. 1157–1199.

[10] Leveson, N. G. (1995), *Safeware: System Safety and Computers*, Addison-Wesley, Reading, MA.

[11] Lutz, R. R. (1996), "Targeting Safety-Related Errors During Software Requirements Analysis," In *Journal of Systems and Software*, 34, 223-230.

[12] Lutz, R. and R. Woodhouse (1997), "Requirements Analysis Using Forward and Backward Search," *Annals of Software Engineering, Special Volume on Requirements Engineering*, 3, pp. 459–475.

[13] Maier, T. (1995), "FMEA and FTA To Support Safe Design of Embedded Software in Safety-Critical Systems," In *CSR 12th Annual Workshop on Safety and Reliability of Software Based Systems*, Bruges, Belgium.

[14] McDermid, J. A. and D. J. Pumfrey (1994), "A Development of Hazard Analysis To Aid Software Design," In *Proceedings of the 9th Annual Conference on Computer Assurance*, IEEE, Gaithersburg, MD, pp. 17-25.

[15] Military Standard, *Procedures for Performing a Failure Mode, Effects and Criticality Analysis* (1980), MIL-STD-1629A.

[16] Miller, S. P. (1998), "Specifying the Mode Logic of a Flight Guidance System in CoRE and SCR," *2nd Workshop on Formal Methods in Software Practice*, Clearwater Beach, FL, forthcoming.

[17] Parnas, D. L. and J. Madey (1995), "Functional Documents for Computer Systems," In *Journal of Systems and Software*, 25, 1, 41-61.

[18] Reese, J. D., "Software Deviation Analysis," Ph.D. thesis, University of California, Irvine, California, 1995.

[19] Reifer, D. J. (1979), "Software Failure Modes and Effects Analysis,"*IEEE Transactions on Reliability*, R-28, 3, 247-249.

[20] RTCA/DO-178B (1992), *Software Considerations in Airborne Systems and Equipment Certification*, RTCA, Inc., 1140 Connecticut Avenue, NW, Suite 1020, Washington, DC, 20036-4001.

[21] Software Productivity Consortium (Nov., 1993), *Reuse-Driven Software Processes Guidebook*, SPC-92019-CMC, v. 02.00.03.

[22] Weiss, D. M. (1997), "Defining Families: The Commonality Analysis," submitted for publication.

[23] Wunram, J. (1990), "A Strategy for Identification and Development of Safety Critical Software Embedded in Complex Space Systems," IAA 90-557, 35-51.