Software Safety and Certification: Reintroducing the FMEA

Rebecca Menes, Ph.D.; SoHaR Inc; Culver City, California, USA

Herb Hecht, Ph.D.; SoHaR Inc.; Culver City, California, USA

Keywords: software certification, software FMEA, model-based design, exception handling

Abstract

As the relative importance of software and hardware in critical systems continues to shift from mainly hardware to a more balanced position, we have to reevaluate how we approach safety and certification of software. Aviation is a field that illustrates how outdated and limited our current efforts are. Current practices are both costly and unable to ensure full coverage of potential failures. We propose to revisit the FMEA (Failure Modes and Effects Analysis) as a framework for streamlining certification of critical software systems. Using model-based design environments we have shown that not only is the FMEA a highly effective framework for software safety analysis and certification, but that we were able to automate a large portion of this effort. The resulting certification process is both cost effective and verifiable. We provide an example of this method and the use of a partially automated software FMEA generator.

Introduction

Safety concerns take on a different focus when we consider software heavy critical systems. Thus we apply a shift from a mainly-hardware oriented certification process to a software/hardware balanced process. This shift should increase the consistency between hardware and software elements both in certification format and thoroughness. An obvious example is that of the certification of flight control systems, particularly for unmanned vehicles. In these systems there is an increasing emphasis on autonomy which implies that the accomplishment of a mission as well as the responsibility for handling exception conditions is implemented in software. The main example presented in this paper will be from this domain.

Currently, hardware vulnerabilities and risks from software faults are treated, in certification, on very different planes. This is the result of habit and the lack of a comparable process. Although hardware and software often still take on different roles, their faults can generate the same failures and/or failures of equal severity. Therefore an equally stringent and comprehensive process of certification should be applied to both elements. In order to illuminate the way in which our method (and the tool we have developed) solves the issue of software certification within the entire system certification process, we first discuss the current certification methods and we focus on their limitations when applied to software-heavy systems. The approach we propose offers a much greater ability to form a consistent process that offers the same format for software and hardware.

We advocate the use of the Failure Modes and Effects Analysis (FMEA) as the scaffold not only for hardware safety analysis and certification, but also for software and for the entire system (hardware and software) as a unit, thus presenting hardware and software on the same footing and allowing for a consistent and *complete* process.

The issue of completeness, which will be discussed in the following section regarding current practices and their limitations, is especially crucial as the current practices do not offer a proof that a certified software element has been evaluated for its complete set of failures and hazards. Our approach is especially effective as it identifies a method for establishing a FMEA, anchored in model-based or object-oriented design environments, that can be shown to be complete as it enumerates and deals with *all* objects that can contribute to failure modes and all their known failure effects and system hazards (regardless whether contained within the software element or with system wide consequences).

A desirable side-effect of the use of a FMEA in assessing hardware safety is that it requires that system engineers and component engineers work together to assess and mitigate safety issues. In the world of software, even safety critical software, the common practice is to leave safety and reliability assessment and testing to the software professionals as they "know the software" best. This introduces a serious deficiency in safety analysis as the software professionals are not always aware of system-wide issues, requirements and ultimately system level consequences of software failures. Most importantly, they may not be aware of all the exception conditions and anomalies that the system as a whole may encounter and that usually have to be handled by software. Specific examples are presented later.

In the following section we discuss limitations of current certification practices. This is followed by the identification of FMEA as a solution, and discussion of a tool for computer-aided FMEA generation in specific design environments.

Current Practices

The objective of most safety certification activities is to assure that the risk of using a device or process is acceptable to the user. The risk is a function of the severity of the consequences of a given failure mode and of the probability of occurrence of that failure mode. The failure modes that can cause the most severe (catastrophic) consequences must be shown to have an extremely low probability, whereas the probability of failure modes that cause less severe consequences can be accepted at a higher level. This general approach to risk assessment is diagrammed in Figure 1. These relations are applicable to hardware as well as to software.

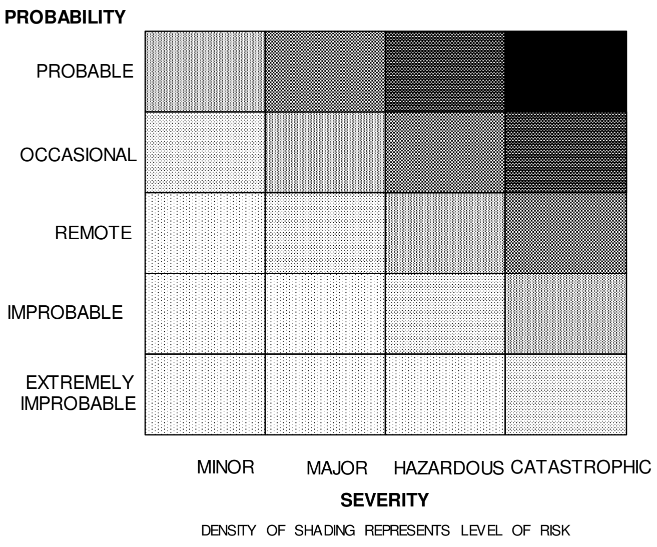


Figure 1— Risk assessment

Risk, safety assessment and certification procedures are highly developed in the field of aviation as the potential failures carry such a high price both in human lives and in equipment. Therefore this is a good area to investigate the adequacy of practices applied to software elements and software-heavy systems.

The acceptable safety risk levels for systems installed on major aircraft are governed by Federal Aviation Regulation 25.1309, and for practical purposes the FAA Advisory Circular AC 25.1309-1A (ref. 1) establishes the ground rules for certification. In the assessment of the probability of failure the AC makes the following observation for software (par. 7i): "In general, the means of compliance described in this AC are not directly applicable to software assessments because it is not feasible to assess the number of kinds of software errors, if any, that may remain after the completion of design, development and test." The AC then relegates software certification to what is currently DO-178B.

1. Introduction
2. System aspects relating to software development
3. Software life cycle
4. Software planning process
5. Software development processes
6. Software verification process
7. Software configuration management proc.
8. Software quality assurance process
9. Certification liaison process
10. Overview of aircraft & engine certification
11. Software life cycle data

Figure 2 — Contents of DO-178B

DO-178B, the compliance document for FAA certification of software for flight controls or similar avionics applications is overwhelmingly focused on control of the software process (ref. 2) as can be seen in the table of content that is reproduced in Figure 2. This approach assigns specific responsibilities to systems engineering (section 2) and software developers (all following sections), thus minimizing the collaboration that is necessary both for efficiency and completeness of the certification process.

Both the AC and DO-178B translate aircraft (system) level severity categories to software level definitions. The DO-178B translation, which is later and more specific, is shown in Table 1. The descriptions are highly condensed from the full versions in par. 2.2.1 and 2.2.2 of DO-178B.

Table 1 — Severity levels

System Level	**Description**	**Software Level**
Catastrophic	Prevents safe flight and landing	A
Severe-Major	Large reduction in safety margins, potential injuries	B
Major	Reduction in safety margins under adverse conditions	C
Minor	Slight reduction in margins or increase in crew workload	D
No Effect	No operational effect	E

The AC (Paragraph 10.b) sets the following quantitative limits on the probability of failure per flight-hour at the system level:

Catastrophic $< 10^{-9}$
Severe-Major $<< 10^{-5}$
Major $<10^{-5}$

The Severe-Major category is only tentatively defined in the AC but detailed in DO-178B. The latter document then defines the process control, configuration management and documentation requirements for levels A – D (level E has no requirements), with the most exhaustive ones applicable to level A.

Challenges of the Current Practices

Limits of Current Safety Certification Process Control: The translation of severity categories from system to software as described above raises two difficulties in terms of risk assessment:

(1) The process control (Chapters 4 – 9 in Figure 2) cannot be analyzed in terms of the quantitative requirements of par. 10.b of the AC, whereas such an analysis from a combination of materials properties, overload testing and redundancy is routinely performed for hardware-only systems. Once more, we do not have a consistent hardware-software process.

(2) The entity or software partition for which the translation is to be made is not defined, but the context of DO-178B implies that it is at a functional unit, for example, one axis of an aircraft attitude control system.

The first of these two issues appears to be accepted by the certifying body – perhaps because nothing much better appeared to be available. But it must be recognized that the reliance on software process control together with its application to large functional partitions is a major factor in the expense and schedule impact of current approaches to software certification.

Since the aviation/aerospace industry has historically been a major source of safety and certification practices to many safety critical industries and programs, this inefficiency affects more and more software-centric decision and control systems that have critical roles and safety-critical vulnerabilities. Certification through process control is a promise for ever increasing personnel levels without assurance of improved safety or complete coverage of failures. The process control focuses on (safety) critical portions of code, not on critical failures or critical methods/objects.

The second difficulty may be even more severe as it indicates a lack of a systematic method to approach software failures in the system level context. Here we encounter also the issue of completeness of the process – how do we enumerate and ensure all failure conditions were met

The inadequacy of the process based approach can be seen in a failure of the Boeing 777 operational program software (OPS) that caused the FAA to issue emergency airworthiness directive (AD) 2005-18-51 for all 777 models. The AD describes "a significant pitch-up event on a Boeing 777-200 while climbing through 36,000 ft altitude. The flight crew disconnected the autopilot and stabilized the airplane during which time it climbed above 41,000 feet, decelerated to 158 knots and activated the stick shaker….These errors were caused by the OPS using data from a faulted (failed) sensor." The root cause is a lack of focus, in the process based approach, on the handling and testing for exception conditions. The FMEA approach specifically targets exception handling because that is where most failures in well-tested systems occur as is shown in references 3 and 4.

Limits of Fault Tree Analysis: Fault trees are an established tool for top-down analysis in safety critical systems. A hazardous condition (top event) is postulated and the possible contributors to this event are identified. The fault tree can be used to compute the failure probability of the top event, given the failure probabilities of the contributors. Most developers and reviewing organizations have tools that provide standard values for the failure probabilities of hardware parts and then calculate the top event probability. For software there are no standard failure probabilities and thus the value of fault tree analysis is considerably limited.

The treatment of software failure probability in the fault tree analysis therefore has to be extremely conservative. This may lead, even when the role of software is limited, to a high incident probability. It is also unsatisfactory because it does not distinguish between software failure modes and thus point to corrective measures that might be taken to improve the reliability.

Delegation of Certification Responsibilities to Software Professionals: The focus of DO-178B (as summarized in the table of contents shown in Figure 2) on the software development process as the basis for certification, is a clear result of the view that software certification is the responsibility of software professionals. This is also the result of the assumption that software professionals understand the vulnerabilities of the software and are familiar with tools and methodologies for testing. However, software engineers, who are not system engineers, are not well qualified to

- Identify the most critical failure effects
- Be aware of early warning signs of critical failures
- Provide means of mitigating the effects of critical failures
- Determine all exception conditions and anomalies that will be encountered by the system as a whole or through interfaces in the system and may affect software performance.

These inputs are essential to the focus of hardware certification and are not only important in terms of covering high risk components; they are also contributors to the reduction in cost and effort involved in the certification. The certification of software, as a system component, could be much more efficient and cost effective if the responsibility were shared both by software and system engineers.

A Framework for Certification of Software Intensive Systems

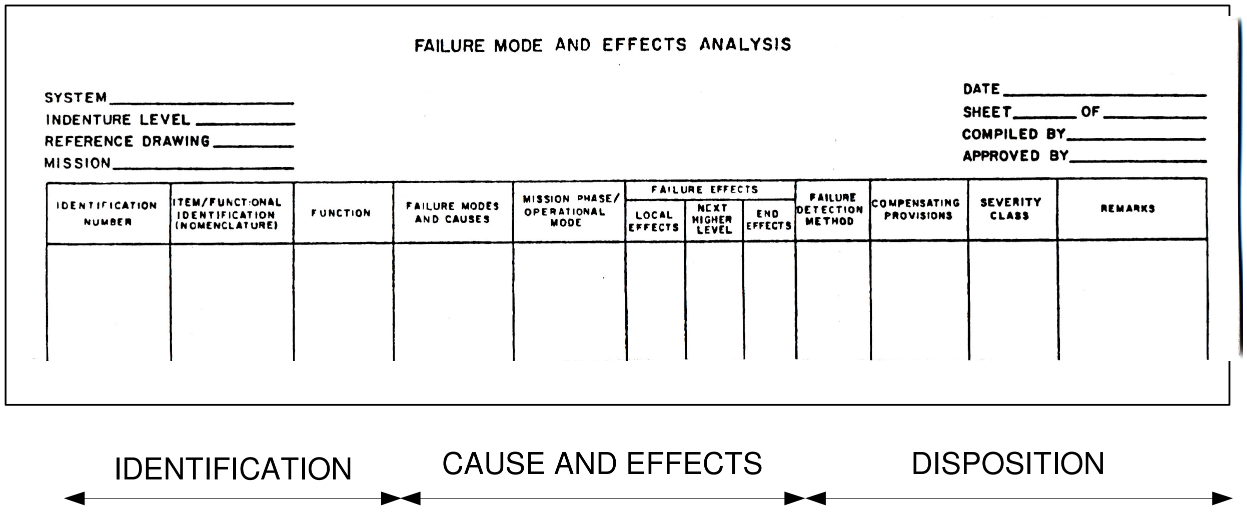
A significant benefit can be found if we are able to establish a certification process that will focus first and foremost on critical failure effects that will be translated into software testing efforts.

Requirements: The general acceptance of the certification process for hardware, and the extensive experience it has provided over decades, are an indication that if we are able to transfer some major components of the process to software as well, we would have a good starting point.
This framework should form a meeting-ground for system engineer and software engineer (as is the case for hardware where the system and component engineers both contribute to the certification process). The framework must be practically implemented in a generic way independent of platform/coding. The framework must be able to ultimately deal with combined software and hardware systems and transition from one aspect to another transparently, posing no obstacles to the system engineers.

Failure Modes and Effects Analysis (FMEA) as the Meeting Ground for System and Software Engineers: Failure Mode and Effects Analysis (FMEA) is a center piece of hardware system certification. FMEA combines the view of the system engineer with the insight of the component designer and it is one of the recognized analysis methods in the AC (par. 9.c). The FMEA not only provides a meeting ground for the complementary disciplines, it also provides a solid structure for identifying the critical elements in a system and ensuring full coverage of vulnerabilities.

A typical hardware FMEA worksheet is shown in Figure 3. The first group of columns identifies the item that is being analyzed. The second group starts with specific failure modes and propagates these to effects at the local, intermediate and system level. This group requires collaborative effort of the hardware specialist and the systems engineer, with the latter having exclusive responsibility for identification of system level effects. The final columns deal with the severity of the effects and measures for detecting and alleviating them and for these the system engineer usually takes the lead. This structured collaboration between the specialist and the system engineer in the analysis of individual failure modes leads to a much more efficient identification of those that are critical.

In the most rigorous hardware FMEA procedure, the identification of parts is based on the bill of materials (BOM). When all of the parts from the BOM have been analyzed the FMEA is complete, and the ability to demonstrate that the FMEA is complete is an important advantage of the part level procedure. Also, for most mechanical and electronic parts the failure modes are well known (e. g., open, short and parameter drift for electronic parts).



FAILURE MODE AND EFFECTS ANALYSIS

SYSTEM__________ DATE__________
INDENTURE LEVEL__________ SHEET_____ OF_____
REFERENCE DRAWING__________ COMPILED BY__________
MISSION__________ APPROVED BY__________

IDENTIFICATION NUMBER	ITEM/FUNCTIONAL IDENTIFICATION (NOMENCLATURE)	FUNCTION	FAILURE MODES AND CAUSES	MISSION PHASE/ OPERATIONAL MODE	FAILURE EFFECTS			FAILURE DETECTION METHOD	COMPENSATING PROVISIONS	SEVERITY CLASS	REMARKS
					LOCAL EFFECTS	NEXT HIGHER LEVEL	END EFFECTS				

Figure 3 — Generic FMEA worksheet format (from MIL STD 1629A)

Alternatively, the FMEA can be conducted using a functional approach, a much less rigorous procedure because the scope of a “function” can be variously interpreted. As an example, it could be an entire power converter, or the input filter, the switching section, and the output filter. Under either interpretation the function comprises a multitude of parts and therefore the failure modes are less well defined. Also, the criterion for completion is much more difficult to define for the functional than for the parts FMEA. For these reasons the functional hardware FMEA is primarily used in early project stages before the parts population is defined.

Many attempts have been made to extend FMEA to the software realm (ref. 5-7) but all of them were faced with two essential differences between hardware and software:

- Hardware is assembled from discrete parts that have catalog numbers and defined characteristics – software is amorphous.
- Most hardware parts have known failure modes – software failures can range over a wider and sometimes unpredictable spectrum.

The first two of the referenced publications (and several more that followed them) use a functional decomposition of software, thus sharing the problems mentioned for the functional hardware approach. The uncertainty of the scope of a function becomes a much more serious concern in software because exception handling may not be identified and analyzed as a separate function. Yet as a result of the Y2K software reviews (and also several prior investigations) it has been recognized that errors in exception handling are responsible for most failures (ref. 3). “The main line software code usually does its job. Breakdowns typically occur when the software exception code does not properly handle abnormal input or environmental conditions - or when an interface does not respond in the anticipated or desired manner" (ref. 4).

The last cited software FMEA article (ref. 7) dispenses with the functional partitioning and instead analyzes the effect of failures in output variables of a software module, one at a time. For some failure modes this may indeed be rigorous, but it neglects processing errors within a module that cause failures of more than one output variable. Analyzing in detail the effects of simultaneous failures in multiple output variables requires so much effort that it is usually impracticable.

Benefits of model based software environments and automating the FMEA: The advent of model-based software development methodologies (such as Simulink) that build up programs from defined primitives provides the opportunity for a fresh approach to constructing a software FMEA. These methodologies provide libraries of primitives (and permit the construction of additional ones) that can be equated to hardware parts in that they have predefined operational characteristics (specific "methods" of operation) and thus it is simple to determine whether they are in a working or failed mode. Further, the listing of the primitives that have been selected for a software module is analogous to a hardware Bill of Materials (BOM). This permits the software FMEA to be declared complete when the entire list has been analyzed. The common format for hardware and software through these methods of operation also allows a transparent transition from one realm to the other enabling a full system FMEA and simplifying the system engineer's role.

An automated implementation is possible in some object-oriented environments: The "objects" ("methods", "classes", "blocks") can be extracted from the design file and a list of failure modes can be generated either automatically (if the information is included in the library) or manually (by the designer/analyst). Moreover, hierarchical information that is included in the definition of the "objects" can be used to automate the process of identification of "higher effects". As in hardware, the final stage of identifying and categorizing "system effects" is handled manually by the system engineer.

Two instances of methodologies that are of interest in this context are the Unified Modeling Language (UML) (ref. 8 and 9) and MATLAB/Simulink (ref. 10) (another instance is that of AADL (ref 11), developed by SAE for the automotive industry). UML and Simulink are both extensively used by industry, and both have a substantial support environment. These environments support external interfacing, which has enabled us to automate the software FMEA process. We will focus on these two environments for the remainder of this paper as they contain the main ingredients for performing the FMEA (manually or automatically) as well as ensuring its completeness.

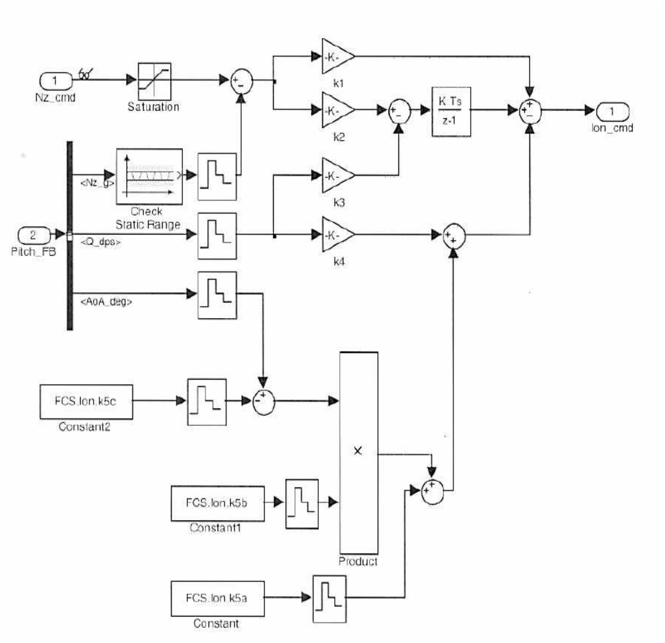
For the UML methodology the primitives of interest are called methods or behaviors, in some cases, classes. In MATLAB/Simulink they are called blocks. Both methodologies permit access to databases from which the names of the primitives, their hierarchical relationship and connectivity can be extracted. The data structures also can be annotated and one interesting use of this feature is the labeling of software exception handling provisions, i. e., primitives (or groupings of them) that are used to detect and/or mitigate failures. The labeling facilitates focused review and testing of exception handling, particularly at the system level.

Examples of this approach using our MOVAT (Model based Verification and Assurance Tool) tool for UML constructs have been previously described (ref 12). The example we used was of a UAV swarm that used a "heart beat" signal among the UAVs to maintain the swarm structure (leader and followers). The automated generation of the FMEA showed how obscure failure modes, that are likely to go unrecognized in a "functional" FMEA, become identified in the computer-aided approach and can be disposed of by collaboration between software and system engineers.

Of particular interest to the aviation control community is the application of a computer-aided approach to the MATLAB/Simulink environment which is the main environment for the development and testing of avionic control systems. The case described in the following section is an example of an automated implementation of the FMEA framework in the MATLAB/Simulink environment which we call MOCET: Model based Certification Tool.

Automated Generation of FMEA of UAV Control System Designed in the MATLAB/Simulink Environment: Figure 4 shows the generic design of an aircraft pitch control system. The computer-aided generation of a software FMEA for this system from MATLAB/ Simulink data by means of the MOCET tool is now described with reference to this figure. Each one of the symbols appearing in the figure is a block with defined functional and failure mode properties. The system responds to the normal acceleration command Nz_cmd by commanding the longitudinal control deflection lon_cmd. The output is also affected by the feedback signals shown as input (2). The first of these is the measured normal acceleration Nz_g. The others are limits to prevent violating dynamic pressure and angle of attack constraints, Q_dps and AoA_deg, respectively.

For safety reasons the commanded normal acceleration is limited by the saturation block. In addition, the measured value of normal acceleration is checked, and if it exceeds the input limit (plus an allowance for disturbances) the output of this channel of the flight control system is disconnected. Other channels or back-up provisions will compensate for the missing output and it can be restored when the acceleration feedback returns to the normal range and other constraints show the program to be operating satisfactorily.

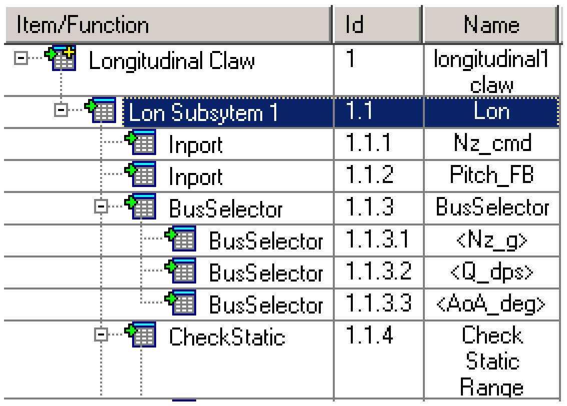


```
Block {
  BlockType               Inport
  Name                    "Nz_cmd"
  Position                [35, 148, 65, 162]
  IconDisplay             "Port number"
}
Block {
  BlockType               Inport
  Name                    "Pitch_FB"
  Position                [15, 278, 45, 292]
  Port                    "2"
  IconDisplay             "Port number"
}
Block {
  BlockType               BusSelector
  Name                    "Bus\nSelector"
  Ports                   [1, 3]
  Position                [60, 206, 65, 364]
  ShowName                off
  OutputSignals           "Nz_g,Q_dps,AoA_deg"
```

Figure 4 — Generic flight control system

Figure 5 — Block structure file (Excerpt)

The diagram shown in Figure 4 is the graphical representation of the block structure text file, portions of which are shown in Figure 5. MOCET operates on this text file to produce a hierarchical listing of the blocks that serves as input to the first two columns of the FMEA worksheet. This parsed file is shown in Figure 6. The automated generation of this hierarchical structure may appear as a trivial accomplishment but is an essential step in bringing software FMEA into equivalence with hardware FMEA. Figure 6 has the format of a Bill of Materials, and in its fully implemented form it is a complete listing of the software primitives that make up the longitudinal control system.



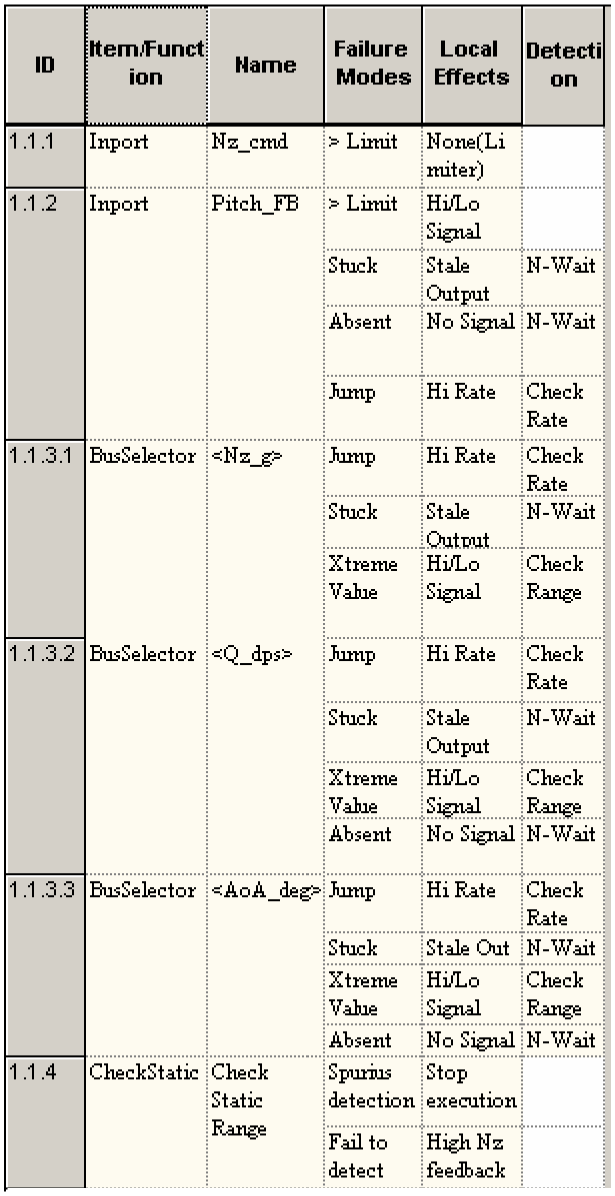
Item/Function	Id	Name
Longitudinal Claw	1	longitudinal1 claw
Lon Subsytem 1	1.1	Lon
Inport	1.1.1	Nz_cmd
Inport	1.1.2	Pitch_FB
BusSelector	1.1.3	BusSelector
BusSelector	1.1.3.1	<Nz_g>
BusSelector	1.1.3.2	<Q_dps>
BusSelector	1.1.3.3	<AoA_deg>
CheckStatic	1.1.4	Check Static Range

Figure 6 — Parsed structure (excerpt)

The four and five digit entries in Figure 6 are the working elements of the longitudinal channel and thus the ones for which failure modes and effects need to be analyzed. These primitives represent units of software but their purpose is to operate on data that represent physical quantities in the flight control system.

Thus the failure modes and effects of these primitives can be evaluated in terms of the physical representations (timing properties that are unique to software are separately analyzed by means of timed Petri nets that are outside the scope of this paper). Starting with the Nz_cmd (1.1.1 in Figure 7) MOCET recognizes that typical failure modes for a command are (a) absence of the command, (b) a jump in command value (exceeding the expected rate of change), and (c) values that exceed the expected command range. These potential deviations from normal operation are entered automatically in the third (Failure Mode) column of the FMEA worksheet as shown in Figure 7.

MOCET also generates the local effects entries in the fourth column. Effects at the intermediate and system level need collaboration with the systems engineer and these entries are currently not generated by the tool.



ID	Item/Function	Name	Failure Modes	Local Effects	Detection
1.1.1	Inport	Nz_cmd	> Limit	None(Limiter)	
1.1.2	Inport	Pitch_FB	> Limit	Hi/Lo Signal	
			Stuck	Stale Output	N-Wait
			Absent	No Signal	N-Wait
			Jump	Hi Rate	Check Rate
1.1.3.1	BusSelector	<Nz_g>	Jump	Hi Rate	Check Rate
			Stuck	Stale Output	N-Wait
			Xtreme Value	Hi/Lo Signal	Check Range
1.1.3.2	BusSelector	<Q_dps>	Jump	Hi Rate	Check Rate
			Stuck	Stale Output	N-Wait
			Xtreme Value	Hi/Lo Signal	Check Range
			Absent	No Signal	N-Wait
1.1.3.3	BusSelector	<AoA_deg>	Jump	Hi Rate	Check Rate
			Stuck	Stale Out	N-Wait
			Xtreme Value	Hi/Lo Signal	Check Range
			Absent	No Signal	N-Wait
1.1.4	CheckStatic	Check Static Range	Spurius detection	Stop execution	
			Fail to detect	High Nz feedback	

Figure 7 — Partial FMEA worksheet

MOCET is programmed for commonly used detection mechanisms for the listed failure modes. Figure 4 shows a range check block in series with the normal acceleration feedback signal. Similar range checks can be inserted into any signal path where out-of-range values can cause serious effects. The implementation of the range check is shown in Figure 8. The instantaneous signal value, designated u in the figure, is compared with preset minimum and maximum constants in the relational operator blocks. Values of u below the preset minimum or above the preset maximum will cause (a) immediate blocking of the output signal and (b) initiation of mitigating action via the assertion path. Abnormal discontinuities in a variable (called jump in Figure 7) are similarly detected by operating on the difference between consecutive signal values. Absence of input or output can be detected by an N-wait module. Because a zero value of a signal may be encountered in routine operation, the instantaneous value is not a suitable failure criterion. However, if the zero value persists for a number of processing cycles it can be taken as an indication of anomalous conditions. The value of N is selected to be a number of cycles for which the variable does not normally dwell at zero (typically this is three cycles). Correct operation of the detection mechanisms is frequently monitored by self-test routines that are invoked during idle times of the computing cycle. All these mechanisms can be included in the block properties for automatic insertion in the FMEA, which would also automatically indicate if the appropriate detection mechanisms were included in the design.

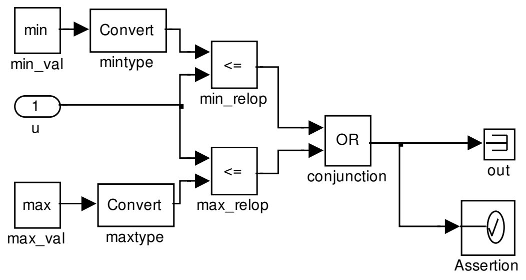


Figure 8 — Range check implementation

Conclusion

The FMEA based approach to software certification promotes collaboration between software professionals and system engineers on a block by block basis. This detailed interaction promises to overcome inefficiencies of current practice where system engineers define criticality levels for major software functions and software professionals assume responsibility for assuring that the entire function is processed to an adequate integrity level.

The individual FMEA row entries then become the meeting ground for interdisciplinary collaboration on certification issues. It is no longer the criticality of a major function that determines the criticality of all software constructs within that function. Instead, for each row in the FMEA the team reviews severity of potential failure effects, adequacy of failure detection and capabilities of failure mitigation. These factors together determine the need for detailed software review and the level of testing required. Model-based and object-oriented design environments such as UML and MATLAB/Simulink naturally support the FMEA structure. They also offer automation possibilities that cannot be achieved in other design environments and are as useful as the tools and structures available for hardware.

We have developed automated tools for these two environments (MOVAT for UML and MOCET for Simulink) that generate software failure modes and effects analysis that is in format and content fully compatible with a part level hardware FMEA. The significant advance over previous attempts to extend the FMEA technique to software is that the row entries are automatically generated from the software development files. This capability:

1. Overcomes the inherent subjectivity of the functional approach for software FMEA (what is a function?)
2. Establishes a clear completion criterion for the FMEA and the certification process (when all objects in the files have been analyzed).

Extensions of MOCET for partial automation of the higher level failure effects entries and for exploration of timing issues are still under development. MOVAT includes computer-aided capabilities for generating timed Petri nets from UML collaboration charts that can identify timing conflicts.

The tools and methodology have been tested on current UAV control systems. Examples include aircraft pitch control system as well as a “heart-beat” monitoring system within a swarm of autonomous UAVs.

References

1. Federal Aviation Administration, *Advisory Circular, System Design and Analysis,* AC 25.1309-1A. June 1988.
2. RTCA (formerly Radio Technical Commission on Aeronautics), *Software Considerations in Airborne Systems and Equipment Certification, DO-178B,* December 1992

3. Hecht, Herbert and Crane, Patrick, "Rare Conditions and Their Effect on Software Failures", *Proc. of the 1994 Reliability and Maintainability* Symposium, pp. 334-337.
4. Hansen, C. K. "The Status of Reliability Engineering Technology 2001",*Newsletter of the IEEE Reliability Society,* January 2001
5. Reifer, D. J., "Software Failure Modes and Effects Analysis*", IEEE Transactions on Reliability*, vol. 28 no. 3, Aug 79
6. Bowles, J. B. and Chi Wan, " Software Failure Modes and Effects Analysis for a Small Embedded Control System", *Proc. of the 2001 Reliability and Maintainability Symposium,* Philadelphia PA, January 2001, pp. 1 – 6.
7. Goddard, P. L. "Software FMEA Techniques", *Proc. of the 2000 Reliability and Maintainability* Symposium, Los Angeles CA, January 2000, pp. 118 – 122.
8. Boggs, Wendy and Michael, *Mastering UML with Rational Rose*, Sybex, 2002
9. www.ilogix.com/uploadedFiles/RhapsodyBrochure.pdf
10. www.cs.ucsd.edu/matlab/pdf_doc/simulink/sl_using.pdf
11. http://www.aadl.info/
12. Hecht, H., An, X., Hecht, M., Computer Aided Software FMEA for Unified Modeling Language Based Software", *Proc. of the 2004 Reliability and Maintainability Symposium,* pp. 243-248, January 2004.

Biography

Rebecca Menes, Ph.D., Researcher, SoHaR Inc., 5731 West Slauson Ave., Suite 175, Culver City, CA 90230 U.S.A, Telephone - 310-338-0990 ext. 101, facsimile - 310-338-0999, e-mail - becky@sohar.com

Rebecca Menes is a researcher in the areas of vulnerability analysis and modeling, and software safety analysis. Before joining SoHaR in 2002, she spent 4 years as a postdoctoral researcher in the University of California, Santa Barbara, in the Materials Research Lab. She received a PhD. in Physics from the Weizmann Institute of Science, Rehovot, Israel.

Herb Hecht, Ph.D., Chief Engineer, SoHaR Inc., 5731 West Slauson Ave., Suite 175, Culver City, CA 90230 U.S.A, Telephone - 310-338-0990 ext. 101, facsimile - 310-338-0999, e-mail - herb@sohar.com

Herb Hecht is a Golden Core member of the IEEE Computer Society, and a past member of its Board. He received a BEE from CCNY and a Ph. D. in Engineering from UCLA