Sharing the Responsibility for Software Dependability

Herbert Hecht
SoHaR Incorporated
herb@sohar.com

Abstract

Software dependability assurance is often regarded as the responsibility of software professionals because they understand the vulnerabilities of the software and are familiar with tools and methodologies for verification. But a more effective approach is sharing the responsibility with the system engineers. The latter are best 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

The collaboration reduces the scope of software that needs intensive analysis and frequently identifies test conditions that demonstrate absence of critical failure modes. The collaboration is guided by a software failure modes and effects analysis (FMEA), and a computer-aided methodology for generating the FMEA is described. Examples of the FMEA and of the actions taken to remove critical failure modes are presented.

Key Words: Certification, Software FMEA

1, Introduction

In evaluating the ability of a computer program to service critical applications the user – on his own or as represented by a regulatory agency – may require demonstration of its dependability or demand protection against severe failure modes by alarms or fault tolerance. These dependability verification tasks are frequently assigned to software specialists with only very general guidance from the system or project engineering group. As an example, 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[1].

The table of content of the document is reproduced in Figure 1. The next section shows how this approach reduces the collaboration between systems engineering and software development, and how the separation of the two functions makes verification more difficult and less efficient.

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
12. Additional considerations

Figure 1 Content of DO-178B

2. Current practices

The objective of most verification 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) failure modes 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 2. These relations are applicable to hardware as well as to software.

The acceptable 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[2] establishes the ground rules for certification. In the assessment of the probability of failure the AC makes the following

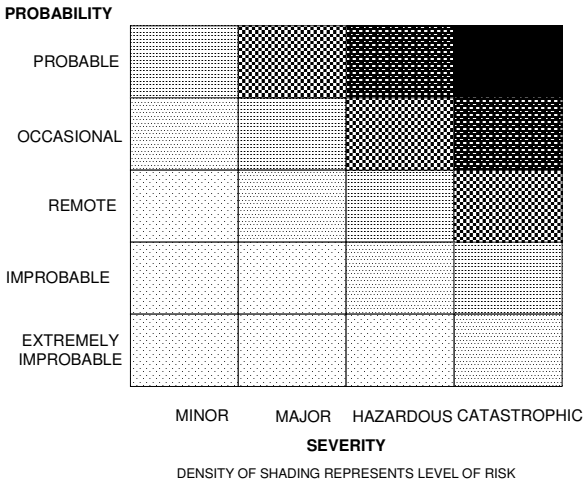

Figure 2 Risk Assessment

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 that we referenced earlier.

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
Catast-rophic	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 Eff.	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 specifically 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.

Two difficulties arise from this system to software translation in terms of risk assessment:

(1) The process controls (Chapters 4 – 9 in Figure 1) 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

(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, e. g., the module that controls elevator deflection or (even higher) the module that controls pitch attitude.

The first of these appears to be accepted by the certifying body – perhaps because nothing much better appears 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 approach to software verification.

At the core of this inefficiency is the very limited interaction between systems and software engineering: the former establishes the worst outcome of a failure at the functional level which then is translated by Table 1 into the required software process controls. This model does not give recognition to the many software failure modes in a functionally critical module that have no or only very mild impact on system safety. The unnecessarily strict controls imposed on these waste resources and, even worse, dilute the effort available for analyzing and testing failure modes that can have the worst system level consequences. In the following heading we present a model of much closer collaboration between the two disciplines.

3. FMEA as the meeting ground

The Failure Mode and Effects Analysis (FMEA) combines the insights of the system engineer and the component designer for hardware and it is one of the recognized analysis methods in the AC (par. 9.c).

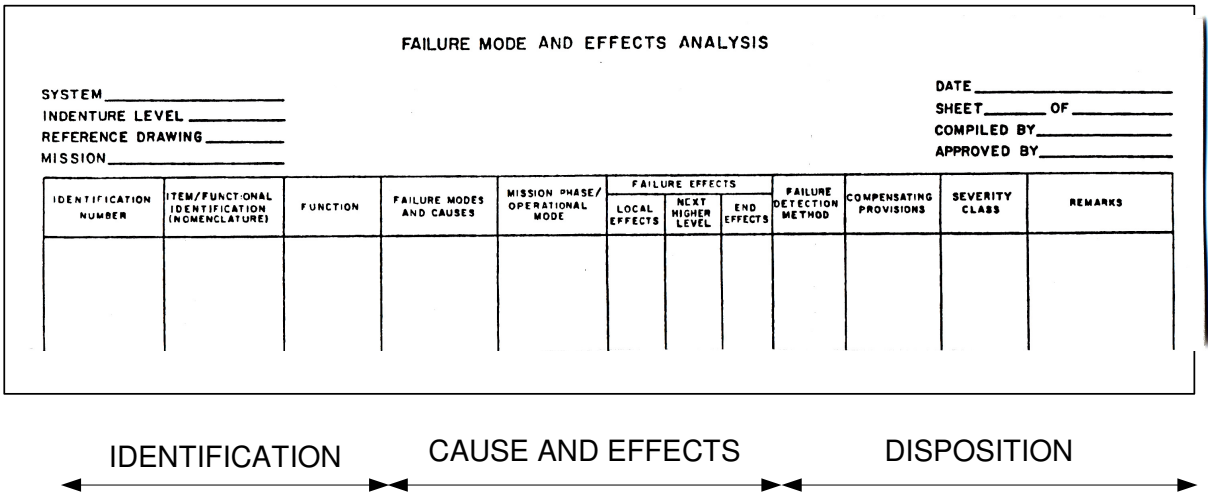

Figure 3 Typical FMEA Worksheet

A typical FMEA worksheet is shown in Figure 3. The first group of columns identifies the item that is being analyzed. This part is usually completed by the hardware or software specialist. 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 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 than the gross identification of critical software segments that had been discussed in the previous section.

In the most rigorous hardware FMEA procedure the identification part 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.

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[3 4 5] 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. "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"[6].

The last cited software FMEA article[5] 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.

4. Computer-aided FMEA

The advent of software development methodologies that build up programs from defined primitives provides the opportunity for a fresh approach to constructing a software FMEA. Two instances of methodologies that are of interest in this context are the Unified Modeling Language (UML)[7] [8] and MATLAB/Simulink[9]. They provide libraries of primitives (and permit the construction of additional ones) that can be equated to hardware parts in that they have predefined operational characteristic and failure modes. Further, the listing of the primitives that have been selected for a software module is analogous to a hardware BOM. This permits the software FMEA to be declared completed when the entire list has been analyzed.

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 the MOVAT tool for UML constructs have been previously described[10]. They show 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. The application of the computer-aided approach to the MATLAB/Simulink environment is described in the following section.

5. An example

The computer-aided generation of a software FMEA for a generic aircraft pitch control system from MATLAB/ Simulink data by means of the MOCET tool is now described with reference to Figure 4. 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.

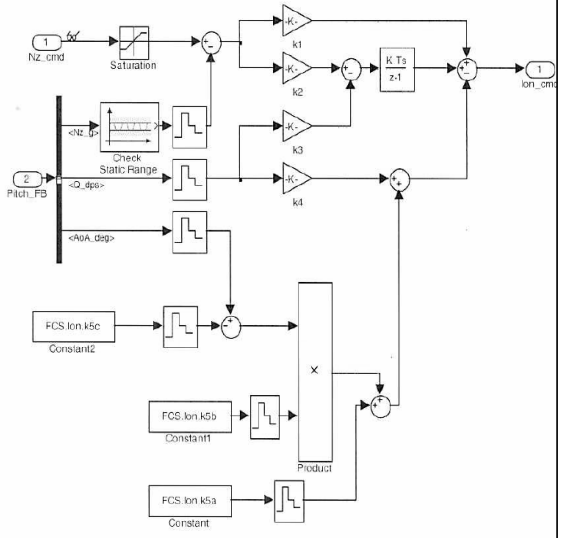

Figure 4 Single Channel Pitch Control

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.

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 BoM, and in its fully implemented form it is a complete listing of the software primitives that make up the longitudinal control system.

```
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 5 Block Structure File (Excerpt)

```
1          longitudinal claw
1.1          Lon
1.1.1          Lon
1.1.1.1            Nz_cmd
1.1.1.2            Pitch_FB
1.1.1.3            Bus\nSelector
1.1.1.3.1            <Nz_g>
1.1.1.3.2            <Q_dps>
1.1.1.3.3            <AoA_deg>
1.1.1.4            Constant
1.1.1.5            Constant1
1.1.1.6            Constant2
1.1.1.7            Discrete-Time\nIntegrator
1.1.1.8            Product
```

Figure 6 Parsed Structure File (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 (1.1.1.1 in Figure 6) 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 in the automatically in

ID	Item/Function	Failure Mode	Local Effect	Detection
1.1.1.1	N_z Command	a. Absent b. Jump c. > Limit	No output Hi rate None (limiter)	N-wait* Chck rate*
1.1.1.2	Pitch FB	See 1.1.1.3		
1.1.1.3	Bus selector	Stuck	No FB	N-wait*
1.1.1.3.1	N_z FB	a. Absent b. Jump c. Xtrm value	No signal Hi rate Hi/lo signal	N-wait Chck rate Chck range
1.1.1.7	Product	a. Absent b. Jump	No signal Hi rate	N-wait* Chck rate*

*not implemented in this version

Figure 7 Partial FMEA Worksheet

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.

However, 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

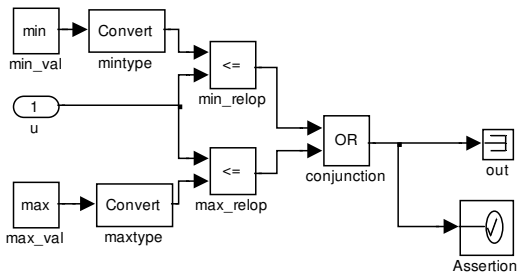

Figure 8 Range Check Implementation

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 an 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.

6. Conclusions

Computer-aided tools, such as MOVAT and MOCET, are available for generation of a 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 verification process (when all objects in the files have been analyzed).

The individual FMEA row entries then become the meeting ground for collaboration on certification issues between system engineers and software professionals. 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.

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 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.

Acknowledgements

Major portions of this research have been sponsored by the Air Force Research Laboratory via a contract with Northrop Grumman under which SoHaR was a subcontractor. The author appreciates the support from David Homan and Raymond Bortner at AFRL and Michael Gomez at NGC.

References

[1] RTCA (formerly Radio Technical Commission on Aeronautics), *Software Considerations in Airborne Systems and Equipment Certification, DO-178B,* December 1992

[2] Federal Aviation Administration, *Advisory Circular, System Design and Analysis,* AC 25.1309-1A. June 1988.

[3] Reifer, D. J., "Software Failure Modes and Effects Analysis*", IEEE Transactions on Reliability*, vol. 28 no. 3, Aug 79

[4] 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.

[5] Goddard, P. L. "Software FMEA Techniques", *Proc. of the 2000 Reliability and Maintainability Symposium,* Los Angeles CA, January 2000, pp. 118 – 122.

[6] C. K. Hansen, "The Status of Reliability Engineering Technology 2001",*Newsletter of the IEEE Reliability Society,* January 2001.

[7] Boggs, Wendy and Michael, *Mastering UML with Rational Rose*, Sybex, 2002

[8] www.ilogix.com/uploadedFiles/RhapsodyBrochure.pdf

[9] www.cs.ucsd.edu/matlab/pdf_doc/simulink/sl_using.pdf

[10] H. Hecht, X. An and M. Hecht, "Computer Aided Software FMEA for Unified Modeling Language Based Software", *Proc. of the 2004 Reliability and Maintainability Symposium,* Los Angeles CA, January 2004, pp. 243-248