MOCET – A Certification Scaffold for Critical Software

Herbert Hecht
SoHaR Incorporated
Culver City CA 90230
herb@sohar.com

1. Introduction

A common problem besetting certification activities, to whatever criteria, has been to find a format that would lend itself to distributing the work to multiple participants and that would define end points to the subtasks as well as to the overall certification effort. It will be shown that software failure mode and effects analysis (FMEA) can provide a structure for partitioning the certification effort and for signaling when it is finished similar to the way in which scaffolds define the outline of a building and provide access to its partitions.

Because a properly completed FMEA is a major step towards acceptance of a hardware item for critical applications there have been a number of suggestions for adapting the originally hardware oriented FMEA format for software but they encountered a significant difficulty: there are no software equivalents to the hardware part or to part type or bill of materials (BOM). The part is the element to which failure modes are assigned, and the part type determines the significant failure modes. The BOM lists the elements that are to be included in the FMEA and when all have been analyzed the FMEA is completed (by convention fasteners and other hardware items and wires are not included in the analysis). As an example, the BOM may list part D8 and identify its part type as power diode. A listing of failure modes[1] (or a failure modes library furnished as part of an FMEA tool) will show the principal failure modes to be high forward resistance, low reverse resistance, open and short. By examining the circuit diagram, the analyst can determine the effect of each of the failure modes on the immediate circuit function and on higher levels of the system hierarchy. Where the higher level effects are critical to the mission or to safety corrective measures need then be taken.

Because there was no analog to the part, the approach in the past has been either to assign failure modes to software functional blocks [2 3] or to consider specific effects due to faulty processing of one input variable at a time[4].

The main disadvantages of the functional approach are that function boundaries are not formally defined (there is no analog to the BOM) and that functions can usually not be assigned to a category that is equivalent to the part type. In previous work we have shown that exception handling is a very frequent cause of failures[5], and not every analyst will consider an exception handling routine as a function. Thus the visibility with regard to exception handling tends to get lost in functional partitioning. Similarly, because the need for exception handling frequently arises from conditions on more than one variable, the other approach has obvious limitations as well.

The advent of object oriented (OO) software development and the availability of comprehensive support tools have changed this picture. The "object" has characteristics similar to those of a part, and the inheritance properties of OO languages can be used to define "part types", classes of objects that share a specific inheritance. This has permitted the computer-aided generation of software FMEAs, making the process much more consistent and also faster. A demonstration of this capability using the Unified Modeling Language (UML) has been previously described[6]. For avionics applications a similar methodology based on the MATLAB-Simulink development tool suite has been found to be particularly easy to apply because of the built-in library of model elements (part types). Details of this approach are presented in the rest of this paper.

2. Representations in MATLAB-Simulink

As an example of generating the equivalent of a BOM from a MATLAB-Simulink file we use a basic aircraft longitudinal (pitch axis) control system. The primary input is the normal acceleration command (Nz_cmd) shown in the upper left corner of Figure 1. The output is the elevator or elevon command (lon_cmd). A from input to output is modified by three pitch feedback signals (input 1 on the left). The most important of these is the achieved normal acceleration (Nz_g). Neglecting secondary terms that are discussed later, when the achieved acceleration is equal to the commanded one the elevator command should be at neutral.

Because extreme acceleration commands can lead to unsafe conditions, the magnitude of the primary input is limited by the saturation element. Also, when the achieved acceleration exceeds a limit (usually selected higher than the command limit) this indicates a possible failure of the control system. This condition is sensed by the check-static-range element.

The other two feedback terms are intended to prevent excessive dynamic pressure (Q_dps) or angle of attack (AoA_deg). Also, to reduce standing errors an integrator has been incorporated in the loop (the square box preceding the output summing junction). Other icons are sample-and-hold elements (stairstep symbol), gain elements (triangles) and constants (rectangles). Each of these icon types is the equivalent of a part type in a hardware BOM, and each individual icon is a part, the failure modes of which are determined by the icon type.

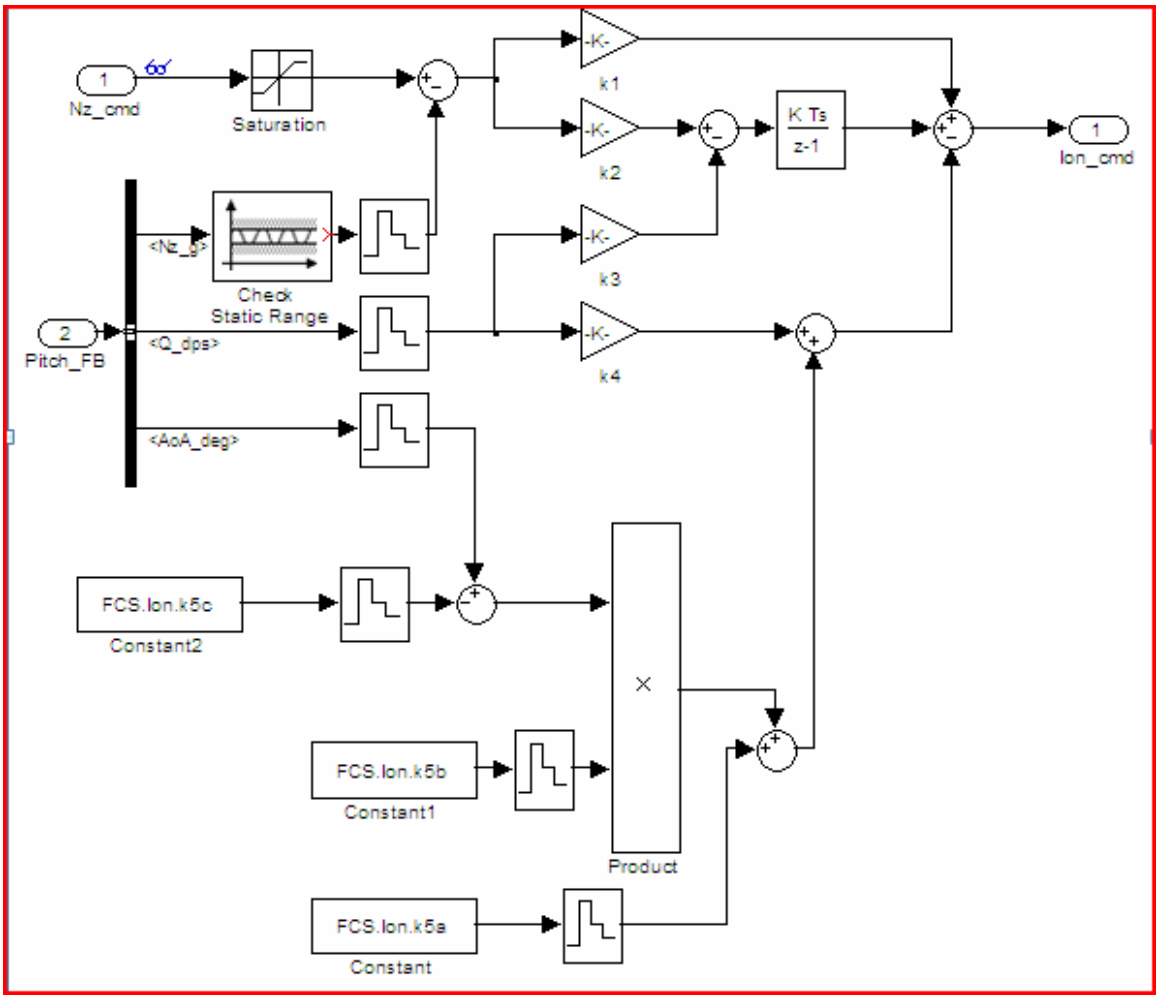


Figure 1 Objects in longitudinal control system

3. Initial Steps for Software FMEA Generation

To generate the initial portion of the FMEA (identification of parts and their failure modes) the following steps are necessary:

1. Extract a listing of all icons from the Simulink file
2. Attach hierarchical identifiers to significant icons (objects)
3. Generate a library of failure modes for each icon type
4. Attach the appropriate failure modes to each listed icon

The implementation of these steps is discussed below..

The control system representation shown in Figure 1 above is generated from a block structured text file that (among other items) lists each instance of a given icon type and the location and legend attached to that item. Thus each icon in Figure 1 can be associated with a block in that file. A partial view of the text file is shown in Figure 2.

```
Block {
  BlockType                Constant
  Name                     "Constant1"
  Position                 [155, 496, 240, 524]
  Value                    "FCS.lon.k5b"
}
Block {
  BlockType                Constant
  Name                     "Constant2"
  Position                 [35, 411, 120, 439]
  Value                    "FCS.lon.k5c"
}
Block {
  BlockType                DiscreteIntegrator
  Name                     "Discrete-Time\nIntegrator"
  Ports                    [1, 1]
  Position                 [395, 160, 430, 200]
  ShowName                 off
  IntegratorMethod         "Forward Euler"
  ExternalReset            "none"
  InitialConditionSource   "internal"
  SampleTime               "FCS.T_Samp"
}
Block {
  BlockType                Product
  Name                     "Product"
  Ports                    [2, 1]
  Position                 [310, 383, 345, 552]
  InputSameDT              off
}
```

Figure 2 Excerpt of text file for Figure 1

To convert this to a format useful for the FMEA it is necessary to eliminate extraneous information and to attach identifiers to each block. The blocks become then become row entries in an FMEA worksheet. The MOCET program automates this and produces a screen shown in Figure 3.

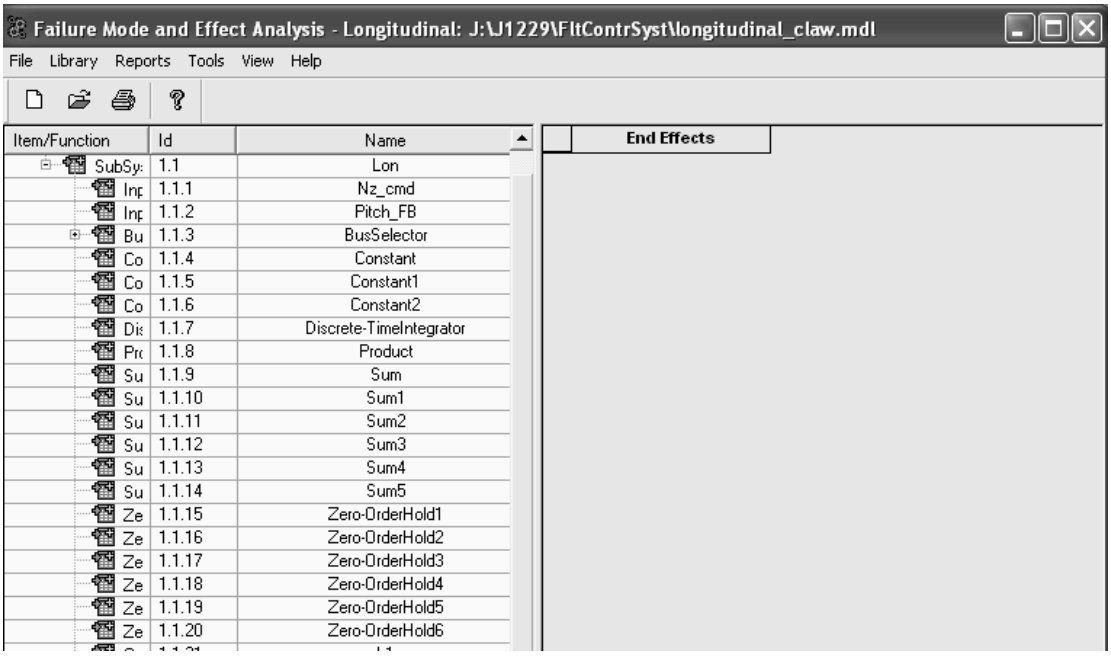


Figure 3 Initial worksheet entries

While FMEA is generally regarded as a bottom-up analysis it has been found helpful to start the process with the definition of critical top level failure effects -- the effects that should be avoided by the later steps in the analysis. Defining these before the detailed analysis begins provides a needed focus and avoids uncertainty in the designation of these effects, e. g., is it “Loss of longitudinal control” or “Failure of longitudinal control”? Once the effects are defined, the analyst will be able to select them from a menu and thus avoid ambiguities.

To call attention of the user to the need for defining end effects, a legend End Effects appears on the right half of the operating area of the screen (Figure 3). To enter the end effects, the user clicks on the Library label in the top task bar, and after selecting Edit can create the screen overlay shown in Figure 4. This is one of the important data entry screens for MOCET. The entries on this screen are preferably generated by (or in consultation with) systems engineering personnel.

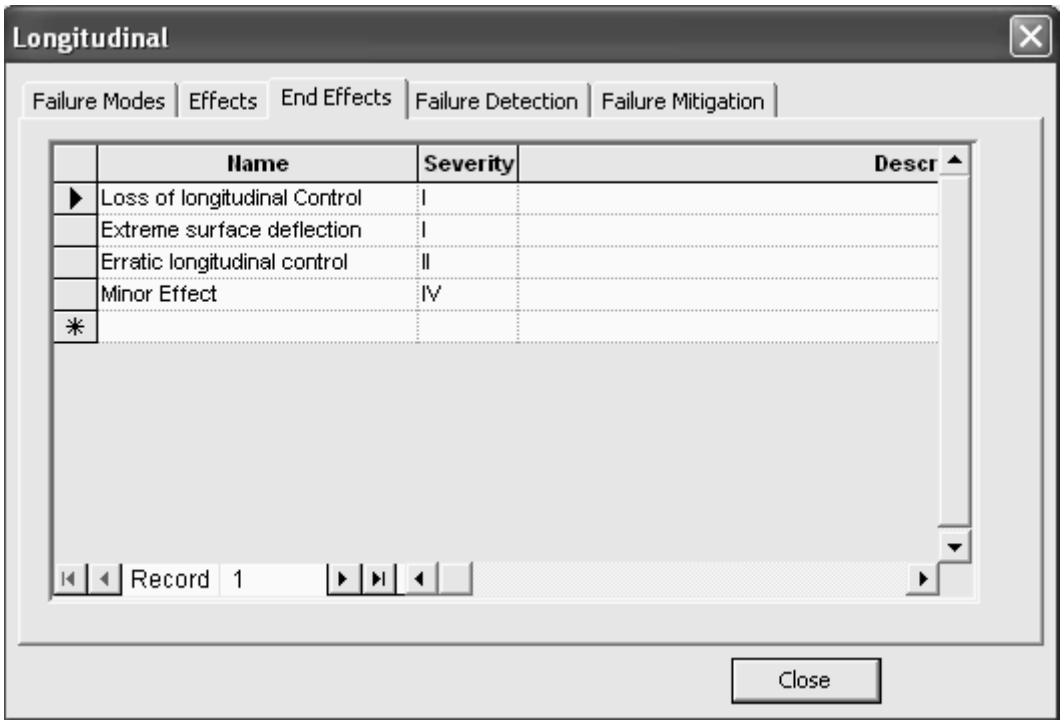


Figure 4 End effects entries

One of the features of this approach is that it promotes collaboration between those whose primary concern is software and those who are responsible for the performance and safety of the system. This assures that all hazards at the system and subsystem level are recognized and focuses the software developer’s and certifier’s attention on the critical areas as shown in Figure 5.

The next step is to generate the failure mode library. The mechanics of this are identical with the drop-down screen format used for end effects but the entries are the responsibility of the software specialists. For practically all objects it may be assumed

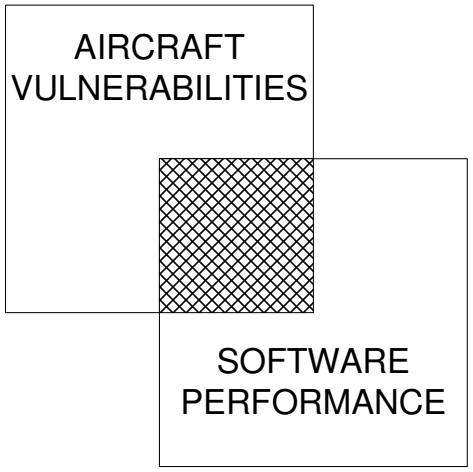


Figure 5 Focusing on the critical area

that they can fail to furnish output; this is designated as the "absent" failure mode. Also, in most cases there may be a possibility that output is not updated correctly and this is referred to as the "stuck" failure mode. In selected cases an output that exceeds a limit (">limit") or a high rate of change in the output ("jump") may lead to undesirable systems effects, and in others a prolonged dwell at zero output (a specialized "stuck") may need to be signaled. Where the object performs a switching function, "failure to switch" when called for and switching when not called for ("spurious switch") are commonly used. Additional failure modes can be added to the library.

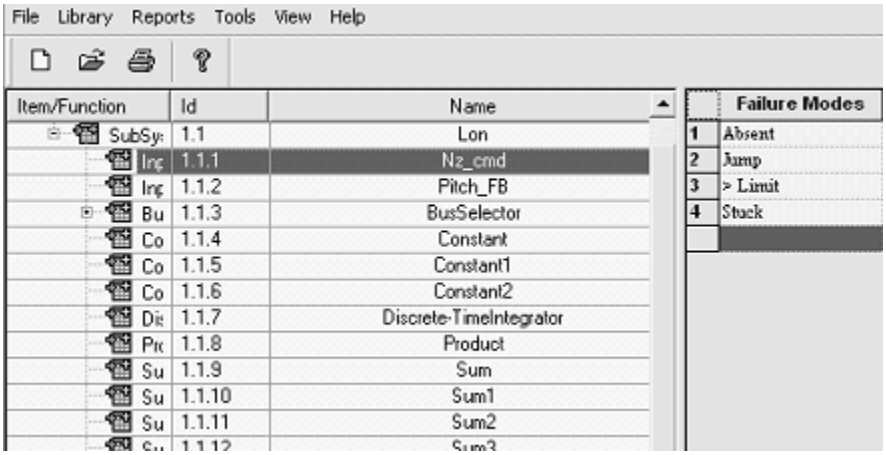


Figure 6 Failure mode entry screen

As shown in Figure 6 the analyst selects the appropriate failure modes from a menu and these are then attached to the highlighted object. In all cases the selection of failure modes is governed by both the software characteristics and the potential effects on the overall system. Typical object oriented development tools include automatic code generators. The output of these can be examined and exercised to determine conditions that can give rise to a failure mode that is of concern.

4. Completing the FMEA

A frequently used format for hardware FMEA worksheets is shown in Figure 7[7]. The identification section has been adapted for MOCET as shown above (Figure 6). The mission phase column is omitted because it is more instructive to generate a separate FMEA report for each mission phase. The three failure effects columns shown in Figure

7 have been replaced by a flexible format that allows fewer or more than the one "next higher level" shown in the figure. Detection and mitigation ("compensating provsions") columns have been retained and the remarks column is optional.

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			SEVERITY CLASS	COMPENSATING PROVISIONS	FAILURE DETECTION METHOD	REMARKS
					LOCAL EFFECTS	NEXT HIGHER LEVEL	END EFFECTS				

IDENTIFICATION CAUSES AND EFFECTS DISPOSITION

Figure 7 Hardware FMEA worksheet

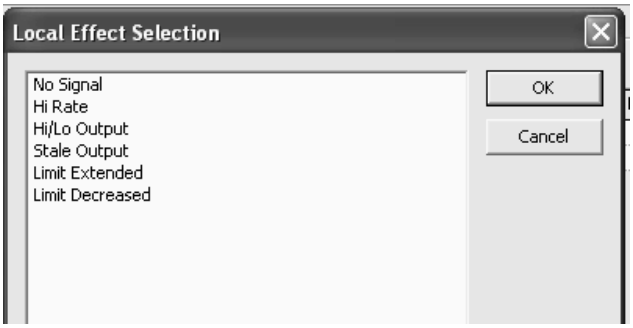


Figure 8 Typical local effects

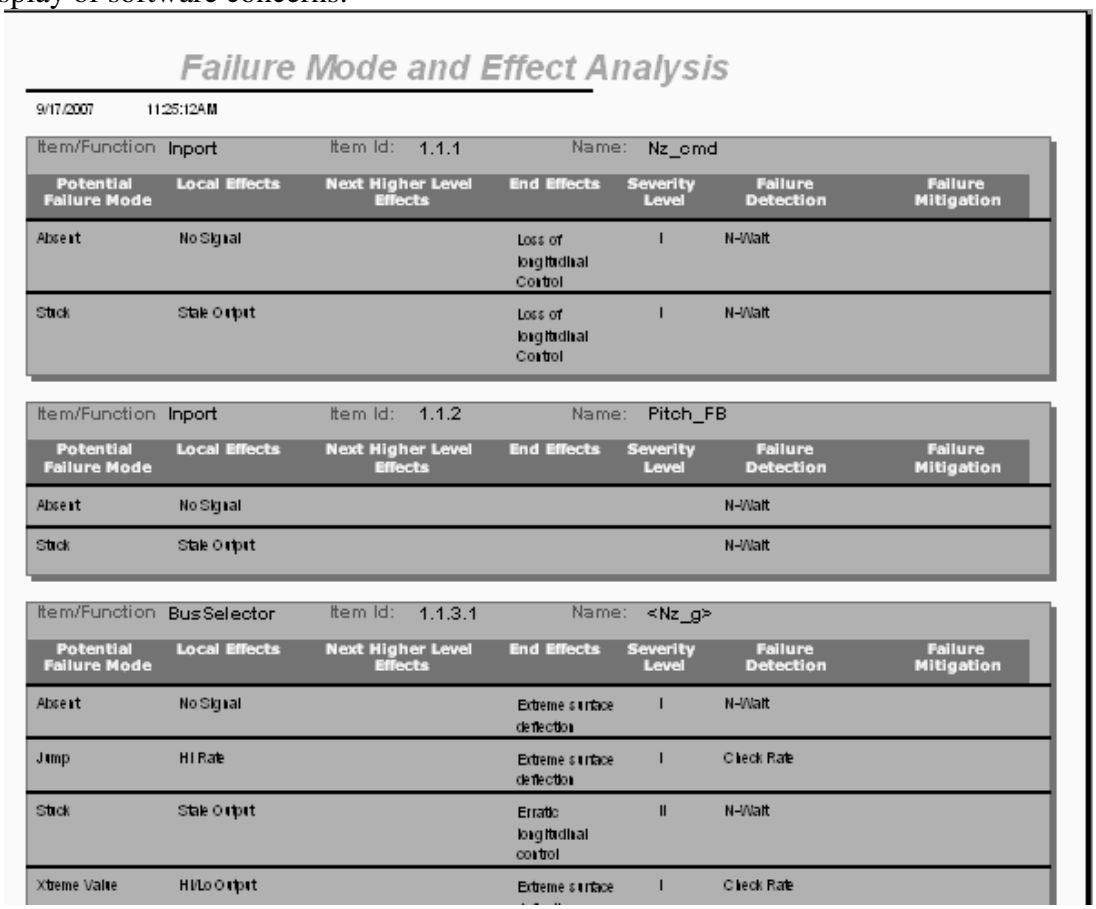
Local effects are those that will be observed in the output of the immediate software function and are best evaluated by the analyst or other software expert. Usually they are closely related to the failure mode. Examples of local effects for the longitudinal control system are shown in Figure 8. The selection of system level effects has already been discussed in connection with Figure 4. The requirement for intermediate levels is determined by the available detection and mitigation provisions. For the longitudinal control system two levels may be of interest. If high pitch rates are detected at a higher level in the software hierarchy, then this will be a suitable "next higher level". Similarly, if flight path deviations are monitored at a still higher level, this may be a good place to conduct additional analysis. If these detection provisions do not exist, then the insertion of intermediate levels is of little benefit.

The generic longitudinal control system shown in Figure 1 has only one effective measure for failure detection (the "check static range" object) and no mitigation provisions. In a practical system, many more detection provisions will be provided. The need for and depth of the mitigation measures depends on the severity of the end effects. Where loss or injury to personnel or severe aircraft damage may result, independently coded software (sometimes combined with redundant hardware) is frequently called for. Other mitigation measures are repetition of a software operation, restart of a program, substitution of a default output or switching to a highly degraded mode for accomplishing

the function that failed. The use of repetition or restart for time critical operations must of course be evaluated in detail.

5. The FMEA Report and its Evaluation

The "native" format of reports generated by MOCET is shown in Figure 9. It is functionally equivalent to the hardware FMEA worksheet (Figure 7) but allows for better display of software concerns.



Failure Mode and Effect Analysis

9/17/2007 11:25:12AM

Item/Function Inport Item Id: 1.1.1 Name: Nz_cmd

Potential Failure Mode	Local Effects	Next Higher Level Effects	End Effects	Severity Level	Failure Detection	Failure Mitigation
Absent	No Signal		Loss of longitudinal Control	I	N-Wait	
Stuck	Stale Output		Loss of longitudinal Control	I	N-Wait	

Item/Function Inport Item Id: 1.1.2 Name: Pitch_FB

Potential Failure Mode	Local Effects	Next Higher Level Effects	End Effects	Severity Level	Failure Detection	Failure Mitigation
Absent	No Signal				N-Wait	
Stuck	Stale Output				N-Wait	

Item/Function BusSelector Item Id: 1.1.3.1 Name: <Nz_g>

Potential Failure Mode	Local Effects	Next Higher Level Effects	End Effects	Severity Level	Failure Detection	Failure Mitigation
Absent	No Signal		Extreme surface deflection	I	N-Wait	
Jump	HI Rate		Extreme surface deflection	I	Check Rate	
Stuck	Stale Output		Erratic longitudinal control	II	N-Wait	
Xtreme Value	HI/Lo Output		Extreme surface	I	Check Rate	

Figure 9 Native report format

Reports can be exported in a number of widely used formats, including MS Word, Excel and pdf.

The evaluation of an FMEA falls into three categories:

- Correctness and completeness of failure modes and low level effects
- Correctness of the propagation to high level failure effects
- Adequacy of detection and mitigation (D&M) provisions and acceptance of residual risk

The first of these is best carried out by software professionals who are experienced in the vulnerabilities of the language and of the model constructs. The second assessment

category is usually assigned to system engineers because of their familiarity with system vulnerabilities such as effects of a delayed output. The third category is primarily the responsibility of project management but may require participation of system engineers in the evaluation of specific D&M provisions.

Absent from this assessment is the evaluation of the completeness of the FMEA as a whole because the transcription of the Simulink model by MOCET into the ID structure of the FMEA removes these traditional concerns. The format of the FMEA worksheet generated by MOCET and the library of pull-down entries also help in facilitating the assessment and ultimately the certification of the overall system.

The assessment described above is directed at meeting safety and availability requirements. It does not by itself assure that these requirements are being met in an optimum manner, i. e., with the minimum resources. The trade-offs for optimization involve moving D&M from lower levels to higher levels in the system hierarchy and shifting the allocation of mitigation between hardware and software functions. The investigation of these alternatives can be automated with MOCET and Simulink, and this should be a fertile area for further research.

References

[1] FMD-97, RIAC, Utica NY, February 1998

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

[3] 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.

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

[5] Hecht, Herbert and Patrick Crane, "Rare Conditions and their Effect on Software Failures", *Proceedings of the 1994 Reliability and Maintainability Symposium,* pp. 334 - 337, January 1994

[6] Hecht, Herbert, Xuegao An and Myron Hecht, "Computer-Aided Software FMEA for Unified Modeling Language Based Software", Reliability and Maintainability Symposium (RAMS), Los Angeles, January 2004

[7] MIL-STD-1629A, Department of Defense, November 1980 (no longer active but still frequently referenced)