Software FMEA Techniques

Peter L. Goddard • Raytheon • Troy

Key Words: Software FMEA, FMEA, Software Safety, Software Reliability, Software Failure Modes

SUMMARY AND CONCLUSIONS

Assessing the safety characteristics of software driven safety critical systems is problematic. Methods to allow assessment of the behavior of processing systems have appeared in the literature, but provide incomplete system safety evaluation. Assessing the safety characteristics of small embedded processing platforms performing control functions has been particularly difficult. The use of fault tolerant, diverse, processing platforms has been one approach taken to compensate for the lack of assurance of safe operation of single embedded processing platforms. This approach raises cost and, in at least some cases where a safe state can be demonstrated, is unnecessary. Over the past decade, the author has performed software FMEA on embedded automotive platforms for brakes, throttle, and steering with promising results. Use of software FMEA at a system and a detailed level has allowed visibility of software and hardware architectural approaches which assure safety of operation while minimizing the cost of safety critical embedded processor designs.

Software FMEA has been referred to in the technical literature for more than fifteen years. Additionally, software FMEA has been recommended for evaluating critical systems in some standards, notably draft IEC 61508. Software FMEA is also provided for in the current drafts of SAE ARP 5580. However, techniques for applying software FMEA to systems during their design have been largely missing from the literature. Software FMEA has been applied to the assessment of safety critical real-time control systems embedded in military and automotive products over the last decade. The paper is a follow on to and provides significant expansion to the software FMEA techniques originally described in the 1993 RAMS paper "Validating The Safety Of Real-Time Control Systems Using FMEA".

1. INTRODUCTION

Failure Modes and Effects Analysis, FMEA, is a traditional reliability and safety analysis techniques which has enjoyed extensive application to diverse products over several decades. Application of FMEA to software has been somewhat problematic and is less common than hardware and system FMEAs. Software FMEA has appeared in the literature as early as 1983. However, the number of papers dedicated to software FMEA has remained small and the number of those which provide descriptions of the exact methodology to be employed have been few. This paper provides a summary overview of two types of software FMEA which have been used in the assessment of embedded control systems for the past decade: system software FMEA and detailed software FMEA. The techniques discussed are an expansion and refinement of those presented in reference 1. System level software FMEA, which was not discussed in reference 1, can be used to evaluate the effectiveness of the software architecture in ensuring safe operation without the large labor requirements of detailed software FMEA analysis. The FMEA techniques described in this paper are consistent with the recommendations of SAE ARP 5580, reference 2.

2. SOFTWARE FMEA

2.1 Software FMEA Application

Software FMEA can be applied to diverse system designs, allowing the analysis to identify potential design weaknesses and allowing design improvements to be recommended. System level software FMEAs can be performed early in the software design process, allowing safety assessment of the chosen software architecture at a time when changes to the software architecture can be made cost effectively. System level software FMEA is based on the top level software design: the functional partitioning of the software design into CSCIs, CSCs, and modules. Detailed software FMEA is applied late in the design process, once at least pseudo code for the software modules is available. Detailed software FMEA is used to verify that the protection which was intended in the top level design and assessed using system level software FMEA has been achieved. Both system and detailed software FMEAs evaluate the effectiveness of the designed in software protections in preventing hazardous system behavior under conditions of failure. Software failure can be the result of errors in software design being expressed due to the specific environmental exposure of the software or of transient or permanent hardware failures. The exact cause of the failure is comparatively unimportant to the analysis results. Software FMEA assesses the ability of the system design, as expressed through its software design, to react in a predictable manner to ensure system safety.

The techniques of system and detailed software FMEA have been used extensively on embedded control systems. Specific applications have included braking, throttle, and steering for automotive applications. Each of these systems has the potential for safety critical failures occurrences. These systems have also had defined safe states which the control system was driven to in cases of failures. However, application of software FMEA techniques, particularly system level software FMEA techniques, does not appear to be limited to systems with safe states. The methodology can be applied to redundant systems to assess the ability of the software and hardware to achieve a known state under conditions of hardware and software failure, allowing redundant elements to effect system recovery. Detailed FMEA may also be required for fault tolerant control processing depending on the hardware protection provided.

2.2 Architectural Considerations

The software FMEA techniques described in the remainder of this paper were developed in response to a need to validate hardware and software designs for embedded control platforms. These embedded control platforms have several unique characteristics which help make software FMEA a valued technique for assessing effectiveness of their safety design.

A typical, and much simplified, hardware architecture for an embedded control system is shown in Figure 1, below. The basic hardware architecture provides for input from a variety of sensors and output of control signals to various control elements such as motors, valves, etc. In modern embedded control systems, the physical hardware is often simplified through the use of highly integrated controllers which include a microprocessor, A to D and D to A conversion capability, multiplexing, and specialized control and communications circuitry on board a single integrated circuit. This can result in the peripheral circuits being limited to those needed to buffer incoming signals to protect the microcontroller and amplifying and providing current sources for output control signals. These highly integrated microcontroller integrated circuits typically have minimal or no memory, internal communications, or processor integrity protection. Thus, analysis methods which assess hardware and software failure effects must include the effects of memory, processing integrity, and communications failures.

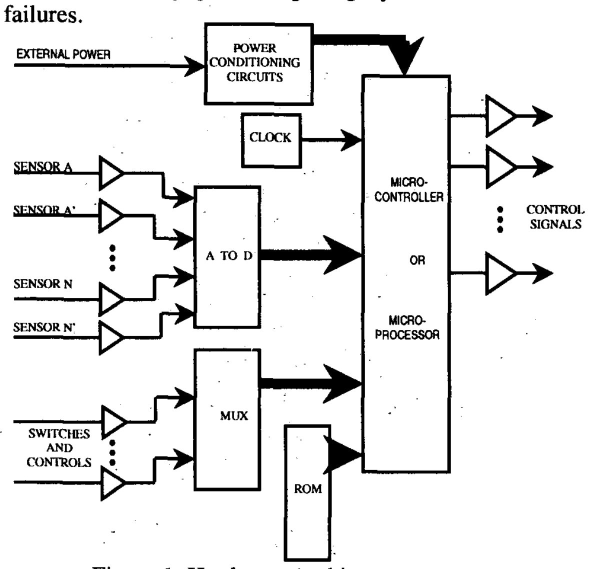

Figure 1. Hardware Architecture.

As shown in the non-italicized pseudo code of Figure 2, embedded control system software follows a straightforward architecture: read sensors, calculate control values, output control signals to actuators. The read-calculate-output loop is repeated endlessly for the control being exercised. Failures of the software or the supporting hardware can result in either incorrect control values, the result of which is detected by the system user, or no system output due to a sufficiently incorrect fault response (e.g. execute no-ops to the end of memory). For safety critical systems, the response of the system to plausible hardware and software failures must be able to be determined prior to failure occurrence. The design must leave the system in as safe a state as is plausible given the occurrence of failure. The requirement for deterministic behavior under failure conditions results in a software architecture which more closely approximates the complete pseudo code of Figure 2: perform self checks, read sensors, validate sensor values, calculate control values, validate control values, validate output hardware condition, enable hardware outputs if output hardware correct, output control to actuators if all checks pass else return to safe state. The technique of continually validating the correctness of the supporting hardware, along with checks to ensure that software has executed the expected routines in the correct order is the minimum necessary for embedded safety critical control systems. Additionally, functional redundancy, implemented in the software through the use of diverse control calculation algorithms and variables is sometimes needed.

```
Program Control
 begin
  sys_valid:=test_all_control_hw();
  initialize;
  done:=false;
  while ((not done) and sys_valid)
   begin
     read_sensors();
     sys_valid:= sys_valid and validate_sensor_values();
     calculate_control_values();
     sys_valid:=sys_valid and validate_control_values();
     sys_valid:=sys_valid and validate_output_hardware();
     if (sys_valid)
       enable_output_hardware();
       output_control_signals();
     sys_valid:=sys_valid and test_critical_hardware();
   end;
   set_system_to_safe_state();
 end.
```

Figure 2. Control System Software Architecture

2.3 Software Hazard Analysis

Unlike hardware and system FMEAs, a software FMEA cannot easily be used to identify system level hazards. Since software is a logical construct, instead of a physical entity, hazards must be identified and translated into software terms prior to the analysis. Prior to beginning the development of a software FMEA, a system preliminary hazard analysis (PHA) for the system should exist. The PHA needs to include all the hazards which can have software as a potential cause. The first step in developing a software FMEA is to translate potential system hazards with possible software causes into an equivalent set of system and software states through the process of software hazard analysis. To perform a software hazard analysis, the analyst begins with each hazard identified in the PHA and performs a fault tree analysis of the potential causes of the hazard. For each potential hazard and potential hazard cause which could be the result of software failures, the analyst must extend the fault trees through the system hardware and software until a sensible set of software input and output variable values is identified. The value set associated with each hazard cause is then identified as a software hazard. Figure 3 shows the form of the output table which results from the software hazard analysis and which is used to determine the criticality of the result of any software failures.

		Critical Software Variables			
		Variable 1	Variable 2	…	Variable n
Hazard 1	Cause 1	Value	Value	…	Value
	Cause 2	Value	Value	…	Value
	⋮	⋮	⋮	…	⋮
	Cause n	Value	Value	…	Value
Hazard 2	Cause 1	Value	Value	…	Value
	Cause 2	Value	Value	…	Value
	⋮	⋮	⋮	…	⋮
	Cause n	Value	Value	…	Value
⋮	⋮	⋮	⋮	…	⋮
Hazard n	Cause 1	Value	Value	…	Value
	Cause 2	Value	Value	…	Value
	⋮	⋮	⋮	…	⋮
	Cause n	Value	Value	…	Value

Figure 3. Software Hazard Analysis Results

2.4 Software Safety Requirements

One of the crucial elements of any safety program for a software intensive system is the development of software requirements to guide the design team in the creation of a software architecture and implementation which includes all the features needed to support safety critical processing. The existence and understanding of these requirements by both the safety and software design groups is crucial to achieving a system design which is adequate for the intended application and to allow the software design group to understand the results of and recommendations from the software FMEA. Safety requirements, appropriate for critical software, can be found in several published sources (Refs. 3-8). A compendium of requirements selected from these sources and tailored for the specific application should be released early in the software design process, ideally prior to the start of top level software design. Discussions of FMEA findings can then be organized to relate to achievement of the previously identified requirements, significantly simplifying the communications process between safety and software engineering.

In addition to requirements imposed directly on the software design, safety requirements will need to be imposed on the software development and execution environments and on development tools. The safety analyst needs to ensure that requirements are imposed which ensure that the behavior of the software is consistent with that expected by the software developer and the analyst. One of the critical elements of the software design which needs to be controlled is the language which is used for software development and the compiler for that language. Compilers which have been carefully tested to the language specification and certified for accuracy of the compiled code must be used in the development of safety critical software if analysis based on the high order language listings for the compiled code is to have validity. Use of the language itself also needs to be limited to those features which are fully defined by the language specifications. Elements of a language whose behavior has been left to the compiler designer to decide should be avoided. A good discussion of the needed controls for the language 'C' can be found in reference 9. The software safety requirements must also specify that indeterminate behavior of the compiler be avoided. Features such as optimization, which can produce indeterminate results in the final object code, must be specified as being disabled. Any operating system or scheduler intended for use with safety critical software also needs to be carefully selected. The executive functions provided by the operating system or scheduler can significantly impact the ability of the developed software to provide the intended level of safety. Requirements which specify the use of a safety certified executive as a part of the software are appropriate if a software FMEA is to have validity.

2.5 System Software FMEA

System software FMEA should be performed as early in the design process as possible to minimize the impact of design recommendations resulting from the analysis. The analysis may need to be updated periodically as the top level software design progresses, with the final system software FMEA update occurring during detailed design, in parallel with the detailed software FMEA. The organization performing the system level software FMEA needs to balance the update periodicity and expected benefits with the associated costs. Labor costs for system level software FMEAs are modest and allow identification of software improvements during a cost effective part of the design process.

Once the software design team has developed an initial architecture and has allocated functional requirements to the software elements of the design, a system software FMEA can be performed. The intent of the analysis is to assess the ability of the software architecture to provide protection from the effects of software and hardware failures. The software elements are treated as black boxes which contain unknown software code, but which implement the requirements assigned to the element. The failure modes which are used to assess the protection provided by each software element are shown in Figure 4. The failure modes to be applied to each software element include: failure of the software element to execute, incomplete execution of the software element, incorrect functional result produced, and incorrect execution timing. Additional 'black box' failure modes may need to be added which are specific to the intended software application. Failure of the software to execute and incomplete execution are particularly important to real time systems. The potential for 'aging' of data in real time control systems must be carefully evaluated. In addition to the failure modes for each software element, the analyst must evaluate the ability of the software design to protect against system failures in hardware and software. As shown in Figure 4, the system level software failure modes evaluate the ability of the system to provide protection against incorrect interrupt related behavior, resource conflicts, and errors in the input sensor and output control circuits.

Element Failure Modes	Fails to execute
	Executes incompletely
	Output incorrect
	Incorrect timing – too early, too late, slow, etc
System Failure Modes	Input value incorrect (logically complete set)
	Output value corrupted (logically complete set)
	Blocked interrupt
	Incorrect interrupt return (priority, failure to return)
	Priority errors
	Resource conflict (logically complete set)

Figure 4. System Level Software Failure Modes

To perform the system level software analysis, the analyst assesses the effect of the four primary and any appropriate additional failure modes for each element on the software. The effect on the software outputs of the failure mode is then compared to the previously performed software hazard analysis to identify potentially hazardous outcomes. If hazardous software failure events are identified, the analyst then needs to identify the previously defined software safety requirement which has not be adequately implemented in the design. If the potentially hazardous failure mode cannot be traced to an existing requirement, the analyst needs to develop additional software requirements which mandate the needed protection. In addition to the failure modes for each software element, the analyst assesses the effect of each of the system level software failure modes on the software outputs and compares the effects against the software hazards and software safety requirements.

The system level software FMEA should be documented in a tabular format similar to that used for hardware FMEAs. Tabular FMEA documentation techniques are well developed in most organizations and familiar to the design engineering staff. Tabular documentation techniques also allow extensive, free form, commentary to be provided as a part of the failure effect documentation. The ability to provide extended commentary on the software design and design requirements is crucial to allowing software engineers to understand the FMEA results and the needed design changes. In many organizations, software engineers can only respond effectively to requirements based presentation of results.

2.6 Detailed Software FMEA

Detailed software FMEA is used to validate that the as implemented software design achieves the safety requirements which have been specified for the design, providing all needed system protection. Detailed software FMEA is similar to component level hardware FMEA. The analysis is lengthy and labor intensive. The results are not available until late in the design process. Thus, detailed software FMEAs are mostly appropriate for critical systems with minimal or no hardware protection of memory, processing results, or communications. For large systems with hardware provided protection against memory, bus, and processing errors detailed software FMEA may be difficult to economically justify.

Detailed software FMEA requires that a software design and an expression of that design in at least pseudo code exist. Implicit in this requirement is the existence of software requirements documentation, top level design descriptions, and detailed design descriptions. Final implemented code may not be necessary if the software elements are described in pseudo code and the software development process provides adequate assurance that the implemented design matches the pseudo code description of the detailed design documentation. To perform the analysis, the analyst postulates failure modes for each variable and each algorithm implemented in each software element. The analyst then traces the effect of the postulated failure through the code and to the output signals. The resultant software state is then compared to the defined software hazards to allow identification of potentially hazardous failures.

If the software hazard analysis has previously been completed to support system level software FMEA, the first step in the detailed software FMEA is development of a variable mapping. The analyst will need to develop, or have produced by automated software development tools, a mapping which shows which variables are used by each software module and whether the variable is an input variable, an output variable, a local variable, or a global variable. As a part of the variable mapping, the analyst needs to clearly identify the source of each input variable and the destination(s) of each output variable. This mapping will be used to allow the analyst to trace postulated failures from the originating location to the output variable set.

Once the variable map is complete, the analyst should develop software 'threads' for the processing being analyzed. The software threads are mappings from an input set of variable through the various processing stages to the system output variables. The software threads will assist the analyst in rapidly tracing postulated failures to system variables and effects. Definition of the software 'threads' will often be available from the software design team through existing design documentation or as a defined output of the automated design tools being used by the design team.

To perform the detailed software FMEA, the analyst next needs to develop failure modes for the processing algorithms as they are implemented in each module. The algorithm failure modes are unique to each software development. A logically complete set of failure modes for each of the variable types also needs to be developed. Reference 1 provides a description of the straightforward process used to develop variable failure modes for simple variable types: Boolean, enumerated, real, integer. Development of a logically complete set of variable failure modes for more complex variables will need to be done based on the specifics of the language in use and the compiler implementation. Since the primary purpose of postulating failure of each variable is to assess the impact of memory failures in processing platforms which do not have effective memory protection, a detailed knowledge of the underlying storage scheme is required. For high order languages, it may be necessary to obtain the needed implementation details from the developer of the compiler and from the language specification.

Once the variable and algorithm failure modes have been developed, the analyst can perform the detailed software FMEA. For each module, algorithm failures are postulated, the effect traced to the module outputs and in turn to the software system output variables using the software threads and the variable map. The system variable effects are then compared against the software hazard analysis to determine whether or not the postulated failure could lead to a system hazard. The analyst then postulates failures for each of the variables used in the module and traces the effects to the system outputs and the defined software hazards in a similar manner. The detailed software FMEA process is analogous to the component level hardware FMEA process except that variables and the variable map substitute for the signals and signal paths of electronic hardware.

If the detailed FMEA identifies failure modes which trace to the defined software hazards, the analyst needs to assess which software safety requirements have not been implemented correctly or if one or more requirements are missing. Similar to system level software FMEA, the most effective way to communicate software design deficiencies is through identification of those requirements which have not been met.

Documentation of the detailed software FMEA can be either tabular or using the matrix documentation recommended in reference 1. Matrix documentation provides some desirable compactness for detailed software FMEA. However, tabular documentation is more familiar to most design groups and allows extensive commentary to be included. The choice of documentation style can be left to the preference of the individual analyst or analysis team.

2.7 Analysis Limitations

Software FMEA can provide insight into the behavior of safety critical software intensive systems, particularly embedded control systems. However, as with all FMEAs, the analysis cannot provide complete system safety certification. Software FMEA examines the behavior of the system being analyzed under conditions of software single point failure. In many cases, the assumption of single point failures may be difficult to fully justify. Many software failures can be induced by failures in the underlying hardware. For systems with minimal memory protection, failures in the memory hardware can appear as errors in variable storage values which can propagate errors through the software into the output variables and subsequently to system behavior. Single point memory failure assumptions can be appropriate for processing memory which has been carefully architected to preclude multiple errors, but may not be safe to generally assume unless the implementation of the storage is known. The implementation details for memory circuitry for highly integrated microprocessors and microcontrollers is likely to be proprietary to the device manufacturer and unknown to the analyst.

Software FMEA does not provide evaluation of the behavior of a software intensive system under conditions of unfailed operation. For many control systems, the stability of the control loop is a crucial parameter in determining safety of operation. Simulation and modeling are appropriate tools for evaluating control stability. FMEA cannot provide the needed evaluation of control loop stability under either normal or failed operation. Similarly, software FMEA provides limited insight into the safety risks associated with changes in timing due to either software or hardware failures. Timing and sizing analysis for worst case interrupt arrivals and resource demands may be needed to provide insight into the effects of some failures postulated during the software FMEA.

3. CONCLUSIONS

Software FMEA has been applied to a series of both military and automotive embedded control systems with positive results. Potential hazards have been uncovered which were not able to be identified by any other analytical approach, allowing design corrections to be implemented. Additionally, system level software FMEA can be applied early in the design process, allowing cost effective design corrections to be developed. System software FMEA appears to be valuable for both small embedded systems and large software designs, and should be cost effective so long as a mature software design process - one which can provide needed software design information in a timely manner - is in use. Detailed software FMEA is appropriate for systems with limited hardware integrity, but may not be cost effective for systems with adequate hardware protections. For designs with limited hardware integrity, detailed software FMEA provides an effective analysis tool for verifying the integrity of the software safety design.

4. REFERENCES

1. Goddard, P. L., "Validating The Safety Of Real Time Control Systems Using FMEA", *Proceedings of the Annual Reliability and Maintainability Symposium*, January 1993.

2. SAE Aerospace Recommended Practice ARP-5580, *Recommended Practices For FMEA,* Draft Version, June 1999.

3. Underwriters Laboratory Standard UL-1998, *Standard For Safety: Safety Related Software*, First Edition, January 1994.

4. NATO Standardization Agreement STANAG 4404, *Safety Design Requirements And Guidelines For Munition Related Safety Critical Computing Systems, Edition 1*

5. United States Air Force System Safety Handbook SSH1-1, *Software System Safety*, 5 September 1985

6. Electronic Industries Association Bulletin SEB6-A, *System Safety Engineering In Software Development,* April 1990

7. Leveson, N. G. , *Safeware: System Safety And Computers*, ISBN 0-201-11972-2, 1995

8. Deutsch, M. and Willis, R., *Software Quality Engineering*, ISBN 0-13-823204-0, 1988

9. Hatton, L., *Safer C*, ISBN 0-07-707640-0, 1994

5. BIOGRAPHY

Peter L. Goddard
Raytheon Systems Company
1650 Research Drive, Suite 100
Troy, Michigan 48083, USA

Pete Goddard is currently employed as a Senior Principal Engineer with the Raytheon Consulting Group in Troy, Michigan. He holds a bachelors degree in Mathematics from the University of LaVerne, and a masters degree in Computer Science from West Coast University. Mr. Goddard has published papers in the proceedings of the Annual International Logistics Symposium, the RAMS Symposium, the AIAA Computers in Aerospace Symposium, and the INCOSE Symposium. He was the principle investigator for the 1984 Rome Labs sponsored "Automated FMEA Techniques" research study and was program manager and part of the research team for the 1991 Rome Labs sponsored "Reliability Techniques For Combined Hardware And Software Systems" research study. He is a co-author of "Reliability Techniques for Software Intensive Systems". Mr. Goddard is an active member of the SAE G-11 Division and is part of the subcommittee on FMEA in the G-11. He is a member of IEEE and an ASQ member and CRE.