Applying FMEA to Software

Rick Homkes, Donna Evanecky, Henry Kraebber
Purdue University College of Technology

Abstract

Failure Mode and Effect Analysis (FMEA) is a well-known industry technique for improving the reliability, quality and safety of products and processes. It "can be described as a systematic group of activities intended to: (a) recognize and evaluate the potential failure of a product/process and the effects of that failure, (b) identify actions that could eliminate or reduce the chance of the potential failure occurring, and (c) document the entire process."[1]

The focus of FMEA is on the design of products and processes. FMEA provides designers with a formal process and "disciplined techniques to identify and minimize the impact of design concerns."[1] It is intended to be used early in the development process, not after problems have become evident. The FMEA procedure utilizes cross-functional teams with open discussion and communications. Effective preventative and corrective actions based on the FMEA findings are essential, and can be very valuable. FMEA studies without proper action on the findings and recommendations will produce little value.

FMEA, however, has generally not been applied to software. As we continue to move to ubiquitous computing, however, software will be more completely integrated into products that consumers use every day. This increase in the use of embedded systems is also matched by the increase in their importance. The effects of some form of failure to the average consumer have thus gone from inconvenience or economic harm to actual injury or death. This means that FMEA, along with other techniques such as Preliminary Hazard Analysis (PHA), Reliability Block Diagrams (RBD), and Fault Tree Analysis (FTA), need to be an integral part of design, development, manufacturing, and maintenance. These techniques should apply not just to individual products, but also to complete systems. In other words, these techniques should not be used with only the hardware of an Antilock Braking System (ABS) controller, but also to the ABS software, the entire braking system and the entire vehicle. This paper starts by making an overview of PHA, RBD, FTA, and FMEA. It then investigates how these methods could be incorporated into the courses that cover programming and software engineering.

Review of Quality / Reliability / Safety Techniques

Quality assurance requirements and techniques have been with us for some time. Even Hammurabi's Code of Law from 1750 BCE has mention of several basic quality assurance programs. These include the corrections in the manufacturing process "If a builder build a house for some one, even though he has not yet completed it; if then the walls seem toppling, the builder must make the walls solid from his own means;" in the failure of a product "If a builder build a house for some one, and does not construct it properly, and the house which he built fall in and kill its owner, then that builder shall be put to death;" and an early type of product warranty in the building of a ship "If a shipbuilder build a boat for some one, and do not make it tight, if during that same year that boat is sent away and suffers injury, the shipbuilder shall take the boat apart and put it together tight at his own expense. The tight boat he shall give to the boat owner."[2] Thus quality is closely related to reliability and safety, but they are not equivalent. For many organizations, quality is defined by the customer, and is related to value or return on expense. Reliability is the susceptibility of a product or process to failure. Safety is freedom from harm and is associated with a reasonable risk for a particular use. Even this must be further defined, as freedom from harm can be broken down into economic harm (an unsecured web site is hacked and credit card numbers are stolen and used) or physical harm (a product failure that causes injury or death to a user or bystander). Lastly, reasonable risk has both societal and legal definitions.

As microcontroller driven products become commonplace in the consumer market, the definitions above become important to software engineers as well as well as hardware engineers. While software people are very familiar with testing, it could probably be stated that more system (hardware and software) thinking is necessary as we build and deploy more mechatronic systems. For example, until recently almost all automotive steering (like all flight control in the past) was a totally hardware system. Electronic power assist steering added sensors and actuators into the system along with a microcontroller. Currently available systems have a steer-by-wire system for rear wheels while maintaining the steering linkage in the front. As we continue down this path, however, we will soon be at the day where weight and cost concerns push us to a total steer-by-wire and brake-by-wire situation. At that point software engineers must be as familiar with risk avoidance and mitigation techniques as their hardware colleagues.

There are several techniques that allow product designers to try and build more quality, safety, and reliability into their products. A quick review includes Preliminary Hazard Analysis (PHA), Fault Tree Analysis (FTA), Reliability Block Diagrams (RBD), and Failure Mode and Effect Analysis (FMEA). While this set of techniques is not complete, it does include techniques that are both deductive (top-down) and inductive (bottom-up). As stated by Amberkar et al, "Deductive techniques focus on systematically identifying causes of undesirable effect, while inductive techniques focus on predicting effects of a priori known problems such as faults."[3] Just as the combination of top-down and bottom-up approaches leads to better software design, a combination of deductive and inductive approaches to quality can be used to build more safety into products.

A PHA (see Table 1) is needed at the beginning of any product development cycle to remind the design team that there are risks associated with the use of the product. Examples from the automotive world include "loss of steering" or "loss of braking." An ex-student related to one of the authors about a particular fuel cell chemical reaction that he was working on that had a possibility of a "sudden and irreversible exothermic event." A simple table containing only a few columns such as Event, Effect, Criticality, and Control reminds the development team, including the software engineers, that risk mitigation and management of these events is needed.

Table 1 - Generic Preliminary Hazard Analysis

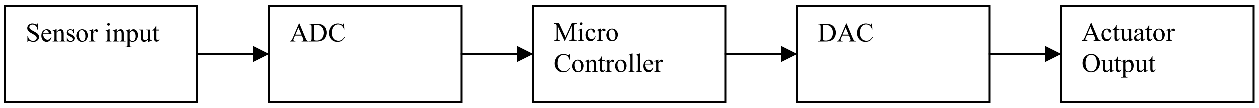
Event	Effect	Criticality	Control
Loss of …	Possible collision	High	Fault tolerant controller

A RBD (see Figure 1) is similar to a Functional Block Diagram (FBD) in that it connects the components of a system. These components could be hardware (e.g. braking calipers) or hardware / software (the ABS controller) or totally software (the ABS program). "The basic steps for creating an RBD for a specific hazard are:

1. Determine the input starting point for the system and the flow of the system from the input to the output.
2. Starting from the input and working toward the output, identify the system components that could contribute to the specific hazard if they failed.
3. For each component, create a block in the RBD and place it in a position relative to its position in the input to output flow of the system.[3]

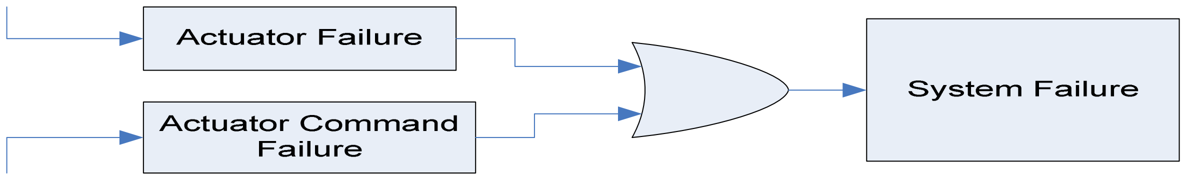
The RBD can then be analyzed for what component or set of components can lead to an undesired event. While the RBD is more detailed than the PHA, it is still very high level, dealing with the block components of the system.

Figure 1 - Generic Reliability Block Diagram



A FTA starts with the unwanted system event that may have been identified in the PHA or RBD above and works its way back to individual failures that cause the event. In other words, a "Fault tree analysis (FTA) is a top-down approach to failure analysis, starting with a potential undesirable event (accident) called a TOP event, and determining all the ways it can happen."[4] The ways that it can happen, or the individual failures along the way, are connected with AND gates and OR gates to get to the final event. For example, Figure 2 shows how a system failure could occur by the failure of either the actuator or the actuator command. While product (and thus software) development focuses on FTA from a projective approach, it could also be used from a historical or diagnostic approach. A recent television show in the Crime Scene Investigation (CSI) series showed this can be done. In the episode, the investigation of a bus accident found that a tire failure caused by a contaminant being placed in the tire along with a bolt failure from a bad part caused the resultant loss of control and crash.

Figure 2 - Generic Fault Tree Analysis



Failure Mode and Effects Analysis

Unlike PHA, RBD, and FTA, FMEA is considered an inductive approach to failure analysis. Here the analysis is done from an individual component (or in the case of a Software FMEA, an individual variable), and following the problem as it influences the subsystem or system. The goal of an FMEA is to identify problems and rank them using a Risk Priority Number, or RPN.

Some sample FMEA table headers are shown in Table 2. The RPN is found using values for severity (S), occurrence (O), and detection (D). One approach to performing an FMEA is to:

1. Identify and list individual components, the function they provide, and their failure modes. Consider all possible modes.
2. For each failure mode, determine the effects of the failure on all other system components and on the overall system
3. Determine the severity of the failure, the potential causes of the failure, and the likelihood that a potential cause will occur.
4. Identify the current design controls that will assure the design adequacy of the failure controls. Determine the ability of the proposed design controls to detect a potential cause or the ability the proposed controls to detect the subsequent failure mode before the component is released for production.
5. Determine the RPN based on the severity, occurrence, and/or detection rankings.
6. For the highest ranking RPN's, recommend actions to take that will reduce the severity, occurrence, and/or detection rankings
7. Re-evaluate the RPN based on the new estimates of the severity, occurrence, and detection rankings.[3]

Table 2 - Sample FMEA Table Headers

Reference Number	Component	Potential Failure	Local Effect	Related Effect	System Effect	S	O	D	RPN	Recommendation

While there are many variations, including a closely related technique called FMECA or Failure Mode, Effects, and Criticality Analysis[4], there are four types of traditional FMEA that can be completed. These are:

System FMEA which focuses on global system functions
Design FMEA which focuses on components and subsystems
Process FMEA which focuses on manufacturing and assembly processes
Service FMEA which focuses on service functions[5]

No matter which type of FMEA is being completed, engineers from the entire problem domain should be included. Depending on the problem or project, this could include mechanical, electrical, manufacturing, systems, and software engineers. One of the author's experiences with process FMEA development at a large automobile manufacturer suggests that this cross functional participation is a key element in the completion of any FMEA. It is the combination of experience and knowledge from all the different engineering competencies involved that adds value to the FMEA process. Generally, though, only one person holds the responsibility of gatekeeper over the follow-up and review of the FMEA. Having this dedicated and involved FMEA agent is also crucial.

Among the list of prerequisites for completing an FMEA are a system definition in regards to interfaces, functions, and operational conditions, the collection of artifacts about the system such as drawings and parts lists, and the collection of information from previous designs that are similar. This information from previous designs, including the failure analysis worksheets, allows the "lessons learned" from previous products to be applied to the current product. This is an area where software engineers could use artifacts that have traditionally been used by hardware colleagues. Part of any analysis should be to compare the FMEA to the FTA. By comparing the top-down FTA to the bottom-up FMEA, the likelihood that an error is missed is reduced. Both artifacts should be updated as the project continues. According to Stamatis, "The FTA always supplements the FMEA and not the other way around. In general, its application may be in a system or subsystem environment with a focus on identifying the root factors that could cause a failure and their independent relationships."[6]

Incorporation of FMEA into courses that cover programming and software engineering

According to Bowles and Wan,

> Software, it is said, is "different", all errors are "logic" or "design errors"; hence, it doesn't fail. The cause of an error and a failure may be far removed from where the error occurs and extensive testing is sometimes viewed as more cost effective than analysis.[7]

Another way of distinguishing between hardware and software is to review the three attributes listed by Pressman. These are that:

1. Software is developed or engineered; it is not manufactured in the classical sense.
2. Software doesn't "wear out."
3. Although the industry is moving toward component-based assembly, most software continues to be custom built.[8]

Of special note is the second attribute above. Quite simple, software doesn't fail like hardware. Hardware has a "bathtub failure curve" of higher initial errors brought about by design and manufacturing problems, fewer errors during the period of normal operation, and then higher failure rates as the product wears out. Even though software does not wear out like hardware (e.g. an EEPROM write limitation), these differences do not preclude software engineers from bypassing predictive safety techniques and only using good design methods and software testing as a means to assure quality. Using the systems or design FMEA mentioned above as a starting point, SAE classifies software FMEA by the way in which the failure mode is recorded and tracked. These are the Functional, Interface, and Detailed FMEA. Unfortunately, "Software FMEAs, as with all FMEAs, tend to be labor intensive. Consequently, it is desirable in most cases to limit software FMEAs, particularly software Detailed FMEAs, to those elements of a software system whose correct functioning is crucial for continued safe (or revenue critical other significant criteria) operation."[9]

Software Functional FMEA "focuses on the functions that an item, group of items, or process performs rather than the characteristics of the specific implementation."[9] Software engineers should look at the architecture, partitioning, and decomposition of the software elements in the system. Assessment can be made as to a modules failure to execute, incomplete execution, execution at the wrong time, or incorrect result. Software Interface FMEA "is intended to focus on failures affecting the interfaces between disparate software and hardware elements."[9] In this case software engineers should assess how a value may not be updated, be updated incompletely, be updated at the wrong time, or be updated incorrectly. Software Detailed FMEA is meant to "assess the effect on the software and the system of failures in the software functionality and in the variables used in the software."[9] In this case incorrect values that may result from an overflow / underflow / out of bounds condition would be traced. In the authors experience a Detailed FMEA of an actual product took a great amount of effort and an off-site retreat of all the product developers. However, for a classroom experience a small exercise using a Detailed

FMEA could easily be placed in most programming classes to check for the conditions mentioned above. As code is placed in the program to handle the situations, the RPN values would change as the occurrence, severity, and delectability factors are changed.

While practice and literature recommend that FTA be used as a supplement to an FMEA, for programming students it may be the required prerequisite. Having the students perform a "What could go wrong?" brainstorming session may be the best way to start thinking about failures. Diagramming the fault tree can then lead to the individual rows in the FMEA table. Use of FBD and RBD, with their analogousness to the class diagrams of an object oriented systems paradigm, is another way for software persons to start the FMEA. Even when the detailed FMEA is being completed, the hardware and software people should remain in communication. There are many cases where diagnostic routines could be inserted into the master loop to check on the hardware components. This will increase the likelihood of detection and thus decrease the RPN. This technique is very applicable to most embedded software development courses.

There are certain actions that increase product reliability that can become a standard boilerplate rows for an FMEA. These have come to embedded programmers as a result of lessons learned the hard way. Foremost of these to one of the authors is that ALL registers of a controller should be initialized, even the ones that are listed in the specification as a "don't care." What happens more often than expected is that different production lots of controllers may have small differences in the register values. These small changes can then cascade into a problem that effects product performance. A second action that can be taken is to do a checksum of all ROM and EEPROM on system startup. While both register initialization and checksumming take valuable time during a system recycle or startup, the resultant error checking can result in a more dependable and safe product. This would be very appropriate to any Computer Engineering Technology senior project as well as any Embedded Controller course.

Conclusion

This paper reviewed several quality, reliability, and safety techniques that are available in the design and development of new products. As our society continues to move towards ubiquitous computing, computers will be interwoven into all of the products that we use. With this increase in use comes an increase in the requirement that the software does no harm to its users. By teaching these techniques to software engineers as well as hardware engineers, we are helping to prepare them to work in team environments, and, more importantly, helping to make the products that we all use safer.

Bibliographic Information

1 Potential Failure Mode and Effects Analysis (FMEA) Reference Manual. (3rd Ed.). (2001). Detroit: DaimlerChrysler Corporation, Ford Motor Company, General Motors Corporation.

2. Hammurabi's Code of Laws. Imaginex Ventures Inc., d/b/a CanadianLawSite.com. Downloaded 1/4/2005 from http://www.canadianlawsite.com/Hammurabi'sCode.htm

3. Amberkar, S., Czerny, B., D'Ambrosio, J., Demerly, J., & Murray, B. (2001-01-0674). A comprehensive hazard analysis technique for safety-critical automotive systems. SAE 2001 World Congress. SAE: Warrendale, PA.

4 Rausand, M., & Høyland, A. (2003). System Reliability Theory; Models, Statistical Methods and Applications, 2nd Ed. Wiley: New York, NY.

5 Failure Modes and Effects Analysis. DRM Associates. Downloaded 1/4/2005 from http://www.npd-solutions.com/fmea.html

6 Stamatis, D. (1995). Failure Mode and Effect Analysis. Quality Press: Milwaukee, WI.

7 Bowles, J., & Wan, C. (2001). Software failure modes and effects analysis for a small embedded control system. 2001 Proceedings Annual Reliability and Maintainability Symposium. IEEE: Washington, DC.

8 Pressman, R. (2001). Software engineering: A practitioner's approach. McGraw Hill: New York, NY.

9 Recommended Failure Modes and Effects Analysis (FMEA) for non-automotive applications. (2000-10-18). Aerospace Recommended Practice ARP5580. SAE: Warrendale, PA.

Biographical Information

Rick Homkes is an associate professor of Computer Technology for the Purdue University College of Technology site at Kokomo, Indiana. He has participated as a faculty intern for many years at Delphi Electronics & Safety, working in manufacturing test, corporate training, and embedded systems.

Donna Evanecky started teaching as an assistant professor for Purdue University School of Technology in 2001 after an eight-year career in the field of quality management at Daimler Chrysler. She teaches for the Department of Organizational Leadership and Supervision at the Kokomo campus.

Henry Kraebber is a professor of Mechanical Engineering Technology in the College of Technology at Purdue University in West Lafayette, Indiana. He teaches manufacturing operations and quality systems.