Chapter 2
Software Failure Mode and Effects Analysis

The process of thinking about the potential problems and knowing how to detect, prevent, or mitigate the problems before they create a catastrophe is what failure mode and effects analysis (FMEA) is all about. FMEA enables engineers to design quality and reliability into a system. A variant of FMEA, failure mode, effects, and criticality analysis (FMECA), prioritizes failures and failure modes based on the failure rate and severity of failure effects. Collectively, we use FMEA to encompass both FMEA and FMECA. This chapter provides a general introduction to FMEA and software FMEA.

2.1 FMEA

FMEA was first documented as part of the US military procedure MIL-P-1269, titled "Procedures for performing a failure modes, effects and criticality analysis," dated back to 1949. Failures were classified by their impact on mission success and personnel or equipment safety. The later military standards MIL-STD-1269 and MIL-STD-1269A were based on MIL-P-1269.

FMEA was practiced mostly in military and aerospace until late 1970. Ford Motor Company was the first to adopt FMEA in the automotive industry. A broader automotive industry adoption happened in the 1980s, and the joint efforts of a few automotive companies eventually led to the industry standard SAE J1739 published in 1994. The use of FMEA now has been embraced by many industries including aerospace, automotive, nuclear power plant, manufacturing, medical device, and health care.

Y.-M. Zhu, *Failure-Modes-Based Software Reading*, SpringerBriefs in Computer Science, https://doi.org/10.1007/978-3-319-65103-3_2

2.1.1 What Is FMEA?

FMEA promotes a systematic way of thinking when a new product or system is developed: what could go wrong, how badly might it be, and how could it be prevented and mitigated. The primary objective of FMEA is to improve design or process. FMEA is a bottom-up, inductive, static analysis method with the effects of failure on lower levels (e.g., individual components) being identified first. FMEA is often performed hierarchically, and the effects identified at the lower level propagate and serve as failure modes at the next higher level [36].

There are many types of FMEA, e.g., system FMEA, mechanical FMEA, electrical FMEA, software FMEA, product FMEA, process FMEA, human-use or misuse FMEA, and health care FMEA. Although the purposes, terminologies, and details vary for each FMEA type, the basic concept is similar.

An FMEA process typically includes the following steps or phases:

(1) Choose the items to be analyzed,
(2) Identify the failure modes and root causes for the failure modes,
(3) Assess the impacts of the failure modes and root causes, and
(4) Prioritize the failure modes and root causes and identify and implement controls (preventive measures, corrective actions, and compensating provision) to improve the product or process design.

To facilitate the analysis, a spreadsheet is typically used, which can be downloaded from many Web sites or other sources. FMEA is an iterative process, repeated multiple times in the course of product or system development. It is a team effort and often includes meetings of people with different backgrounds and diverse skills and knowledge.

To assess the risk associated with issues identified during an FMEA process and to prioritize corrective actions, some quantitative methods are included. The most commonly used methods for assessing criticality are risk priority number (RPN), criticality number ranking as in MIL-STD-1629A, and multi-criteria Pareto ranking. In the case of RPN, an FMEA team rates the severity of the failure effect, the likelihood of occurrence of the failure cause, and the detection of the failure cause and calculates the RPN as the product of the above three numbers. The rating scales for severity, occurrence, and detection are typically 1 to 5 or 1 to 10. An organization defines a threshold for the RPN value, above which some mitigation is mandatory.

Many authors have written about the benefits of FMEA. FMEA improves product or process reliability and quality and, as a result, increases customer satisfaction. It enables a development organization to identify and eliminate potential failure modes and root causes, which reduces and minimizes late changes and their associated costs. FMEA itself captures an organization's knowledge, catalyzes teamwork, and promotes idea exchanges across teams.

2.1.2 FMEA Standards

FMEA has been adopted in a variety of industries, and industry-specific standards have been established. Aerospace and defense companies usually use the MIL-STD-1629A published by the US Department of Defense in 1980. The standard was canceled in 1998, and users instead were referred to various national and international standards. The automotive industry mostly uses the SAE J1739, the first version of which was published in 1994 and the latest in 2009 (the standard for non-automobile applications is SAE ARP5580). Other industries generally adopted the IEC 60812, titled "Analysis techniques for system reliability—procedure for failure mode and effects analysis (FMEA)," published by the International Electrotechnical Commission, and the latest version was issued in 2006. Many professional societies and organizations have published FMEA handbooks or established their own FMEA processes as well, e.g., International Marine Contractors Association Guidance 166 on FMEAs.

2.2 Software FMEA

Reifer was widely credited for introducing the FMEA to software engineering for requirements analysis in 1979 [51]. However, there was some precursor work before Reifer, e.g., the software error effects analysis in 1973 [22]. Although software FMEA has been practiced for about 40 years, there is no specific, widely accepted guideline or standard for software FMEA. Researchers and practitioners frequently refer to IEC 60812 for software FMEA. Like hardware FMEA, software FMEA is primarily used to discover the software design issues during software development.

Software is different from hardware. Hardware fails due to aging, wearing, or stressing, but software modules do not fail and just exhibit incorrect behaviors [17]. For software components, the failure modes are generally unknown and depend on the dynamic behavior of the system, since if a failure mode would be known, it would be corrected [17]. The definition of failure modes is one of the hardest parts of software FMEA. Analysts have to apply their own knowledge about the software and postulate relevant failure modes. During software FMEA, possible design failure modes and sources of potential nonconformities must be determined in all software artifacts, mostly codes, under consideration. A complete FMEA for a software-based system should, however, include both hardware FMEA and software FMEA, and the effects should be assessed on the final system functions.

2.2.1 Types of Software FMEA

Software FMEA in practice is often performed at different levels, system, subsystems, and components, which corresponds to architectural partitions or levels of abstraction. The software FMEA file, e.g., a spreadsheet, is treated as a living document, and analysis at different levels can be rolled up. As design and implementation proceed, more details of the system are revealed, which enables meaningful low-level analysis. Goddard suggested the use of software FMEA at a system level and a detailed level on embedded control systems [25], where a system-level software FMEA can be based on the top-level software design and performed early in the software design process and a detailed-level software FMEA can be based on the software module design such as pseudo code and applied late in the design process. Software FMEA at the detailed level can be labor-intensive. Similarly, the committee on the safety of nuclear installations at the Nuclear Energy Agency of the Organization for Economic Cooperation and Development proposed software FMEA at system/division, unit, module, and basic component levels [14].

When conducting software FMEA, analysts take different viewpoints. Bowles and Wan extended Goddard's work and introduced functional, interface, and detailed software FMEA [8], where functional software FMEA is related to functional requirements, interface software FMEA is concerned with the interfaces between software modules and between software and hardware, and detailed software FMEA is conducted on individual variables. The authors suggest to use the results of functional software FMEA to reduce the amount of effort required for the interface and detailed analysis and the results of functional and interface analyses to reduce the effort required for the detailed analysis.

Although researchers and practitioners discussed many different viewpoints, Neufelder's work is most comprehensive and she identified eight viewpoints: functional, interface, detailed, maintenance, usability, serviceability, vulnerability, and software process [42]. The functional viewpoint is mostly useful for software requirements FMEA, the interface viewpoint is for software/software and software/hardware interface FMEA, the detailed viewpoint is applicable to design and implementation FMEA, the maintenance viewpoint is related to software maintenance FMEA, the usability viewpoint is for system use and misuse FMEA, the serviceability and vulnerability viewpoints, as their name suggested, are related to software serviceability and vulnerability, and the software process viewpoint is for software process FMEA.

During software development, many software artifacts are generated. Software FMEA can be tailored to analyze requirements, design, and code implementation. Perhaps the most frequently used areas of software FMEA are requirements analysis and code analysis. In fact, Reifer applied software FMEA to requirements analysis [51]. Code-level FMEA tends to be tedious, and tool assistance is highly desired.

2.2.2 Software FMEA Steps

In general, a software FMEA follows the same or similar steps sketched in Sect. 2.1.1. Some authors wrote about an elaborate preparation or planning phase [42], where the hierarchical level (system, subsystems, and components) is decided at which the FMEA is performed, the riskiest parts of the system or subsystem are chosen, as well as the viewpoints to conduct the FMEA are determined. The outcome of an FMEA is documented in a worksheet, and the analysis team needs to pick and agree on the worksheet format. The planning phase can also include team member identification and team formation, resource allocation, software tool customization, documentation (e.g., requirements specification, architecture description, design document, and code modules) identification and preparation, and general agreement on ground rules, assumptions, failure definitions, and policies.

Do not underestimate the effort to prepare information or documentation to support a software FMEA, in fact any FMEA, which may include system architecture or structure, system environment, system boundary, block diagrams, system functional structure (both definition and representation), system initialization/operation/control/maintenance, system and component modeling, failure significance, and compensating provision. This information or documentation defines the extent of analysis and provides the right input to the analysis. It is understandable that, as system design and implementation proceed, they will have different levels of maturity and accuracy.

There is no single widely accepted guideline or standard on the software FMEA procedure. However, there are papers, chapters, and books that discuss software FMEA and its procedures. El-Haik and Shaout elaborated a 12-step software FMEA procedure [17], which is summarized in the following nine steps.

- Define the software FMEA scope. The team lead, along with team members, defines what functions, critical areas, or subsystems and components will be subject to the software FMEA exercise. The team may consult the project scope and architecture description to identify systems, subsystems, or components boundaries.
- Identify potential failure modes. The team can research past failure modes and brainstorm new failure modes with subject matter experts based on new insights on the software under concern. Common failure modes include faulty requirements, interfaces, communications, timing, sequence/order, logic, data, data definition, memory allocation/management, installation, error detection/recovery, false alarm, synchronization, algorithms/computation, user mistakes, user recovery from mistakes, user instructions, abusive user, and incompatibility. While failure modes are identified, related failure causes shall also be identified. Software failure modes are typically caused by design faults or deficiencies, e.g., violation of design principles and best practices. A database or categorization of the failure modes is an important part of corporate knowledge.
- Identify potential failure effects. A potential effect is the consequence of the failure. These effects can be local effects, the effects on the subsystem and

system levels, and the effects on the operator, environment, or other people involved. Fishbone diagrams are often used to capture the cause/effect relationship.

- Rate severity. Severity is a subjective rating and measures how bad or serious the effect of the failure mode is. It is typically rated on a discrete scale of 1 to 5 or 1 to 10, corresponding to negligible to catastrophic effects. Organizations need to provide guidance on how to rate severity, and real examples are highly desirable. Failure effects should be propagated to the system level. The most serious failure modes often need a control plan to mitigate the risks, and the development team needs to develop proactive and preventive design recommendations.
- Rate occurrence. Occurrence is the likelihood that the failure cause occurs in the course of the intended life. It is a subjective rating as well. For software FMEA, it usually assumes that if the failure cause happens, so does the failure mode. Thus, occurrence also measures the likelihood of the failure mode. It is rated on a scale of 1 to 5 or 1 to 10, with highest occurrence corresponding to highest probability and vice versa. The occurrence is just a ranking scale, not the actual probability which would be within 0 and 1 inclusive. Organizations can use historical data to derive the failure rate and use it to guide the assignment of occurrence rating. Researchers also suggested some proxies such as complexity metrics as an indirect measure for occurrence.
- Assess current controls. The purpose of software design controls is to identify, as early as possible, issues such as nonconformities, including not meeting requirements specifications and violating design principles or best practices, deficiencies, or vulnerabilities. To this end, the software FMEA team shall review past similar failure modes and how they were detected, or brainstorm how failure modes can be recognized and detected with new technologies. That is, current controls prevent the cause of failure from occurring. In addition to specific techniques and design verification, design controls also include general design guidelines, best practices, and standards and procedures adopted by the project team such as design review and operator training.
- Rate detection. Detection is also a subjective rating that quantifies the likelihood that a detection method will detect the failure of a potential failure mode before the impact of the effect is materialized. Again, it is a ranking scale of 1 to 5 or 1 to 10, not the actual probability. Common detection methods include review or inspection, assertions, and data validation. The software FMEA team shall assess the capability of each detection method, which may be used in different stages during the development based on their capabilities. The team then reviews all the detection methods, builds a consensus on their detection ratings, and rates the methods. It is important that failure modes with the highest severity are detected directly, close to the failure cause or source, and are compensated immediately and effectively. For failure modes with the lowest severity, they can be detected away from the source, e.g., by the effects of the failure modes; they can be compensated by default values, retry, or other exception handling mechanisms.

- Compute RPN. RPN is simply the product of severity, occurrence, and detection ratings, and it is used to prioritize potential failure modes and root causes. The software FMEA team agrees on a threshold, and when the RPN of a failure mode is above that threshold, the team proposes recommended actions (see next step). There is no universal threshold, since the ratings may be industry or organization specific. The severity, occurrence, and detection, thus RPN, shall be reassessed after a risk mitigation is implemented.
- Recommend actions. Where the risk of a potential failure is high (large RPN), a control plan shall be developed to improve the situation. Potential actions include the following:

  (a) Transfer the risk of failure to other systems,
  (b) Prevent failure altogether, and
  (c) Mitigate the risk of failure by

  – Reducing severity,
  – Reducing occurrence, and/or
  – Increasing the detection capability.

2.3 Software FMEA in the Software Development Life Cycle

Various processes or methods are selected for the software development projects, which are known as the software development life cycle models, including the waterfall model, the V model, the iterative model, the spiral model, and the agile model. The V model is nothing but a verification and validation model and often used as a reference model or framework to understand the software development process.

The V model is schematically shown in Fig. 2.1. On the left branch of the V are, from top to bottom, requirements definition, system design, subsystem design, and

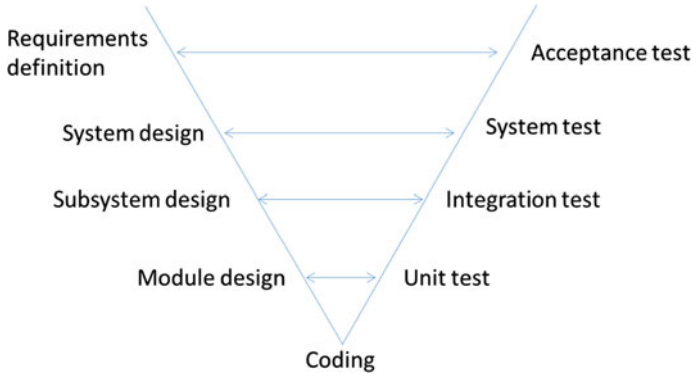

Fig. 2.1 The V software development model

module design. On the bottom is implementation or coding. On the right branch of the V are, from bottom to top, unit testing, integration testing, system testing, and acceptance testing, which correspond to the activities on the left branch of V.

The code is often reviewed or inspected to detect anomalies. It can also be analyzed with other static analysis tools. Often the time code or the software unit is unit-tested by execution (dynamic testing). A unit test is to verify that the software unit is implemented per module design. When multiple software modules or components are integrated together, the emphasis is typically on the interfaces among those modules and assumptions one module has on others. System testing is against system requirements specifications and confirms that the system is implemented correctly. The acceptance testing makes sure the right system is implemented from the user's perspective.

Tracing is an important concept in the software development, and the horizontal lines with arrows indicate the tracing between requirements or designs and testing. Every requirements specification and design element shall be tested, and every test shall be traced to system functions. A software system is often implemented using a divide-and-conquer strategy, and the system is decomposed recursively until the lowest element can be adequately dealt with. Thus, there is a tracing on the left side of V from higher-level to lower-level system functions or designs. In line with the hierarchical decomposition, requirements specifications can also be recursively decomposed and refined at system, subsystem, and component levels, which gives rise to system requirements specification, subsystem requirements specification, and component specification, respectively.

Software FMEA can be applied on the left branch of the V, including the implementation at the bottom of the V. The hierarchical levels of software FMEA naturally correspond to the hierarchical decompositions of the system. Software FMEA at a system level can be applied to the requirements definition and system design phases; software FMEA at the subsystem level can be applied to the subsystem design phase; and lastly, software FMEA at the component level can be applied to the module design and implementation phases.

When conducting software FMEA, analysts can take different viewpoints [42]. When analyzing system-, subsystem-, or component-level requirements specifications, a functional viewpoint can be assumed. When analyzing the design, particularly the interface aspects of the interacting components, an interface viewpoint can be adopted. When analyzing the detailed design or implementation, a detailed viewpoint can be taken. When analyzing the human interface or human–computer interaction, a usability viewpoint can be appropriately assumed.

We shall point out that while it is true that software FMEA has been used in safety-critical system, that does not mean other types of software applications will not benefit from it. To the contrary, software FMEA can be applied to any software system to prevent problems from happening, which improves software quality and increases system reliability.

In Chap. 3, we will discuss software peer review and how it is applied in the software development life cycle (see Sect. 3.3). The rest of the book, Chaps. 4–9, discusses failure-modes-based software reading that merges the strengths of

software FMEA and peer review and avoids their respective weakness, and illustrates how the failure-modes-based reading techniques can be employed in software requirements (Chap. 5), design (Chap. 6), implementation (Chap. 7), usability (Chap. 8), and testing (Chap. 9).

2.4 Summary

This chapter overviewed the history of FMEA, sketched different types of FMEA and main steps in FMEA, and surveyed FMEA standards and guidelines. It then focused on software FMEA. The differences between hardware FMEA and software FMEA were highlighted. The software FMEA steps were illustrated, and how the software FMEA can fit in a software development life cycle was briefly touched upon.

We shall point out that software FMEA is not meant to replace software reliability methods. Rather, software FMEA provides a systematic thinking tool that allows developers to anticipate issues and improve designs. Software FMEA is applicable not just to safety-critical software. All kinds of software projects can benefit from software FMEA.

http://www.springer.com/978-3-319-65102-6

Failure-Modes-Based Software Reading

Zhu, Y.-M.

2017, XI, 99 p. 12 illus., Softcover
ISBN: 978-3-319-65102-6