SFMEA assistant design method for control system using requirement modeling

Mengyue Liu, Junshu Wang, Dong Li, Yihong Liu
China Electronic Product Reliability and Environmental Testing Research Institute, Guangzhou 510610, P.R. China

Abstract. Software Failure Modes and Effects Analysis (SFMEA) is an effective reliability analysis technique which is widely used in control systems. However, it is difficult to analyze the failure mode using traditional SFMEA method, as control system becomes more complicated. This paper proposed an assistant design method of SFMEA based on system requirement model, which would remedy the deficiencies in analyzing complex control system. An experiment shows that this method can be applicable to SFMEA for complex control system.

1 Introduction
Software plays an important role in various systems, such as real-time computer control system, embedded computer system, etc.[1]. As being widely used in control systems that were once operated by humans only, software has more influence on system reliability and safety nowadays. The SFMEA methodology is a software reliability analysis technique based on the bottom-up approach. It may improve software reliability and safety by identifying possible software failure modes in control system. After proposed by Reifer in 1979, SFMEA has been widely applied to the design of control system reliability[2].

While control system becomes more and more complicated, the application of SFMEA is less effective due to its implementation way. The analysis of SFMEA is mainly carried out manually; and its expression is inaccurate and ambiguous. But a successful SFMEA analysis requires precise and efficient representation for the structures, functions and behaviours of control system.

Tribble described how to realize FMEA analysis automation with assistant technologies in a report [3]. However, only some theoretical concepts were mentioned. This paper proposed an assistant design method of SFMEA from system requirement model, in order to solve the problems come across in analyzing complicated or large scale control system. The rest of this paper is organized as follows. The process of SFMEA in control system is described in section 2. The assistant design method and procedure is described in section 3. A modeling experiment is described in Section 4. And section 5 gives the conclusion.

2 The process of SFMEA in control system
By applying SFMEA in the early phase of software development, this methodology provides a systematic strategy to find out failures and weaknesses in software design [4]. The results of SFMEA in turn affect the development process and make the software meet the requirement of reliability and safety. The process of SFMEA in control system includes the following steps [5]:

(1) Confirmation of software functions
Generally, a control system can be divided into several software units. And the functions and structures of these units can be confirmed during this step. Also, the software flow charts can be drawn out.

(2) Identification of software failure modes
After analyzing the system structure and function accurately and effectively, all the possible software failure modes in the system, which are used to describe system behaviours when software fails, should be identified.

(3) Assessment of the failure mode effect(s), cause(s), etc.
All the following contents for a failure mode should be found out: failure mode effect(s), failure mode cause(s), corrective or preventive measures for failure mode(s). Then record all the information details above in a SFMEA Table.

(4) Reassessment in the new iteration
Modify the software requirements and specifications according to SFMEA results, and set up another SFMEA cycle as software development carried out until the SFMEA results show the reliability and safety of software is up to the standard.

With SFMEA, potential faults of control system software can be revealed in the early software detection. And the system reliability and safety can be enhanced thereby. On emphasis, software function confirmation is the basic step of the entire process above, while directly affecting the following steps.

3 SFMEA assistant design method using requirement modeling
The purpose of this research work is to standardize the process and results for confirming software functions. The SFMEA assistant design method based on system requirement model can provide a comprehensive description of system requirements via using Semi-Formal graphic language, formal mathematical language or pure computer language to describe system requirements, which is more objective and accurate compared with using natural language. SFMEA assistant design starts from analyzing the characteristics of control system, after the outline of system requirement modeling been confirmed then, puts forward the modeling methods of system interfaces, functions and operation states.

3.1 Process of control system requirement modeling
Control system has been widely applied in aerospace, industry, power system, devices instrumentation, and so on [5]. It has specific characteristics as follows:
• Complexity of interface and data
   Due to the complicated interactive interfaces and relationships, control system needs to handle large amounts of data for acquisition, processing and analysis. Hence, only with the accurate and clear description of the interfaces and data objects between the system and external environment, can we sufficiently analyze the influence on system functions if there are any exceptions in the interaction.
• Diversity of software functions
   The most important function of control system, which is complicated generally, is accomplished by software. During software development, many influential factors including functional logic, time sequence, and abnormal data should be taken into consideration and handled correctly. Thus, the number and complexity of software functions may increase rapidly. In order to completely analyze all the affections to system safety caused by function failures, the software functions should be described accurately.
• Variation of operating states
   Control system has many tasks to implement and may go through various operating status. Transitions between operating states may cause uncertainties in operation or data process. Only describing the operating states accurately, can the influence of states transition on system safety be analyzed comprehensively.

   Via characteristic analysis, we formed a bottom-up modeling process: interface –software function –operating state (shown in Figure 1).
The requirement modeling of control system has three main modules as: Interface model, Software function model, Operating state model which can be build by using text, charts and formal languages according to the software requirement and interface specification.

3.2 Modeling method and process of interface

The process of system interface modeling is as follow:

Step 1: Confirming the control system interface

Confirm the internal and external boundary according to modeling system object. If there is any redundancy design in control system, then the redundancy design strategy has to be raised, and its expression form can be annotated by natural language or redundancy model.

Step 2: Confirming the external connection equipment

Confirm external connection equipment of system object, such as sensors, other systems, according to system interface requirement or software interface specification. For the complicated external connection equipment, dividing it into sub-components can simplify the process. If there is a physical limitation or redundancy design, the information can be described by natural language.

Step 3: Modeling of external input interface and data objects

According to the system interface requirements and software interface specifications, each input interface type and data element between external connection equipment and system object should be confirmed at first. Next, confirm the data essentials information of input interface, include name, range, communication format, period, failure processing, etc. If there are many external interfaces and data information, charts will be useful.

Step 4: Modeling of external output interface and data objects

Similar to last step, relying on the system interface requirements and software interface specifications, the external output interface types and data elements should be confirmed at this step.

Step 5: Confirming the external operation environment

Confirm the external operation environments and its changes according to the system requirements and software requirement specifications, such as temperature, humidity, altitude, pressure, etc. each of which can be described with natural language.

Step 6: Review

Review all the connection equipment, input or output interfaces and make sure they are completely covered. If “true”, finish the entire process, otherwise, repeat the steps mentioned above.

A semi-formal system external interface modeling can be defined as: $IO = \{I, O, E\}$, I means a set of external input interfaces, O means a set of external output interfaces, E means a set of external connection equipment.

(1) Set of external input interfaces
If a system has N external input interfaces, set I can be expressed as I= \{I_1, I_2, ..., I_n\}. Each of external input interface I_i can represent in multi-components:

\[ I_i = (i_{name}, i_{source}, i_{type}, i_{value}, i_{period}, i_{constraint}, i_{backup}, i_{check}, i_{error}, i_{Original}, i_{Safe}) \],

- **i_{name}**: the name of input interface
- **i_{source}**: source of input interface, \( i_{source} \in E \)
- **i_{type} \in \{0,1,2,3,4,5,...\}**: type of input interface, the meaning of each number can be user-defined. For example, 0 can be defined as analog.
- **i_{period}**: input period
- **i_{constraint}**: constrained relationship of the input interface
- **i_{value}**: values of input interface, such as:

  - If 'i_{type}' is a discrete magnitude, then integers can represent the input values, such as \( i_{value} \in \{0,1\} \). If 'i_{type}' is continuous, then interval can represent the input values, such as \( i_{value} \in [L_B, U_B] \), \( L_B \) means lower boundary, \( U_B \) means upper boundary. If 'i_{type}' is open interval or unconscious interval, then describe it respectively. If 'i_{type}' is data bus, then it can be represented as a tetrad, \( i_{value} = (i_{PortChannel}, i_{PortType}, i_{DataLength}, i_{Data}) \). 'i_{PortChannel}' means bus ID, 'i_{PortType}' means bus type, like \( i_{PortType} \in \{422,1553B,...\} \), \( i_{DataLength} \) means data length, \( i_{Data} \) means data frames which can be continuous or disperse.
- **i_{backup}**: set of redundancy for input interface, \( i_{backup} \in I \)
- **i_{error} = \{A, B, C, D,...\}**: the failure diagnosis and processing of input interface, each letter of this set can be user-defined, for example, A means frame format error, etc.
- **i_{Original}**: original of external input when restarting
- **i_{Safe}**: default value or safety value of input while fault occurs, this value can be relevant to the specific function, also as one or more fixed values or taking value intervals.

(2) Set of external output interfaces

If a system has N external output interfaces, set O can be expressed as O= \{O_1, O_2, ..., O_n\}. Each of external output interfaces O_i can represent multi-components as below:

\[ O_i = (O_{name}, O_{destination}, O_{type}, O_{value}, O_{period}, O_{constraint}, O_{backup}, O_{check}, O_{error}, O_{Original}, O_{Safe}) \],

- **O_{name}**: the name of output interface
- **O_{destination}**: destination of output interface, \( O_{destination} \in E \)
- **O_{type} \in \{0,1,2,3,4,5,...\}**: type of output interface, meaning of each number can be user-defined.
- **O_{period}**: output period
- **O_{value}**: values of output interface, the same format as \( i_{value} \)
- **O_{backup}**: set of redundancy for output interface, \( O_{backup} \in O \)
- **O_{error} = \{A, B, C, D,...\}**: the failure diagnosis and processing of output interface, each letter of the set can be user-defined.
- **O_{Original}**: original of external output when restarting.
- **O_{Safe}**: default value or safety value of output when fault occurs.

(3) Set of external connection equipment

If a system has N external connection equipment, set E can be expressed as E= \{E_1, E_2, ..., E_n\}. Each of external connection equipment E_i can represent multi-components as below:

\[ E_i = (E_{name}, E_{type}, E_{source}, E_{value}, E_{period}, E_{constraint}, E_{backup}, E_{check}, E_{error}, E_{Original}, E_{Safe}) \],

- **E_{name}**: the name of connection equipment
- **E_{type}**: type of connection equipment, \( E_{type} \in \{0,1\} \). 0 means this equipment is a sending device (sending data to the system object), 1 means this equipment is a receiving device (receiving data from the system object).
- **E_{source}**: the input interfaces of system object relating to the E_i equipment, \( E_{source} \in I \).
- **E_{value}**: the output interfaces of system object relating to the E_i equipment, \( E_{value} \in O \).

### 3.3 Modeling method and process of software function

The process of software function modeling is as follow:

**Step 1:** Modeling of software function objects
Confirm the modeling software function, and determine function boundary. If there are some redundancy relationships between other software functions and function objects, the natural language can be used to annotate software object models.

Step 2: Associating the interface models

Confirm the input/output data of software function objects and associate them with the information of interface models, such as name, range, communication format, period, failure processing, etc.

Step 3: Modeling of software function process

a) Select “Data process” or “Control process” according to the software requirement specifications; the main difference between them is the description of the processing.

b) Divide the process of software function objects into indivisible sub-processes. Here, “indivisible” means an independent judgment condition for logic or time in software requirement, such as a sub-procedure for data or control.

c) Describe the processing mode of sub-processes according to the software requirement specifications including specific behavior, such as control processing, time processing, failure processing, redundancy processing, etc.

d) List sub procedures: confirm the conditional execution relationship of sub-processes, according to the judgment conditions; confirm the array of sub-processes sequential or concurrent according to the control relation between sub-processes; confirm control direction between sub-processes according to their relation.

Step 4: Review

Review whether all the software functions, input or output data processes are covered entirely. If “Yes”, then finished it; if “No”, repeat the above-mentioned steps.

Software function modeling can be defined as set $F$, $F = \{ (F_{\text{Name}}, f) | f \in F \times F \rightarrow FO \} = \{ F_{\text{Name}}, FI, FO, FP \}$. $F_{\text{Name}}$ means function name, FP means function processing, FI means set of function input interfaces, FO means set of function output interfaces.

(1) Set FI as function input interfaces

Set each function input interface $FI_i$ can be expressed as $\{FI_{\text{range}}, FI_{O_i}, FI_{num}\}$.

- $FI_{\text{range}}$: the range of function input interface data, which is the sub set of the input interface data value set $I_{\text{value}}$, $FI_{\text{range}} = \{UP, LOW\}$. $UP$ means upper limit value, $LOW$ means lower limit value.
- $FI_{O_i}$: Set of function output interfaces mapped from the $FI_i$ function input interface via function processing $FP$, $FI_{O_i} \in FO$.
- $FI_{num}$: number of function input interfaces

(2) Set FO as function output interfaces

Set each function output interface $FO_i$ can be expressed as $\{FO_{\text{range}}, FO_{O_i}, FO_{num}\}$.

- $FO_{\text{range}}$: the range of function output interface data, which is the sub set of output interface range or in accordance with output interface value range $O_{\text{value}}$, $FO_{\text{range}} = \{UP, LOW\}$.
- $FO_{O_i}$: set of function input interfaces mapped into the $FO_i$ function output interface via function processing $FP$, $FO_{O_i} \in FI$.
- $FO_{num}$: number of function output interfaces

(3) Set FP as function processing procedure

Function processing FP can be recorded as $FP = \{P_i, P_o, P_{\delta}, P\}$

- $PI$: the data processing of input interface $FI_i$ in function $F_i$, $PI$ can be described by combination of constraint symbols and essentials in $FI_i$.
- $PO$: the data processing of output interface $FO_i$ in function $F_i$, $PO$ can be described by combination of constraint symbols and essentials in $FO_i$.
- $PJ$: set of judgment nodes of function $F_i$. Judgment nodes can be described by combination of constraint symbols and specification essentials in $FI_i$ and $FO_i$.
- $P$: the processing of function $F_i$, include conditional execution, sequential execution, concurrent execution. Conditional execution can be defined with judgment nodes together.

3.4 Modeling method and process of operating state
The process of operating state modeling is as follow:

Step 1: Modeling of operating state objects

Confirm the operating state and the state boundary according to the system specifications. Referring to the software function models, obtain the relevant information associated with the state, including the functions required to be performed or not.

Step 2: Modeling of the state changing condition

Confirm the conditions of states entry, exit and change. Obtain the data information relevant to the state transition conditions, and then describe these logic judgment factors which are composed by data or time information.

Step 3: Combination of the state objects and conditions

Combine the system operating state objects and accordingly changed conditions into the scenes which have different perspectives, granularities and levels.

Operating state modeling can be defined as set $S$, $S = \{S_{\text{name}}, S_{\text{Func}}, S_{\text{tranto}}, S_{\text{tranfrom}}, S_{\text{condition}}, S_{\text{I}}, S_{\text{O}}\}$.

- $S_{\text{name}}$: name of operating states
- $S_{\text{Func}}$: set of software functions which operating in state $S_i$
- $S_{\text{tranto}}$: the subsequent states of the states $S_i$, $S_{\text{tranto}} \in S$
- $S_{\text{tranfrom}}$: the previous states of the states $S_i$, $S_{\text{tranfrom}} \in S$
- $S_{\text{condition}}$: the changing condition of state, can be described by combination of constraint symbols and essentials in $F_i$.
- $S_{\text{I}}$: set of external input interfaces related with state $S_i$
- $S_{\text{O}}$: set of external output interfaces related with state $S_i$

4 SFMEA assistant design experimentation

The objective of this research work is to enlighten the use of SFMEA assistant design for a control system through the development of an experiment. The experimentation of SFMEA assistant design methodology proposed in this paper is based on an airborne electronic control system, which is constituted by controller, sensors, mechanical transmission device and electrical equipment. The main functions of this system include receiving signals and data from airplane cabin and sensors, after processing, sending control commands to mechanical driving devices, in order to accomplish tasks in various operating states. Based on these functions, this experimentation will respectively build models for interface, software functions, operating states. Finally, SFMEA result will be given.

• Interface model

Considering airborne electronic control system, the external devices have interactive relationships with the system include airplane cabin, position sensors, transmission mechanism and electrical equipment. Involved external input/output interfaces include connection commands, mechanical position information, control instructions for transmission mechanism and power signal. The connected relations between the airborne electronic control system and others are shown in figure 2.

![Interface Diagram of the Control System](image-url)
Next, we should analyze data essentials to each input and output interface, and build interface model. Using charts to describe the data essentials of interface models is highly recommended. As an example, here only gives a chart of the mechanical position interface (shown in Table 1).

• Software functions model
The system software has three main function units including servo control, failure detection, limiting position protection. By analyzing the input/output and data processes of these units, we can build three software function models. Using flow-process diagrams to describe the software function models is recommended. Here takes a graphical description of servo control function unit as an example (shown in Figure 3).

• Operating state model
System operating states, related to servo control, failure detection, limiting position protection, include initialization, control administration and stop. Then build model State Transition Diagram that can be used to describe system operating states conveniently (shown in Figure 4).

The SFMEA analysis can be developed with the models of interfaces, software functions and operating states. Due to space limitation, this paper only demonstrates sectional processes and results. Here take servo control function as an example. Via analyzing the data essentials of the mechanical position interface model, we found that redundancy design in the model miss the precaution while failures existed in both A and B channels. Put this possible failure mode into the processing procedures and outputs of servo control function model, the following results were given in Table 2.

![Figure 3: Model of Servo Control Function Unit](image1)

![Figure 4: Model of Operating States](image2)
Table 1: Model of Mechanical Position Interface

| Name               | Mechanical Position Interface |
|--------------------|-------------------------------|
| Type               | analog signal                 |
| Source             | Sensors                       |
| Destination        | Airborne Electronic Control System |

Data elements

| Data name/ID | Position Data |
|--------------|---------------|
| Data type    | analog signal Float 32 |
| Data range   | [1, 99]       |
| Data unit    | %             |
| Time requirement | Period 20ms   |
| Fault diagnosis | Control loop diagnosis |
| Default value | 1%            |
| Redundancy    | Dual redundancy of A and B channels; Switch to another after one is failure |

Comments

Environment

Unstable factors from changes of sensors working environment

Table 2: SFMEA for Servo Control Function

| Servo Control Function | Failure modes                                      | Failure causes                                                                 | Failure effects | Precautions                                      |
|------------------------|----------------------------------------------------|--------------------------------------------------------------------------------|-----------------|--------------------------------------------------|
|                        | Error result of Control law computation            | Connection command is effective, but failures existed in both A and B channels of “position data” | Servo control failure | Add alarm mechanism when both A and B channels failure |
|                        | continues output of default control current value | Connection command is effective, but failures existed in both A and B channels of “position data” | Servo control failure | Add alarm mechanism when both A and B channels failure |

5 Conclusion

SFMEA is a bottom-up analysis of software failure modes within system. It is used to identify potential weaknesses, so that they can be disposed in early phases of software development. By taking SFMEA analyzing procedure as a breakthrough point, this paper proposed an assistant design method of SFMEA based on system requirement modeling. This method would remedy the deficiencies in analyzing complex control system compared with traditional SFMEA method. In this way, SFMEA would be applicable to complex control systems. Meanwhile, a more automated SFMEA would be possible.

Acknowledgements

This work was supported by the National Foundation “JSZL2015610C002”.

References

[1] Goddard P L. “Software FMEA Techniques”, Reliability and Maintainability Symp, pp. 118-123, (2000).

[2] Ozarin N. “Failure Modes and Effects Analysis during Design of Computer Software”, Proceedings of International Symposium on Product Quality and Integrity, pp. 201-206, (2004).
[3] Tribble A C, Miller S P. “Software Safety Analysis of a Flight Management System Vertical Navigation Function-A Status Report”, Proceedings of the 22th Digital Avionics Systems Conference, pp. 12-16, (2003).

[4] Zeng F P, Yang S K, Lu M Y. “Study on Computer Aided Design for System Software FMEA”, Journal of Computer Science, 9, pp. 106-109, (2009).

[5] Dillibabu R. “Development and Application of SFMEA Model to Software Testing Environment”, International Journal of Industrial Engineering Research and Development, 4, pp. 61-72, (2013).