LETTER

Software FMEA for Safety-Critical System Based on Co-analysis of System Model and Software Model

Guoqi LI[†a)], *Member*

SUMMARY Software FMEA is valuable and practically used for embedded software of safety-critical systems. In this paper, a novel method for Software FMEA is presented based on co-analysis of system model and software model. The method is hopeful to detect quantitative and dynamic effects by a targeted software failure. A typical application of the method is provided to illustrate the procedure and the applicable scenarios. In addition, a pattern is refined from the application for further reuse.

key words: *Software FMEA, safety critical system, co-analysis*

1. Introduction

As early as in 1949, Procedures for conducting FMEA (Failure Mode and Effects Analysis) were described in US armed forces military procedures document MIL-P-1629 [1]. In 1979, Reifer firstly introduced FMEA into software analysis [2]. Software FMEA is valuable and practical for embedded software engineering. For example, at early stage of embedded software lifecycle, it is used for determine the critical level of the software and at verification stage, it is valuable for efficient and effective testing.

John B. Bowles and Chi Wan presented a completed demonstration of software FMEA in their paper in 2001 [3], which is a good tutorial for software FMEA and was referred frequently by related papers. However, the object being analyzed is a simple experimental system, so the method is limited for large and complicated software in practical industry of nowadays. Recent years, researches on software FMEA is more active than ever. Chris Price and Neal Snooke provided a method for automated Software FMEA [4], which is mainly for source code analysis. Many researchers show enthusiasm on UML aided method [5], [6] and model-based method [7]. Failure mode database [8] and knowledge oriented methods [9] are also well invested. What's more, tools are developed to aid Software FMEA [10].

However, the current status and heritages of Software FMEA are still not satisfying for practical requirements. In this paper, we present a novel method based on co-analysis of system model and software model to enhance the analysis in 3 aspects:

1. It is hopeful to provide not only qualitative but also quantitative effects of failure;
2. Figure out dynamic effects of failure;
3. Suit to large and complicated embedded software and systems.

Our method has benefit a lot from researches on hardware and software co-design [11] and system FMEA [12]. In the following sections, we first describe the method. Then, a typical application of the method is provided to clarify the procedure and the applicable scenario. Finally, we draw conclusions and address our future works.

2. Procedure of Software FMEA

To do FMEA, no matter for hardware or software, the procedures are the same [13]:

1. Identification of systems and functions;
2. Identification of failure modes;
3. Determination of effects of failure modes;
4. Identification of possible causes;
5. Documentation and risk reduction.

The novel method is the same as the traditional ones in the above-mentioned steps 1,2,4 and 5, except that in step 1, both of the system models and software models must be provided by the developers, which is practical due to the popularity of the model-based development in safety-critical embedded systems.

During the analysis, System/software is analyzed from bottom to top. Analyzer recognizes failure modes at low level, traces their transiting and finds their effects on higher levels. For step 2, to identify the failure mode of a software component, many heuristic rules are available. Besides, failure mode database and expert system aided methods are also provided.

For step 4, Snooke states that there are three causes of failure [14]:

1. Abnormal value input to the software from its environment;
2. Failure in the hardware upon which the software is executed;
3. Logical/algorithmic/semantic error in the implementation code (a bug).

The third step "Determination of effects of failure modes", practically speaking, is the hardest one, since this step will be various according to specific applications. For

Manuscript received June 30, 2011.
Manuscript revised March 28, 2012.

[†]The author is with the Faculty of School of Reliability and System Engineering, Beihang University, Beijing China
a) E-mail: gqli@buaa.edu.cn

DOI: 10.1587/transinf.E95.D.3101

software FMEA, the potential failures are captured by software or originate from software. They are disguised, combined and ultimately affects system behaviors. Finding out the effects is tedious and boring. The novel part of our method is for the step.

2.1 Why do We Need a Novel Step 3?

To find out effects of failures, original method is based on informal system models, such as descriptions of system in natural language. It is highly subjective and dependent on the skill of the practitioner and the analysis will be unlikely complete, consistent, or error free. Therefore, people provide model-base method [7]. The main idea of the method is to combine software model, system model and fault models to one extended model and find the effects of the failure by simulation or model checking. However, system developers and software developers share different knowledge background and different modeling tools. It is difficult to combine models together, especially for large scale embedded systems. So, the analysis is usually conducted by ignoring the details of the implementation and obtains a qualitative analysis. However, we need a quantitative analysis to meet the challenge of the market and the requirements of increasingly rigorous safety.

2.2 Main Idea and Procedure of Step 3 in the Novel Method

Firstly, the novel method accepts the fact that it is impossible to combine system and software models, which are developed and maintained separately. Co-analysis of the two kinds of models is aided by traceability.

In many software standards for safety-critical systems, such as DO-178B [15], traceability between high level and low level requirements is required. High level requirements describe the software to do “What” for system and low level requirements describe “How” the software completes the transactions listed in high level requirements. Figure 1 illustrates the traceability. A high level requirement usually is a transaction and a low level requirement is usually a software module/component, such as a function. A high level requirement could be traced to many low level requirements and vice versa. For example, a transaction of high level may be implemented by many functions and a function may be invoked by different transactions. Consequently, the two levels requirements are from different points of view. High level requirements are stories about system. On the contrary, Low level requirements are closed to source code. If the low level requirements are well modeled, many tools could translate low level requirements into source code automatically.

Usually system and software are organized by different models. High level requirements are talking about system and could be mapped to system model. Contrarily, low level requirements could be mapped to software models. System model is defined as an abstraction of a system. A typical system model is a block diagram of a control system, Fig. 2 is an example. Each block in the block diagram establishes a relationship between signals. System models are based on rules of physical world, for system itself is designed and built to change something in the real world. Essentially, a block diagram of a control system is a series of differential equations, representing features of system. On the other hand, a software model is a description of structure or dynamic features of software, such as a data flow chart or a state machine. Software models are based on Turing machine, for software itself is designed and built to process information on computer.

So, to find out effects of a failure, we should:

1. Find out its effects on software according software models;
2. Trace the effects to system modules aided by traceability of high level and low level requirements.
3. Figure out the quantitative and dynamic effects on system according to system models.

Even though such method is believed to be practical for engineering, we need to look further into its theoretical basis, whereas “pattern”, a general reusable solution to a commonly occurring problem within a given context, which is successful in the fields of object-oriented design [16], could be resorted to. It is a description or template to be used in many different situations. We could accumulate patterns of co-analysis of system model and software model for FMEA in the future. The following section is an effort for this purpose.

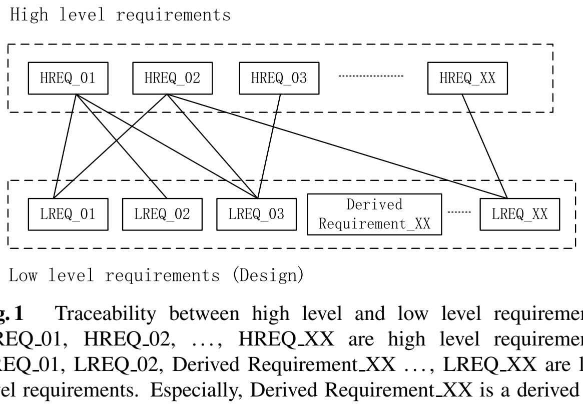

Fig. 1 Traceability between high level and low level requirements. HREQ_01, HREQ_02, ..., HREQ_XX are high level requirements. LREQ_01, LREQ_02, Derived Requirement_XX ..., LREQ_XX are low level requirements. Especially, Derived Requirement_XX is a derived requirement which could not be traced to any requirements on high level.

3. A Typical Application

3.1 Description of the System being Analyzed

The system to be analyzed, shown in Fig. 2 is a control system with PID (Proportional, Integral and Derivative) controller. The PID controller, which consists of proportional, integral and derivative elements, is widely used in feedback control of industrial processes. A PID controller calculates an “error” value $e(t)$ as the difference between a measured

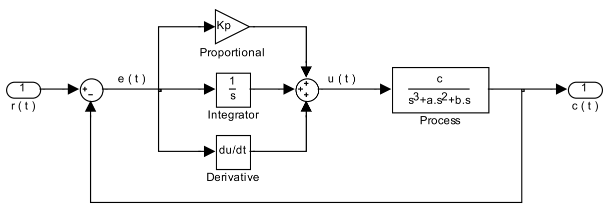

Fig. 2 The block diagram of a control system with PID controller.

Table 1 Modes and corresponding parameters.

Mode	a	b	c	P_K	T_I	T_D
Mode_1	6	5	1	18	1.4000	0.3360
Mode_2	5.4	2	1	6.48	2.2217	0.5331
Mode_3	20.2	4	0.2	242.4	1.5708	0.3770

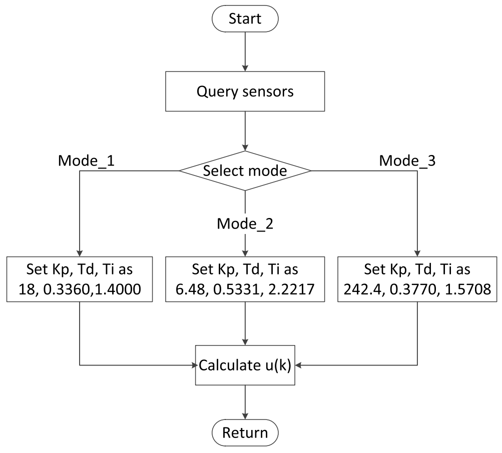
Start
Query sensors
Select mode
Mode_1
Mode_2
Mode_3
Set Kp, Td, Ti as 18, 0.3360,1.4000
Set Kp, Td, Ti as 6.48, 0.5331, 2.2217
Set Kp, Td, Ti as 242.4, 0.3770, 1.5708
Calculate u(k)
Return

Fig. 3 The Flow chart of the software module/component to implement PID controller.

process variable $c(t)$ and a desired setpoint $r(t)$. The controller attempts to minimize the error by adjusting the process control inputs. $e(t) = r(t) - c(t)$ is the input of PID controller and $u(t)$ is the output. The Process is the object to be controlled.

The transfer function of the PID controller is:

$C(s) = K_P\{1 + \frac{1}{T_I s} + T_D D(s)\}$

Where K_P, T_I, and T_D are parameters of proportional, integral and derivative respectively.

The transfer function of the Process is:

$P(s) = \frac{c}{s^3+as^2+bs}$

Where a, b, c and K_P, T_I, T_D are determined by specific mode. Table 1 lists the modes and corresponding parameters.

The block diagram is a system model. Blocks in the model are implemented by software or hardware. Specifically, in this case, the $P(s)$ is implemented by hardware and $C(s)$ is by software. All the blocks cooperated to complete missions assigned to the system.

A function written in C language is designed to implement the PID controller. Figure 3 shows its flow chart. Initially, sensors are to be checked by "Query sensors" to clarify the condition of the system. The parameters of $P(s)$ are changed by the the condition automatically. Based on the data of sensors, correct model is selected, then corresponding K_P, T_I, and T_D are assigned. Lastly calculate the output of the PID controller $u(t)$ according the model in Fig. 2. In fact, the $u(t)$ is calculated by difference equations according to specific PID controller arithmetic. Details are here omitted for the maturity of this technology as well as the fact that errors are generally impossible.

3.2 Procedure of the Analysis

If a fault, which is quite possible, in one of the sensors appeared, and provided error data for "Query sensors", what will be the effects to the system? We must pay due attention to this problem.

From the software model, shown in Fig. 3, we can see that if "Query sensors" get error data, "Select mode" would assign wrong data for K_P, T_I, and T_D. From the traceability between the system model and the software model, it is clear that the K_P, T_I, and T_D in Fig. 3 correspond to the coefficient of the proportional, integral and derivative elements respectively in the system model, shown in Fig. 2. Even though the information of the traceability is not explicitly described in the previous subsection, it is generally required in safety critical system development.

Now, let's turn to the system model and find out what the effects of error assignment of K_P, T_I, and T_D in the system model. Figure 4, Fig. 5 and Fig. 6, show the step response of every mode of the system with every possible PID controllers. The first column of the figures are situations in which there are no PID controller. The second column of the figures are correctly matched PID controllers. Other columns are error matched. From the figures we can see:

1. When $P(s)$ in Mode_1, if error data of sensors leading error matched PID controller, there will be no damage and system will work well.
2. When $P(s)$ in Mode_2, if error data of sensors leading error matched PID controller, the effect it produces to the system will depend on which mode PID controller is for: if it is for Mode_3, there will be no damage; but if Mode_2, there will be a noticeable shock before the system becomes stable and thus the system will be damaged.
3. When $P(s)$ in Mode_3, if error data of sensors leading error matched PID controller, no matter PID controller is for Mode_1 or for Mode_2, the system will be unstable and the output of the system is in danger.

The analysis result is summarized in Table 2.

3.3 The Obtained Pattern for Reuse

The scenario of the application is popular in practical engineering of safety-critical systems, such as a controller for jet engine, UAV or nuclear reactor, though practically there usually have more modes.

We call the pattern "Mismatch pattern", which has 3

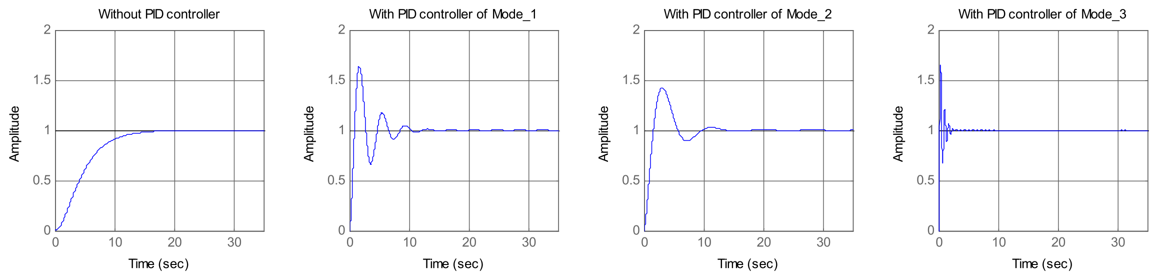

Fig. 4 The step response of system. $P(s)$ is in Mode_1.

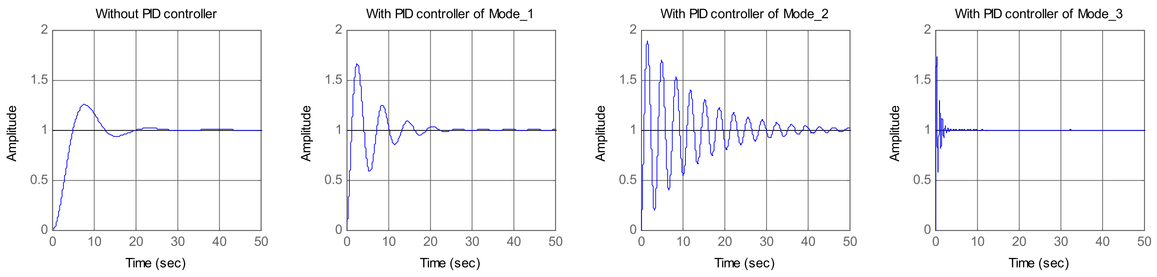

Fig. 5 The step response of system. $P(s)$ is in Mode_2.

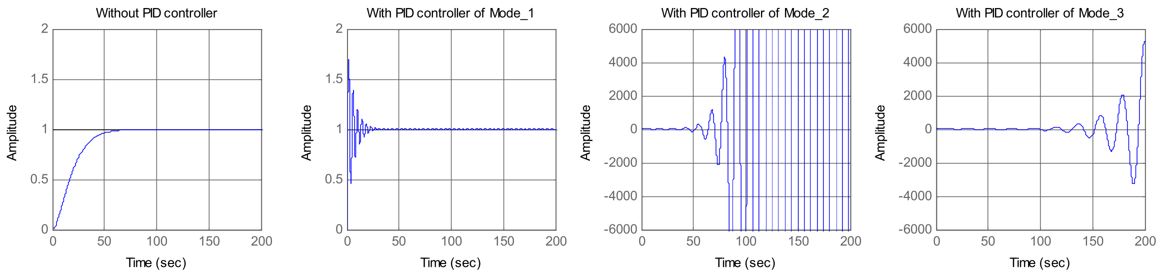

Fig. 6 The step response of system. $P(s)$ is in Mode_3.

Table 2 Software FMEA Table.

Items	Contents
`Component being analyzed`	PID controller
`Failure mode`	"Query sensors" on the flow chart of software model "PID" get wrong data
`Local Effect`	If there should be Mode_3 and recognized as Mode_1 or Mode_2, the PID module will give danger outputs.
`System Effect`	Step response of the system will be unstable, shown in Fig. 7.
`Causes`	Electric error of sensors or software logic errors when query sensors.
`Recommended measures`	Add additional protection module to monitor $c(t)$. If $c(t)$ is beyond threshold, ignore the PID module and set $u(t) = e(t)$.
`Remarks about tractability`	The data flow diagram shown in Fig. 3 maps to the blocks of proportional, integral and derivative in Fig. 2.

features:

1. object to be controlled has many modes.
2. controller is implemented by software.
3. Coefficients of the controllers are determined by sensors.

4. Conclusion and Future Works

The main contribution of this paper is to provide a co-analysis based method for determining effects of software failure on system. It is highly hopeful to find out the quantitative and dynamic effects of a software failure on the system with the method. In our future works, we will make deeper exploration on the topic and provide more patterns for co-analysis based software FMEA.

The C and Matlab source code used in Sect. 3 is available by email: gqli@buaa.edu.cn.

References

[1] "Procedures for performing a failure mode, effects and criticality analysis," U.S. Departmento fo Denfense, Nov. 1949.
[2] D.J. Reifer, "Software failure modes and effects analysis," IEEE Trans. Reliability, vol.R-28, no.3, pp.247–249, 1979.
[3] J.B. Bowles and C. Wan, "Software failure modes and effects analysis for a small embedded control system," Proc. Reliability and Maintainability Symposium, 2001.
[4] C. Price and N. Snooke, "An automated software fmea," Proc. In-

ternational System Safety Regional Conference, Singapore, April 2008.

[5] W. Wang and H. Zhang, "Fmea for uml-based software," Proc. 2009 WRI World Congress on Software Engineering, Los Alamitos, CA, USA, pp.456–460, IEEE Computer Society, 2009.

[6] H. Hecht, X. An, and M. Hecht, "Computer aided software fmea for unified modeling language based software," Annual Symposium of Reliability and Maintainability, 2004.

[7] A. Joshi, M. Whalen, and M.P. Heimdahl, "Model-based safety analysis final report," tech. rep., NASA, 2005.

[8] W.M. Goble and J.V. Bukowski, "Development of a mechanical component failure database," Reliability and Maintainability Symposium, 2007. RAMS '07. Annual, Orlando, FL, pp.451–455, Jan. 2007.

[9] X. Zhao and Y. Zhu, "Research of fmea knowledge sharing method based on ontology and the application in manufacturing process," DBTA, pp.1–4, 2010.

[10] J. Elmqvist and S. Nadjm-Tehrani, "Tool support for incremental failure mode and effects analysis of component-based systems," Proc. Conference on Design, Automation and Test in Europe, DATE '08, New York, NY, USA, pp.921–927, ACM, 2008.

[11] M. Ishikawa, G. Saikalis, and S. Oho, "Cpu model-based mechatronics/hardware/software co-design technology for real-time embedded control systems," IEICE Trans. Electron., vol.E90-C, no.10, pp.1992–2001, Oct. 2007.

[12] V. Motevalli and M.S. Mohd, "New approach for performing failure analysis of fuel cell-powered vehicles," International Journal of Automotive Technology, vol.10, pp.743–752, 2009.

[13] H. Pentti and H. Atte, "Failure model and effects analysis of software-based automation systems," tech. rep., STUK, 2002.

[14] N. Snooke, "Model-based failure modes and effects analysis of software," Proc. DX04, Carcassonne, pp.221–226, France, June 2004.

[15] RTCA, "Do-178b software considerations in airborne systems and equipment certification," 1992.

[16] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1994.