**Proceedings of the SICE Annual Conference 2018**
**September 11-14, 2018, Nara, Japan**

# A Proposal for Fault Tree Analysis for Embedded Control Software

Masakazu Takahashi[1†], Tomohiro Suzuki[2], Yunarso Anang[3], Reiji Nanba[4], and Yoshimichi Watanabe[5]

[1, 2, 5]Department of Engineering, University of Yamanashi, Yamanashi, Japan
(Tel : +81-55-220-{8585, 8492, 8651}; E-mail: {mtakahashi, stomo, nabe}@yamanashi.ac.jp)
[3]Dept. of Computational Statistics, Institute of Statistics, Jakarta, Indonesia
(Tel: +62-21-819-1437; E-mail: anang@stis.ac.id )
[4]Department of Civil and Environmental Engineering, Daiichi Institute of Technology, Kagoshima, Japan
(Tel : +81-995-45-0640; E-mail: r-nanba@daiichi-koudai.ac.jp)

**Abstract:** In this paper, we propose a method of Fault Tree Analysis for embedded control software. The characteristics of the proposed method are as follows; using FT templates corresponding to software instructions, using FT development rules using reverse software slicing technique, and tools that develop FT mechanically. As a result of applying the proposed method and the tool to the existing problem, we confirmed that the proposed method can develop FT equivalent to FT created by a trained technician.

**Keywords:** Embedded Control Software, Safety Analysis, Fault Tree Analysis,

## 1. INTRODUCTION

This paper proposes an adequate Fault Tree Analysis (FTA) method for a specific fault of embedded control software (ECSW) that is written in the C language and installed into industrial products. We only deal with ECSW written in the C language because many types of ECSW are still written in the C language [1].

This paragraph describes the technical terminology used in this paper related to software safety [2]. Nonconformance produced when software is being developed is referred to as defects. Such defects existing in software cause unexpected behavior. This behavior is referred to as a fault. As a result of leaving such a faulty state, the state becomes worsen, and the software fails to fulfill the required functions. This condition is referred to as a failure. Failure causes damage to the users and the environment and is referred to as an accident.

Today, the action of industrial products where high reliability is required, such as automobiles, industrial plants, aircraft, and space equipment, is controlled by ECSW. Whereas improvement of ECSW has increased the functionality of industrial products, it has also made such products more complicated. Complexity in ECSW has tended to cause defects and faults within the products that trigger failures. These failures have resulted in causing accidents.

Therefore, software developers have been required to adequately verify the reliability of ECSW, while removing defects (simple program bugs) that could cause accidents. To remove relatively simple defects, software testing techniques have been used. However, defects in ECSW are not only those of bugs but also those that could occur stochastically, those that could occur by the specific operational conditions, and those that could cause by the functional degradation of sensors. Therefore, safety analysis methods have become important for removing these complicated defects.

This paper proposes an FTA method to detect defects of ECSW written in the C Language. FTA that targets software is a method that analyzes the defects that could cause specific accidents. This method traces the faults from system level to program level via module level so as to clarify the fundamental defects. Faults that can occur to ECSW include, for example, the value of an electrical current variable becomes greater, and the value of an acceleration variable becomes greater. Defects include hardware failures and abnormal timing issues, such as frequent interrupts, and the timing of the invalid data update. FTA can also be used to detect simple defects. However, software testing technique can detect defects more efficiently. Therefore, this paper does not focus on detecting simple defects. The process to trace defects is graphically described using logic symbols. This graphic description has a tree structure so that it is called a Fault Tree (FT). The characteristics of the proposed method are as follows:

(1) Prepare FT templates in advance that correspond to the statements in C.
(2) Prepare FT development rules to develop FT by combining the FT templates.
(3) In accordance with the reverse execution sequences, FT templates corresponding to the statements are combined based on the FT development rules.

## 2. RELATED WORKS

Previous studies are divided into the establishment of standards associated with software safety and the examination of safety examination methods for ECSW.

First, let us focus on the establishment of standards associated with software safety for each industry. Various standards that have been established in the past include ISO 26262 regarding ECSW development in the automobile industry [3], GAMP5 in the pharmaceutical industry [4], ICE 62304 in the medical equipment industry [5], and DO-178C in the aviation industry [6], and JAXA JMR-001 in the space equipment industry [7]. The contents of these standards are complicated. In reality, it is difficult to enhance safety only based on the analysis depending on the experience of engineers.

† Masakazu Takahashi is the presenter of this paper.

Next, let us focus on the safety analysis methods. The safety examination methods are divided into methods to exhaustively examine failure possibilities in the design phase to clarify the causes, and methods to clarify the cause of specific failures in the operation phase.

As for the former, Takahashi et al. defined standard ECSW's failure modes by analyzing the existing pharmaceutical production facilities and proposed FMEA method using those modes [8]. Snooke et al. proposed code-level FMEA method by tracing the failure between ECSW's instructions [9]. Those FMEA methods became to be applied to ECSW development gradually. As for the latter, Leveson et al. proposed FT templates for fundamental instructions. FT was developed by combining the template [10]. Chen proposed a safety analysis method that described software as the finite state machine and conducted FTA to it [11]. However, it was difficult to apply those FTA methods to the ECSW, because those methods did not define FT development procedure.

## 3. PROPOSED FTA METHOD

This section describes the proposed method. Subsection 3.1 describes Leveson's FTA method for software which provides the fundamentals for this paper. Then, subsection 3.2 outlines the FT templates and FT development rules. Finally, subsection 3.3 describes the outline of the proposed method.

### 3.1 Software FTA Method Proposed by Leveson

This subsection describes Leveson's FTA method for software, the basis for this paper [10].

A software program consists of a set of basic statements, such as an assignment statement, a conditional branching statement, and a module (function) call statement. Therefore, FT templates that correspond to these statements are prepared in advance. Then an FT is developed by combining the FT templates using logic symbols along the reverse execution sequence. By repeating, this method clarifies the defect.

The following section outlines the FTA procedure. First, the target fault is determined. Faults that occur to ECSW are as follows; outputting an abnormal value, executing a function or a module at the wrong timing, and the impossibility of executing a function or module. Second, a statement that causes the target fault is identified. An action that may cause such an undesirable situation is called an event. Especially, an event that causes fault is called the top event, and events that are causes of top event are called the intermediate event. The intermediate events might include the followings: the inadequacy of the algorithm, an abnormal value input into the statement, and the execution of the statement at the wrong timing. In the case of the inadequacy algorithm, the algorithm is corrected. In the case of an abnormal value input into the statement, further analysis is conducted. In the case of the execution of the statement at the wrong timing, necessary correction is conducted, such as an execution cycle, and timing for disabling and enabling interrupts. Additionally, the relationships between the fault and the intermediate events are described by using FT templates and logic symbols. Third, a statement that causes intermediate events is identified and analyzed into lower intermediate events. In the same way, the relationships between upper and lower intermediate events are described by using FT templates and logical symbols. Those steps are repeated until the intermediate events can no longer be traced. The intermediate events which finally remain are the defects. However, the rules for tracing events and combining FT templates were not defined. Hence, the integrity of the developed FT depends on the engineer's skills. As a result, another engineer who has not enough skills cannot develop the adequate FT.

### 3.2 FTA using reverse slicing and development rules

This section outlines the fault occurrence process in ECSW, FT templates, and FT development rules.

**(1) Fault occurrence process in ECSW**

Let us consider how a fault occurs in ECSW. Suppose the following case that takes place: There is an ECSW that operates properly until a certain statement is executed, but when the next statement $I_0$ is executed, an event that differs from the normal condition occurs. This event is considered to be the cause of a fault and described as <$Event_0$>. As the statements are executed, the ECSW's status gradually becomes to differ from the normal condition. Suppose that <$Event_j$> occurs when $I_j$ is executed. Finally, suppose that <$Event_n$> occurs when $I_n$ is executed while this event is recognized as a fault. The chain of statement execution, from the execution of $I_0$ until the fault <$Event_n$> occurs after $I_n$ is executed, is shown in the description below.

$I_0$<$Event_0$>$I_1$<$Event_1$>...$I_{j-1}$<$Event_{j-1}$>$I_j$<$Event_j$>...$I_n$<$Event_n$>

Therefore, by regarding with <$Event_n$> and $I_n$, where the fault occurs, and $I_n$ as the starting point, tracing the chain of statement executions inversely can reach <$Event_0$> and $I_0$. Here, $I_{j-1}$ that is executed previously from $I_j$ can be obtained by inversely tracing the program slicing (hereinafter, slicing) results. Slicing is a method to extract all the statements in the program that affect the execution result of a particular statement in the program. A set of statements extracted is referred to as a slice. The slice includes statements that are data-dependent and that are control-dependent. Here, $I_{j-1}$ is data-dependent on $I_j$. This indicates that the values of variables set in $I_{j-1}$ might be referred to by $I_j$. $I_{j-1}$ is also control-dependent on $I_j$. This indicates that $I_{j-1}$ is a branch statement that includes $I_j$, or that $I_{j-1}$ is a loop statement that includes $I_j$. Accordingly $I_{j-1}$ is a statement that assigns values to variables used for $I_j$ or that satisfies the preconditions (the conditions satisfied before the statement's execution) for executing $I_j$. As for reverse tracing procedure, FT development rules that are preliminarily prepared can be used to trace execution procedure mechanically. The trace from <$Event_j$> to <Eventj-1> related to $I_j$ is prepared as the FT Templates. The templates can determine the change mechanically.

For all these reasons, the proposed method uses reverse program slicing and FT templates for tracing the statement's execution sequence of ECSW inversely. After the following subsection, from the understandability of the algorithm, the subscript numbers of Event and I are in reverse order. Namely, $I_0$ and <$Event_0$> indicate the point in time when a fault occurs, while $I_n$ and <$Event_n$> indicate the point in the time when the cause occurs.

**(2) FT templates**

This subsection describes the details about FT templates that are used for FT development. The C language has numerous statements. It takes long time to prepare FT templates for all the statements. Therefore, we prepared nine FT templates that are frequently used. In the case that a new statement appears, a new FT template is developed. As templates merely define the relationship between events, adding new FT templates do not affect other templates.

**(a) Assignment statement**

Fig. 1 shows the FT template for the assignment statement. This template indicates that assignment statement causes the event because the value assigned and/or the operand used causes the event.

**(b) Block if statement**

Fig. 2 shows the FT template for the block if statement. This template indicates that block if statement causes the event because one or more conditions cause the event. Moreover, this template also indicates that the event is caused when the i-th condition is satisfied and the i-th clause causes the event.

**(c) While statement**

Fig. 3 shows the FT template for the while statement. This template indicates that while statement causes the event because the statement itself is not executed because of failing to satisfy the repetition conditions and/or because of executing the repetition n times.

**(d) Function call**

Fig. 4 shows the FT template for the function call. This template indicates that this function call causes the event while failing to call the function and/or while successfully calling the function.

**(e) Interrupt**

Fig. 5 shows the FT template for the interrupt. This template indicates that interrupt causes the event because the interrupt occurs or did not occur. The former indicates that the event occurs when the interrupt occurs and the interrupt routine is executed. The latter can be divided into the case where an interrupt does not occur and the case where interrupts are disabled. Where an interrupt does not occur, the event occurs because no interrupt occurs and the interrupt routine is not executed. Where interrupts are disabled, the event occurs because the interrupts are disabled.

**(f) Global variables**

Fig. 6 shows the FT template for global variable. This FT template indicates that one or more global variables used at n locations in ECSW cause the event.

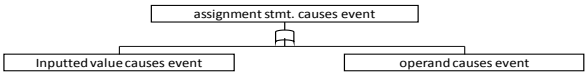


Fig.1 The FT templates for the assignment statement.

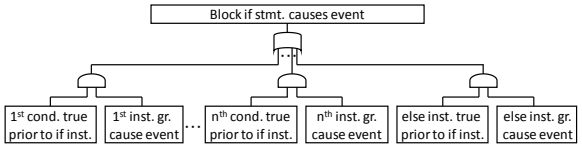


Fig.2 The FT template for the block if statement

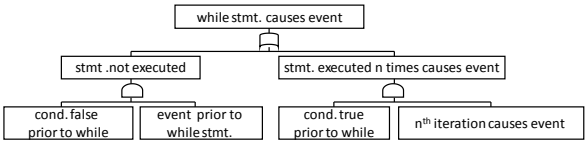


Fig.3 The FT template for the while statement

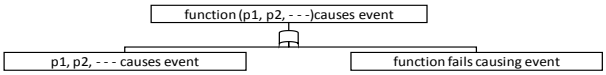


Fig.4 The FT template for the function call

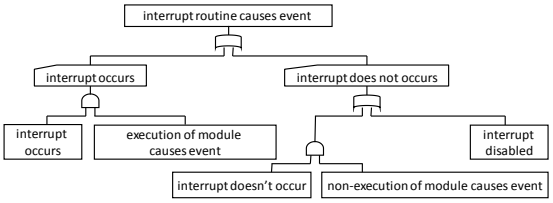


Fig.5 The FT template for the interrupt

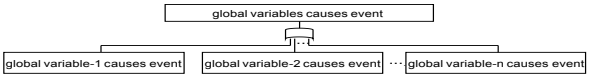


Fig.6 The FT template for the global variable

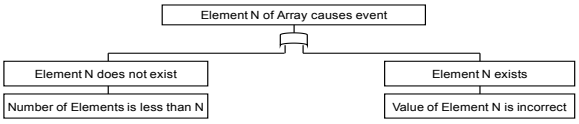


Fig.7 The FT template for the array

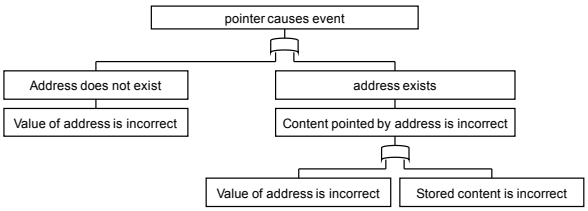


Fig.8 The FT template for the pointer

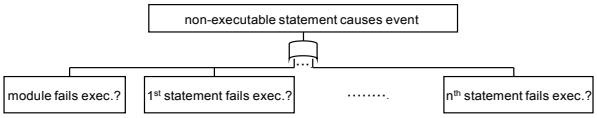


Fig.9 The FT template for the non-execution

## (g) Array

Fig. 7 shows the FT template for the array. This FT template indicates that the array causes the event because there is no n-th element (out of range, illegal index access) and/or the n-th element contains an improper value.

**STEP1**: Identify the statement that cause the top event (fault).
A) j = 0.
B) Define the target event, and describe it as the top event.
C) Identify the statement ($I_j$) that causes the top event.
D) Select a FT template that corresponds to $I_j$, and regard it the developed FT (initial state of the developed FT)
E) Fill the developed FT.
F) Identify the $Event_j$ when the $I_j$ causes the event.
G) Go STEP2

**STEP2**: Identify the statement(s) that is (are) previously executed.
Identify the statement (or statements) $I_{j+1}$ that is (or are) executed before $I_j$ by analyzing $Event_j$.
A) When $Event_j$ contains global variables.
   i. Identify where all variables (global and local variables) exist (as set), and decide the all statements $I_{j+1}$
   ii. Connect FT template for global variables to the developed FT, and Fill the developed FT using $I_{j+1}$.
   iii. Go STEP3.
B) When $Event_j$ contains Local variables.
   i. Identify where all variables exist, and decide the all statements $I_{j+1}$ that are sets the values to the local variables immediately before
   ii. Connect FT template for assignment to the developed FT, and Fill the developed FT using $I_{j+1}$.
   iii. Go STEP3.
C) When $Event_j$ contains global and local .
   i. Conduct a + b.
   ii. Go STEP3.
D) When $Event_j$ cannot trace any more.
   i. Finish FTA.

**STEP3**: Develop partial FT corresponding to the statement that cause the event
A) When $I_{j+1}$ is block statement (initial number of nest number is k=1.)
   i. When nest number k is greater than (>) nest number that trace target statement exists.
      a. FT template corresponding to statement that has the nest number k is connected with (added to) the developed FT.
      b. Fill the FT template corresponding to statement that has the nest number k.
      c. k = k +1.
      d. Return to top of STEP3.
   ii. When nest number k equals to (=) nest number that trace target statement exist.
      a. FT template corresponding to trace target is connected with (added to ) the developed FT.
      b. Fill the FT template corresponding to $I_{j+1}$.
      c. The content in FT template is set into $Event_{j+1}$.
      d. j = j + 1.
      e. Return STEP2.
B) When $I_{j+1}$ is not block statement.
      a. FT template corresponding to $I_{j+1}$ is connected with (added to) the developed FT.
      b. Fill the FT template corresponding to $I_{j+1}$.
      c. The content in FT template is set into $Event_{j+1}$.
      d. j = j + 1.
      e. Return STEP2.

Fig.10 FT development procedure

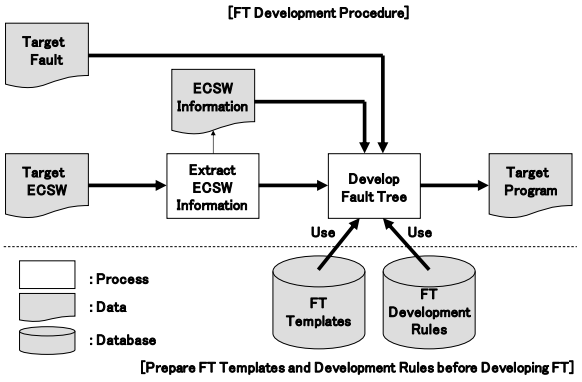


Fig.11 Outline of the proposed method

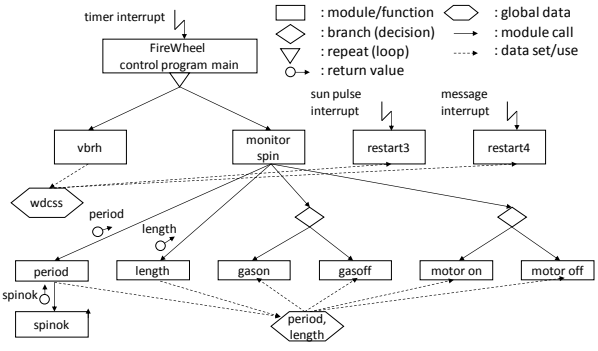


Fig.12 Overview of ECSW

```
000: int SUNP , MAGP ;
002: int DNCTR , DNMAX , THETA ;
003: int WDCSS , WDCTR , LASTP, WDLOST, L1 ,
L2 ;
009:
010: int SAMPLE ( int SAMPLE_ARG ) {
011:         return SAMPLE_ARG ;
012: }
013:
014: int PERIOD ( ) {
015:         int MS ;
016:         int SUN , MAG ;
017:         bool ifc1 , ifc2 ;
018:         ifc1 = SPINOK ( SUNP ) ;
019:         ifc2 = SPINOK ( MAGP ) ;
020:         if ( ifc1 == true ) {
021:                 MS = SUN ;
022:         }
023:         else {
024:                 if ( ifc2 == true ) {
025:                         MS = MAG ;
026:                 }
027:                 else {
028:                         MS = SUN ;
029:                 }
030:         }
031:         if ( MS == SUN ) {
032:                 return SUNP ;
033:         }
034:         else {
035:                 return MAGP ;
036:         }
037:}
```

```
050: void RESTART4 ( ) {
051:         if ( DNCTR != 1 ) {
052:                 DNCTR = DNCTR - 1 ;
053:         }
054:         else {
055:                 DNCTR = DNMAX ;
056:                 THETA = ( THETA + 1 ) %
057:         }
058:         WDCSS = WDCSS + 1 ;
059:         WDCTR = WDCTR + 1 ;
060:
061:         int sw_cond ;
062:         sw_cond = WDCTR % 16 ;
063:
064:         if ( sw_cond == 14 ) {
065:                 TL1 = SAMPLE ( L1 ) ;
066:         }
067:         else if ( sw_cond == 15 ) {
068:                 TL2 = SAMPLE ( L2 ) ;
069:         }
070: }
071:
072: void RESTART3 ( ) {
073:         SUNP = min ( LASTP , WDCSS ) ;
074:         int restkari ;
075:         restkari = ( SUNP + 64 ) / 128 ;
076:         DNMAX = min ( restkari , 255 ) ;
077:         DNCTR = DNMAX ;
078:         THETA = 0 ;
079:         LASTP = WDCSS ;
080:         WDCSS = 0 ;
081: }
```

Fig.13 Program code of ECSW -Excerpt-

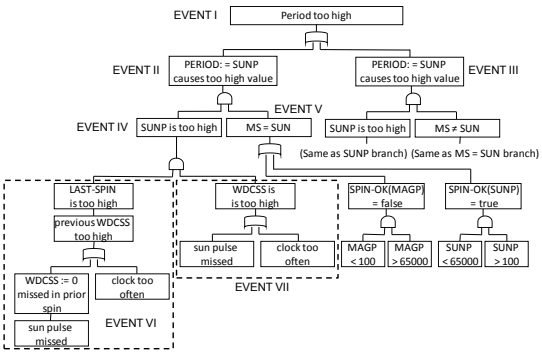


Fig.14 Leveson's FT for "PERIOD too high" -Excerpt-

## (h) Pointers

Fig. 8 shows the FT templates for the pointer. This template indicates that the pointer causes the event because there is no address that the pointer refers or the address referred by the pointer contains an improper value.

## (i) Non-Execution

Fig. 9 shows the FT template for the non-execution of statements. This template indicates that the non-execution of the target statement (n-th statement from the head of the function) causes the event because of the non-execution of the function or the non-execution of the i-th (i < n) statement (the process cannot reach the n-th statement because of an infinite loop etc.).

The FT template for the nested statements is considered. Where the trace target statement is included in control target statement, such as the block if and the while statement, the sequence of the inverse trace is from the trace target statement to the control target statement. As the control target statements including the trace target statement are considered to be one statement as $I_j$, and an FT is developed for this statement $I_j$. Hereinafter, these control target statements including the trace target statement are called block statements. Where a control target block statement is nested with multiple layers, the scope of the outermost nest is regarded as $I_j$. Those nests are numbered from external nest to internal nest (the outermost nest is 1, while the innermost nest is n).

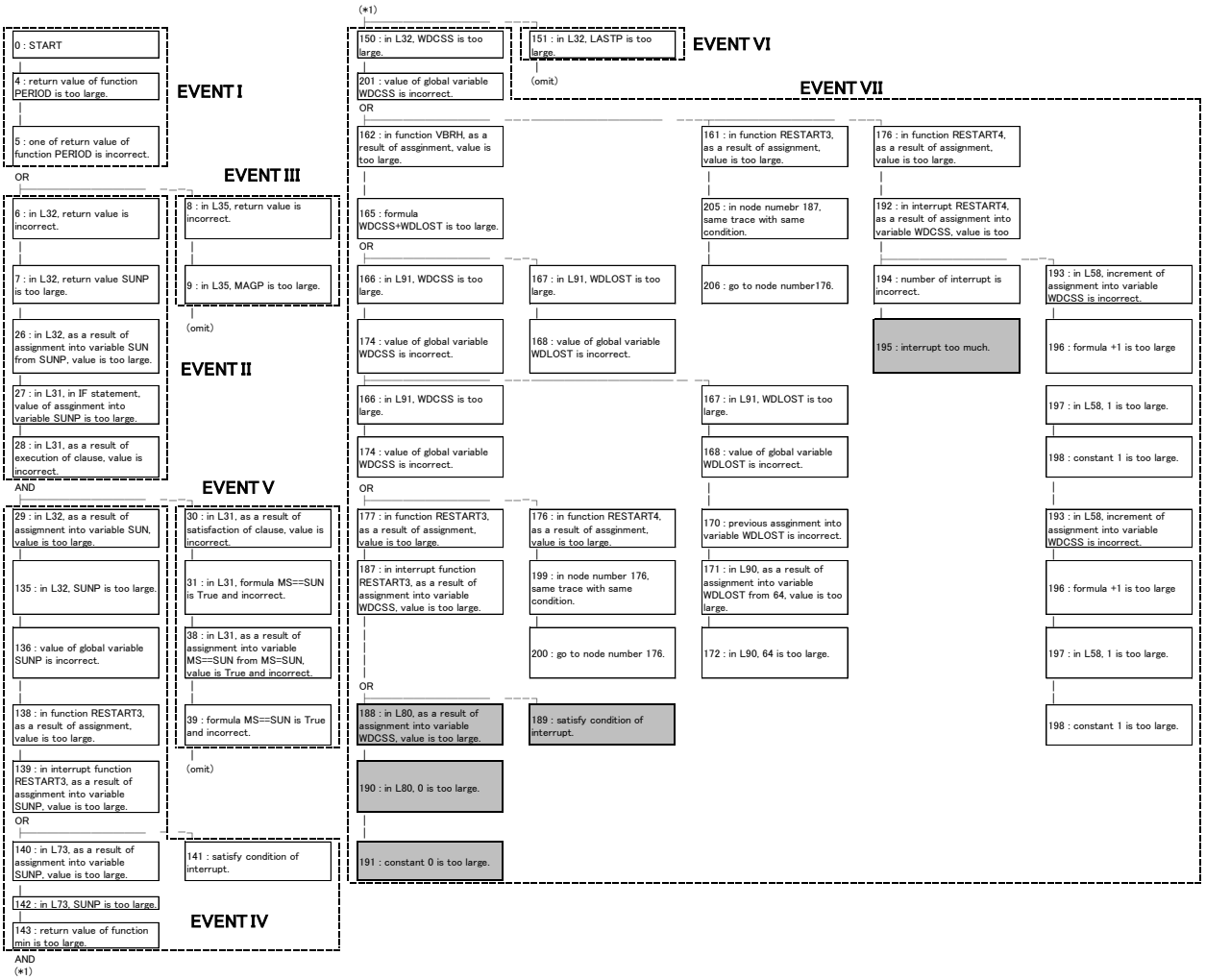


Fig.15 Our FT for "PERIOD too high" –Excerpt-

**(3) FT development rules**

Fig. 10 shows the proposed FT development rules. The FT development rules consist of three steps. In step 1, a statement that causes a fault is clarified. While the FT template that corresponds to the statement is defined as the developed FT, the event is described. In step 2, according to the valid scope of the variables described in the events included in the developed FT, FT templates for the global variables and the local variables are connected to the developed FT. This process can clarify all the statements that might cause the event concerned. In step 3, the FTs that correspond to all the statements that might cause the event are connected to the developed FT in order to clarify an event. Where statements are block statements, the FTs that correspond to the nested statements are connected to the developed FT in order to clarify the event. The development process for the nested statement then returns to step 2. These steps are repeated for the depth of the nest layers. Sometimes, the process gets out of the nests. Steps 2 and 3 are repeated until the event can no longer be traced. The events that are reached finally are identified as the defects.

### 3.3 Outline of the proposed method

Fig.11 overviews the proposed method. The proposed method consists of the preparation phase and the implementation phase.

In the preparation phase, the existing ECSW is analyzed to clarify statements that are frequently used (described in section 3.2 (2)), and FT templates are developed. The rules associated with FT development are also defined as FT development rules (described in section3.2 (3)). Those templates and rules are used in the implementation phase.

In the implementation phase, an FT is developed according to the FT templates and the FT development rules. First, the target ECSW structure is analyzed in order to extract ECSW information that is used, such as variables, functions, assignments, and various statements. Second, an FT is developed applying the FT templates and the FT development rules based on the target ECSW (source code), the target fault, and ECSW information. In this study, we tried to develop an FTA support tools for the proposed method. We developed these tools in C based on Windows 7 OS. FTA support tools consist of the ECSW analysis tool and the FT development tool.

## 4. APPLICATION AND EVALUATION

We evaluated the proposed method by applying it to Leveson's example (spinning satellite which spins too fast). Two engineers with three or more years of experience in failure analysis evaluated the Leveson's FT and an FT that is developed applying the proposed method (hereinafter, our FT) individually.

An over-speed in spinning produces excessive centrifugal force and damages the satellite's booms. With regard to this accident, Leveson implemented system-level FTA. It is confirmed that a fault can cause an accident when the value of the variable PERIOD became greater (PERIOD too high) or the value of the variable LENGTH became smaller (LENGTH too low).Fig.12 overviews the ECSW, and Fig.13 shows the program (that is rewritten in the C language because original program is written in Pseudo Pascal). Fig.14 shows Leveson's FT (Fig.14 is shown in reference [10] as Fig.8), and Fig.15 shows our FT (To improve readability, display parameters of Fig.15 were edited).

We compared and evaluated these two FTs of "PERIOD == SUNP" causes too high value (EVENT II in Fig.16)" in "PERIOD too high". Leveson's FT included 18 events, while our FT developed by the proposed method included 72 events. There were various reasons why our FT included a larger number of events. The analyzer of Leveson's FT omitted an interim progress, while the analyzer of our FT did not omit anything because of strictly applying the FT development rules, and assignment statements with multiple arguments were split into a combination of assignment statements with two arguments (for example, EVENT VII in Fig.14 includes only 3events, while EVENT VII in Fig.15 includes 40 events). We analyzed the correspondence between Leveson's FT and our FT. Groups of events included in our FT, which are indicated by areas enclosed by dotted lines, corresponds to each event in Leveson's FT. This analysis confirmed that both FTs have almost the same structure. Finally, we examined detected defects. Leveson's FT detected the following defects: the sun pulse was missed, and the clock ran too fast. On the other hand, our FT detected the following defects: RESTART3 did not satisfy the condition of the interrupt, and RESTART4 interrupted too frequently (Those are gray painted events). RESTART3 is triggered by the sun pulse interrupt. While the indication differs, the defect that the sun pulse was missed and the defect that RESTART3 did not interrupt are the same in meaning. Since RESTART4 is triggered by the clock interrupt, the defect that the clock ran too fast and the defects that RESTART4 interrupted too frequently are the same meaning. These confirmed that the proposed method detected defects properly.

For the fault of "LENGTH too low," as a result of the comparison, it is confirmed that the proposed method detects defects properly.

As a result, the proposed method can develop an FT properly, and the FT structure itself remains the same. Readability of our FT was more understandable than the Leveson's, because of no omission of middle events.

## 5. FUTURE WORKS

In this paper, we proposed an FTA method for ECSW and prototyped support tools. We applied our proposed method and support tools to the faults of Leveson's example. As a result, the proposed method could develop an appropriate FT and detected defects properly. Our support tools could implement FTA automatically. Use of our proposed method and support tools can develop an FT without any omission of interim progress. This provides the reproducibility of FT development that does not depend on the experiences or skills of the engineer. On the other hand, our FT makes indication complicated because of many intermediate events. We will develop the method to simplify indication using logical operation. Additionally, we will apply the proposed method and tools to large-scale ECSW, and we reflect the results to them.

## ACKNOWLEGEMENT

This research was supported by the Scientific Research Grant of the SUZUKI foundation "A Safety Analysis Method Cooperating FMEA, FTA, and HAZOP for Embedded Control Software."

## REFERENCES

[1] Information-Technology Promotion Agency, White paper of Embedded Software Development, 2017.
[2] N. Suzumura, "Section2 Understanding of Fundamental of Safety Design", Design Wave Magazine, No.109, pp.27-33, 2006.
[3] International Organization for Standardization, ISO26262 Road vehicles – Functional safety, 2011.
[4] International Society for Pharmaceutical Engineering: GAMP5 A Risk-Based Approach to Compliant GxP Computerized Systems, 2008.
[5] International Electro technical Commission, ICE 62304 Medical Device Software, 2006.
[6] Radio Technical Commission for Aeronautics, DO-178C Software Considerations in Airborne Systems and Equipment Certification, 2011.
[7] Japan Aerospace Exploration Agency, JMR-001 System Safety Standard, 2008.
[8] M. Takahashi, R. Nanba, and Y. Fukue, "A Proposal of Operational Risk Management Method Using FMEA for Drug Manufacturing Computerized System", Trans of SICE, Vol.48, No.5, pp.285-294, 2012.
[9] N. Snooke and C. Price, "Model-driven automated software FMEA", Proc. of Reliability and Maintainability Symposium, 7pages, 2011.
[10] N. Leveson and P. Harvey, "Analyzing Software Safety", IEEE Trans. on Software Engineering, Vol.9, No.5, pp.596-579, 1983.
[11] C. Chen, F. Zeng, and M. Lu, "A Verification Method for Software Safety Requirement by Combining Model Checking and FTA", Proc. of International Industrial Informatics and Computer Engineering Conference, pp1359-1364, 2015.