JOURNAL OF APPLIED INTELLIGENT SYSTEMS & INFORMATION SCIENCES
VOL. 1, ISSUE. 1, PP. 11-23, FEBRUARY 2020.
Available at: **www.journal.research.fanap.com**
DOI: **10.22034/jaisis.2020.102415**

DESIGN SAFE SOFTWARE VIA UML-BASED SFTA IN CYBER PHYSICAL SYSTEMS

Shahrzad Oveisi[1,2*], Mohammad Ali Farsi[3], Ali Kamandi[1]
[1]*Department of Algorithms and Computation, School of Engineering Sciences, University of Tehran, Tehran, IRAN*
[2]*Research Center, FANAP Co., Tehran, IRAN; shahrzad.oveisi@gmail.com*
[3]*Aerospace Research Institute (Ministry of Science, Research and Technology), Tehran, IRAN*

ABSTRACT

In cyber physical systems (CPSs), hazards can lead to injuries, deaths, destructions or loss of vital equipment or environmental damages. In these systems, software controls the behavior of mechanical and electronic components as well as their interactions; therefore, it plays a special role in creating system hazards and its safety plays a crucial role in a risk management process in cyber-physical systems. Many methods can be used to establish safety in software components of these systems and the software fault tree analysis (SFTA) is among the main methods. The main purpose of SFTA is to identify possible deficiencies in software requirements, design or implementation, which may result in undesirable events in software. On the other hand, unified modeling language (UML) is among the methods used for guaranteeing the construction of object-oriented software. In this paper, a sequence diagram generated in the software production process and the SFTA are used to evaluate safety. The proposed method can play a major role in designing safe systems. The proposed method for designing safe software is implemented in a real CPS and due to the use of uncertain data the reliability of the system is calculated using SFTA-based Fuzzy.

KEYWORDS: Software Safety, SFTA, UML, Cyber Physical Systems, Fuzzy.

1. INTRODUCTION

Cyber physical systems (CPS) are integrated systems of computation, networking, and physical processes. Embedded computers and networks monitor and control the physical processes, with feedback loops where physical processes affect computations and vice versa. Examples of CPS include aerospace systems, transportation vehicles and intelligent highways, robotic systems, intelligent environments and spaces, etc. (Rajkumar & Lee, 2012; Wu et al., 2011; Oveisi & Farsi, 2018).

The discovery of flaws has become more difficult with the increasing complexity of CPSs. Software are the cornerstone of CPS. The complexity of these software with millions of lines of code can cause dangerous consequences regarding the failure of these software. Final flaws in requirements, design, or execution of software can lead to unpredictable events at the integration level of software (Kim et al., 2010). Software failure can cause huge disasters, for example, on 4 June 1996, the Ariane 501 satellite launch failed catastrophically 40 seconds after initiation of the flight sequence, incurring a direct cost of approximately $370 million. The inquiry board report (IBR), clearly identifies the proximate cause of the disaster as a software failure (Dowson, 1997).

[*]*Corresponding Author*

RECEIVED: 25 NOVEMBER, 2019; **ACCEPTED**: 29 JANUARY, 2020; **PUBLISHED ONLINE**: 1 FEBRUARY, 2020

There are several ways to establish safety in hardware and software components. Fault tree analysis (FTA) is an approach to establish safety in software components, which can be implemented at different levels of software development for various purposes. FTA is a technique used in the field of safety (Bobbio et al., 2001). Initially, FTA was introduced in1960s with the primary goal of identifying events that could cause a system to reach a dangerous or insecure state (Towhidnejad et al., 2003). FTA is a powerful tool for the analysis of complex systems, focusing on how an event occurs and examining the hierarchy of the relevant causes. Therefore, FTA is often recognized as a top down approach (Oveisi & Ravanmehr, 2017; Vyas & Mittal, 2015).

According to studies conducted by the national institute of standards and technology, the cost of software error in the US economy is approximately 59.5 billion US dollars per year (nearly 0.6% of GDP). It is also estimated that over one-third of costs (i.e. 22.2 billion US dollars) can be cut with the improvement in infrastructure, including the use of safety analyzes and tests that detect, anticipate, and eliminate the flaw. Considering the above mentioned points, the application of a method to increase the software reliability seems to be essential in these systems (Romani et al., 2010).

Requirements analysis during all stages of software development plays the most important role in determining the safety of the whole software. If problems and errors leading to software failure are identified and their risks are reduced at this stage, then, the level of risk and system failure significantly decreases during later software development processes (Kamalrudin et al., 2018; Martins & Gorschek, 2017). A modeling language that is used in different phases of software development, including the requirements phase, is unified modeling language (UML). It also provides standardization in specifying, documenting, writing blueprint and visualizing the artifacts of software-intensive system under development (Kamandi et al., 2006; Hovsepyan et al., 2014; Paiboonkasemsut & Limpiyakorn, 2015). Several object-oriented designs use UML, which is standardized and commonly used by the software development community. UML applies a number of diagrams and views to describe software systems such as sequence diagram and use case diagram.

In this paper, the software fault tree analysis (SFTA) is used in requirements analysis phase of software design, and a method has been presented to confirm it using a sequence diagram. Then, the reliability of the system is calculated using fuzzy analysis.

2. Research Background

Software safety techniques play an important role in software development and are a valuable factor in the life cycle. Several studies have been so far conducted in different phases of software development cycle to increase the safety and reliability of software. A number of safety-related works using the mentioned methods are briefly cited below.

A manual four-step solution has been presented to integrate SFMEA and SFTA for the analytical process of use-case based requirements. In this approach, the UML use-case model is translated to a software fault tree for the analysis of safety based on system behavior. A text-based use-case model, known as use-case specifications, is used in this approach to produce the SFMEA to reduce the fault effects and results (Tiwari et al., 2012). Vyas & Mittal (2012) proposed another approach to extract safety requirements in a manual systematic form of use-case requirements. They have validated the results of their approach using a real case study in elevator control system. An effective method has been suggested to organize the information of fault tree and reuse SFTA information to produce the software fault tree (Romani et al., 2010). Four different phases have been designed for this method. In the first step, information of software fault tree is described by a semiformal method in the form of elements such as nodes, relations, target functions, and target software modules. Then, a knowledge base is constructed for information of software fault tree. For this purpose, different attributes of each node are considered. Finally, a reusable fault tree is automatically produced from the knowledge base using compliance between the texts with intelligent relations. This approach has been applied in the aerospace software systems.

The analysis process of requirements for software reliability is presented based on traditional safety analysis techniques and software system requirement specifications by Li et al. (2014). The main purpose of this process is to define a number of necessary requirements for software reliability. This approach is composed of four steps: specifying the criticality rate for each requirement, selecting requirement to be analyzed, utilizing the safety analysis techniques, and finally identifying the reliability attributes such as precision, compatibility, correctness, and maintenance.

The rules and algorithms for automatic transfer of a risk presented by fault tree to machine state diagram are suggested by Kim et al. (2010). The UML state diagrams are appropriate for the discrete behavior of specific subsystems. The main purpose of this method is to develop an algorithm to transfer fault tree risks to UML status diagram for safety analysis of system behavior. Oveisi & Ravanmehr (2017), after reviewing the major techniques of software reliability and safety in CPS, presented a software fault tree analysis (SFTA)-based approach for analysis of operational use-cases (UC) in a CPS system. Extends fuzzy fault tree analysis methodology to petrochemical process industry in which fire, explosion and toxic gas releases are recognized as potential hazards by Lavasani et al. (2014), Mahmood et al. (2013) explained and reviewed fuzzy theory application in reliability and maintenance analysis.

3. Proposed Approach

Although several methods have been suggested to increase safety and guarantee in the software development process, less attention has been paid to the evaluation of a method based on these inaccurate data given the uncertainty of the events leading to failures before a software is used. Accordingly, the work flow method proposed to enhance the reliability and to reduce the risk is presented in Fig. 1. The issues discussed in the proposed method are then evaluated below.

3.1 *Work flow of proposed method*

In the proposed method, the system and software requirements are first carefully evaluated, and a sequential diagram is developed by checking use cases and their diagrams. Then, according to the resulting diagram, the SFTA will be plotted and the reliability of the system is calculated based on the fuzzy method.

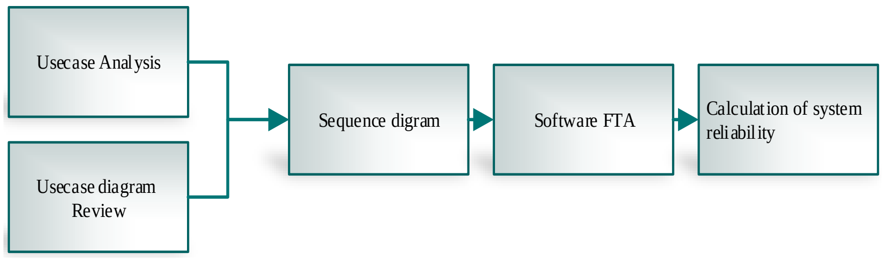

Fig. 1. Work flow of the proposed method

3.2 *SFTA using UML modeling*

The main objective of SFTA during software development is to identify the weak points in the specifications of requirements. To this end, the weak requirements are changed or other requirements are added. All the conditions having a direct impact on the safety of the system are identified. When requirements with safety considerations are identified, these requirements are tracked throughout the lifecycle development (Towhidnejad et al., 2003; Oliveira et al., 2016).

During the software design, the software is displayed in the form of a number of modules in which the interfaces, inputs, and outputs are specified using UML, and SFTA at this stage is applied to identify modules (objects, methods or functions) that can directly affect the safety of the system. It should be noted that the generation of fault trees for the system at requirements phase upon design selection is a more efficient choice than their generation at the implementation phase, since it is a heavy and hard task demanding intensive work during the latter phase.

Object-Oriented Design (OOD) can be chosen as a tool to apply SFTA at the design stage. There are two main reasons to select OOD: 1) several recent software designs use OOD, and the software are implemented by OO languages, 2) Recently, many OODs use UML, which is standardized by the software community and is commonly applied in this community. UML uses a number of views and diagrams to describe software systems.

In the next section, two sequence diagrams that are important and applicable in the software development process are selected, followed by the presentation of guidelines for converting the sequence diagram into a Fault tree.

3.3 Sequence diagram mapping to fault tree

1. The objects associated with the occurrence of a particular event are identified and each object can be displayed as an intermediate event. Each of these objects is displayed with the OR gate.
2. The functions that create another object are base events and OR gate inputs.
3. When an object is created by another object, the created one is shown as the base event or an intermediate event.
4. If a message is sent to multiple objects or received from them, then this transfer operation can be displayed by forwarding a logical gate represented by a triangle.
5. The problem of scheduling in the sequence diagram created a challenge for the development of Fault tree:

When the runtime of each of the functions is assumed in sequence diagram and some objects are simultaneously created, these objects are the middle events of the gate inputs.

- In one case, event *A* must be completed before the event *B* is started, and in the second case, it is not obligatory for the event to be completed before the event *B* is started. The scheduling problem is handled as follows:
- For a case in which event *A* must start before event *B* is complete, we can AND schedule the result of event *A* with event *B* to execute this order.
- To handle the operations to determine the sequence of two events that are simultaneously activated (e.g. it is necessary to activate the first event before the second event). For example, the object *A* prototypes the object *B*, and then both objects perform simultaneous activities. To illustrate this scheduling sequence, we show the object *A* with two sub-objects (*A*1, *A*2), where the sub-object *A*1 represents the activity that should be completed after the completion of an instance of object *B*, and *A*2 sub-object represents the beginning of activity of object *A* along with an instance of object *B*.

4. Quantitative Evaluation of Fault Tree

The quantitative probability evaluation of the final event in Fault tree is done in two ways: Boolean algebra and direct application of values. In the first method, Boolean algebra and the logic structure of Fault tree are used to combine and integrate the base events, and in the second method, the probability rules and the logical structure of Fault tree are used to combine and incorporate the base events. Given the advantages and disadvantages of the two methods mentioned by Billinton (1984), Boolean algebra method has been used in this paper.

In this method, the definition of the final event begins according to middle, incomplete, and median events, and each final event continues in terms of a base and incomplete event in such a way that all intermediate events are eliminated.

4.1 Fuzzy analysis

Using the developed fault tree and considering the fundamental events, fuzzy logic was used to determine the probability rate given that there was no information to determine the failure probability of the basic event (Khakzad et al., 2011; MIRI, et al., 2011; Renjith et al., 2010). These phases began by selecting a team of five relevant experts and ended by estimating the probability (Chen & Hwang, 1992). To estimate the probability, the center of gravity and Onisawa formulas were used. The implementation of fuzzy logic is described below.

To determine the weight importance of experts, the work experience and education criteria have been used. The scoring method of experts is shown in Table 1.

Table 1. Scoring based on expert characteristics

Row	Condition	Classification	Score
1	Experience in the specialized branch	1-2 year	1
		2-5 year	2
		>5 year	3
2	Education	BSc	1
		MSc	2
		Ph.D.	3
3	Field of study	Aerospace engineering	1
		Mechanical engineering	2
		Electrical engineering	3
		Computer hardware engineering	4
		Computer software engineering	5
		System reliability engineering	6
		Software reliability engineering	7
4	Title	Expert in the relevant department	1
		Academic staff of the relevant department	2
		Director, Project manager	3

Table 2. Weighted scores of selected experts

Expert number	Experience (year)	Education	Field of study	Title	Weighted index	Expert weight
1	4	MSc	Electrical engineering	Expert	8	0.166
2	6	MSc	Mechanical engineering	Expert	8	0.1666
3	10	Ph.D.	Computer software engineering	Academic	13	0.27
4	2	BSc	Aerospace engineer	Expert	4	0.083
5	15	Ph.D.	System reliability engineer	Project manager	15	0.312

The weights of experts evaluation criteria were determined after specifying these criteria in the previous phase. The final weight score of each expert is achieved by dividing the sum total of obtained scores by the total scores of all the experts participating in the study. The weight score of each expert based on the criteria set out in the previous stage is shown in Table 2.

Linguistic variables have been used to quantify expert opinions or to determine the weights of their views on basic events. The five linguistic variables include very low, low, moderate, high, and very high, which are summarized as *VL*, *L*, *M*, *H*, and *VH*. To fuzzify this section, a trapezoidal fuzzy number has been used. The corresponding membership functions are shown in Eq. (1).

$$FVH(x)=\begin{cases}0 & \text{x} \le 0.8\\ \dfrac{x-0.8}{0.1} & 0.8<\text{x} \le 0.9\\ 1 & 0.9<\text{x} \le 1\end{cases}$$

$$FH(x)=\begin{cases}\dfrac{x-0.6}{0.15} & 0.6<\text{x} \le 0.75\\ \dfrac{0.9-x}{0.15} & 0.75<\text{x} \le 0.9\\ 0 & \text{Otherwise}\end{cases} \tag{1}$$

$$FM(x)=\begin{cases}\dfrac{x-0.3}{0.2} & 0.3<\text{x} \le 0.5\\ \dfrac{0.7-x}{0.2} & 0.5<\text{x} \le 0.7\\ 0 & \text{Otherwise}\end{cases}$$

$$FL(x)=\begin{cases}\dfrac{x-0.1}{0.15} & 0.1<\text{x} \le 0.25\\ \dfrac{0.4-x}{0.15} & 0.25<\text{x} \le 0.4\\ 0 & \text{Otherwise}\end{cases}$$

$$FVL(x)=\begin{cases}0 & x>2\\ \dfrac{0.2-x}{0.1} & 0.1<\text{x} \le 0.2\\ 1 & 0<\text{x} \le 0.1\end{cases}$$

The weights of linguistic terms of experts whose views have been used in quantification are shown in Table 3.

Table 3. Weight of linguistic variables in quantifying the views of experts

Weight of linguistic terms				Language variable
0.2	0.1	0	0	Very low(VL)
0.4	0.25	0.25	0.1	Low(L)
0.7	0.5	0.5	0.3	Moderate (M)
0.9	0.45	0.75	0.6	High (H)
1	1	0.9	0.8	Very High (VH)

4.1.1. *Expert consensus*

Although there are different viewpoints on the probability of basic events, we must integrate the views into one view. For this purpose, there are several ways to integrate fuzzy numbers. One of the methods presented by Clemen & Winkler (1999) is as follows: to reach consensus among experts, the weight of each expert is multiplied by their linguistic variables according to the following Eq. (2):

$$M_i = \sum_{j=1}^{m} W_j A_{ij} \, (i = 1, \ldots, n) \tag{2}$$

where A_{ij} is the linguistic variable in relation to each basic event of i by the expert j, W_j weight of j expert, m number of base events, n number of experts, and M_i fuzzy number of experts consensus in relation to each basic event i.

4.1.2. *Defuzzification*

Defuzzification of fuzzy numbers is an important method for decision making in fuzzy environments. In this research, the center of gravity method has been selected for defuzzification, which is the most accurate defuzzification method developed by Sugeno in 1985 (Nguyen, 1999). Defuzzification of the trapezoidal fuzzy number A= (a_1, a_2,…, a_4) is obtained using the following formula (Romani et al., 2010):

$$X^* = \frac{\int_{a_1}^{a_2} \frac{x-a_1}{a_2-a_1} + \int_{a_2}^{a_3} xdx + \int_{a_3}^{a_4} \frac{a_4-x}{a_4-a_3} xdx}{\int_{a_1}^{a_2} \frac{x-a_1}{a_2-a_1} + \int_{a_2}^{a_3} dx + \int_{a_3}^{a_4} \frac{a_4-x}{a_4-a_3} xdx} = \frac{1}{3} \frac{(a_4+a_3)^2 - a_4a_3 - (a_1+a_2)^2 + a_1a_2}{(a_4+a_3-a_2-a_1)} \tag{3}$$

The number obtained from the previous step in relation to each basic event is equivalent to expert opinion and is still possible. At this stage, these numbers are defuzzified using the center of gravity model Eq. (2) of the trapezoidal formula.

4.1.3 *Conversion of possibility formula to probability*

The number obtained from the defuzzification step is still possible form. Since the fault tree accepts probability, the number achieved from the previous step should be converted from possibility to probability. For this purpose, the formulas presented by Onisawa are used.

$$FP = \begin{cases} \frac{1}{10^k} & \text{CFP} \neq 0 \\ 0 & \text{CFP=0} \end{cases} \; ; k = \left[\frac{1-CFP}{CFP} \right]^{\frac{1}{3}} * 2.301 \tag{4}$$

In this equation, FP is the probability rate of each basic event and CFP is the possibility number derived from the defuzzification stage.

5. Case Study

We applied the results of our approach to a part of a real CPS known as command issuing set, the architecture of which is shown in Fig. 2.

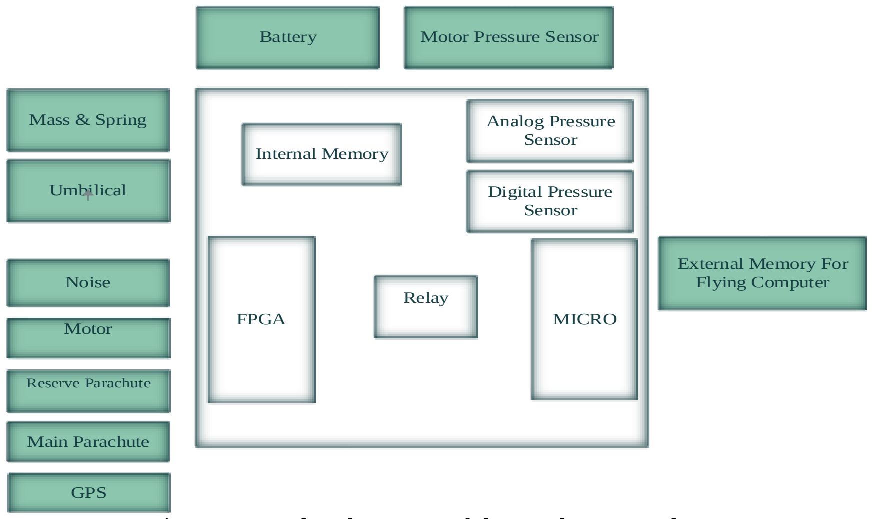

Fig. 2. General architecture of data and command unit

The most important goal of data and command unit is the timely release of commands for separation of nose, engine, and parachutes based on the simulated time and height during a sounding rocket flight. To begin working, this section needs to detect the start of movement, and in fact, it must receive the start signal. The start signal, which results from simultaneous cut of cord and compression of mass and spring switch, is a command to start the operations of the two system processors, which use data from pressure sensors and timeline of their internal timers to perform their operations.

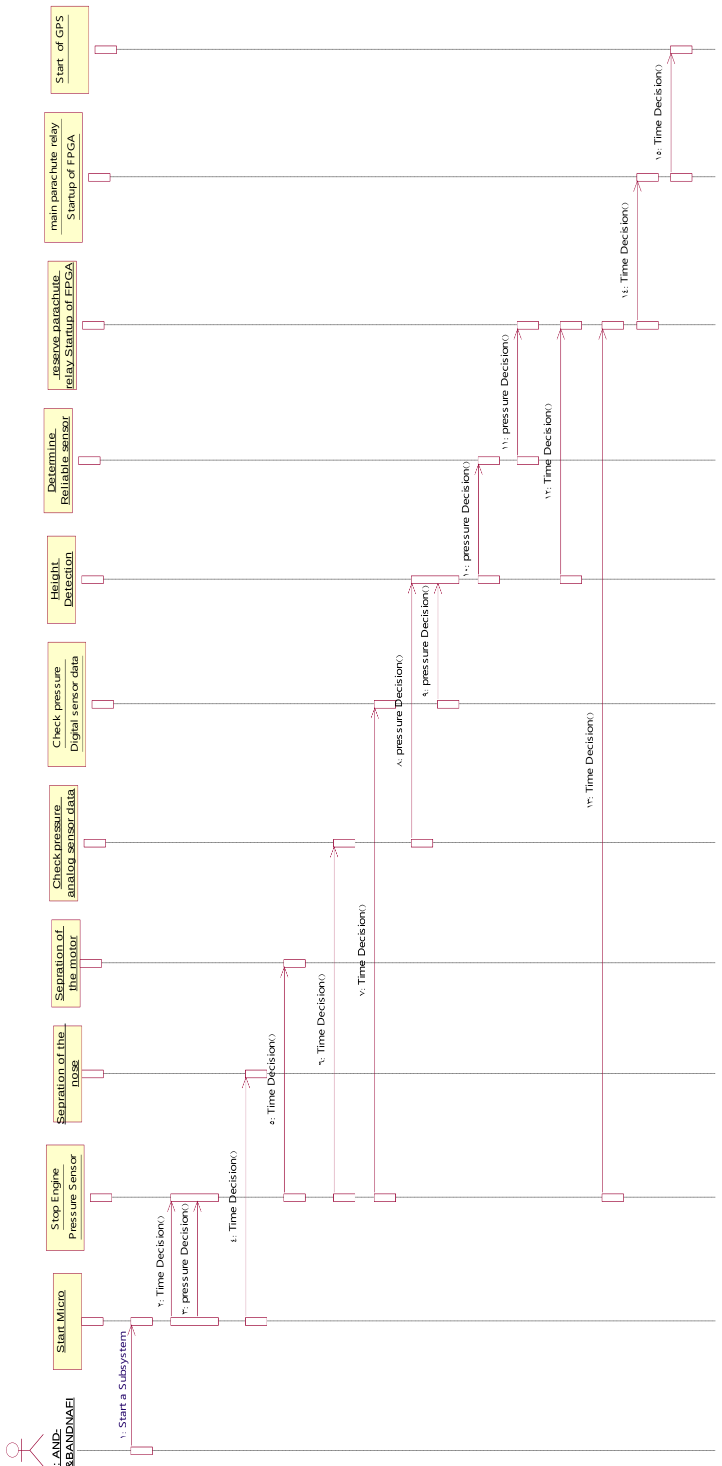
: AND-BANDNAFI
Start Micro
Stop Engine Pressure Sensor
Sepration of the nose
Sepration of the motor
Check pressure analog sensor data
Check pressure Digital sensor data
Height Detection
Determine Reliable sensor
reserve parachute relay Startup of FPGA
main parachute relay Startup of FPGA
Start of GPS
١: Start a Subsystem
٢: Time Decision()
٣: pressure Decision()
٤: Time Decision()
٥: Time Decision()
٦: Time Decision()
٧: Time Decision()
٨: pressure Decision()
٩: pressure Decision()
١٠: pressure Decision()
١١: pressure Decision()
١٢: Time Decision()
١٣: Time Decision()
١٤: Time Decision()
١٥: Time Decision()

Fig. 3. Sequence diagram of data and command unit

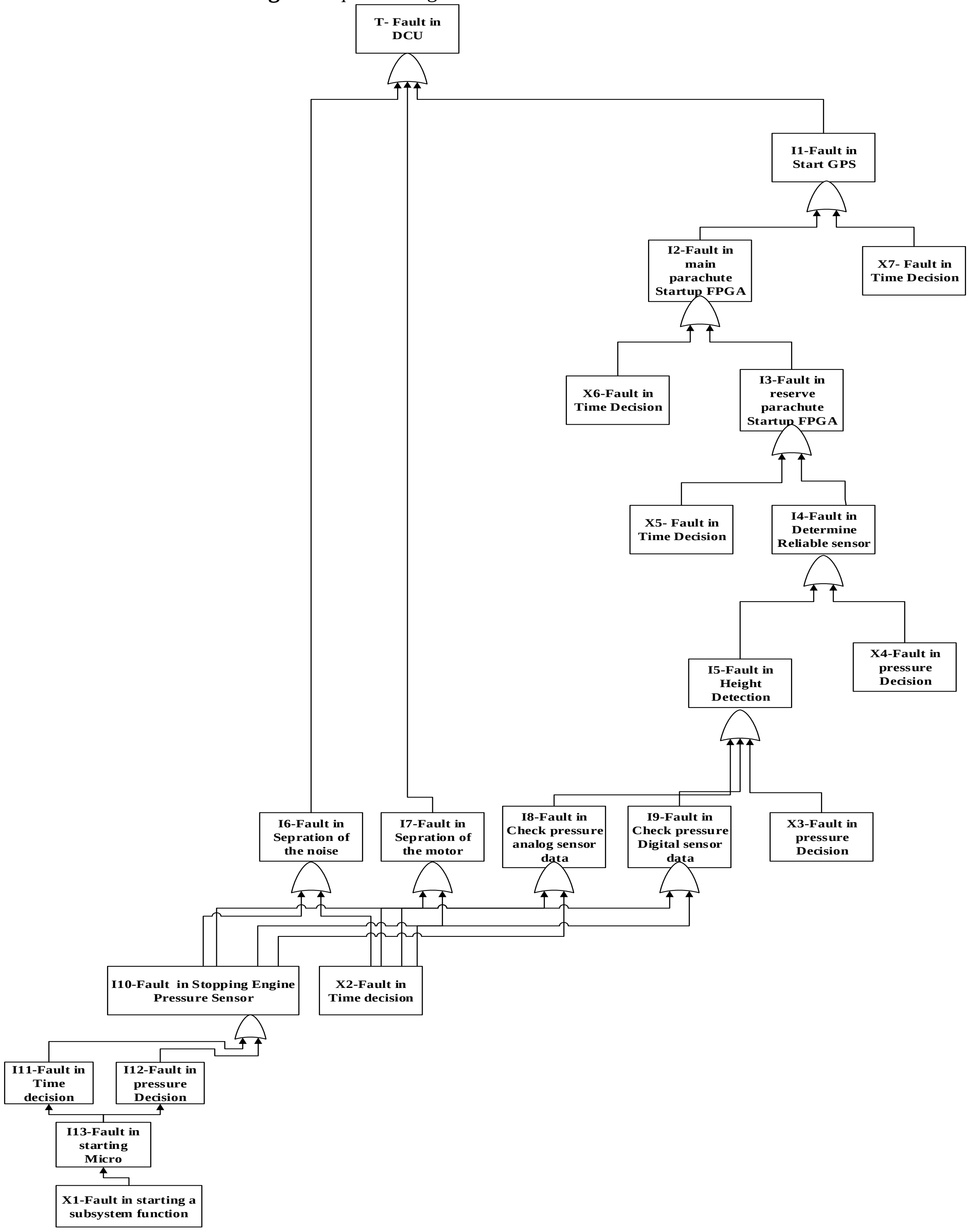
T- Fault in DCU
I1-Fault in Start GPS
I2-Fault in main parachute Startup FPGA
X7- Fault in Time Decision
X6-Fault in Time Decision
I3-Fault in reserve parachute Startup FPGA
X5- Fault in Time Decision
I4-Fault in Determine Reliable sensor
I5-Fault in Height Detection
X4-Fault in pressure Decision
I6-Fault in Sepration of the noise
I7-Fault in Sepration of the motor
I8-Fault in Check pressure analog sensor data
I9-Fault in Check pressure Digital sensor data
X3-Fault in pressure Decision
I10-Fault in Stopping Engine Pressure Sensor
X2-Fault in Time decision
I11-Fault in Time decision
I12-Fault in pressure Decision
I13-Fault in starting Micro
X1-Fault in starting a subsystem function

Fig. 4. SFTA of data and command unit

5.1 Evaluation of the designed software

The data and command unit is one of the most important subsystems of space and aerial systems. As discussed earlier, since the use of code-level SFTAs is a heavy and difficult task, it has been attempted to provide analytics to establish software safety from early steps of the software life cycle.

Our focus in this paper is to analyze the design of software used in data and command unit. Accordingly, after examining the use case diagram and the existing documentation, sequence diagram was generally achieved for the entire software suite as well as for all subsystems by means of the use case profile table, and the sequence diagram of the whole set is presented in Fig. 3. After analyzing the existing sequence diagram and according to the presented SFTA method, the SFTA plot (shown in Fig. 4) and then its fuzzy analysis are obtained.

5.2. Fuzzy analysis

In the table below, the failure probability (FP) calculation results related to each base event for the failure risk of DCU subsystem software are shown, and in Table 5, the final and middle probability rates for the failure risk of the mentioned items are presented.

Table 4. Sequence diagram of data and command unit

Failure probability	Event name	Event symbol
0.0043	Fault in starting a subsystem function	X1
0.0065	Fault in time decision1	X2
0.0042	Fault in pressure decision1	X3
0.0047	Fault in pressure decision2	X4
0.0065	Fault in time decision2	X5
0.0063	Fault in time decision3	X6
0.0061	Fault in time decision4	X7

Table 5. SFTA of data and command unit

Event rate	Event name	Event symbol
0.0782	Fault in DCU	T
0.058	Fault in starting of GPS	I1
0.0519	Fault in main parachute relay startup by FPGA	I2
0.0456	Fault in reserve parachute relay	I3
0.0391	Fault in determining reliable sensor	I4
0.0344	Fault in height detection	I5
0.0108	Fault in separation of the noise	I6
0.0108	Fault in separation of the motor	I7
0.0151	Fault in check pressure analog sensor data	I8
0.0151	Fault in check pressure digital sensor	I9
0.0086	Fault in stopping engine pressure sensor	I10
0.0043	Fault in time decision	I11
0.0043	Fault in pressure decision	I12
0.0043	Fault in starting micro	I13

As shown in Table 5, using expert opinions and fuzzy analysis, the system evaluation was done as follows. The total system failure rate was estimated 0.0782 using the Boolean algebra, and thus the reliability of the software subsystem was 0.9218.

6. Conclusion

Nowadays, the detection of failure is more difficult as the CPSs become more and more complex. The software with millions of lines of code play a key role in the failure or success of a system; therefore, the goal is a safety software design from the very beginning of the software development cycle. Therefore, in this paper, safety was examined in the analysis phase of requirements using the sequence diagram. Then, a method was presented to generate SFTA from the sequence diagram. Using Delphi technique, experts' opinions, and fuzzy analysis, the reliability of the software subsystem has been calculated. In future research, we examine the reliability by developing a software using UML and converting it into a state method.

REFERENCES

Billinton, R. (1984*). Reliability Evaluation of Power Systems*. Springer USA.

Bobbio, A., Portinale, L., Minichino, M., & Ciancamerla, E. (2001). Improving the analysis of dependable systems by mapping fault trees into bayesian networks. *Reliability Engineering and System Safety*, 71(3), 249-260.

Chen, S. J., & Hwang, C. L. (1992). *In Fuzzy Multiple Attribute Decision Making,* Springer, Berlin, Heidelberg, 289-486.

Clemen, R., & Winkler, R. (1999). Combining probability distribution from experts in risk analysis. *Risk Analysis,* 19(2), 187-203.

Dowson, M. (1997). The Ariane 5 software failure. *ACM SIGSOFT Software Engineering Notes*, *22*(2), 84.

Hovsepyan, A., Van Landuyt, D., Michiels, S., Joosen, W., Rangel, G., Fernandez Briones, J., & Depauw, J. (2014). Model-driven software development of safety-critical avionics systems: An experience report. *ACM/IEEE 17th International Conference on Model Driven Engineering Languages & Systems*, Valencia, Spain.

Kamalrudin, M., Ow, L.L., & Sidek, S. (2018). Requirements defects techniques in requirements analysis: A Review. *Journal of Telecommunication, Electronic and Computer Engineering*, 10(1-7), 47-51.

Kamandi, A., Azgomi, M.A., & Movaghar, A. (2006). Transformation of UML models into analyzable OSAN models. *Electronic Notes in Theoretical Computer Science*, 159, 3-22.

Khakzad, N., Khan, F., & Amyotte, P. (2011). Safety analysis in process facilities: Comparison of fault tree and Bayesian network approaches. *Reliability Engineering and System Safety*, 96(8), 925-932.

Kim, H., Eric, W.W., & Debroy, V. (2010). Bridging the gap between Fault trees and UML state machine diagrams for safety analysis. *The 17th Asia Pacific Software Engineering Conference*, Sydney, Australia.

Lavasani, S.M., Zendgani, A., & Celik, M. (2014). An extension to fuzzy fault tree analysis (FFTA) application in petrochemical process industry. *Process Safety and Environment Protection*, 93, 75-88.

Li, L., Lu, M., & Gu, T. (2014). A reuse-oriented auxiliary construction method for software fault tree and tool implementation. *IEEE International Conference on Reliability Maintainability and Safety,* Guangzhou, China.

MIRI, L. M., Wang, J., Yang, Z., & Finlay, J. (2011). Application of fuzzy fault tree analysis on oil and gas offshore pipelines, 1(1), 29-42.

Mahmood, Y. A., Ahmadi, A., Verma, A. K., Srividya, A., & Kumar, U. (2013). Fuzzy fault tree analysis: a review of concept and application. *International Journal of System Assurance Engineering and Management*, 4(1), 19-32.

Martins, L.E.G., & Gorschek, T. (2017). Requirements engineering for safety-critical systems: Overview and challenges. *IEEE Software*, 34(4), 49-57.

Nguyen, H.T., & Prasad, N.R. (1999). *Fuzzy Modeling and Control: Selected Works of Sugeno.* CRC press. Florida.

Oliveira, A.L.D., Braga, R.T., Masiero, P.C., Papadopoulos, Y., Habli, I., & Kelly, T. (2016). Model-based safety analysis of software product lines. *International Journal of Embedded Systems, 8*(5-6), 412-426.

Oveisi, Sh., & Ravanmehr, R. (2017). Analysis of software safety and reliability methods in cyber physical systems, *International Journal of Critical Infrastructures*. 13(1), 1-15.

Oveisi, S., & Farsi, M.A. (2018). Software safety analysis with UML-Based SRBD and fuzzy VIKOR-Based FMEA. *International Journal of Reliability, Risk and Safety: Theory and Application*, 1(1), 35-44.

Paiboonkasemsut, P., & Limpiyakorn, Y. (2015). Reliability tests for process flow with fault tree analysis. *The 2nd International Conference on Information Science and Security,* Seoul, South Korea.

Rajkumar, R., Lee, I. (2012). Cyber-Physical systems: The next computing revolution. *The 47th ACM/IEEE Design Automation Conference,* Anaheim, USA.

Renjith, V.R., Madhu, G., Nayagam, V.L.G., & Bhasi, A.B. (2010). Two-dimensional fuzzy fault tree analysis for chlorine release from a chlor-alkali industry using expert elicitation. *Journal of Hazardous Materials*, 183(1-3), 103-110

Romani, M., Lahoz, C., & Yano, E. (2010). Identifying dependability requirements for space software systems. *Journal of Aerospace Technology and Management*, 2(3), 287-300.

Tiwari, S., Rathore, S.S., Gupta, S., Gogate, V., & Gupta, A. (2012). Analysis of use case requirements using SFTA and SFMEA Techniques. *The 17th International Conference on Engineering of Complex Computer Systems*, Paris, France.

Towhidnejad, M., Wallace, D.R., & Gallo, A.M. (2003). Validation of object oriented software design with fault tree analysis. *In 28th Annual NASA Goddard Software Engineering Workshop,* Greenbelt. United Kingdom.

Vyas, P., & Mittal, R., K. (2012). Eliciting additional safety requirements from use cases using SFTA. *1st IEEE International Conference on Recent Advances in Information Technology,* Dhanbad, India.

Vyas, P., & Mittal, R.K. (2015). The applications of SFTA and SFMEA approaches during software development process: An analytical review. *International Journal of Critical Computer-Based Systems*, 6(1), 29-49.

Wu, F.J., Kao, Y.F., & Tseng, Y.C. (2011). From wireless sensor networks towards cyber physical systems. *Pervasive and Mobile Computing*, 7(4), 397-413.