

CRANFIELD UNIVERSITY

JIAWEN MAO

SAFETY ASSESSMENT METHODS FOR AVIONICS SOFTWARE  
SYSTEM

SCHOOL OF AEROSPACE, TRANSPORT & MANUFACTURING

MSc by Research  
Academic Year: 2016 - 2017

Supervisor: Dr Huamin Jia  
Associate Supervisor: Dr Irfan Madani  
November 2017



CRANFIELD UNIVERSITY

SCHOOL OF AEROSPACE, TRANSPORT & MANUFACTURING

MSc by Research

Academic Year 2016 - 2017

JIAWEN MAO

SAFETY ASSESSMENT METHODS FOR AVIONICS SOFTWARE  
SYSTEM

Supervisor: Dr Huamin Jia  
Associate Supervisor: Dr Irfan Madani  
November 2017

This thesis is submitted in partial fulfilment of the requirements for  
the degree of MSc by Research

© Cranfield University 2017. All rights reserved. No part of this  
publication may be reproduced without the written permission of the  
copyright owner.



## **ABSTRACT**

Nowadays, the avionics software has been becoming more and more critical for both civil and military aircraft. However, the software may become crazy sometimes and may cause the catastrophic result if any failure in software. Therefore, the software safety assessment is not only crucial to the specific software, but also for the system and aircraft. Although there are some industry standards as guidelines for development of software system, applications of these standards to practical software systems are still challenged and hard to operate in practice. This thesis tries to solve this problem.

After analyses and summaries of the system safety assessment process and existing software safety assessment process in different fields, research wants to propose the systematic and comprehensive software safety assessment process and method for avionics software.

The thesis presents the research process, and proposes one suitable avionics software safety assessment process. Meanwhile, thesis uses a real functional block in flight management system as a case study, and then conducts the software safety requirement assessment based on the proposed software safety assessment method.

After analysis the result of case study, this proposed software safety assessment process and methods can quickly and correctly identify the software design errors. So, this analysis can use to prove the feasibility and validity of this proposed software safety assessment process and methods, which will help engineers modify every software design errors at the early stage in order to guarantee the software safety.

Keywords:

Software Safety, Software Development Process, Software Safety Assessment Process, DO-178C, ARP4754A, Functional Hazard Assessment, Fault Tree Analysis, Failure Mode and Effects Analysis, Formal Method, NuSMV



## **ACKNOWLEDGEMENTS**

I would like to express my special thanks of gratitude to my supervisor Dr Huamin Jia. His patient guidance and constant encouragement is so helpful and great throughout the study.

I would like to thank all my friends for their care and encouragement through this year.

Finally, I would like to express a deep sense of gratitude to my parents who try their best to give their love, encouragement and supporting for the study in Cranfield University.





# TABLE OF CONTENTS

ABSTRACT .....	i
ACKNOWLEDGEMENTS.....	iii
LIST OF FIGURES.....	vii
LIST OF TABLES .....	ix
LIST OF ABBREVIATIONS.....	x
LIST OF SYMBOLS .....	xi
LIST OF DEFINATION .....	xii
1 Introduction.....	1
1.1 Background.....	1
1.2 Research Objectives.....	2
1.3 Methodology .....	2
1.4 Thesis Structure.....	4
2 Literature Study .....	7
2.1 Introduction .....	7
2.2 The Importance of Software Safety.....	7
2.3 System and Software Safety Assessment Process .....	12
2.3.1 System Safety Assessment Process.....	12
2.3.2 Software Safety Assessment Process.....	17
2.3.3 Relationship between System Safety Assessment and Software Safety Assessment .....	23
2.4 Software Safety Assessment Techniques.....	24
2.4.1 Traditional Software Safety Assessment Methods .....	24
2.4.2 Emerging Software Safety Assessment Methods .....	31
2.5 Summary .....	41
3 Avionics Software Safety Assessment Process .....	43
3.1 Introduction .....	43
3.2 Software Development Process in DO-178C.....	43
3.3 Avionics Software Safety Assessment Process.....	44
3.3.1 The Objectives and Tasks of Each Steps.....	46
3.3.2 Transition Criteria of Proposed Avionics Software Safety Assessment Process.....	48
3.4 Summary .....	49
4 Proposed Methods for Avionics Software Safety Assessment .....	51
4.1 Introduction .....	51
4.2 Methods for Software Safety Requirements Assessment Process .....	51
4.2.1 Identify hazard.....	53
4.2.2 Identify safety requirements .....	54
4.2.3 Formal Verifications.....	55
4.3 Methods for Software Architecture Safety Assessment Process .....	61
4.3.1 Software Development Assurance Level Assignment Process .....	62

4.3.2 Comparison of Existing Software DAL Assignment Process.....	65
4.3.3 Software Development Assurance Level Assignment Process Case Study.....	66
4.4 Consideration of Software Development Assurance Level Assignment Process.....	71
4.5 Summary .....	73
5 Case Study.....	75
5.1 Introduction .....	75
5.2 Overview of Flight Management System .....	75
5.3 Overview of Position Calculation Function.....	79
5.4 Verification Tool NuSMV .....	81
5.4.1 NuSMV Installation.....	82
5.4.2 NuSMV Architecture.....	84
5.4.3 NuSMV Input Language .....	86
5.5 Position Calculation Safety Requirement Elicitation Process.....	87
5.5.1 Identify Hazards .....	88
5.5.2 Identify Safety Requirements .....	89
5.6 Formal Modelling .....	94
5.7 Result Analysis .....	96
5.8 Summary .....	99
6 Conclusion and Further Work.....	101
6.1 Conclusion .....	101
6.2 Further Work.....	102
REFERENCES.....	105
APPENDICES .....	111

## LIST OF FIGURES

Figure 1-1 Process of Research Methodology .....	3
Figure 2-1 Software in Military Aircraft [13] .....	8
Figure 2-2 Growth of Software Volume in Civil Aircraft [14] .....	9
Figure 2-3 Sequence of Software Error Lead to the Failure Condition [5] .....	9
Figure 2-4 Percentage of Hull Losses by Accident Category 1997-2016 [19] ..	11
Figure 2-5 Safety Assessment Process in ARP4754A [10] .....	13
Figure 2-6 System Safety Assessment Process in ARP4754A .....	14
Figure 2-7 System Safety Process in MIL-STD-882E [6] .....	16
Figure 2-8 Software Safety Assessment Process in Military [24] .....	18
Figure 2-9 Basic Idea of Software Criticality Index Assignment Process in Software Safety Assessment Process .....	19
Figure 2-10 Relationship between System Safety Assessment Software Safety Assessment .....	23
Figure 2-11 Functional Hazard Assessment Process [9] [10] .....	25
Figure 2-12 Construction step of Fault Tree Analysis [29] .....	28
Figure 2-13 Failure Mode and Effects Analysis Process [31] .....	30
Figure 2-14 Difference between FTA and FMEA [32] .....	31
Figure 2-15 Model Checking Formula [35] .....	33
Figure 2-16 State Transition Graph for Kripke Structure [38] .....	34
Figure 2-17 Linear Temporal logic [40] .....	35
Figure 2-18 Example of LTL Formula .....	36
Figure 2-19 Computation Tree Logic [40] .....	36
Figure 2-20 $AG\ p$ .....	37
Figure 2-21 $AF\ p$ .....	38
Figure 2-22 $EG\ p$ .....	38
Figure 2-23 $E\ (p\ U\ q)$ .....	39
Figure 3-1 Software Development Process in DO-178C .....	43
Figure 3-2 Proposed Avionics Software Safety Assessment Process .....	45
Figure 3-3 Emphasis of Research .....	46

Figure 4-1 Process of Software Safety Requirement Assessment .....	52
Figure 4-2 Process of Hazard Identification .....	53
Figure 4-3 Verification Process of Safety Requirement [16].....	56
Figure 4-4 State Transition Graph .....	59
Figure 4-5 Verification Result 1 of Microwave Problem.....	60
Figure 4-6 Verification Result 2 of Microwave Problem.....	61
Figure 4-7 Process of Software Architecture Safety Assessment .....	62
Figure 4-8 Assign Development Assurance Level to Functional Failure Set Members [10].....	65
Figure 4-9 Cost of Different Development Assurance Level Software.....	66
Figure 4-10 Fault Tree Example-1 [10].....	68
Figure 4-11 Fault Tree Example 2 [10].....	70
Figure 4-12 FDAL/IDAL Assignment Process .....	72
Figure 5-1 Overview of FMS Components [50] .....	76
Figure 5-2 Overview of FMS Function.....	77
Figure 5-3 FMS Block Diagram [51] .....	79
Figure 5-4 Position Calculation Logical [51] .....	80
Figure 5-5 NuSMV Installation Result .....	84
Figure 5-6 Architecture of NUSMV [53].....	85
Figure 5-7 Kripke Structure in NuSMV [55] .....	87
Figure 5-8 Fault Tree of “The both sides of PFD indicate the wrong navigation and position information” .....	90
Figure 5-9 Subtree of “The both sides of PFD indicate the wrong navigation and position information” .....	91
Figure 5-10 Sensor State Transition Graph.....	95
Figure 5-11 Monitor Function State Transition Graph .....	96
Figure 5-12 NuSMV Result of Specification 1 .....	97
Figure 5-13 NuSMV Result of Specification 2 .....	97
Figure 5-14 NuSMV Counterexample Result .....	98
Figure 6-1 The Relationship between Objectives and Achievement .....	101

## LIST OF TABLES

Table 2-1 Activities of Each Step in Software Safety Assessment Process-1 [23] [24] .....	20
Table 2-2 Activities of Each Step in Software Safety Assessment Process-2 [23] [24] .....	21
Table 2-3 Objective and Proposed Methods in Software Safety Assessment Process [25].....	22
Table 2-4 Symbol of Fault Tree Analysis [28].....	27
Table 2-5 Step of Software Fault Tree .....	28
Table 2-6 Comparison of Model Checkers .....	41
Table 4-1 Software FHA [8] [10].....	54
Table 4-2 Output of Software Failure Mode and Effects Analysis .....	55
Table 4-3 General Principle for DAL Assignment [5] .....	64
Table 4-4 Accepted Assignment of FDAL and IDAL of Example 1 .....	67
Table 4-5 Unaccepted Assignment of FDAL and IDAL of Example 1.....	68
Table 4-6 Accepted Assignment of FDAL and IDAL of Example 2.....	70
Table 4-7 Unaccepted Assignment of FDAL and IDAL of Example 2.....	71
Table 5-1 Installation Steps of NuSMV Source Code [43].....	83
Table 5-2 Installation Step of NuSMV Binary Code.....	84
Table 5-3 SMV Language Example.....	86
Table 5-4 Failure Condition for Position Calculation.....	88
Table 5-5 Basic Event list of Fault Tree.....	93
Table 5-6 FMEA for Navigation Source Selection Logical Failure .....	94
Table 5-7 FMEA for Navigation Source Monitor function failure.....	94

## LIST OF ABBREVIATIONS

CCA	Common Cause Analysis
CTL	Computation Tree Logic
DAL	Development Assurance Level
DME	Distance Measuring Equipment
FDAL	Functional Development Assurance Level
FFS	Functional Failure Set
FHA	Functional Hazard Assessment
FTA	Fault Tree Analysis
FMC	Flight Management Computer
FMEA	Failure Mode and Effects Analysis
FMS	Flight Management System
FMSA	Flight Management System Application
GPS	Global Positioning System
IDAL	Item Development Assurance Level
IRS	Inertial Reference System
LTL	Linear Temporal Logic
LOR	Level of Rigor
MIL-STD	Military Standard
PSSA	Preliminary System Safety Assessment
RNAV	Area Navigation
RNP	Required Navigation Performance
RPN	Risk Priority Number
SwCI	Software Criticality Index
SSA	System Safety Assessment
TNAV	Time Navigation
VOR	VHF Omni directional radio range

## LIST OF SYMBOLS

$S$	Finite Set of State
$S_0$	Initial State of Kripke Structure
$\delta$	Transition Relation
$AP$	Set of Atomic Propositions
$L$	Labelling Function
$\neg$	Negation
$\wedge$	Conjunction
$\vee$	Disjunction
$\rightarrow$	Implication
$\leftrightarrow$	Equivalence
$\&$	Logical And
$!$	Logical Not
$ $	Logical Or
$\pi$	Path
$A$	Path Quantifier
$E$	Path Quantifier
$X$	Temporal Operator, means next time
$F$	Temporal Operator, means eventually
$G$	Temporal Operator, means always
$U$	Temporal Operator, means until

## **LIST OF DEFINATION**

Software Failure: Before the software delivery, any error or fault found in software requirement, design, code, and integration process may cause software partial function and/or entire software cannot perform as its expected.



# 1 Introduction

## 1.1 Background

With rapid development of computer, software already has been applied in different fields. Compared with human, software has the incomparable capability, and it can make decisions unemotionally and more accurately. Due to this reason, the scale and complexity of software have increased tremendously, which make its proportion of the critical function in the complicated modern systems also has risen sharply. Nowadays, software is playing a vital role in hazard control and operation of safety-critical function or system [1]. For example, the development cost of software in Fly-by-wire system can occupy about 60-70% of the total cost of the entire Fly-by-wire systems [2]. In the integrated avionics system of F-22 Raptor, the implementation of avionics function and system is up to 80% by using software [3].

Software can bring lots of benefits. In the meanwhile, it can cause some serious problems. Comparing with hardware, the safety and quality of airborne software are lower due to the software characteristics. In some circumstances, software may not be performed according to its design, which can have adverse effects on the airborne system, and even lead to mishaps. In recent years, the number of accidents caused by software failure are increasing. Software failures may cause severe damage to aircraft, which may threaten and harm life [1]. For example, Turkish Airlines Flight 1951 crashed during the approach on 25 February 2009, and six passengers and three pilots died in this accident. The final accident investigation report shows that the radio altimeter system failure is the major factor, which made aircraft stall. However, the report also points out the airborne software on Flight 1951 lacked the programme which could deal with the stall situation, and this reason also contributes to the accident [4]. So that, software has critical influence on the airborne system safety, especially the airborne software system.

The software safety assessment process aims to solve software unsafe problem, and software safety assessment methods focus on identifying the software failure

which could lead to the system or aircraft level hazard. Meanwhile, the software safety assessment process and method can reduce the development cost of software.

## 1.2 Research Objectives

This MSc-by-Research project is designed to study and develop technical methods and processes for safety-critical avionics software safety assessment.

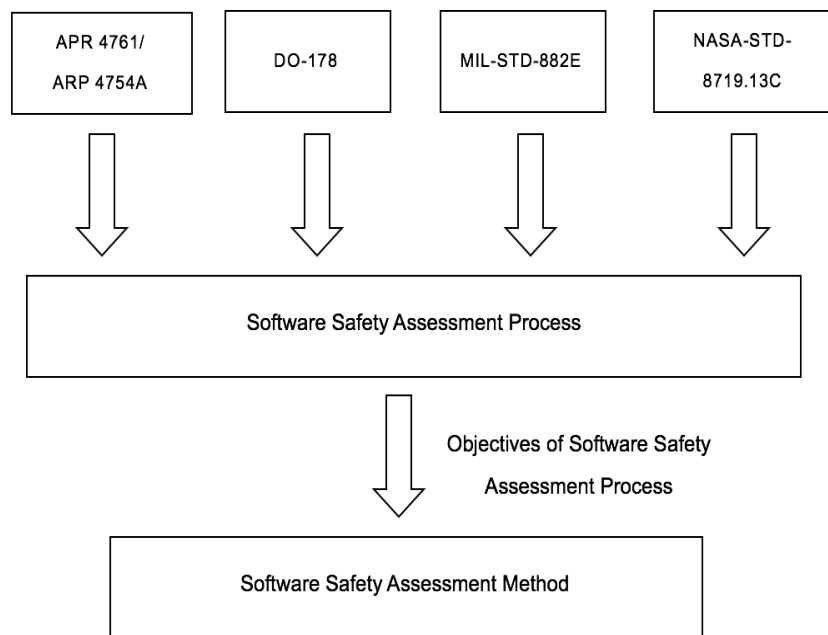
- On the basis of the airborne software lifecycle introduced in **Software Considerations in Airborne Systems and Equipment Certification** (DO-178C) [5], work out the systemic software safety assessment process for avionics software.
- According to objectives of each process in software development, briefly introduce various safety assessment methods in accordance with each process.
- Focus on objectives of requirement safety assessment, proposes the proper safety assessment methods.
- Based on the existing standard and requirement safety assessment result, applies the Development Assurance Level assignment process to software level.

Through the proposed software safety assessment process and suitable methods, software error and fault can be identified and modified, which it can ensure software safety can be guaranteed during each stage of software lifecycle. Thereby, the proposed safety assessment process and method can be used to prove software safety in different phase of software lifecycle and improve reliability of software, and even entire system.

## 1.3 Methodology

Before work on the software safety assessment process, the research starts with getting familiar with the software lifecycle. Currently, lots of standards and guidelines are accepted by the authority and manufacturer, such as DO-178C [5], **Standard Practice for System Safety** (MIL-STD-882E) [6] and so on. Among these standards and guidelines, **Software life cycle processes** (ISO/IEC

12207) [7], **NASA Software Safety Guidebook** (NASA-GB-8719.13) [8] and DO-178C [5] gives detailed description of the software life cycle which includes the software development process. Furthermore, **Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment** (SAE ARP 4761) [9] and **Guidelines for Development of Civil Aircraft and Systems** (SAE ARP 4754A) [10] provide detailed safety assessment process for aircraft and system level, and suggest several safety assessment methods, such as Functional Hazard Assessment, Fault Tree Analysis and Failure Mode and Effects Analysis. Meanwhile, the standards such as DO-178C provide the objectives and activities of software lifecycle. Based on the standards, guidelines and the summaries of the software safety assessment process from other aspects, works out the comprehensive and practicable avionics software safety assessment process, and determines the objectives and activities of the proposed avionics software safety assessment process. The Figure 1-1 shows the flowchart of research methodology.



**Figure 1-1 Process of Research Methodology**

After determination of the proposed software safety assessment process, consider the proper software safety assessment methods for each step according

to objectives and tasks found in the previous study. Safety assessment methods not only use at aircraft and system level, but also use on software, such as Fault Tree Analysis and Failure Mode and Effects Analysis. Meanwhile, some of the emerging methods can be used in the proposed software safety assessment process. At the end, the research uses case study to prove the rationality and feasibility of new avionics software safety assessment process and methods.

## **1.4 Thesis Structure**

The thesis includes 6 chapters.

Chapter 1: Introduction. Briefly describes the background and existing problem of avionics software safety. Lists all the objectives of this research, and gives the precise description of the research methodology.

Chapter 2: Literature Study. Briefly introduces the literature study on the software safety assessment process in different aspects, and the software lifecycle described in DO-178C. And then, detailed introduces the traditional and emerging software safety assessment methods. The literature study is the fundamental of the proposed software safety assessment process and methods.

Chapter 3: Avionics Software Safety Assessment Process. Detailed discusses the content of proposed avionics software safety assessment process, and lists objectives and activities of each sub-process. The Chapter also recommends suitable software safety assessment methods for each sub-process. At last, this Chapter gives the general transition criteria, which makes the proposed software safety assessment more comprehensive.

Chapter 4: Proposed Methods for Avionics Software Safety Assessment Process. Detailed describes the content and proposed methods for software safety requirement assessment and software architecture safety assessment process. This Chapter gives the several examples for presenting the feasibility of the proposed method.

Chapter 5: Case Study. Detailed describes how to apply the avionics software safety assessment process and methods to the practical software function in civil

aircraft. The result of case study proves the feasibility of proposed avionics software safety assessment process and methods.

Chapter 6: Conclusion and Further Work. This Chapter summarizes all the achievements in the research, and points out the relevant work which has not been fully solved in this research.



## **2 Literature Study**

### **2.1 Introduction**

In the literature study, the author has researched on several parts. Firstly, the Chapter shows the investigation of usage of software in military and civil aircraft, and several air accidents caused by software failure, which presents the importance of software safety. Second, research of both system safety assessment process and kinds of software safety assessment process gives ideas of developing the proposed avionics software safety assessment process. Last, the Chapter discusses the traditional and emerging safety assessment methods, such as Fault Tree Analysis, Formal Methods.

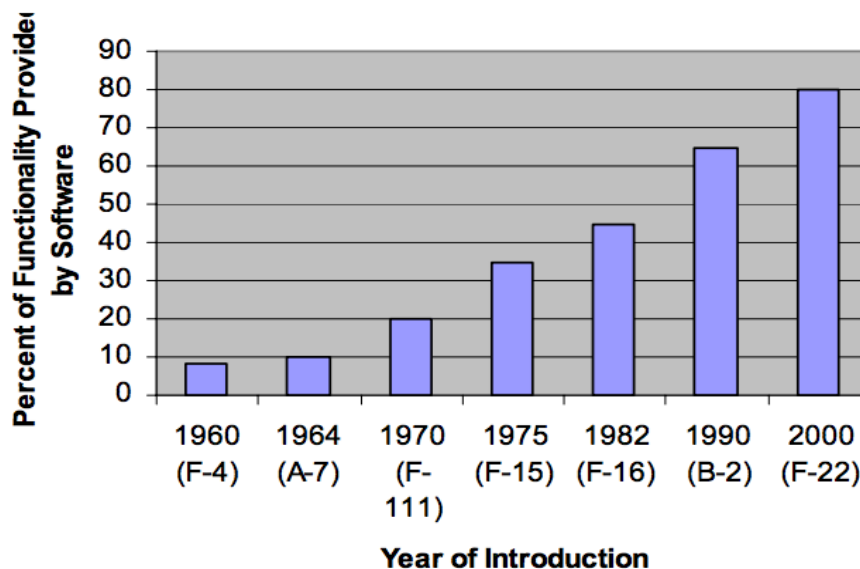
### **2.2 The Importance of Software Safety**

Safety is defined as the state that the risk is acceptable [10]. Safety in software can be understood as the capability of maintaining in the state which the risk has been reduced at an accredited level. Software safety can make sure that the related software hazards are under control and will not propagate to a system. Therefore, software safety is the critical property for the system safety. The importance of software safety can be experienced from all aspects, and this section gives the introduction from two points, one is the proportion of function or system implemented by software in the aircraft system, and the other is to demonstrate several catastrophic air accidents which were caused by software failure.

In modern avionics, the system functions are usually implemented by some complicated computer software [11]. The progress of computer and software has changed the aviation industry [12]. Nowadays, many safety-critical systems and functions are implemented and controlled by computer and software in modern aircraft.

According to the NASA study of flight software complexity, the percent of function which provided by software in military aircraft has risen from 8% in 1960 to 80% in 2000, as shows in Figure 2-1. The software size has grown rapidly, and the

software size has increased from 1000 lines of code to 1.7M lines of code. For example, the F-22 has 2.5M lines of code [13].

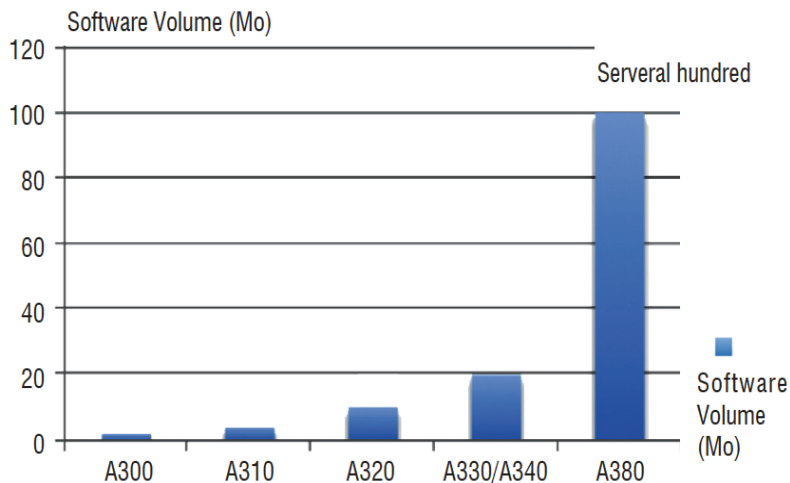


**Figure 2-1 Software in Military Aircraft [13]**

Same as the military, many airborne systems and functions, especially the safety-critical systems and functions, have been implemented by software in civil aircraft. According to the journal from Aerospace Lab [14], the airborne system has been changed a lot because of the development of computer and software technology over the last 30 years, and software volume has grown rapidly in these years. For example, avionics system in Airbus A380 has more than 100 million lines of code. The Figure 2-2 shows the growth of software volume in different types of Airbus aircraft.

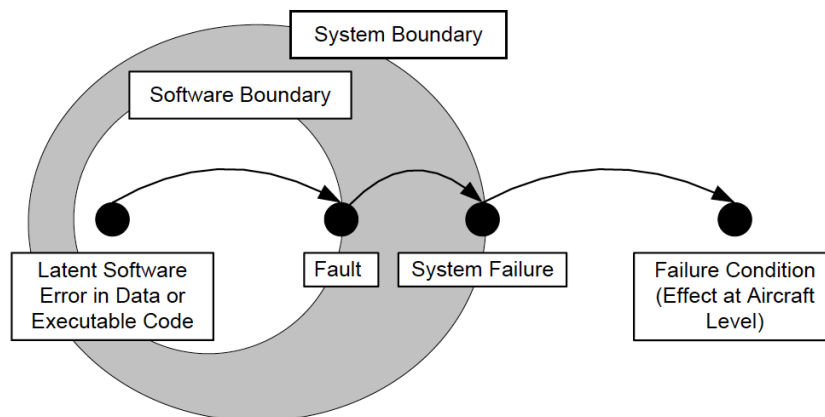
The reason of software volume growth is easy to understand, because faster computer and software can achieve more flexibility of the aviation system. Since the widespread use of software, the engineer has been concerned with the software failure and its implication for related safety-critical functions. If one software is unsafe and people trust a lot, the destruction of software failure can be erroneous [15]. Even the smallest error can cause the most severe consequence, which might lead to significant damage or loss of life.





**Figure 2-2 Growth of Software Volume in Civil Aircraft [14]**

As the Figure 2-3 shows, any error found in the entire software life cycle, such as the error found in the requirement, design, code, and integration process, can contribute to the most severity result like loss of human life. These errors may arise at anytime and anywhere in the software life cycle [16]. Software safety is to identify the hazard which can lead to the system failure. Furthermore, software safety could decrease the error rate during the software lifecycle, and reduce the probability of hazard occurrence and the risk level.



**Figure 2-3 Sequence of Software Error Lead to the Failure Condition [5]**

The failure of safety-critical software could cause the severe harm and damage to the system and aircraft, and potentially, lost the human life. Some software-related air accidents are described below.

- Korean Air Flight 801 [15]

The Korean Air Flight 801, crashed in Guam on August 1997, and it killed 225 of 254 persons on board. National Transportation Safety Board considered that there were two identified software errors in Minimum Safe Altitude Warning, which might be the contributing factors in this crash.

- American Airlines Flight 965 [15]

The American Airlines Flight 965 crashed into the west slope of the mountain on December 20, 1995. This accident caused 159 deaths, and the aircraft was totally destroyed. The reason of this accident was that the flight management system incorrectly understands the waypoint identifier when pilot type the similar identifier. Although the final report pointed that the major factor of this accident was the pilot error, the poor design of flight management system also contributed to this crash.

- British Airway Flight 027 [17]

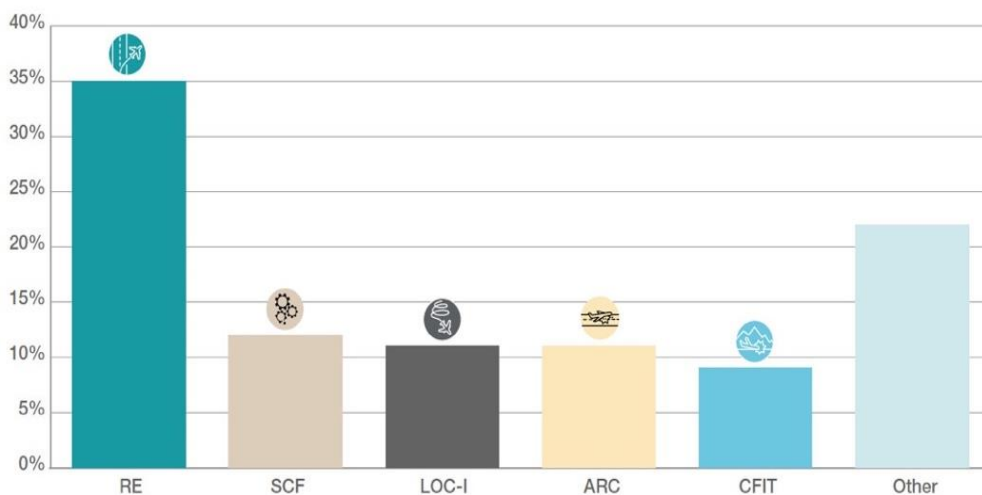
The British Airways Flight 027 nearly collided with an aircraft from Korean Air Cargo during the flight at the Chinese airspace region. The altitude between two aircraft was only 600 feet. Fortunately, the pilot found this problem immediately and took the corresponding actions which prevented this catastrophe. After the investigation, the reason of this accident was due to the failure of the collision avoidance system installed of the Korean Air Cargo aircraft. The failed collision avoidance system determined wrong altitude of cargo aircraft and provided the wrong command, and this collision avoidance system was damaged during the maintenance.

- Qantas Flight 72

On October 20, 2008, the Qantas Flight 72 departed from Singapore to Australia with 303 passengers. During the cruising at 37,000ft, the autopilot was disconnected automatically. When the pilot has evaluated the situation, the aircraft began to pitch down, and then descended 650ft. After 15 Seconds when the pilots controlled aircraft, aircraft pitched down again and descended 400ft at this time. About 1 hour later, the aircraft landed at Learmonth. Fortunately, no

one died in this accident. According to the Australian transportation safety report, the failure of air data inertial reference units installed in the aircraft was one major factor of this accident. The aircraft installed three sets of air data inertial reference units, and one of the three had sent the incorrect data to aircraft system before the autopilot disconnected [18].

According to the Airbus statistical analysis of commercial aviation accidents happened during 1958-2016, accident caused by System, Component Failure or Malfunction included the failure of power plant, software and database systems approximately occupy 12% of the total number of hull losses accidents since 1997 [19].



**Figure 2-4 Percentage of Hull Losses by Accident Category 1997-2016 [19]**

Any failure of software can lead to a potential risk situation which could lead to the loss of property and lives. Therefore, software failure has emerged as one of the new sources of hazard [20]. Over-trusting or underestimating the complexity of software could lead the occurrence and propagation of hazard, which may contribute to the system, even aircraft failure.

Therefore, people want the correct and safe software. Unfortunately, it is difficult to correctly produce the software, even harder to ensure the software safety, because safety is an incredibly complex issue that depends on many factors. For example, safety engineers hope to detect and correct all the errors before implemented. In fact, some errors can be identified and some cannot be found.

Meanwhile, new errors might be introduced. So that, this is one reason that some software may have the defects after implemented, and some software was deemed as safe, but it still can cause the hazard.

Software safety is not a software-specific issue, and is a systems issue [21]. If all the errors or failure can be found and corrected at the early time, or can be detected during the whole software life cycle, the software will reduce the probability of hazard occurrence. Therefore, software safety should be integrated into software lifecycle and system development process. Through the assessment of safety-critical software during the software lifecycle, it will help to guarantee the correctness of software requirements, code and implementation. Thereby, the software safety can be guaranteed, and the confidence level of software can be kept at a higher level. For system safety, the software safety assessment can be used as evidence to prove the system hazard has been controlled and eliminated. The next section will discuss two system safety assessment processes which are introduced in APR4754A and military standard.

## **2.3 System and Software Safety Assessment Process**

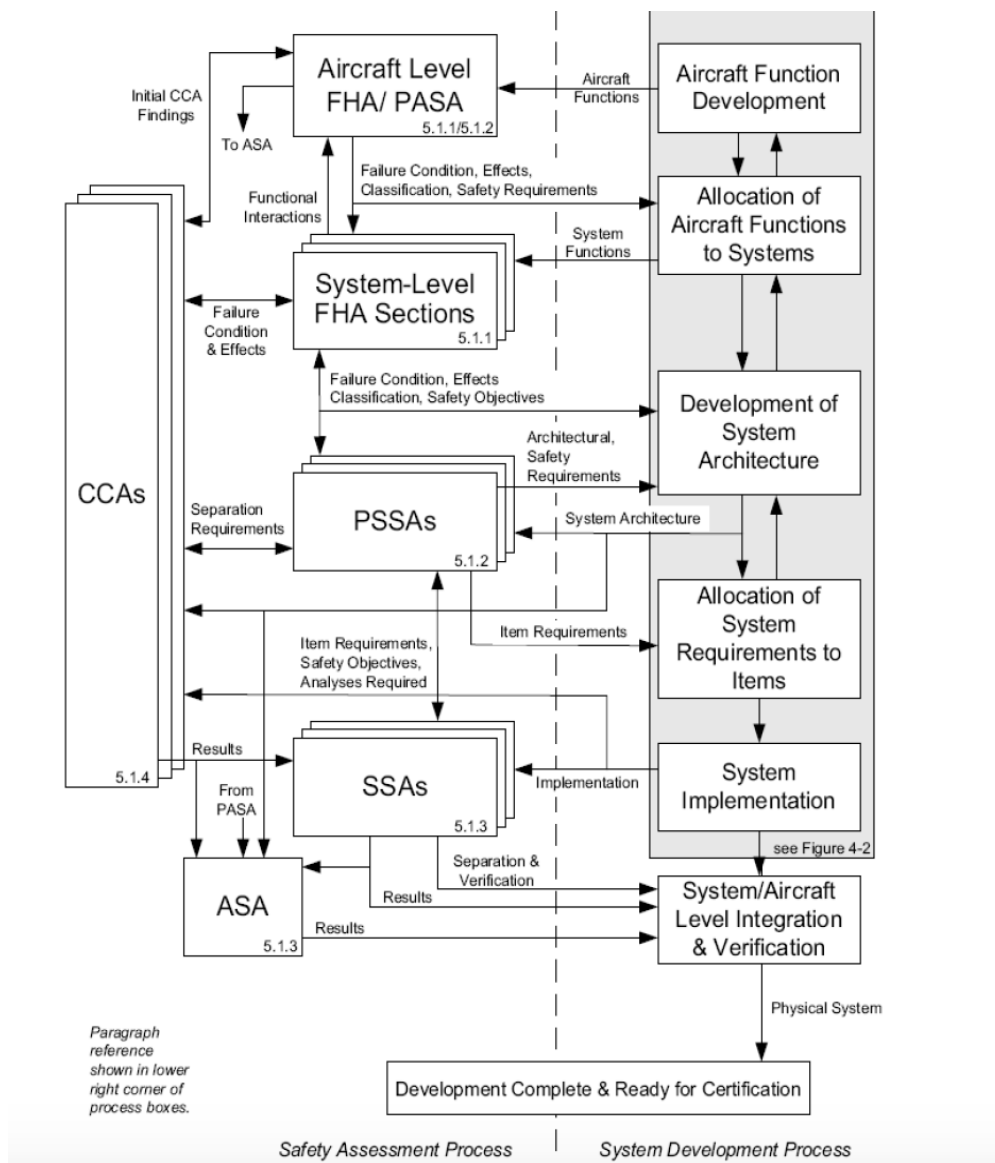
For the safety-critical system, it is essential to perform the safety assessment, which can identify, control and reduce all of the safety risks at the acceptable risk level. This safety assessment not only conducts at system-level, but also involve with software-level [22].

This section mainly discusses the system safety assessment process, software safety assessment process, and the relationship between these two processes. The section lists several system safety assessment processes and software safety assessment processes which are acceptable to the aviation and military. And then, the section analyses the relationship between system safety assessment and software safety assessment.

### **2.3.1 System Safety Assessment Process**

- System Safety Assessment Process in Civil Aircraft

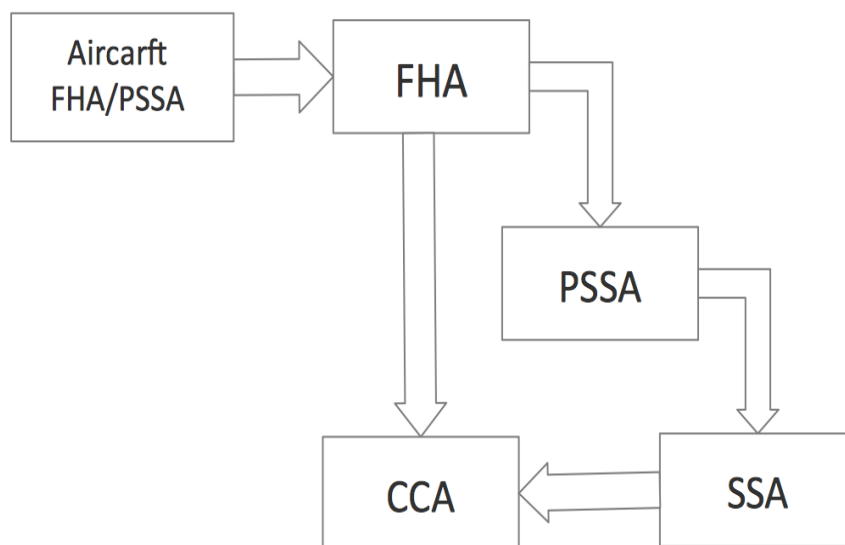
In civil aircraft, ARP 4754A and ARP4761 are two wide acceptance guidelines related to aircraft and system safety assessment. These two guidelines provide the detailed safety assessment process and various safety assessment methods. ARP4754A provides the detailed system safety assessment process which is to establish the safety objectives and to demonstrate compliance with other related safety requirements during the period of system design [10]. ARP4761 provides the guidelines and methods for conducting the safety assessment on system level or at aircraft level of civil aircraft for certification [9].



**Figure 2-5 Safety Assessment Process in ARP4754A [10]**

In APR4754A, the system safety process uses various assessment processes and methods to evaluate system design and functions, which ensure all the hazards have been found and controlled. Figure 2-5 shows the system safety process.

The system safety assessment process in ARP4754A includes four parts, which are functional hazard assessment (FHA), preliminary system safety assessment (PSSA), system safety assessment (SSA), and Common Cause Analysis (CCA) respectively. Before conducting the system safety assessment process, the outputs of aircraft PSSA and FHA are used for the input file of system safety assessment, such as the list of aircraft level failure conditions, and high-level safety requirements. After the aircraft functions are allocated to system-level, it can start to conduct system safety assessment. The system assessment process can be summarized as shows in Figure 2-6.



**Figure 2-6 System Safety Assessment Process in ARP4754A**

The first step is to conduct system-level functional hazard assessment. This process is to identify the system-level failure condition according to the list of the aircraft-level failure conditions, safety objectives and system functions, and to determine the severity of each failure condition according to its effect. After finished the system-level FHA, preliminary system safety assessment and common cause analysis can be started. PSSA is to complete the failure

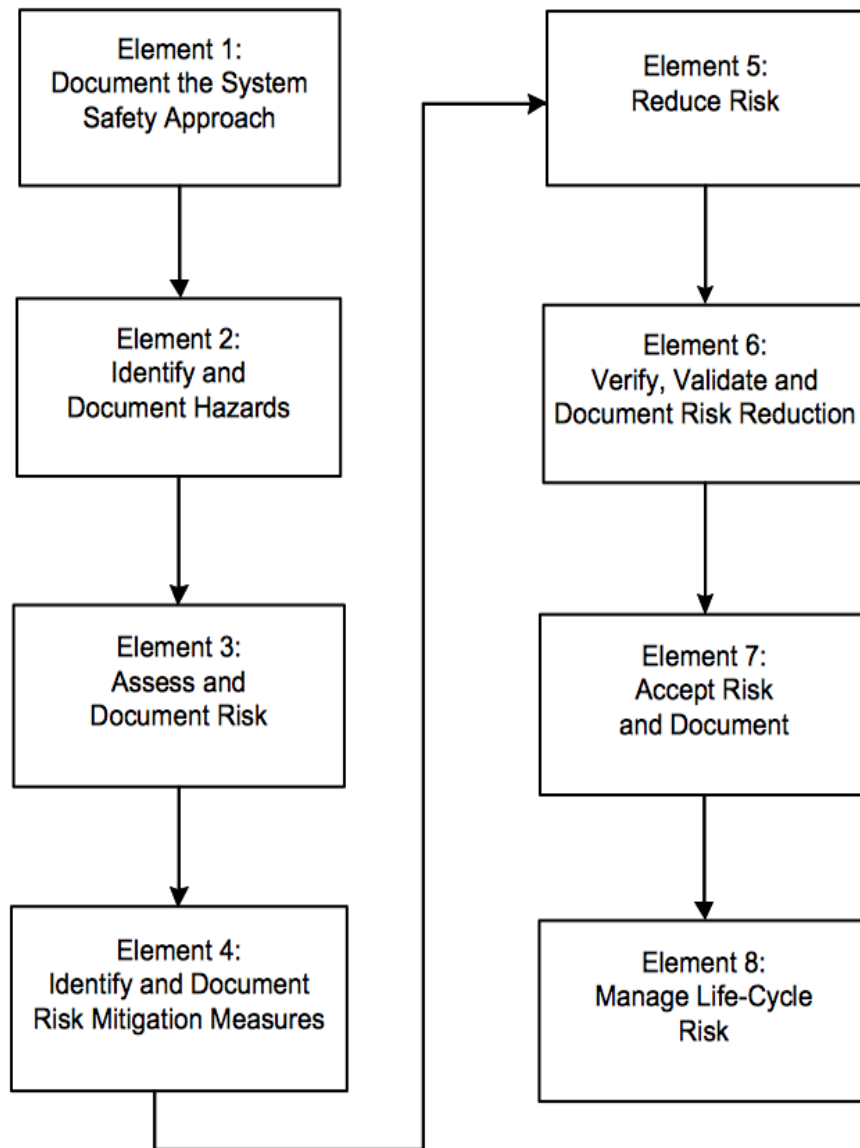
conditions list, to generate lower-level safety requirements, and to verify the proposed system architecture by reviewing related safety requirements. PSSA is the iterative process, and it can be conducted during the entire software lifecycle. At the lowest level, PSSA can be used for identifying the software or hardware safety requirements. Common cause analysis is to verify and determine independence of the system, function or item. During the CCA, it generates the system safety requirements for PSSA to validate the proposed architecture. The last step is system safety assessment. SSA evaluates the implemented system-level function to prove whether the relevant safety requirements are satisfied or not. In SSA, the common safety assessment methods are fault tree analysis and failure mode and effects analysis. Those two methods help to determine the causes of the related failure conditions. After the SSA, it can start the software and hardware level safety assessment process. Figure 2-6 shows the primary relationships in the system safety assessment process. In fact, each process can be entered or re-entered depending to the requirement. Meanwhile, there are many feedback loops between each process.

The second part is the introduction of system safety assessment process in military standards, MILT-STD-882E [6]. The basic idea of the military system safety assessment process is similar with the ARP4754A, which is to identify and assess the hazard.

- System Safety Assessment Process in MILT-STD-882E

The system safety assessment process has eight components. Figure 2-7 gives the flowchart of system safety assessment process in MILT-STD-882E [6]. Here gives the brief introduction of each element.

- Element 1 is Document the System Safety Approach. The document should include the result of risk management, specified and derived system requirements, and the description of the acceptable process of hazard and risk by authority, and the Hazard Tracking System report.



**Figure 2-7 System Safety Process in MIL-STD-882E [6]**

- Element 2 is Identify and Document Hazards. Documented all the hazard identified in this safety assessment process, and the hazard should include software and hardware, interfaces, operational environment.
- Element 3 is Assess and Document Risk. By using the table of severity category and probability level, assessed all the hazards. The severity category table is to identify the potentially harmful, such as death or injury, the effect on the environment, and the monetary loss. The



probability level table is for assessing the likelihood of each hazard occurrence. After assigning the severity and probability, it used the Risk Assessment Code to evaluate each risk. Last, document all the hazard in Hazard Tracking System report.

- Element 4 is Identify and Document Risk Mitigation measures. Identified methods of the risk mitigation and reduction, such as using design alternative to eliminate or reduce the hazards. All the alternatives should be documented in Hazard Tracking System report.
- Element 5 is Reduce Risk. Selected the applicable mitigation methods, and controlled the hazard to stay at a reasonable and receivable risk level. This should consider the cost, effectiveness and feasibility of the selected method.
- Element 6 is Verify, Validate, and Document Risk Reduction. Through the verification and validation of implementation, proved the effectiveness of the selected mitigation methods. Documented all the verification and validation result in Hazard Tracking System report.
- Element 7 is Accept Risk and Document. The appropriate authority should accept all the risk before the hazard happened.
- Element 8 is Manage Lifecycle Risk. Through the entire system lifecycle, the system safety process should be used for identifying and maintaining the Hazard Tracking System.

The system safety process aims to identify, control, reduce and document all the system-level hazards during the system lifecycle. The process can also focus on selecting the proper method of control, reduce and eliminate the hazards.

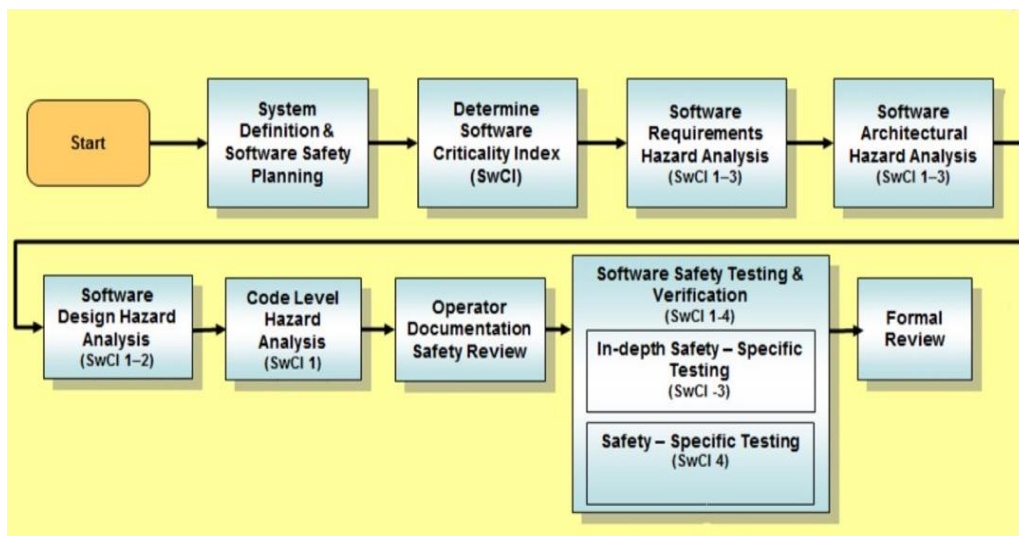
### **2.3.2 Software Safety Assessment Process**

In this section, the author summarizes software safety assessment process from different aspects. The software safety assessment process focuses on the content and selected safety assessment methods.

- Software Safety Assessment in Military

Military standards such as MIL-STD 882E, consider that the software safety assessment process conducts various safety assessment activities and tasks on safety-critical software and functions according to the level of rigor [23] [24].

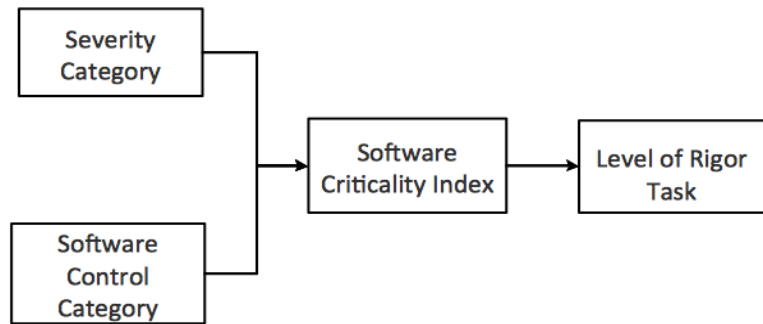
The process has nine sub-processes and involves software development process. This software safety assessment process aims to reduce the risk during the software lifecycle, which can add robustness for software. Figure 2-8 shows the software safety assessment process in Naval Surface Warfare Centre.



**Figure 2-8 Software Safety Assessment Process in Military [24]**

The first step is to determine the safety scope of program and tasks, and the second step is to determine Software Criticality Index (SwCI) for each software safety-significant function. The rest of steps in this safety assessment process have a closed relationship with the software development process, which focus on software safety requirement, architecture, code and testing verification.

The basic idea of the military software safety assessment process is to conduct various verification and validation actions on different software functions according to its Software Criticality Index. Figure 2-9 shows the basic idea of the Software Criticality Index assignment process in the military software safety assessment process.



**Figure 2-9 Basic Idea of Software Criticality Index Assignment Process in Software Safety Assessment Process**

The assignment process can be summarized as the following steps [6]:

- Step 1: After the system safety assessment, starts to identify the software safety-significant functions and hazards related to these functions.
- Step 2: Starts to determine the software control category of the identified hazard by using the software control category table.
- Step 3: Through a combination of software control category and the severity category table, determines the software safety criticality index for each safety-significant function.
- Step 4: Determines the level of rigor (LOR) tasks by using software Safety Criticality Index.
- Step 5: Reviews the result and execution of level of rigor tasks:
  - Step 5a: If LOR tasks are fully completed, uses results to assign the software hazards to the system.
  - Step 5b: If LOR tasks are not performed or not completed, assigns risk level to software based on MIL-STD-882E;

This process uses Software Criticality Index to determine the level of each software safety-significant function, and perform the various assessment activities for the different Software Criticality Index. Assessment activity includes software safety assessment sub-process and proper safety assessment methods. Table 2-1 and Table 2-2 show the activities of each step in this process, and give the recommended safety assessment methods for each step.

**Table 2-1 Activities of Each Step in Software Safety Assessment Process-1 [23]  
[24]**

Steps	Activities (includes method)
System Definition and Software Safety Planning	<ul style="list-style-type: none"> <li>a) Define program safety scope.</li> <li>b) Conduct Functional Hazard Assessment and identify the software Safety-Significant Functions for each subsystem.</li> </ul>
Determine Software Criticality Index	<ul style="list-style-type: none"> <li>a) Assign Software Criticality Index to each software Safety-Significant Functions according to Figure 2-9.</li> </ul>
Software Requirements Hazard Analysis	<p>If Software Criticality Index is 1-3, it need to conduct assessment tasks:</p> <ul style="list-style-type: none"> <li>a) Conduct Software Requirements Hazard Analysis for each software Safety-Significant Functions.</li> </ul>
Software Architectural Hazard Analysis	<p>If Software Criticality Index is 1-3, it need to conduct assessment tasks:</p> <ul style="list-style-type: none"> <li>a) Review all architectural related documents to assess current hazards and review functional hazard analysis.</li> </ul>
Software Design Hazard Analysis	<p>If Software Criticality Index is 1-2, it need to conduct assessment tasks:</p> <ul style="list-style-type: none"> <li>a) Collect design explanation documents, such as the explanation of how the proposed architectural is implemented within the design.</li> <li>b) Conduct Software Causal Factor Analysis.</li> <li>c) Review all design documents and previous architectural analysis by using Conceptualized Control Flows.</li> </ul>
Code Level Hazard Analysis	<p>If Software Criticality Index is 1, it need to conduct assessment tasks:</p> <ul style="list-style-type: none"> <li>a) Conduct Data Structure Analysis.</li> <li>b) Conduct Data Flow Analysis.</li> <li>c) Conduct Compliance Checklist.</li> </ul>
Operator Documentation Safety Review	<ul style="list-style-type: none"> <li>a) Review operation documents such as the user manual, to identify new hazards and ensure adequacy of procedural controls.</li> </ul>

**Table 2-2 Activities of Each Step in Software Safety Assessment Process-2 [23]  
[24]**

Steps	Activities (includes method)
Software Safety Testing and Verification	If Software Criticality Index is 1-4, it need to conduct assessment tasks: a) Determine the safety testing requirements, such as the safety-specific or the in-depth safety testing in accordance with the Software Criticality Index by using AOP-52 and the JSSSEH. b) Analyze test results.
Formal Review	a) Provides enough evidence that the software failure related to the system risk has been identified and defined, and that risk level keeps at the accepted level by the appropriate authorities.

- Software Safety Assessment in Computer Science

**IEEE Standard for Software Safety Plans** [25] is used for the safety-critical software lifecycle, such as software development, maintenance. In this standard, it describes that the detailed steps in software safety analysis, and it includes the following steps [25]:

- Software safety analyses preparation
- Software requirements safety analysis
- Software design safety analysis
- Software code safety analysis
- Software test safety analysis
- Software change safety analysis

IEEE Standard for Software Safety Plans introduces the detailed objectives and activities of each sub-process of software safety assessment process. It also suggests the proper safety assessment methods according to objectives. Table 2-3 shows the objectives and proposed methods in each step of software safety assessment process.

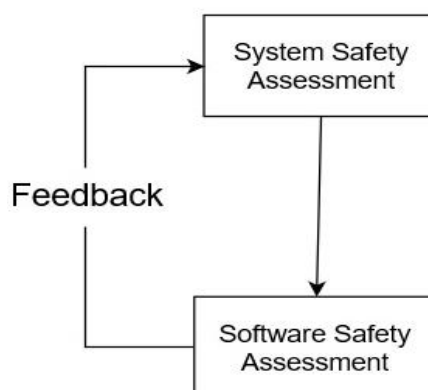
**Table 2-3 Objective and Proposed Methods in Software Safety  
Assessment Process [25]**

Steps	Objectives	Method
Software safety analyses preparation	Allocates documents to support the software safety analysis process, such as system FHA result, software requirements and the interface between hardware and software.	a) Review
Software requirements safety analysis	Evaluates the software requirements such as functional, interface and safety. Identifies errors and failure that could lead to the hazard.	a) Criticality analysis b) Specification verification c) Software system and function assessment
Software design safety analysis	According to the safety requirements, verifies the design of safety-critical function, and proves that it doesn't introduce any new hazard during the design.	a) Functional assessment Software element analysis
Software code safety analysis	Verifies the design of safety-critical functions is entirely implemented in the code correctly.	a) Code Logic analysis b) Code Data analysis Code Data Flow Analysis
Software test safety analysis	Proves that the safety requirements have been correctly implemented by software design. All the software functions have been tested, and the result is acceptable.	a) Software Unit Test b) Interface testing c) Integration testing
Software change safety analysis	Proves that the change doesn't introduce the new hazard; doesn't effect on the previously solved hazard, and doesn't cause the more severe effect on existing hazards.	a) Review

### 2.3.3 Relationship between System Safety Assessment and Software Safety Assessment

System safety assessment starts with identifying the related hazards and generates the safety requirements for system design which achieve to control, reduce and mitigate the hazard. The software safety assessment has the same purpose, and it applies the assessment process at the software level. As mentioned above, software safety is the system issue. Any failure of software may cause the system failure. So that, the software safety assessment process should contribute to the assurance of system safety by assuring the software safety.

Software safety assessment should be involved in the system safety assessment, and becomes the part of the system safety assessment. The output of system safety assessment is used as the input of software safety assessment, and the output of software safety assessment will be the feedback for system assessment. The relationship between software safety assessment and system safety assessment shows in Figure 2-10. For example, the system safety assessment determines the development assurance level of system-level functions or items. After this, the software assurance level assignment is based on the output of system safety assessment and proposed software architecture. Meanwhile, the software development assurance level will be returned to the system for verification and validation.



**Figure 2-10 Relationship between System Safety Assessment Software Safety Assessment**

Therefore, software safety assessment is not only important to the specific software, but also for the related system and whole aircraft. Software safety assessment aims to increase the confidence level that the software will perform as expected by identifying and controlling all the related hazards.

The next section will discuss the traditional and emerging software safety assessment methods, and these methods focus on identifying the failure or error that could lead to a hazard, and mitigate the software contributors to hazards.

## **2.4 Software Safety Assessment Techniques**

Safety assessment includes kinds of safety assessment techniques to satisfy different objectives. The common safety assessment techniques, such as Fault Tree Analysis, Failure Modes and Effects Analysis, start to be used for safety-critical software. Meanwhile, some new methods such as formal method apply mathematics and logic to assess the software. This section will mainly discuss the working principle and working steps of both the traditional and emerging software safety assessment techniques.

### **2.4.1 Traditional Software Safety Assessment Methods**

This section mainly discusses three safety assessment methods which are functional hazard assessment, fault tree analysis, and failure mode and effects analysis respectively.

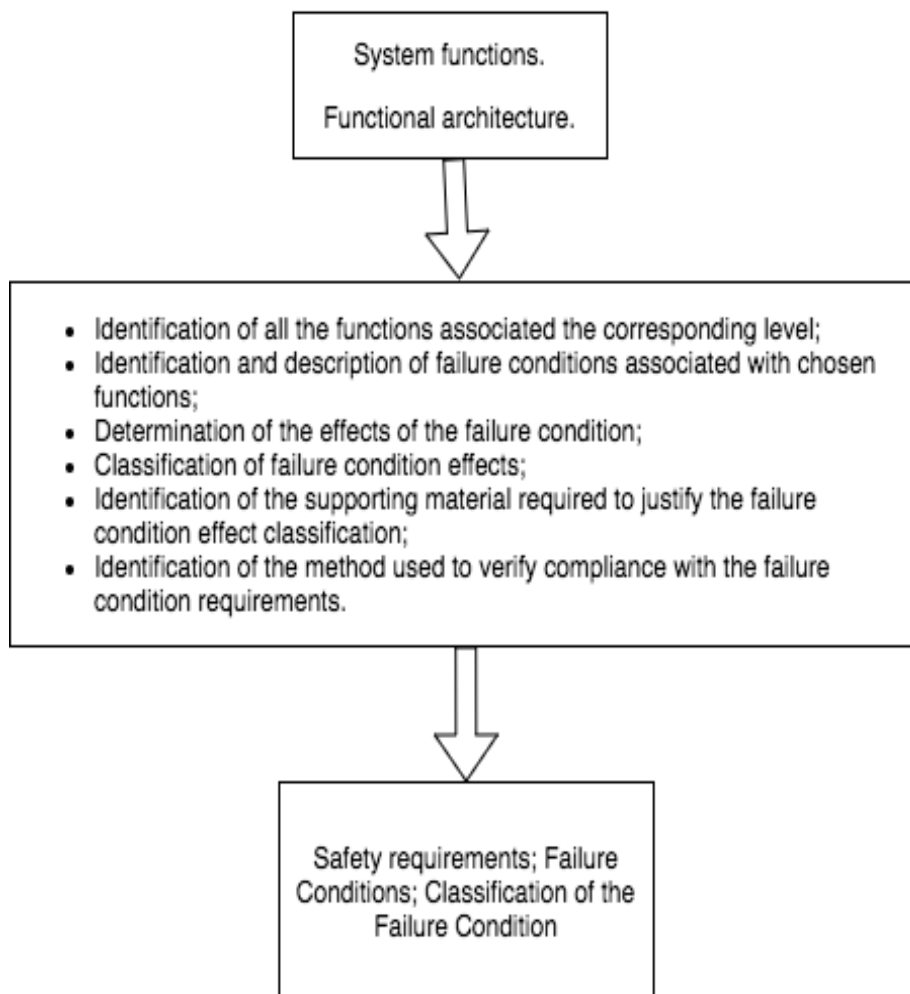
- **Functional Hazard Assessment**

Functional Hazard Assessment (FHA) is the comprehensive functional assessment process to identify the failure condition and to classify the failure condition according to its severity [9]. FHA can conduct in both aircraft and system level, and it usually begins in the early stage of aircraft or system development process. The aircraft level FHA is a high-level assessment, and it mainly assesses functions at the beginning of an aircraft development. The identified hazards from the aircraft level FHA are related to the aircraft function. The system level FHA is to assess the system function, thereby identifies and classifies the single or combination hazards which influence on aircraft. The FHA



aims to identify all associated failure conditions and classify the severity level according to its effects. The classification of failure conditions establishes the safety requirements of aircraft, system or software.

Generally, FHA conducts at the higher level, such as aircraft and system level. But it can also identify hazards at software level. The principle and process of software FHA and aircraft/system FHA are same. The general process is shown in Figure 2-11:



**Figure 2-11 Functional Hazard Assessment Process [9] [10]**

The basic idea of conducting FHA is to identify the scope of assessment firstly, and to determine all the possible failure conditions by the assumption of the function failure, such as “loss control” and “loss warning”. Last, to define the classification and effect by the related supporting materials.

The input of FHA is the system functions and functional architecture. The output of FHA should include several information associated with aircraft, system or software:

- The related failure conditions.
- The effects of failure condition, which include the effect on aircraft, crew and passenger.
- Classification of each failure condition according to its effect. The Classification is divided into five categories, Catastrophic, Hazardous, Major, Minor, and No Effect.
- Assumed environment is considered during the FHA.


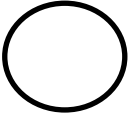
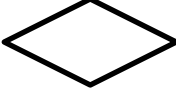




The output of FHA is the list of failure conditions and its classification, which is the fundamental of the fault tree analysis. The fault tree analysis can use to identify failure conditions as its top event. The next section will introduce how to decompose the failure condition and how to find the basic cause of the related failure condition.

- Failure Fault Tree Analysis

Fault Tree Analysis (FTA) is a top-down analysis approach that was developed in 1962 under U.S. Air Force Ballistics Systems Division at Bell Laboratories [26]. FTA can be applied to almost every phase of software lifecycle. For example, it can be used in software requirement assessment for eliciting safety-related requirement, and can also be used in software design.

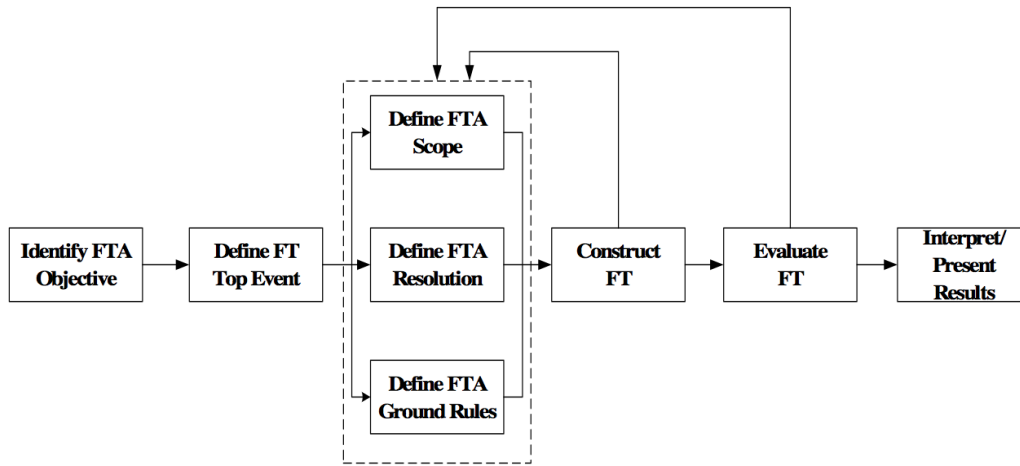
FTA is to identify the software errors or mistakes which caused the failure condition or hazard. By using the graphical presents the combination of component failures or errors that leads to the high-level failure [27]. The typical fault tree is constructed by symbols, and each symbol expresses the different meanings or attributes. Table 2-4 shows some frequently used fault tree construction symbols [28].

**Table 2-4 Symbol of Fault Tree Analysis [28]**

Symbol	Name	Description
	Intermedia Event	The event can be analysed and decomposed further.
	Basic Event	It presents the event is the basic reason of the hazard, and cannot decomposed any more.
	Undeveloped Event	This is used for some event cannot developed further due to the lack of information or out of scope.
	AND gate	If all of the input faults occur, the out fault happen.
	OR gate	If at least one of the input faults occurs, the output fault happens.
	Transfer In	The tree will be developed further as the occurrence of the symbol of Transfer out.
	Transfer Out	The fault tree must be attached at the symbol of Transfer in.

The FTA starts with the list of failure conditions that have been found in system-level FHA. Each failure condition should have one individual tree, and the primary procedure of FTA is to assume the failure condition has happened, and to go

backward to identify all the possible causes. The general process of fault tree construction shows in Figure 2-12:



**Figure 2-12 Construction step of Fault Tree Analysis [29]**

Software Fault Tree Analysis is the approach that applies normal FTA to software level, which helps to identify the software errors or failure. The construction steps are same with the normal FTA. However, it needs to take account of software function and software architecture [8]. Before construction, software FTA needs to specify the purpose and scope of this fault tree. Then, uses the list of failure conditions found in the system functional hazard assessment as the top events of software FTA. After defining top event of the fault tree, it can start to decompose until found the basic events of this fault tree. The general procedure of SFTA shows in Table 2-5:

**Table 2-5 Step of Software Fault Tree**

Step 1	Define the purpose and scope of the software fault tree analysis.
Step 2	Define all the undesired event (failure conditions).
Step 3	Identify causes for top-level fault (upper tier).
Step 4	Identify next level of events (intermediate tier).
Step 5	Identify root causes.
Step 6	Analysis the result.

Cut set analysis can be performed to determine the minimal cut sets of the corresponding fault tree after the construction. This analysis gives the several sets of the basic events that can cause the top event individually.

Fault tree analysis can identify all the possible causes of the related failure condition, which can help engineer to recognize the weakness of the system or software. The other advantage of FTA is that can help engineer identify the human error. However, FTA has some disadvantages, such as FTA only focuses on the cause of the failure, but it cannot provide any suggestion or information if the failure occurs. FTA doesn't care about the effects and solution of each failure condition.

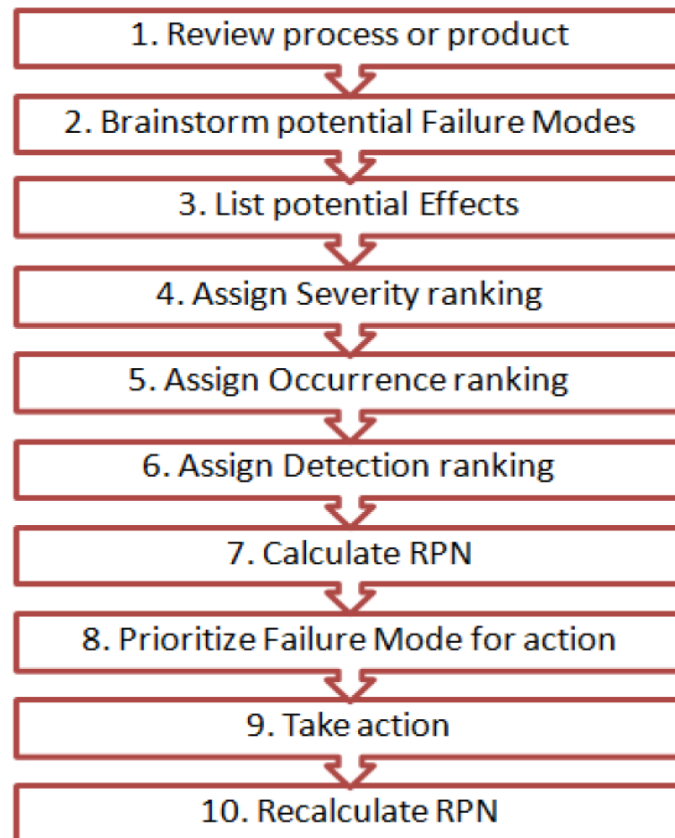
- Failure Mode and Effects Analysis

Due to the working principle of FTA, FTA cannot consider the common mode failure, which means one error or mistake may cause more than one hazard. So, many standards and researches recommend the bi-direction approach which can work backwards and forwards [16]. Failure Mode and Effects Analysis (FMEA) is one of the bottoms-up analysis methods, and it was developed by the military for studying the possible failure of the military system in 1950s [30]. This method cares about how each component failed, and how the component failure propagated to the system. FMEA is one of the fault analysis methods to identify, control and eliminate the possible failure mode during each phase of the development, and it focuses on failure prevented.

The Risk Priority Number (RPN) is the important part of FMEA analysis, and it is used to rank each identified failure mode. The RPN is using the values of Severity (S), Occurrence (O), and Detection (D). The severity means the consequences of the failure. The occurrence means the frequency of the failure occurred, and the detection means the probability of the failure detection. These three factors can be ranked from 1 to the 10, and 10 is the most serious [31]. RPN value is to multiply the ranking of three factors and the formula shows as below:

$$RNP = \text{Severity} \times \text{Occurrence} \times \text{Detection}.$$

The RPN should be used for each possible failure modes for determining the effects. However, this RPN may lead to the reversible ranking. One failure has a less severity value, but receives a higher RPN than one more severe failure mode. For example, the RPN of a failure mode which the severity ranking is “2”, may be lower than a failure mode which the severity ranking is “1”, because of the higher occurrence or detection ranking.

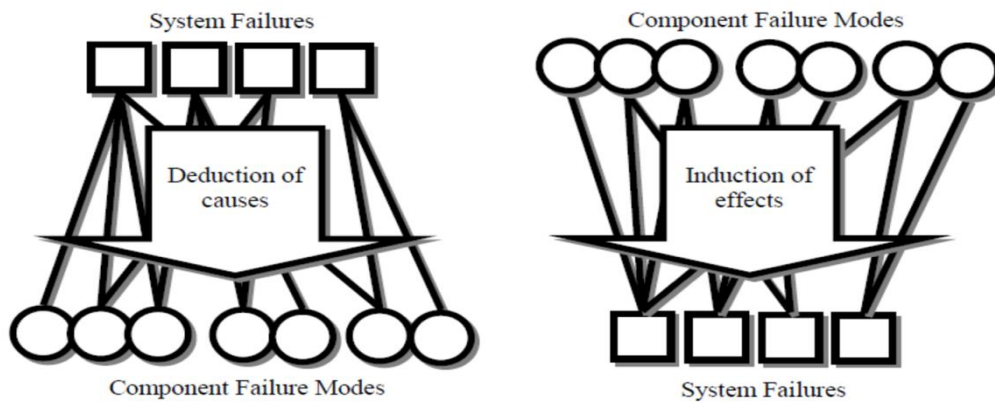


**Figure 2-13 Failure Mode and Effects Analysis Process [31]**

The output of FMEA typically includes the list of identified components or functions, the failure modes related to identified function, and the effect of this failure [9]. The process of FMEA shows in Figure 2-13.

Software Failure Mode and Effects Analysis is to apply FMEA to software. The procedure of Software Failure Mode and Effects Analysis is similar with FMEA, and it starts to identify all possible failure modes related software. And then, it works forward to find the effects on the system or aircraft.

FTA and FMEA are both analytical methods for fault analysis, but they focus on different purposes.



**Figure 2-14 Difference between FTA and FMEA [32]**

The working principle of FTA and FMEA shows in Figure 2-14. FTA focuses on the consequences of the failure condition. Therefore, this method is to identify all the possible basic causes related to the hazard by checking all the available components. The cause of a hazard can be the single failure or the combination. However, FMEA focuses on the individual failure modes, and it exams all the possible failure modes for determining effects on the higher-level component or system. [33].

### **2.4.2 Emerging Software Safety Assessment Methods**

Besides traditional safety assessment methods introduced in the previous section, there are many new methods for software safety assessment, and some of them have been already used in the practical software safety assessment project. This section briefly introduces the formal methods, and gives the detailed explanation of model checking.

- **Formal Method**

Formal Method is not a single method, and it is the set of techniques and tools. Formal method is to describe and verify the safety problem by using mathematics and logic. This method aims to guarantee the safety of system and software.

Formal method includes three parts of activities, and they are respectively formal specification, program refinement and formal verification [34].

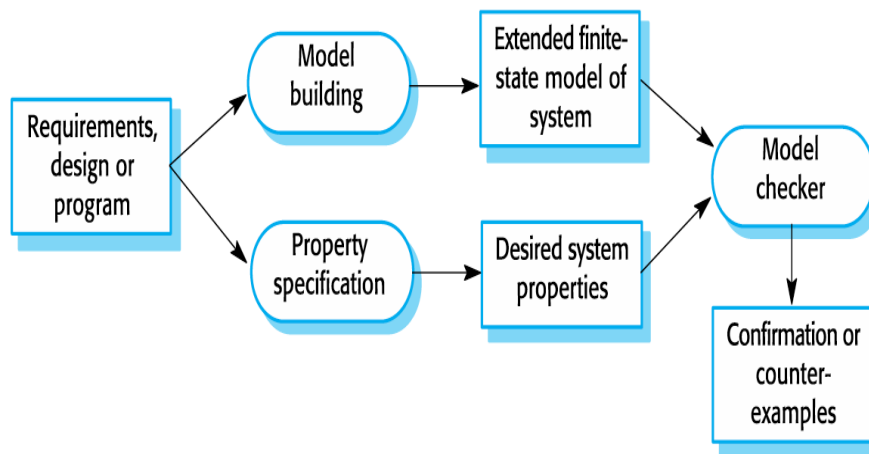
Formal specification refers to a collection of methodologies for system objects, and its operations. Formal specification also can be the description of the behaviour of each object in the development process. Formal verification is the next step after the formal specification, and it usually uses two techniques, which are theorem proving and model checking. Program refinement is the new technology which combines the formal method and automated reasoning.

Theorem proving uses the logical formula to regulate the system and its properties. A formal system will provide axioms or regulations, and theorem proving applies these axioms or rules to prove whether the system has specific properties or not. Model checking is a technique to check the desired properties of one finite state model. This technique can examine each possible state of the system to check whether the desired properties hold for a model or not. Properties can be related to system or software safety, liveness and functional [16]. Compared with theorem proving, model checking can be fully automated and conducts the verification task without human intervention. However, model checking has a disadvantage which is the state explosion. The different states rise exponentially in the concurrent system due to the increasing scale of the system. However, the development of Binary Decision Diagrams and Symbolic Model Checking has solved this problem. In the next section, the author will mainly discuss the related knowledge and the working principle of model checking.

- Model Checking

As mentioned above, the working principle of model checking is to exhaustively and automatically search the given model and to verify whether the desired properties are held in this model or not. If the answer is yes, it proves the model and specification is consistent. Otherwise, it gives the encounter example which points out the error. The working principle can be described as Figure 2-15.





**Figure 2-15 Model Checking Formula [35]**

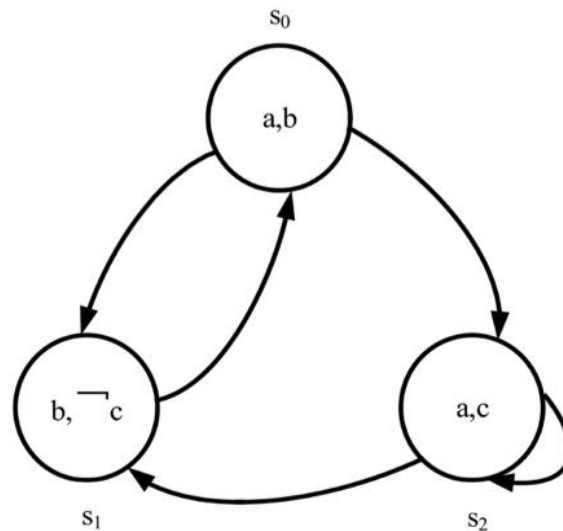
In the formal modelling techniques, a model is a formal description of the object, such as a function, a system, and a piece of equipment [34]. Kripke structure is one of the modelling languages.

- Formal Modelling Language - Kripke structure

A Kripke structure is used to express the model in the model checking. The Kripke structure was proposed by Saul Kripke in 1963, and used to present the behaviour and transition relationship of the system [36]. The Kripke structure can describe all the internal relationships of a state transition system.

A Kripke structure consists of five tuples, and it presents as  $K = (S, S_0, \delta, AP, L)$ .  $S$  is a finite set of states.  $S_0$  is the initial state or the set of initial state, and  $S_0$  belongs to  $S$ .  $\delta$  is the transition relation between each state.  $AP$  presents the set of atomic propositions, and atomic propositions are the propositions cannot be further divided, such as the state is true.  $L$  means the labelling function, and it presents as  $L: S \rightarrow 2^{AP}$ .  $L(a)$  is the set of atomic propositions, and it expresses as the value of  $L$  is true when under the state  $a$  [37]. The Kripke structure can be express as the graph which has the label, root and direction. The set of the node equals to the set of finite states  $S$ ; the set of the edge equals to the set of transition relation  $\delta$ ; the label of the node equals to  $L$ , and the root of the graph equals to the  $S_0$ .

So, a Kripke structure can be translated into one state transition graph. However, the state transition graph is not the formal model, and is the way in helping to get the better understating of the software behaviour. Figure 2-16 shows example of a state transition graph, and this graph can be expressed in a Kripke structure.



**Figure 2-16 State Transition Graph for Kripke Structure [38]**

The Kripke structure of Figure 2-16 can be expressed as:

- $S = \{S_0, S_1, S_2\}$
- Initial state is  $S_0$
- Transition relation  $\delta = \{(S_0, S_1), (S_1, S_0), (S_0, S_2), (S_2, S_1)\}$
- $L(S_0) = \{a, b\}, L(S_1) = \{b, \neg c\}, L(S_2) = \{a, c\}$

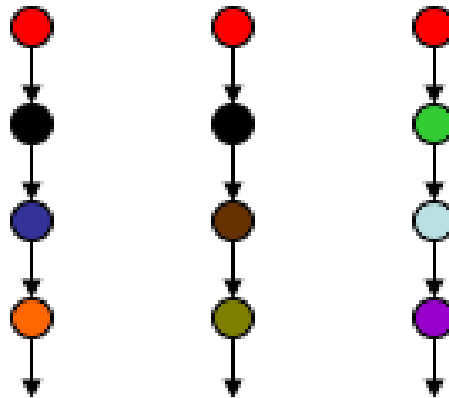
The informal model can be formally modelled by the Kripke structure, and the properties should be also translated into the formal specification. In model checking, it usually uses temporal logic to express the property.

- Temporal Logic

Temporal logic is to specify the property over time, but it doesn't really care about the time. It cares about the states and their relative position in the model. Temporal Logic can also express the dynamic state of the system. Generally, temporal logic mainly is divided into two categories. One is Linear Temporal Logics, and the other is Computation Tree Logic.

– Linear Temporal Logic

Linear Temporal Logic (LTL) was introduced by Pnueli in the 1970s [39]. It is widely used to express the behaviour of events on a calculation path. LTL formula consists of a formula  $p$  and a temporal logic operators “A”, such as  $A p$ . Figure 2-17 is the graphic representation of Linear Temporal Logic, and shows that every node has a unique successor. Each path can use to describe one LTL formula.



**Figure 2-17 Linear Temporal logic [40]**

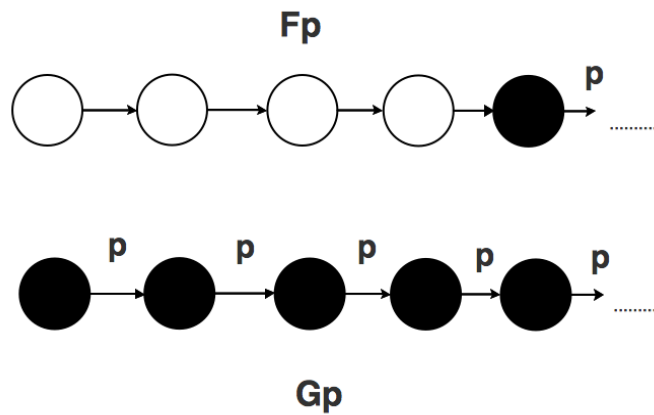
The LTL formula consists of the following components [38] [37]:

- Given the atomic proposition set  $AP$ . (State label  $p \in AP$ ).
- Basic Boolean Operators.  $\neg$ (Negation),  $\wedge$ (Conjunction).
- Basic Temporal Operators.  $X$  (next time),  $F$  (eventually),  $G$  (always),  $U$  (until).

The syntax of LTL formula is defined as following [38] [37]:

- If all the atomic proposition  $p \in AP$ , then  $p$  is the valid LTL formula.
- If  $p$  and  $q$  are two valid LTL formulas, then  $\neg p$ ,  $p \vee q$ ,  $p \wedge q$ ,  $p \rightarrow q$ ,  $p \leftrightarrow q$ ,  $Xp$ ,  $Fp$ ,  $Gp$ ,  $(p U q)$  are also valid LTL formulas.

Use the syntax can describe the state transition in one model, such as “Always after  $p$  eventually  $q$ ”. It can be formalized as  $AG (p \rightarrow AFq)$ . Figure 2-18 is shown the typical LTL formulas.

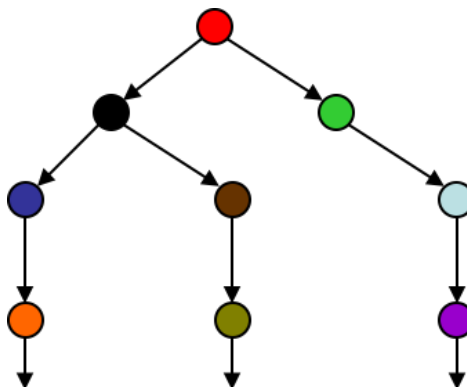


**Figure 2-18 Example of LTL Formula**

If there is a given Kripke structure  $K = (S, S_0, \delta, AP, L)$ , and LTL formula  $p$ . In LTL model checking algorithm, the model checker will exam whether the path  $\pi$  satisfy  $p$  or not.

– Computation Tree Logic

Computation Tree Logic (CTL) was created by E. Emerson and E. Clarke in 1979. It is used for presenting the property of Branching Time Temporal in the program at the initial stage of development [41]. Figure 2-19 shows graphic representation of Computation Tree Logic and shows every node has several successors. Compared with LTL, CTL uses the computation tree instead of a linear path, and introduces the path quantifiers to present that the properties must be held in all states or some states starting from the initial state.



**Figure 2-19 Computation Tree Logic [40]**

In CTL, there are two path quantifiers. One is “A” which means all executions or paths, and the other is “E” which means for some execution. CTL syntax requires that one of path quantifiers and one of temporal operators should be used in a pair, and path quantifier should be written before the temporal operator, such as AG and EX. So, XA and XF are not allowed in CTL syntax.

The syntax of CTL formula, it combines with the basic temporal operators X (next time), F (eventually), G (always), U (until), and the path quantifiers (A, E) to describe state transition:

$$\begin{matrix} [A] \\ [E] \end{matrix} \begin{matrix} [X] \\ [F] \\ [G] \\ [U] \end{matrix} p$$

In CTL formula, EX, EG and EU are the basic combination operators, and rest of combination operators can be transformed, such as  $\neg EX p = AX\neg p$ .

This section gives several CTL formula examples and its descriptions. Thesis uses one computation tree as a running example, and this computation tree is with varying distribution of the red and black states. The given formula  $p$  and  $q$  are true if the black states satisfy  $p$  and red states  $q$  [37].

$AG p$  is true when  $p$  satisfies all the states on all paths which starts from the initial state in a computation tree. Figure 2-20 shows the  $AG p$  in a computation tree.

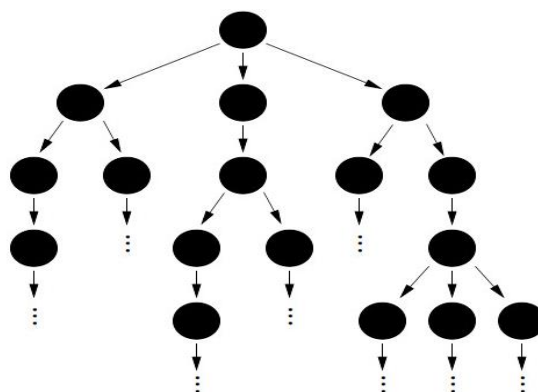


Figure 2-20  $AG p$

AF  $p$  is true when there is one state which satisfies  $p$  on all paths starting from the initial state in a computation tree. Figure 2-21 shows the AF  $p$  in a computation tree.

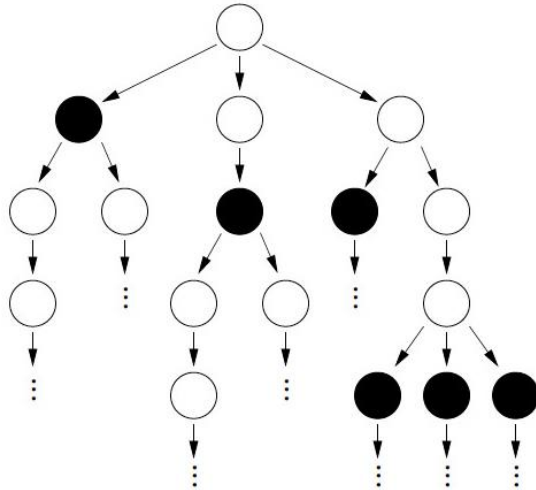


Figure 2-21 AF  $p$

EG  $p$  is true when every state satisfies  $p$  on a path which starts from the initial state in a computation tree. Figure 2-22 shows the EG  $p$  in a computation tree.

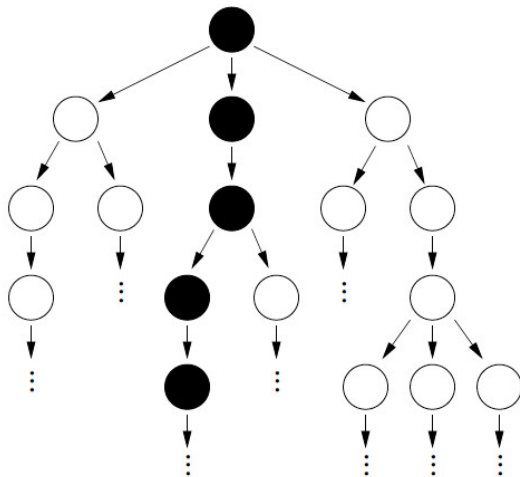
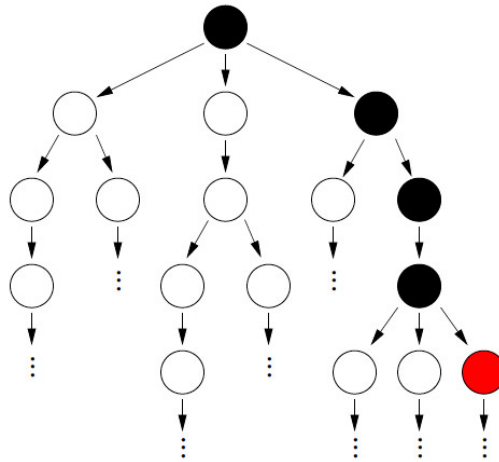


Figure 2-22 EG  $p$

E  $(p \cup q)$  is true if there is a path starting from the initial state satisfies  $p$  until reaches one state which satisfy  $q$ . Figure 2-23 shows the E  $(p \cup q)$  in a computation tree.



**Figure 2-23 E ( $p \text{ U } q$ )**

The syntax of CTL formulas need to follow the several rules, and shows as below [37]:

- If all the atomic propositions  $p \in AP$ , then  $p$  is the valid CTL formula.
- If  $p$  and  $q$  are two valid CTL formulas, then  $\neg p, p \vee q, p \wedge q, p \rightarrow q, p \leftrightarrow q, AXp, EXp, AFp, EFp, A(p \text{ U } q), E(p \text{ U } q)$  are also valid CTL formulas.
- The valid CTL formula  $p$  can only be created by applying the first two steps within a limitation times. Otherwise, the formula  $p$  is not a valid CTL formula.

CTL model checking is to verify whether all the states or some states in the path  $\pi$  of a Kripke structure  $K$  satisfy the CTL formula  $p$  or not. The syntax of CTL model checking can be described as below:

$K, S \models p$  expresses that one formula  $p$  is true when at state  $S$  in Kripke structure  $K$ .  $\pi = \{S_0, S_1, S_2, \dots\}$  means the path in Kripke Structure, and  $S_0$  is the current state which is also the initial state.  $S_{i+1}$  is the successor state of  $S_0$ . So, the semantics of CTL can be defined as [37] [38]:

- $K, S \models p$  if  $p \in L(S)$
- $K, S \models \neg p$  ( $p$  is false at state  $S$ ).
- $K, S \models p_1 \wedge p_2$ , if  $p_1 \in L(S)$  and  $p_2 \in L(S)$ .

- $K, S \models AX p$ , if for all paths  $\pi = \{ S_0, S_1, S_2, S_3, \dots \}$ , starting in  $S_0$ ,  $p$  is true when at state  $s_1$ .
- $K, S \models EX p$ , if there exists a path  $\pi = \{ S_0, S_1, S_2, S_3, \dots \}$ , starting in  $S_0$ ,  $p$  is true when at state  $S_1$ .
- $K, S \models E (p1 \cup p2)$ , if there exists a path  $\pi = \{ S_0, S_1, S_2, S_3, \dots \}$ , starting in  $S_0$ ,  $p1$  is true until  $p2$  is true.

There is some CTL formula used to describe safety features:

- $AG p$  means invariant. In requirement specification, it will use for describing the status of system or software function cannot be changed at any time, if  $AG p$  is true.
- $EF p$  means potential. In requirement specification, it will use for describing the status of system or software function might be changed at a particular time, if  $EF p$  is true.
- $AF p$  means Inevitable. In requirement specification, it can describe the hazard.

Compared with CTL, LTL focuses on the individual path, but CTL focuses on the multiple paths. The syntax of LTL is simpler than CTL, because CTL has two path quantifiers. For semantically, two logical are Incomparable. So, depending on the target problem and its requirements to choose the proper temporal logic.

#### ● Model Checker

Based on the different platforms and purposes, lots of research institutions and labs developed various types of model checkers, such as SPIN developed by Bell Labs and NuSMV developed by Carnegie Mellon University.

SPIN is one of the typical Explicit-State Model Checkers, and it can verify the multi-threaded software efficiently. So, this model checker is suitable for verifying the concurrent software design [42].

NuSMV is one of the symbolic model checkers which re-implemented the SMV model checker by Carnegie Mellon University, and this checker supports the



verification of properties expressed in both CTL and LTL formula [43]. Table 2-6 briefly shows the comparison between two model checkers.

**Table 2-6 Comparison of Model Checkers**

Name	Input Language	Properties description language	Platform
SPIN	Promela	LTL	Windows, Linux
NuSMV	SMV	CTL, LTL	Windows, Linux, MacOS

## 2.5 Summary

After the literature review, there are several findings in software safety assessment process and methods.

First, the software safety assessment process is essential for software, system, even the aircraft. This process helps engineer to identify the software hazards which contributes to the high-level failure. This process can control, reduce and eliminate the hazard at the initial stage of development. The software safety assessment process guarantees correctness of software and adds rigor and robustness to safety significant software.

Software safety assessment is not a simple and independent process, and it has a closed relationship with other processes, especially the software development process. The software safety assessment process should be integrated into the entire software lifecycle, and there is a one-to-one correspondence relationship between software safety assessment process and software development process. For example, the development process includes the requirement process, and the proposed software safety assessment process should have one sub-process which focuses on requirement assessment and verification.

Last, software FTA and software FMEA are two recommended approaches for identifying the software failure which contributes to system hazards. However, these two fault analysis approaches don't have the capability of assuring the

correctness and consistency. Model checking is one of the emerging approaches in software assessment method, and it checks whether a desired property to be held in a target model or not. If the answer is yes, it can confirm the correctness and consistency of the model and properties. So, combined FTA, FMEA and model checking not only can elicit the software safety requirements, but also can prove the correctness and consistency.

# 3 Avionics Software Safety Assessment Process

## 3.1 Introduction

This Chapter discusses the proposed software safety assessment process for avionics software. As mentioned in pervious, the software safety assessment should have the one-to-one corresponding relationship with software lifecycle, especially the software development process. Several guidelines or standards discuss software development process model for different purposes, such as DO-178C [5].

## 3.2 Software Development Process in DO-178C

DO-178C is one of the widely accepted standards to assure safety of airborne software, and this document provides the guidelines for development and safety assurance of airborne software based on software lifecycle [5].

Software lifecycle is divided into five parts, and software development process is the secondary part of the DO-178C. And the software development process includes four sub-processes, and they are respectively software requirement process, software design process, software coding process and integration process. Figure 3-1 shows content and objectives of the software development process.

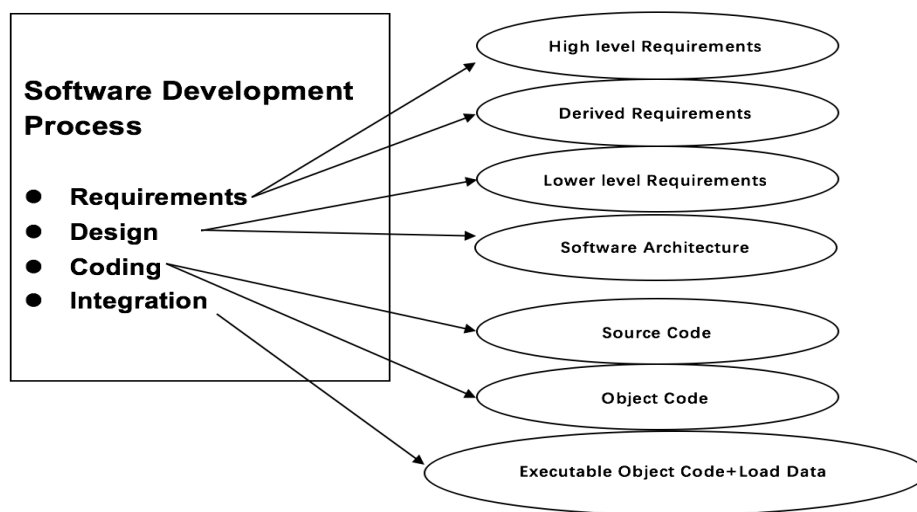


Figure 3-1 Software Development Process in DO-178C

The software requirement process is to develop the high-level software requirements which include functional, safety and performance requirements. The software architecture process is to develop software architecture and lower-level requirements that can be used for the software coding process. The coding process is to implement software design included software architecture and lower-level requirements. The last is the integration process, and this process is to produce and load the executable object code and related files.

Through the objectives of each sub-process in software development process, three objectives are necessary for considering during software safety assessment, which are software requirement, software architecture and code. Therefore, the software safety assessment process should focus on these components, and assess each individually. However, requirements are divided into different categories, such as the functional, safety, interface and so on. In this research, the author focuses on the safety-related requirements, which may involve software function or interface.

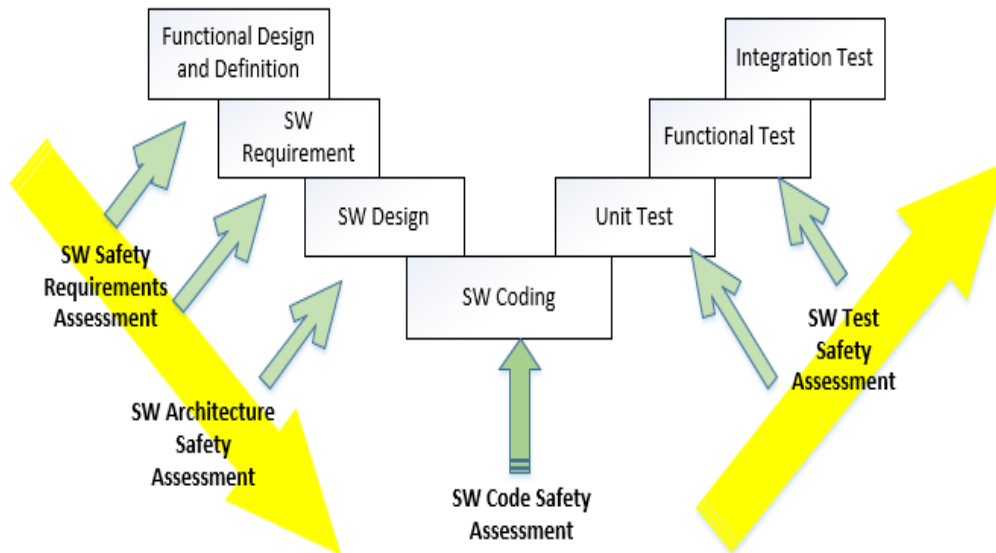
Besides software development process, the safety assessment process should also include the software testing. Software testing provides evidence to prove the confidence level of the software development process by demonstrating whether the software satisfied its requirements or not [5]. As mentioned in Chapter two, different kinds of software safety assessment process include that all the sub-processes which is from the beginning of software development to the end of software testing.

So, the completely proposed avionics software safety assessment process should consist of four parts, and they are requirement assessment, architecture assessment, software code assessment and software test assessment. The next section discusses the detailed objectives and tasks of each sub-process.

### **3.3 Avionics Software Safety Assessment Process**

Based on the avionics software development process and existing software safety assessment process, the proposed avionics software safety assessment process should include software safety requirement assessment, software

architecture safety assessment, software code safety assessment and software test safety assessment. Figure 3-2 shows the interactive relationship between the development process and proposed software safety assessment process.



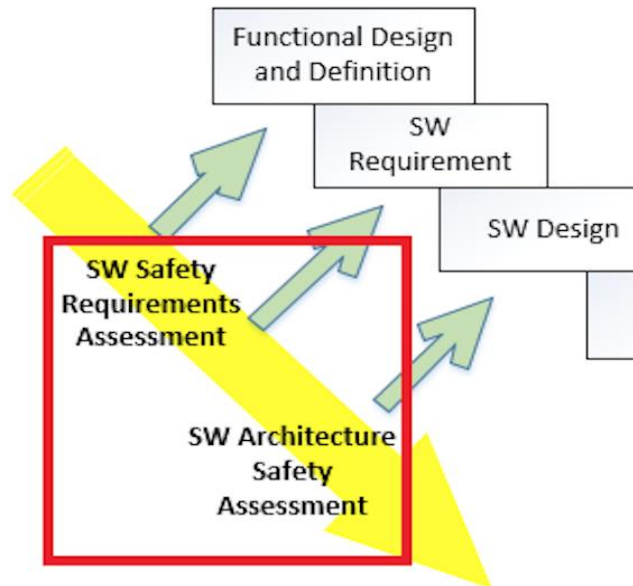
**Figure 3-2 Proposed Avionics Software Safety Assessment Process**

The software safety requirement assessment is the first step of the entire software safety assessment, and it mainly verifies correctness of software safety requirements and consistency between the safety requirements and software functions. The software architecture safety assessment is to assign and re-assign the Development Assurance level of software, which ensures the software architecture conforms to the assigned Development Assurance level. The software code safety assessment is to verify whether the code implements all the requirements as intended or not. The final sub-process is the software test safety assessment, which is to check all the software testing document and test cases are correct, and all the found hazards are controlled or eliminated.

However, software safety assessment process contains many contents, which are hard to focus on all sub-processes of the proposed software safety assessment process. So, this research places emphasis on the previous two sub-

processes which show in Figure 3-3. One is software safety requirement assessment, and the other is software architecture safety assessment.

Chapter four will discuss these two sub-processes, and Chapter five will use the practical case study to prove feasibility of safety requirement assessment process.



**Figure 3-3 Emphasis of Research**

The next section gives the brief introduction of the entire proposed software safety assessment process, and proposed the proper methods for each sub-process according to its objectives.

### **3.3.1 The Objectives and Tasks of Each Steps**

This section gives the description of each sub-process, and points out the objectives and available safety assessment methods according to its objectives.

- **Software Safety Requirement Assessment**

The objective of Software Safety Requirement Assessment is to identify and elicit software safety requirements, and then to verify the correctness and consistency of safety requirements. In this process, it has two tasks. One is the safety requirements elicitation and the other is requirements verification. The first task needs to identify the associated hazards by examining the consequences of the whole software failed, or some software components failed. Then, it to find the

basic causes related to this hazard. For this task, some traditional safety assessment methods can help to identify the hazards and basic events, such as FHA and FTA. The second task is to verify the safety requirements which elicited from the previous task, and some emerging methods such as formal method can be used.

- Software Architecture Safety Assessment

The objective of Software Architecture Safety Assessment is to verify whether the proposed software architecture satisfies the assigned development assurance level of software or not. In this process, it provides two kinds of software Development Assurance Level assignment process. The activity of this process is to assign and re-assign the software development assurance level. Through comparing the results, the engineer can check the software architecture.

- Software Code Safety Assessment

The objective of Software Code Safety Assessment is to ensure software code which is consistent with the relevant requirements. According to different analytical purpose, code analysis has the various tasks. For example, code logic analysis is to find and correct logic errors. There are lots of analytical tools based on the type of programming language, such as SPARK Toolset for Ada, and Eclipse for language C. Code data analysis focuses on the data structure. Data-flow analysis can help the compiler to optimise the program, and formal inspection of source code is to assess the quality of code.

- Software Test Safety Assessment

The objective of Software Test Safety Assessment is to prove all the hazards which found in the entire safety assessment process have been controlled or reduced, and all the hazards has been maintained at an acceptable level. This process mainly conducts review or checking to perform the analysis. For example, it will check and review the safety-related testing documents, test cases and other testing material related to safety and quality of software.

### **3.3.2 Transition Criteria of Proposed Avionics Software Safety Assessment Process**

Transition criteria are the entrance and exit requirements for each process, and determine whether one process can be entered or exited. The general transition criteria are shown as below, and might need to be more specific when apply to the practice.

- Software Safety Requirement Assessment
  - The entry criteria for the Software Safety Requirement Assessment are: Preliminary system safety analysis has been finished, and system architecture has been established and formally released. All system level requirements have been allocated to the software level.
  - The exit criteria for the Software Safety Requirement Assessment are: Software safety-related requirements have been elicited and verified. The verification results and specifications of software safety-related requirement have been documented and formal released. All the hazards found in this process have been documented.
- Software Architecture Safety Assessment
  - The entry criteria for the Software Architecture Safety Assessment are: Initial software architecture has been established. System function has been allocated to software level, and the development assurance level of the related system has been assigned and verified.
  - The exit criteria for the Software Architecture Safety Assessment are: Enough evidence generated to prove the final software architecture satisfies the software development assurance level, and software architecture has been documented and formally released. All the hazards found in this process have been documented.
- Software Code Safety Assessment
  - The entry criteria for the Software Code Safety Assessment are: Review of software architecture has been finished and all source code has been generated.
  - The exit criteria for the Software Code Safety Assessment are: All the source code could be traced to the corresponding safety requirements,



and source code could be proved the consistent with the related safety requirements. All the hazards found in this process have been documented.

- Software Test Safety Assessment
  - The entry criteria for the Software Test Safety Assessment are: the corresponding verification activity has been performed.
  - The exit criteria for the Software Test Safety Assessment are: enough evidence generated to prove all of the hazards have been eliminated or controlled at an acceptable level, and all identified error or deviation is documented and feedback as a Problem Report.

### **3.4 Summary**

The Chapter firstly introduces the software development process in DO-178C, and produces the proposed avionics software safety assessment process according to the software development process and literature study. The Chapter also lists the objectives of each sub-process, and recommends the software safety assessment methods for each sub-process. The Chapter also discusses the general transition criteria between each sub-process.



## **4 Proposed Methods for Avionics Software Safety Assessment**

### **4.1 Introduction**

As mentioned in Chapter 3, this research focuses on the first two sub-processes of the avionics software safety assessment process proposed in Chapter 3. This Chapter will discuss in detail the assessment process for software safety requirement and architecture, and proposes the proper software safety assessment methods for each sub-process.

This Chapter is divided into two parts. The first part is to discuss the software safety requirement assessment process and the proposed safety assessment method. The second part is to introduce the assignment process of software development assurance level and to use for verification.

### **4.2 Methods for Software Safety Requirements Assessment Process**

In the *IEEE Recommended Practice for Software Requirements Specifications* (IEEE Std 830-1998) [44] gives the 13 categories of software requirements and includes function, interface, and safety requirement. Safety requirement is the requirement that indicates the “shall” and “shall not” behaviour of software, system and aircraft, such as the light shall not be turned on until the button is pressed.

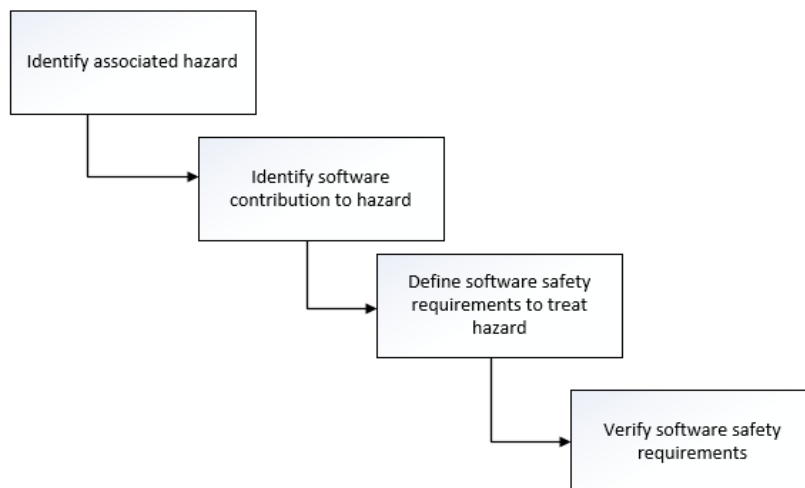
The scope of the safety requirement is widely. Sometime, the safety requirement will involve with all the requirements which include functions for safety-critical system and software. Sometimes, the safety requirements will consist of those requirements which protection operations, fail-safe design or other design related to safety.

Safety requirement is the set of safety objectives obtained from aircraft-level, system-level, and software-level safety assessment process. Usually, it is decomposed the higher-level requirements from the top-level to the lower-level. At aircraft level, the safety requirements are those requirements generated from

the aircraft FHA based on aircraft functions. At the system level, the safety requirements are all those system-level requirements generated from the system FHA. So, the elicitation of software safety requirement is the same process with the aircraft level and system level.

Through the literature study, the author found FTA and FMEA are two useful approaches for eliciting safety requirements, and the model checking is the proper method for verification. So, the proposed method for safety requirement elicitation and verification is to combine fault analysis approaches and model checking together.

The software safety requirement assessment process includes four steps. The first one is to identify associated hazards, and then to determine software errors or fault which caused the hazard. Third is to create safety requirements formal specification by using temporal logic, and last is to conduct the formal verification by using model checker tools. The next section will discuss each step in detail. Figure 4-1 shows the general software safety requirement assessment process.

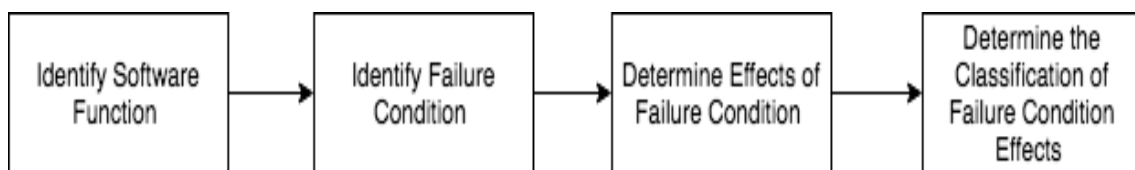


**Figure 4-1 Process of Software Safety Requirement Assessment**

The previous two steps aim to obtain the safety requirements by using process the traditional safety assessment methods. The entire safety assessment process begins with the identification of the hazards associated with software, and identifies all the possible causes that can contribute to the failure conditions.

### 4.2.1 Identify hazard

The first step of the software safety requirement assessment process is to identify all the hazards related to the target software. In this step, it will use software functional hazard assessment, but it doesn't need to perform all the actions of normal FHA procedure. Software functional hazard assessment only focuses on identifying and classifying the failure conditions. And then, it determines the effects of each failure condition. Figure 4-2 shows the process of software FHA for identification.



**Figure 4-2 Process of Hazard Identification**

First, software FHA is to determine all the functions associated with target software. After preliminary system safety analysis, it can get the software function list and software initial architecture. This function list is the important input of software FHA, and is utilized to identify the failure condition. However, the hazard is very general, such as loss of control. If loss of control hazard needs to be examined, it might identify several failures related to many systems. So, it is necessary to define the scope of the safety assessment process, and specify the system function.

Second, software FHA is to identify the related failure conditions according to its function. It should consider the consequences when software function failed, and then identify the hazards associated with this function. Environmental and emergency configuration list needs to be created.

Third, software FHA is to determine the effects of each failure condition should be considered from three aspects, and respectively are effect on passenger, aircraft and crew. Last, it is to determine the classification of the failure condition according to its effect. The classification criterion should be accorded to the classification of software level in DO-178C. In DO-178C, it gives five levels to a

failure condition which ranges from level A to level D, and level A is the most serious. The output should be documented in the Table 4-1 .

**Table 4-1 Software FHA [8] [10]**

Functional	Failure Condition	Flight Phase	Effect of Failure Condition to Aircraft/Crew/Passenger	Classification

The output of software FHA is the list of hazards and its classification. This list is the input of the next step, and also is crucial supporting materials for software development assurance level assignment process.

#### **4.2.2 Identify safety requirements**

After software FHA, it can start to identify all the possible causes related to the hazard by using the traditional fault analysis methods. In this step, it uses the bi-directions approaches, software fault tree analysis and software failure mode and effects analysis to go backward and forward for eliciting the safety requirement.

- **Software Fault Tree Analysis**

Software Fault Tree Analysis is to find the software error and failure which lead to the occurrence of the failure condition. The process of software fault tree analysis has been shown in Chapter two. Software fault tree analysis will be finished until the event cannot be developed anymore or the event doesn't belong to the scope of software, such as the damage of electronic components.

After construction, each software fault tree needs to conduct the minimum cut set analysis, and the minimum cut sets are used as the input of software development assurance level assignment process.

- **Software Failure Mode and Effects Analysis**

The critical step of Software Failure Mode and Effects Analysis is to list all the possible failure modes of software functions or components. In this research, the potential failure modes of software FMEA are the list of errors derived in the

software FTA and several possible software failure modes recommended in NASA software safety guidebook, such as incorrect logic, incorrect input and output [8].

The output of software failure mode and effects analysis should be documented in the table, and should list the following information: 1) function or component; 2) function description; 3) failure mode; 4) effects; and 5) comment. The software FMEA format is shown in Table 4-2.

**Table 4-2 Output of Software Failure Mode and Effects Analysis**

Component/Function:			
Failure Mode	Effect on Software	Effect on System	Comment

- Safety Property

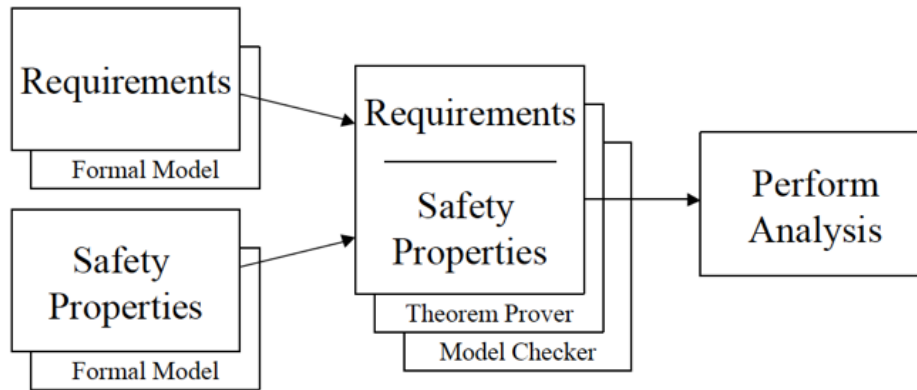
After finished the previous steps of software safety requirement assessment process, it can get several basic events which compose a list of software errors or mistakes contribute to the hazard. This list can be regarded as the software safety requirements. However, some safety requirements may be generally. So, it is necessary to specify the safety requirement according to its function description and architecture.

The last step is to verify the specified safety requirement elicited from the previous steps, and the proposed safety requirement verification method is model checking.

### 4.2.3 Formal Verifications

In order to conduct formal verification on software requirement, it must be translated into a verifiable form firstly. So, it needs to establish the formal model for software requirement at the beginning. And then, the safety requirements need to be translated into the formal description by using the temporal logic. After created a formal model and a formal requirement specification, it can apply both model and specification into the model checker. Finally, verification tool conducts

the assessment automatically. Figure 4-3 shows the verification process of software safety requirements.



**Figure 4-3 Verification Process of Safety Requirement [16]**

The first step of verification process is to establish the formal model of the software, and it uses the Kripke structure.

- Modelling the software

Before using formal modelling techniques, it needs to finish preparatory work. To get the related software information is necessary for establishing the model. Preparatory work includes gathering related documents and information such as the system architecture, system function description, system requirement, target software function description, and software architecture and requirements. By using these documents and information, understand the relationship between the system and software, and this relationship includes the functional allocation relationship, logic mode, and so on. Software modelling is the process to transform the relationship in the abstract.

At the beginning of modelling, it needs to divide software into several functions. The entire formal model is made up of the several sub-models. Then, analyzes software function and its architecture to determine how many states do this function have, what kind of transition relationship between each state, and to identify the propositions of each state. The state transition diagram can help to establish.



In practice, formal modelling is the hardest part of the entire project due to the complexity of software. So, the simplest way of modelling is to separate the software according to its architecture and functions.

- Defining the Safety Properties

As mentioned in Chapter two, model checking uses temporal logic to represent the property. No matter what kind of temporal logic, the translation is straightforward.

Firstly, all the safety requirement needs to be checked whether the definition is clear and unambiguous or not. Then, uses temporal logic to substitute the informal specification, and it is attention that the time sequence and logical relationship of the informal specification are two critical factors of formalisation. Here shows one example from a NASA project.

“HDG switch lamps shall be lit when HDG mode is active” [16] [45] is one of safety requirements for Lateral Modes function in Flight Guidance system. This requirement has two variables, lamp and mode. Lamp has two states, on or off. So, does mode. This requirement requires when the mode is active, the lamp must turn on. So, this property will be translated as:

$AG (HDG\_Lamp=true \leftrightarrow HDG\_Mode\_active)$

HDG\_Lamp is the variable to determine the state of HDG Lamp, and HDG\_Mode\_active is to determine the state of HDG mode. Both types of variables are Boolean, true or false. AG means verify this property for all the state in all the path.

The next section is to verify the safety requirement by using a model checker. The type of model check has been discussed in the previous Chapter, and the working principle of model checker will be introduced in the case study.

- Microwave Oven Problem

To make this process more concrete, this section presents one small examples. Microwave Oven is the most representative example in model checking. The

microwave functions are easy to understand, so this is a best example to present the entire model checking process [46].

### - System Modelling

First, it needs to understand and define the system function and system architecture. The microwave oven has following functions:

- Microwave oven can cook the food;
- Microwave oven can open or close the door.
- Microwave oven can be reset if there is a wrong input (door open).

Microwave oven also has several safety requirements:

- If the microwave oven is cooking, the door shall not open.
- If the Start button was pressed when the door was opened, the microwave enters the error mode.
- If the Reset button was pressed, the error mode shall not be active.

According to related information, it can start to create the model of the microwave oven. There are five variables in this model, and they are “start”, “reset”, “closed”, “error”, and “cook”. Figure 4-4 is the state transition graph for a microwave system. The Kripke structure of microwave system can be described as:

The state set S is  $S = (S1, S2, S3, S4, S5)$ .

S1 is the initial state. The transition relation is

$$R = \{(S1, S2), (S1, S3), (S3, S1), (S4, S1), (S2, S5), (S3, S4), (S4, S3), (S5, S3), (S5, S2), (S4, S4)\}$$

The labels of each state are:

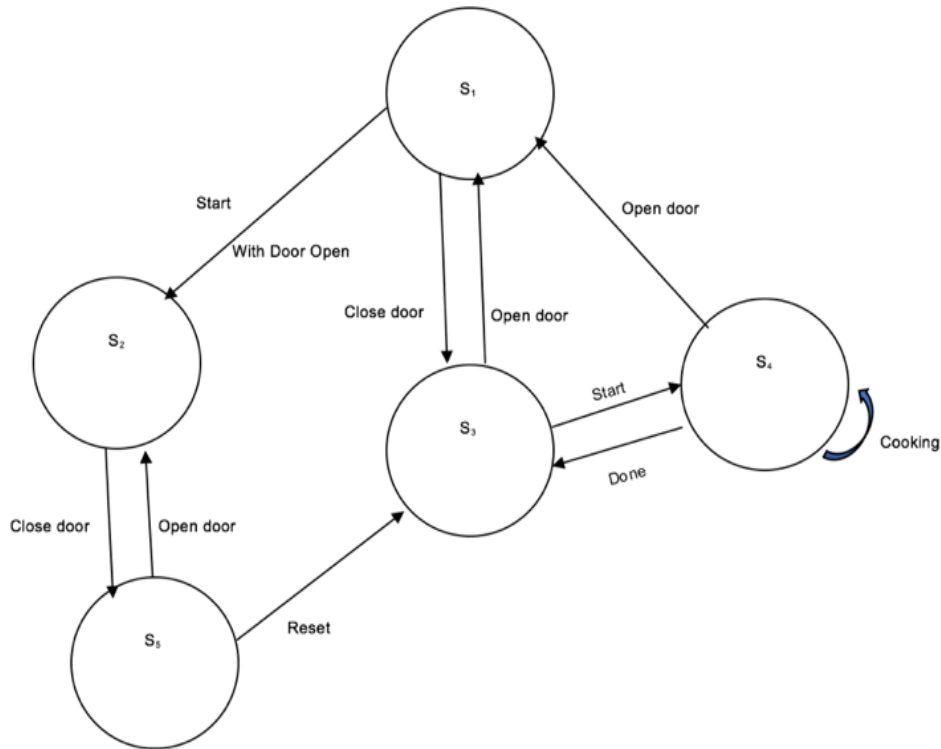
$$L(S1) = \{\neg\text{close}, \neg\text{start}, \neg\text{cook}, \neg\text{error}, \neg\text{reset}\}$$

$$L(S2) = \{\neg\text{close}, \text{start}, \neg\text{cook}, \text{error}, \neg\text{reset}\}$$

$$L(S3) = \{\text{close}, \neg\text{start}, \neg\text{cook}, \neg\text{error}, \text{reset}\}$$

$L(S_4) = \{\text{close, start, cook, } \neg\text{error, } \neg\text{reset}\}$

$L(S_5) = \{\text{close, start, } \neg\text{cook, error, reset}\}$



**Figure 4-4 State Transition Graph**

**– Properties Specification**

The second step is to translate the requirement into temporal logic. Here choose two properties. One is “No Cook while door is open”. This requirement means the close state should be ALWAYS happen before the cook state at any time. This property can be formalized as  $\text{SPEC AG} ( \neg\text{closed} \rightarrow \text{AX} ( \neg\text{cook} ) )$ . The second one is “The state of cook will be eventually happened at some time” and this can be translated as  $\text{SPEC EF}(\text{cook})$ . The temporal operator EF can be explained as there exists one state which can satisfy this specification in this model.

**– Translate model and specification into NuSMV**

Last, translates both Kripke model and CTL specification into NuSMV by using SMV input language. The NuSMV code shows as below:

```

MODULE main
VAR
start:boolean;
reset: boolean;
closed : boolean;
error: boolean;
cook : boolean;
ASSIGN
init(error):= FALSE;
init(cook):= FALSE;
next(error):=
case
(start & !closed): TRUE;
(closed & reset): FALSE;
TRUE      : error;
esac;
next(cook):=
case
(start & closed) : TRUE;
(!closed) : FALSE;
TRUE: cook;
esac;
SPEC AG(!closed-> AX(!cook))
SPEC EF (cook)

```

The verification result generated by NuSMV shows in Figure 4-5 and Figure 4-6:

```

ubuntu@ubuntu-SVE14A27CXH: ~
ubuntu@ubuntu-SVE14A27CXH:~$ /home/ubuntu/Desktop/nusmv/NuSMV-2.6.0-Linux/bin/Nu
SMV /home/ubuntu/Desktop/11.smv
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:36:56 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

-- specification AG (!closed -> AX !cook) is true
ubuntu@ubuntu-SVE14A27CXH:~$

```

Figure 4-5 Verification Result 1 of Microwave Problem

```
ubuntu@ubuntu-SVE14A27CXH: ~
ubuntu@ubuntu-SVE14A27CXH:~$ /home/ubuntu/Desktop/nusmv/NuSMV-2.6.0-Linux/bin/Nu
SMV /home/ubuntu/Desktop/11.smv
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:36:56 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

-- specification EF cook is true
ubuntu@ubuntu-SVE14A27CXH:~$
```

Figure 4-6 Verification Result 2 of Microwave Problem

#### – Result Analysis

In this microwave example, the author chose two safety requirements for comprehensively presenting the software requirement formal verification process. The verification result proves the consistency between the formal model and formal specification, and also proves the correctness of safety requirements.

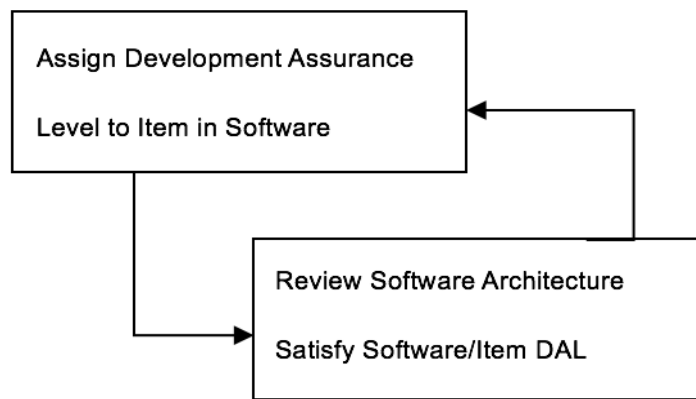
Through this example, NuSMV can be proved as a convenient and easy-understanding verification tool, because the transition between an informal model and SMV language is straightforward. In a word, NuSMV is an effective method for safety requirement verification, and can be applied to the practical case.

### 4.3 Methods for Software Architecture Safety Assessment Process

The purpose of architecture safety assessment is to determine whether software architecture satisfies safety requirement or not, especially the Development Assurance Level (DAL). Architecture safety assessment can give an early assessment of the credibility of a proposed software architecture, which can help

to reduce the risk that safety problems are uncovered in the later stage of software lifecycle where they are expensive to correct or modify.

The software architecture safety assessment process is the iterative process. It should begin at the initial stage of software development, and repeat during the entire software lifecycle. Figure 4-7 shows the process of software architecture safety assessment.



**Figure 4-7 Process of Software Architecture Safety Assessment**

If there are any changes or new hazards introduced in the software architecture, it needs to review the initial DAL of software, and to check the new software architecture whether satisfies the software DAL or not. If the architecture requires the higher level of safety, it needs to re-assign the DAL of software until the current design and architecture can satisfy the software DAL. The next section will discuss and compare two different kinds of software DAL assignment process.

#### **4.3.1 Software Development Assurance Level Assignment Process**

In ARP4754A [10], it has already explained the detailed DAL assignment process. The research will apply this DAL assignment process to the software architecture safety assessment process as one of the assessment methods for software architecture. Before explaining the software DAL assignment process, the Chapter will introduce the Development Assurance Level firstly.

- Introduction of Development Assurance Level

Modern aircraft have integrated a lot of complex systems and functions, and the safety margin may be reduced by the unsafe system, software or item. The authority concerns on the error introduced during the development process that will cause the severe failure condition. So, the authority wants to use the series of activities to identify and control all the hazard during the product lifecycle [10]. Development Assurance Level (DAL) is the process used to identify the error through a series of planned and systematic actions during the system or software lifecycle. DAL not only means the software level, but also includes the set of guidelines and activities to guarantee the safety during each stage of the lifecycle. The objectives of assurance activities are to identify, control and eliminate the errors, such as requirement, design and code error.

In ARP4754A [10], it divides DAL into two categories, the functional development assurance level (FDAL), and item development assurance level (IDAL). FDAL is to perform the assurance activities on functions which guarantee the safety of functional development, and the FDAL is usually used for aircraft-level and system level. However, the IDAL is to perform the assurance activities on item level, such as software. In software standard, such as DO-178C, item development assurance level is a software level.

When assigned the FDAL or IDAL, it needs to consider two kinds of independence attributes, which are functional independence and item development independence. Functional Independence is to ensure that the likelihood of common mode error has been minimized by implementing and performing the different functions. Item Development Independence is to ensure that the likelihood of common mode error between two items has been minimized by several actions, such as using different operating systems. So that, the independence attributes can eliminate the common mode error, and establish the acceptable confidence level of function or item.

The next section will discuss the software DAL assignment process. One is assignment without the software architecture consideration, and the other is assignment with the software architecture consideration.

- Software DAL Assignment without Software Architecture Consideration

In general, the FDAL and IDAL assignment process are a top-down decomposed process, and it starts with the list of the failure conditions and fault trees constructed in fault tree analysis.

When assigned software DAL without consideration of software architecture, the software DAL assignment process is to assign all the software FDAL and IDAL by using Table 4-3.

**Table 4-3 General Principle for DAL Assignment [5]**

<b>Top Level Failure Condition Severity Classification</b>	<b>Associated Top Level Function FDAL Assignment</b>
Catastrophic	A
Hazardous/Severe Major	B
Major	C
Minor	D
No Safety Effect	E

Firstly, it assigns the top level FDAL of software according to the classification of related failure condition which identified in high level FHA. And then, IDAL for all items in this function should be designated as the same level of the top-level function FDAL. The above table shows the corresponding relationship between the severity classification of failure conditions and DAL.

- **Software DAL Assignment with Software Architecture Consideration**

The assignment process with software architecture consideration requires the enough evidence to prove that the functional independence and item development independence of Functional Failure Sets (FFS) member has been satisfied. FFS is uses to express the combination of errors or faults which can lead to the hazards. Conceptually, FFS is equivalent to the result of the minimal cut set analysis in FTA.

Basically, if members within a given FFS can be proved that its functional independence to be satisfied, their FDAL can be assigned a lower level than the classification of related top-level failure condition according to Figure 4-8. In IDAL assignment, if the members of FFS has item development independence, IDAL



can use the same row which is used for FDAL assignment. The IDAL also can be gracefully degraded.

TOP-LEVEL FAILURE CONDITION CLASSIFICATION	DEVELOPMENT ASSURANCE LEVEL (NOTES 2 & 4)		
	FUNCTIONAL FAILURE SETS WITH A SINGLE MEMBER	FUNCTIONAL FAILURE SETS WITH MULTIPLE MEMBERS	
		OPTION 1 (NOTE 3)	OPTION 2
Column 1	Column 2	Column 3	Column 4
Catastrophic	FDAL A (NOTE 1)	FDAL A for one Member, additional Member(s) contributing to the top-level Failure Condition at the level associated with the most severe individual effects of an error in their development process for all applicable top-level Failure Conditions (but no lower than level C for the additional Members).	FDAL B for two of the Members leading to top-level Failure Condition. The other Member(s) at the level associated with the most severe individual effects of an error in their development process for all applicable top-level Failure Conditions (but no lower than level C for the additional Member(s)).
Hazardous/ Severe Major	FDAL B	FDAL B for one Member, additional Member(s) contributing to the top-level Failure Condition at the level associated with the most severe individual effects of an error in their development process for all applicable top-level Failure Conditions (but no lower than level D for the additional Members).	FDAL C for two of the Members leading to top-level Failure Condition. The other Members at the level associated with the most severe individual effects of an error in their development process for all applicable top-level Failure Conditions (but no lower than level D for the additional Members).
Major	FDAL C	FDAL C for one Member, additional Member(s) contributing to the top-level Failure Condition at the level associated with the most severe individual effects of an error in their development process for all applicable top-level Failure Conditions.	FDAL D for two of the Members leading to top-level Failure Condition. The other Members at the level associated with the most severe individual effects of an error in their development process for all applicable top-level Failure Conditions.
Minor	FDAL D	FDAL D for one Member, additional Member(s) contributing to the top-level Failure Condition at the level associated with the most severe individual effects of an error in their development process for all applicable top-level Failure Conditions.	
No Safety Effect	FDAL E	FDAL E	

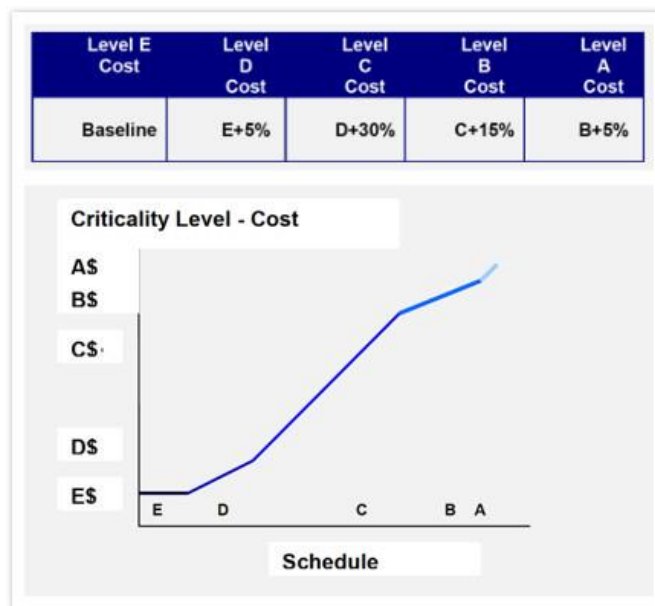
**Figure 4-8 Assign Development Assurance Level to Functional Failure Set Members [10]**

### 4.3.2 Comparison of Existing Software DAL Assignment Process

The purpose of DAL is to reduce the probability occurrence of failure during the entire software lifecycle. The difference between two kinds of software DAL assignment process is that if both functional independence and item development independence can be satisfied, the DAL of some FFS members would be degraded.

The assignment process without software consideration is more conservative. According to Table 4-3, all the software IDAL should be assigned to the same level as the classification of its related top-level failure condition. For example, if a failure condition is catastrophic, and the corresponding software IDAL should

be assigned as A level. Although this process can guarantee the safety, the higher development assurance level implies a higher level of rigour and more costly development and assurance activities. Figure 4-9 shows that the cost of development, especially in the verification tasks, will be significantly increased when the critical level is higher [47].



**Figure 4-9 Cost of Different Development Assurance Level Software**

In order to guarantee safety and reduce the cost, ARP 4754A provides a way for software DAL assignment process, which can help to degrade some IDAL in some circumstance.

### **4.3.3 Software Development Assurance Level Assignment Process Case Study**

This section gives four cases of FDAL and IDAL assignment process. The division of cases is related to independence of function and item development.

- No evidence to prove both the Independence Functional and Item Development can be satisfied

If there is no evidence to prove both the functional and item development independence have been satisfied, assessment process of all FDAL and IDAL will use Table 4-3. The FDAL and IDAL are assigned to the same level as the

top-level FDAL. For example, if top-level FDAL is level B, the rest of all FDAL and IDAL will be assigned to level B.

- Both Functional Independence and Item Development Independence can be proved

If both functional independence and item development independence are presented, it can allow both FDAL and IDAL to use option 1 or option 2 in Figure 4-8. Figure 4-10 shows one example of this situation. First, it needs to calculate the FFS of this fault tree. In FFS calculation, " \* " expresses the AND gate and " + " expresses the OR gate. So, calculation is shown as below:

$$FC = F1 * F2 \quad (1)$$

$$F1 = F1 + I1 \quad (2)$$

$$F2 = F2 + I3 \quad (3)$$

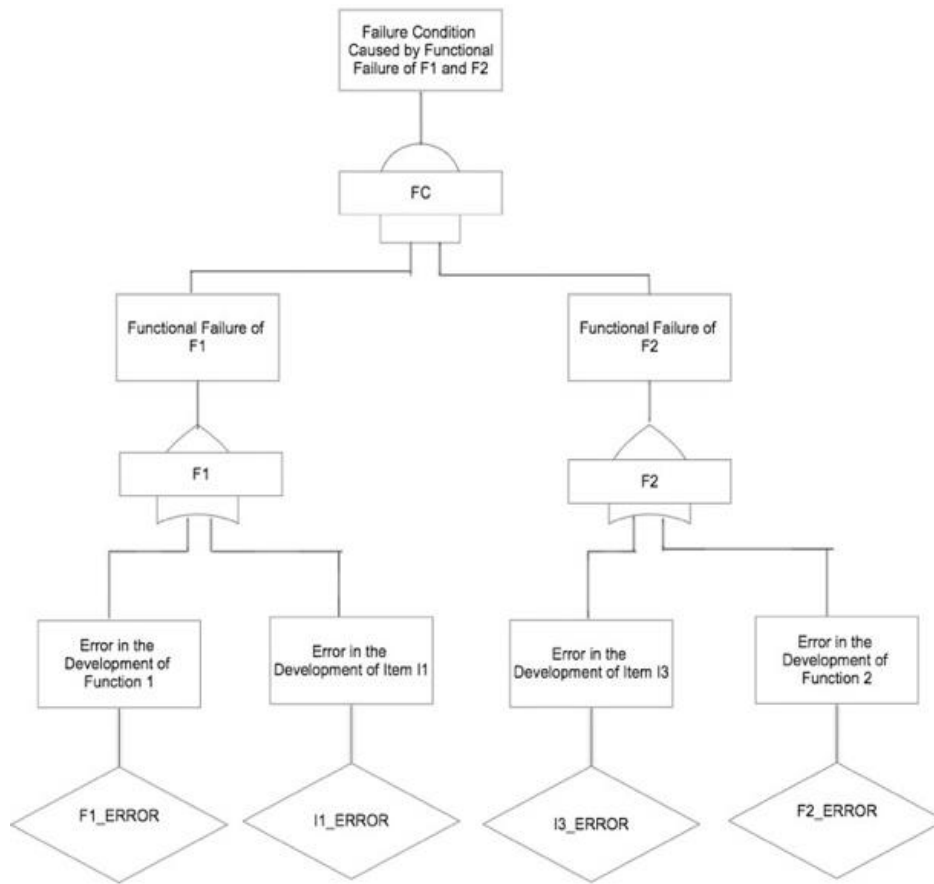
Put (3) and (2) into (1),

$$FC = (F1 + I1) * (F2 + I3) = F1 * F2 + F2 * I1 + F1 * I3 + I1 * I3$$

So, the FFSs for the failure condition are {F1, F2},{F1,I3},{I1,F2},{I1,I3}. First, assign FDAL to F1 and F2. If this failure condition is a catastrophic failure condition and both F1 and F2 satisfy the functional independence, the assignment of F1 and F2 can use option1 or option2 in Figure 4-8. The IDAL can also use the option1 or option2 in Figure 4-8. So, the correct assignment result of FDAL and IDAL shows in Table 4-4:

**Table 4-4 Accepted Assignment of FDAL and IDAL of Example 1**

FDAL Assignment		IDAL Assignment	
F1	F2	I1	I3
A	C	A	C
C	A	C	A
B	B	B	B
A	B	A	B
B	A	B	A



**Figure 4-10 Fault Tree Example-1 [10]**

In fact, if the F1 is A level and F2 is level C can be accepted, the F1 is level A and F2 is level B won't be considered anymore. Because the development cost of level B is higher than level C.

Here give several unaccepted results of assignment process in Table 4-5.

**Table 4-5 Unaccepted Assignment of FDAL and IDAL of Example 1**

FDAL Assignment		IDAL Assignment	
F1	F2	I1	I3
A	C	C	A
C	A	A	C

F1 and I3, or I1 and F2 are two members of FFS in this fault tree. This means that if both F1 and I3 happen, it will lead to the top-level hazards. The assignment

of FFS members must be followed with the general principle. For example, the top-level failure condition is catastrophic, and more than one failure could lead to this failure. So, one failure can be assigned level A, and the rest failures can be assigned at least C level. Or, all the failures can be assigned at the level B. So, both the DAL of F2 and I2 assigned at level C cannot be accepted.

- Functional Independence is proved but Item Development Independence is not satisfied

If independent Functions can be proved, but Items doesn't satisfy the independence, and one of the item error can lead to a common mode error. The IDAL of the non-independent items should be assigned to the same level of the related failure condition. Figure 4-11 shows one example under this situation.

First, it needs to calculate FFS of this fault tree.

$$FC = (F1 + I1 + I2) * (F2 + I3 + I2)$$

$$FC = F1 * F2 + F1 * I3 + F1 * I2 + I1 * F2 + I1 * I3 + I1 * I2 + I2 * F2 + I2 * I3 + I2 * I2$$

Because  $I2 * I2 = I2$ , so it equals to:

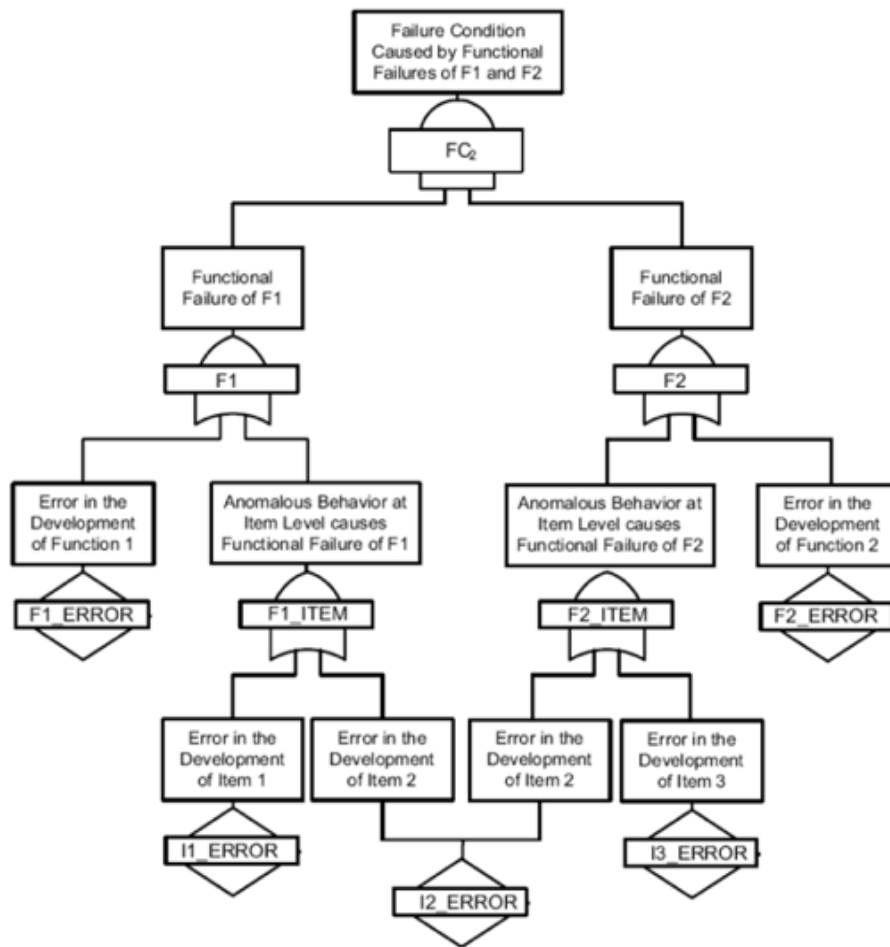
$$FC = F1 * F2 + F1 * I3 + F1 * I2 + I1 * F2 + I1 * I3 + I1 * I2 + I2 * F2 + I2 * I3 + I2$$

In this fault tree, Item I2 will cause both F1 and F2 failure, so I2 can individually lead to the catastrophic top-level failure condition. So, Item I2 is a single member of FFS. So, the formula can be simplified as:

$$FC = F1 * F2 + F1 * I3 + I1 * F2 + I1 * I3 + I2$$

The FFSs for the failure condition is: {F1,F2},{F1,I3},{F2,I1},{I3,I1},{I2}.

The I2 is a common mode error of this fault tree, so the IDAL of I2 should be assigned same as the related top-level failure condition classification.



**Figure 4-11 Fault Tree Example 2 [10]**

So, IDAL of I2 should be assigned as level A. The correct result of assignment result of FDAL and IDAL shows in Table 4-6 :

**Table 4-6 Accepted Assignment of FDAL and IDAL of Example 2**

FDAL Assignment		IDAL Assignment		
F1	F2	I1	I2	I3
A	C	A	A	C
C	A	C	A	A
B	B	B	A	B

The assignment of rest FDAL and IDAL are same with Table 4-5. So, the explanation of Table 4-6 will not show anymore.

Here shows some unaccepted assignment cases and explanation.

**Table 4-7 Unaccepted Assignment of FDAL and IDAL of Example 2**

FDAL Assignment		IDAL Assignment			Comment
F1	F2	I1	I2	I3	
B	B	B	B	B	The I2 cannot be level B due to it is the common mode error.
A	C	C	A	A	Same reason with line2 in Table 4-5.
C	A	A	A	C	Same reason with line2 in Table 4-5.

- Only Item development independence can be proved

The FDAL is assigned to the same level as the top-failure condition by using Table 4-3. All the IDAL can be degraded by using the option 1 or option 2 in Figure 4-7. However, the failure of each independent item won't lead to the top-level failure condition. Otherwise, it should be assigned at the same level of top-level failure condition.

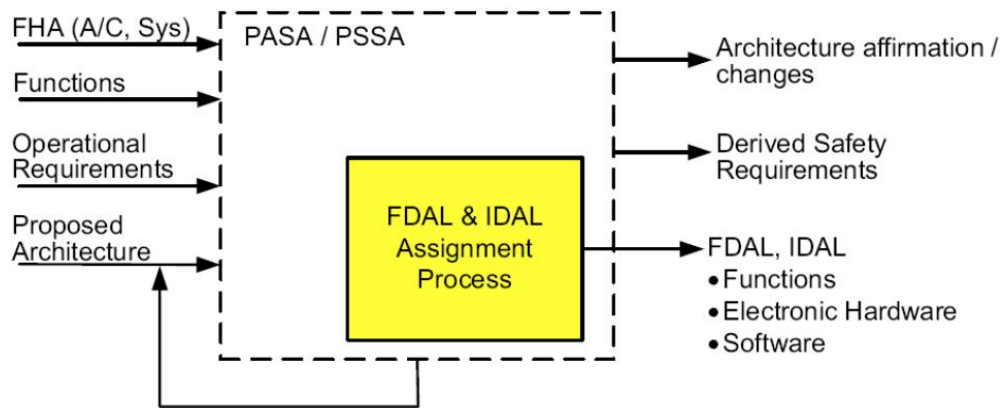
#### **4.4 Consideration of Software Development Assurance Level Assignment Process**

The DAL assignment process and architecture assessment is the complex and time-consuming process, and this process need lots of supporting material to support the assignment process and the verification process. Here discusses the preparation work before the assignment process and the questions need to be cared during the assignment process.

- Preparation Work

The goal of software development assurance level assignment is that assign the correct and proper software DAL to each software components according to software architecture and requirements. Before the assignment, the research needs to obtain some supporting information for software DAL assignment

process. Figure 4-12 shows the input and output of FDAL and IDAL assignment processes.



**Figure 4-12 FDAL/IDAL Assignment Process**

Firstly, it conducts FHA to identify the all failure condition related to software, and the identified failure conditions and its severity classification are a precondition for the entire DAL assignment. Second, it needs to conduct SFTA. SFTA need to undertake two kinds of tasks. One is assignment DAL by using the fault tree structure, and the other is to determine the FFS of each failure condition. Third, proposed system and software architecture are the important supported documents for FDL and IDAL assignment process. Last, the assignment process may require the additional documents, such as initial software function list, system and software function requirements, safety requirements and operational requirements. These materials are the fundamental of software architecture safety assessment.

- Independence of Function and Item development

As mentioned, independence attribute is one of the most important part of architecture safety assessment process. If software function and items can be claimed independence, the software DAL can be degraded.

The item development independence can be achieved as designed and implemented by different teams and different processes, and may install in a



different operating system. If there is no evidence to show any common errors in components, item development independence could be substantiated.

The functional independence can be achieved as by using different requirement to implement one function and partition. By using different requirement to implement one function, such as the aircraft navigation function can be implemented by GPS navigation system or IRS navigation system.

Partition is for functional independence that avoids the occurrence of common mode error during the development process. If there is the partition for each function which implemented in the common design, the DAL of the partitioning function would be assigned at the same level of classification of the top-level hazard during its development. If the partition not be used or if its independence cannot be proved, the FDAL of function should be re-assigned the same level of the IDAL of common design, or the function should be re-allocated to the lower level in order to spate the common design and independent part.

#### **4.5 Summary**

This Chapter discusses the content and proposed methods of software safety requirement assessment process and software architecture safety assessment process. By using several examples, the Chapter detailed describes each step of the software safety requirement assessment process, and different software DAL assignment situations in the software architecture safety assessment process.



## **5 Case Study**

### **5.1 Introduction**

The research uses the practical case study to exam the proposed software safety requirement assessment process and software architecture safety assessment process. The case study chooses the flight management system which is one of the highly software intensive system in avionics system. Due to the complex and complicated of the flight management system, it is hard to analyse all the system functions at once. So, the author chose one typical function in FMS which is the position calculation function for this case study.

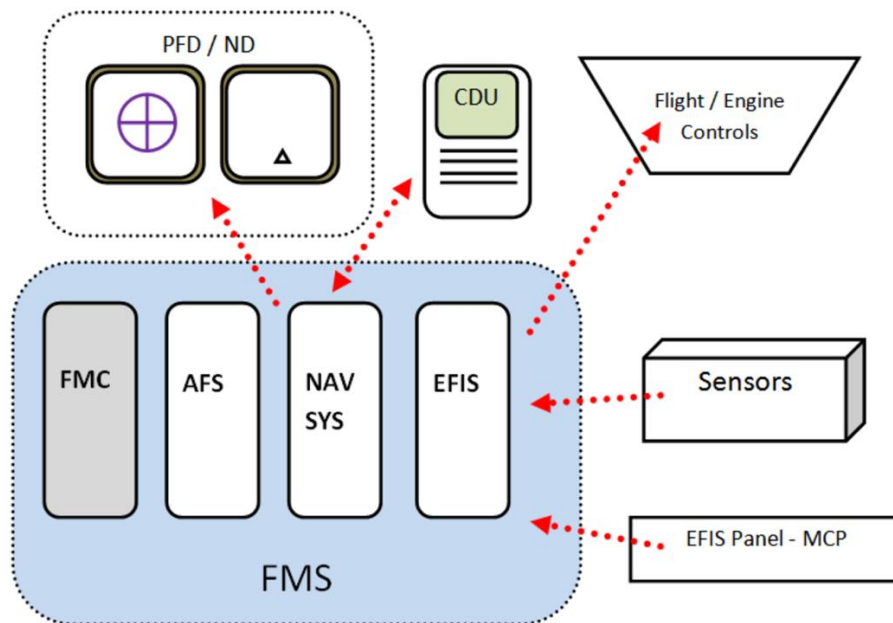
The Chapter illustrates the procedure of applying the safety requirements assessment process to the position calculation function in flight management system, and it is organized as follows. Section 5.1 provides an overview of the flight management system. Section 5.2 presents the logic of position calculation function in FMS. Section 5.3 describes formal verification tool, NuSMV. Section 5.4 presents the safety requirement elicitation process by using traditional safety assessment methods. Section 5.5 provides the formal modelling of position calculation function. Section 5.6 shows the analysis of safety requirements assessment result.

### **5.2 Overview of Flight Management System**

The initial requirement of Flight Management System is for navigation which can help the pilot to arrive the desired destination. At 1970s, an area navigation (RNAV) computer began to be installed on the aircraft. At the same time, the fuel crisis drove the development of aircraft performance management, which helped to optimize the commercial aircraft navigation and to improve the efficiency of aircraft operation. In late 1970s, Boeing Company organized a flight desk technology group for the development of initial FMS, and wanted to combine flight management computer and control display units together. This system has become as the core part of aircraft flight planning and navigation function [48].

The development of FMS starts with the Flight Management Computer (FMC) which is the key component. Besides FMC, the current FMS has three main components [49]

- The Automatic Flight Control or Automatic Flight Guidance System
- The Aircraft Navigation System
- An Electronic Flight Instrument System



**Figure 5-1 Overview of FMS Components [50]**

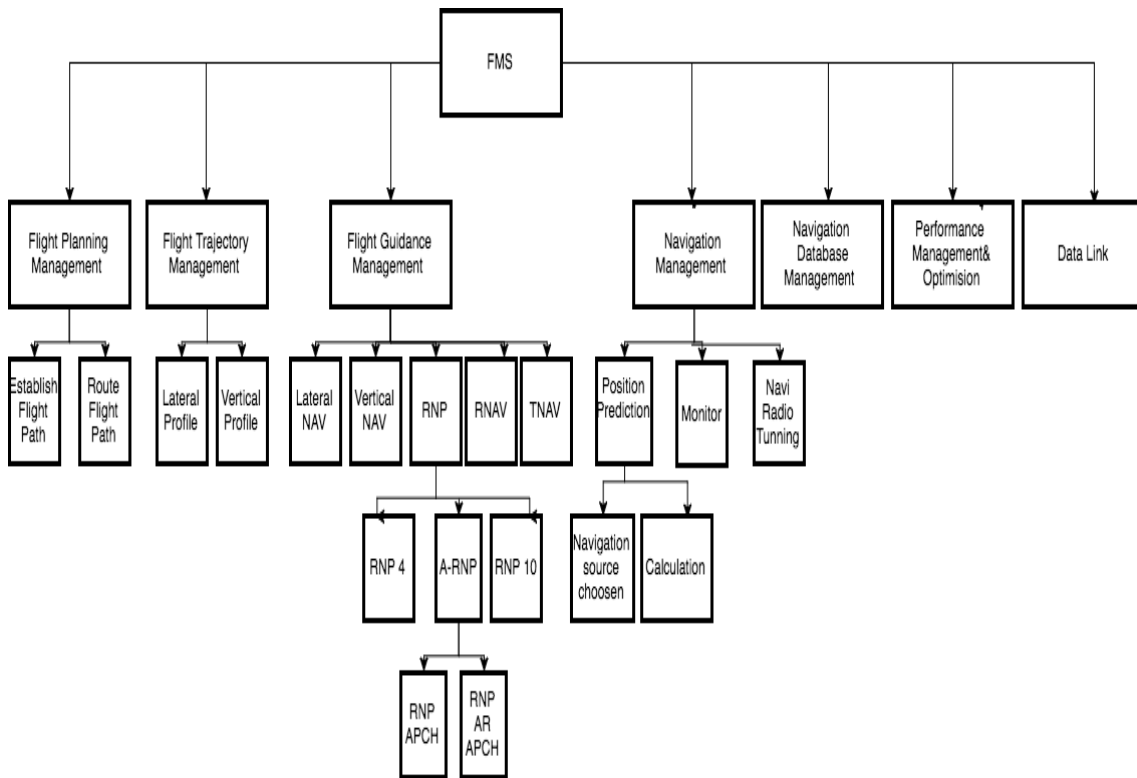
Flight Management Computer is the computer system that uses database to proceed with various tasks, such as pre-schedule and modify the flight path. One of main tasks of FMC is to continuously update with the aircraft current position by using multiple and available navigation sensor information.

The Automatic Flight Control System receives various sensor information from different systems. The Automatic Flight Control System can allow pilots to choose different operation modes, such as Autopilot and manual control. Depending on operation mode, Automatic Flight Control System will automatically control the flight control surface or display the control command on PDFs for the pilot to follow it.

The task of Navigation System is to calculate the aircraft position continuously. It uses multiple navigation sensor information, such as Global Positioning System, Radio navigation system, Inertial Reference System and other information.

FMS can help the pilot to carry out various tasks, which lead FMS to become one of the important avionics systems in modern aircraft. The current FMS has been developed into an integrated system, which combined multiple functions such as trajectory prediction, navigation, guidance and performance optimization.

Typically, FMS provides several functions, flight planning, flight trajectory management, flight guidance management, navigation management, and performance management optimisation. Figure 5-2 shows the organization of main FMS functions and briefly introduces each function.



**Figure 5-2 Overview of FMS Function**

Flight Planning extracts the navigation data from navigation database and establishes the complete flight path by using navigation information, which includes airport information, waypoint, route, approach procedure and departure procedure. Flight Planning function allow pilot to select the current or alternative

flight plan, to create the new or user-defined route, and to edit the existing route information and restriction, such as altitude, speed, and the time of arrival.

Flight Trajectory Management predicts and establishes the lateral and vertical flight profile data in each waypoint for the current, alternative or temporary flight plan according to aircraft performance information.

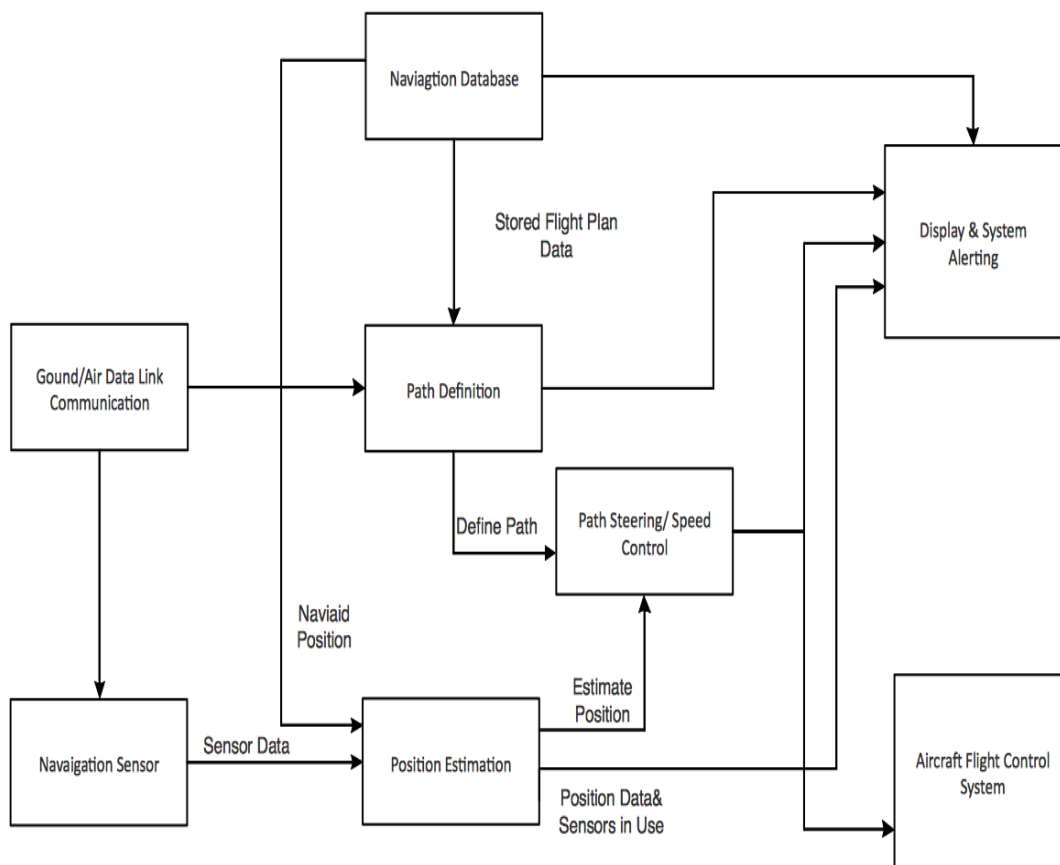
Flight Guidance Management is to calculate the vertical and lateral steering commands according to pre-scheduled flight path and current aircraft status information. The Flight Guidance Management aims to lead the aircraft to fly in accordance with the desired path. Flight Guidance Management also provides required navigation performance (RNP), area navigation (RNAV) and time navigation (TNAV).

Navigation Management combines various navigation information which come from each navigation sensor, and calculates the aircraft position, speed and attitude information. Aircraft position calculation is one of the ways to determine aircraft position during the flight. It will use the sensor data provided by navigation sensor, and the ground-based Navaid position information extracted from the navigation database. After getting the estimated position, the position data and the selected sensor data will be displayed on PFDs for the pilots. Meanwhile, position data will also be used for generating path steering which used for aircraft Flight Control system. Navigation system also has navigation radio tuning, which manages and tunes all the navigation radio equipment. This function provides automatic navigation tuning equipment which can be used for aircraft position calculation, and automatic ILS approach. Meanwhile, navigation system also provides the navigation radio tuning interface for manual.

Performance Management Optimise is to calculate the optimal height, speed or other performance data of the flight path. It can calculate the other information, such as time of arrival, distance or other data during the flying.

There are lots of interactive relationships between each function in FMS. For example, the navigation database provides navigation sensor and navaid position

data for position calculation function, and provides stored flight plan data for path definition function. Figure 5-3 shows the FMS block diagram.



**Figure 5-3 FMS Block Diagram [51]**

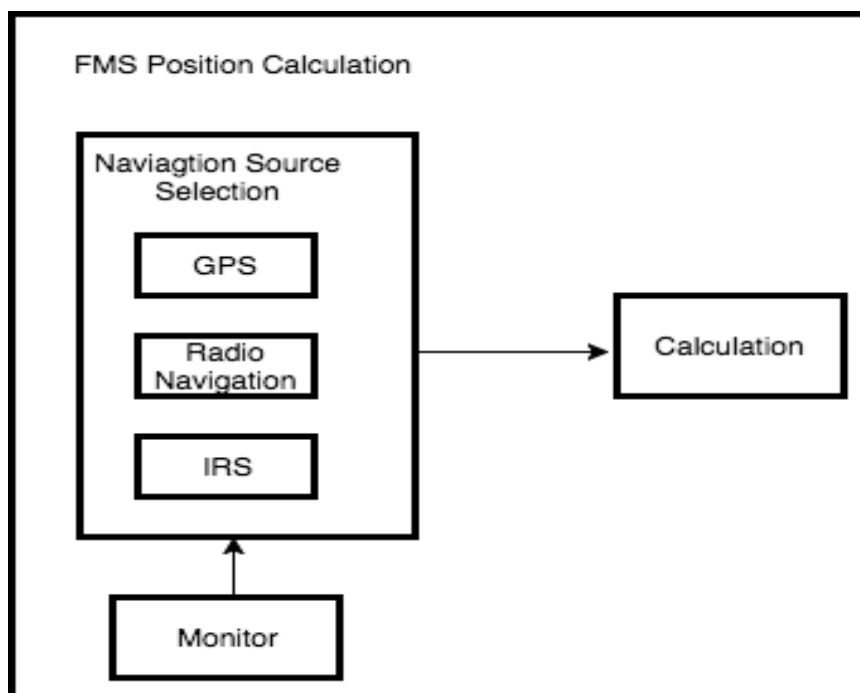
For this case study, the author chose the position calculate function in navigation management, and assumed that the aircraft is only installed the single FMS, which means it only has one set of FMSA. The next section gives the detailed description of position calculation function logic.

### **5.3 Overview of Position Calculation Function**

To satisfy the navigation requirements, FMS uses various sensor data from GPS, Radio Navigation AIDs, Inertial Reference System (IRS), Air Data Computer, and Attitude/Heading Reference. These sensor data are used to determine the aircraft position, direction, and speed information.

Position calculation function is one of the primary functions in FMS navigation system. There are several navigation sensor receivers installed in aircraft, such as the GPS receiver, the Radio Navigation Source Receiver, which includes VOR receiver and DME receiver. The position calculation selects the proper and available sensor that provides the best navigation plan for calculating and estimating the aircraft position.

The working principle of position calculation function is to choose the available navigation sensor and to calculate the aircraft position and other data. Position calculation function has three parts, which are navigation source selection logic, calculation algorithm and monitor function respectively. In this case study, the author focuses on navigation source selection logic and monitor function. The navigation source selection logic is to choose the proper and available navigation sensors. In this case study, the author assumed there are three navigation sensors installed on FMS, which are GPS, Radio Navigation sensor, and IRS. The basic logic of position calculation shows in Figure 5-4.



**Figure 5-4 Position Calculation Logical [51]**

In FMS position calculation function, the basic selection sequence is:

1. GPS.



2. Radio Navigation Source.
3. IRS.

The selection logic is that FMS will use GPS data firstly. However, if the GPS sensor is detected any failure or loses of GPS signal, which means GPS is not available anymore. FMS will select the secondary navigation source of selection sequence, and set the failed GPS sensor into rest mode automatically. This means when GPS sensor is not available, the FMS will choose Radio Navigation sensor automatically. However, if the radio navigation sensor is detected any failure or loses signal, FMS will select the third order of selection sequence for navigation and set failed radio navigation sensor into rest mode automatically. When both GPS and radio navigation sensor are not available, the FMS will use IRS for navigation. However, if all of navigation sensor failed, the FMS position calculation will be turned off automatically.

FMS position calculation also has the monitor function. It continuously monitors all the navigation sensors to make sure that each sensor provides the valid and correct information to the FMS. If it finds any failure of navigation sensor or other navigation equipment, it will set the failed sensor into rest mode. The monitor function mainly monitors two aspects. One is navigation sensor status, and other is the data that the sensor sent to FMS. If sensor status is abnormal or sensor data is incorrect, the monitor will take action to this sensor.

This case study will focus on conducting software safety assessment on position calculation function. The next section will briefly introduce the verification tool, NuSMV.

## **5.4 Verification Tool NuSMV**

There are lots of formal verification tools, and some are already discussed in the previous Chapter. In this case study, the author chose to use NuSMV. NuSMV is a symbolic model checker, which is re-engineered and re-implemented of SMV by Carnegie Mellon University. NuSMV is an open source and flexible methods for model checking. This section introduces the installation of NuSMV, the software architecture of NuSMV, and the input language of NuSMV.

### 5.4.1 NuSMV Installation

NuSMV is the open source software and operates under Linux system. Other operating system such as windows or MacOS needs to simulate the Linux system firstly before install NuSMV. There are two ways to install NuSMV, and here gives the briefly description.

#### ● **First Step: The Preparation**

As Chapter mentioned before, the precondition of NuSMV installation is to install the Linux operating system. Nowadays, it has various brands of the Linux system, such as Ubuntu or Red Hat. NuSMV can be used in almost Linux operating system, so there is no requirement for operating system choosing. However, it needs to make sure that all the following aid software already installed before NuSMV installation.

- ANSI C compiler;
- GNU tar and gzip tool;
- GNU Bison v.1.25 or latest version;
- GNU Flex v.2.6.0 or latest version;
- GNU make.

This case study chose Ubuntu Linux system. Ubuntu is one of the most famous open source operating system, and it is the Linux distribution which based on the Debian architecture [52]. The installation of Ubuntu won't discuss in thesis.

#### ● **Second Step: The Installation**

After finished all the preparation work, it can download the latest version of NuSMV from the official website. It can start to install the NuSMV, and here gives two ways of installation. The results of two kinds of installation are totally same. The first installation way is to use NuSMV source code.

- The NuSMV Source Code Installation

Source Code NuSMV is already compiled for most common operating system and architecture. However, for some unfamiliar operating systems and some people who want to re-engineer, NuSMV provides the source code

version to install. The NuSMV user manual [43] provides the detailed installation steps, and this section only gives the briefly introduction of source code installation. Table 5-1 shows the installation steps [43].

**Table 5-1 Installation Steps of NuSMV Source Code [43]**

Step	Function	Command
1	Move to the directory where build NuSMV.	# cd /home/nusmv
2	Unpack the distributions.	# gzip -dc /tmp/NuSMV-2.6.0.tar.gz   tar xf -
3	Create a directory for building.	# mkdir build
4	Enter this new built directory.	# cd build
5	Configure by invoking cmake.	# cmake
6	Enter the directory before compile NuSMV.	# pwd <TOPDIR>/NuSMV/build
7	Compile NuSMV.	#make
8	Set the path of master. nusmvr into environment variable.	# export NUSMV_LIBRARY_PATH =<TOPDIR>/NuSMV/share/nusmv

– The NuSMV Binary Code Installation

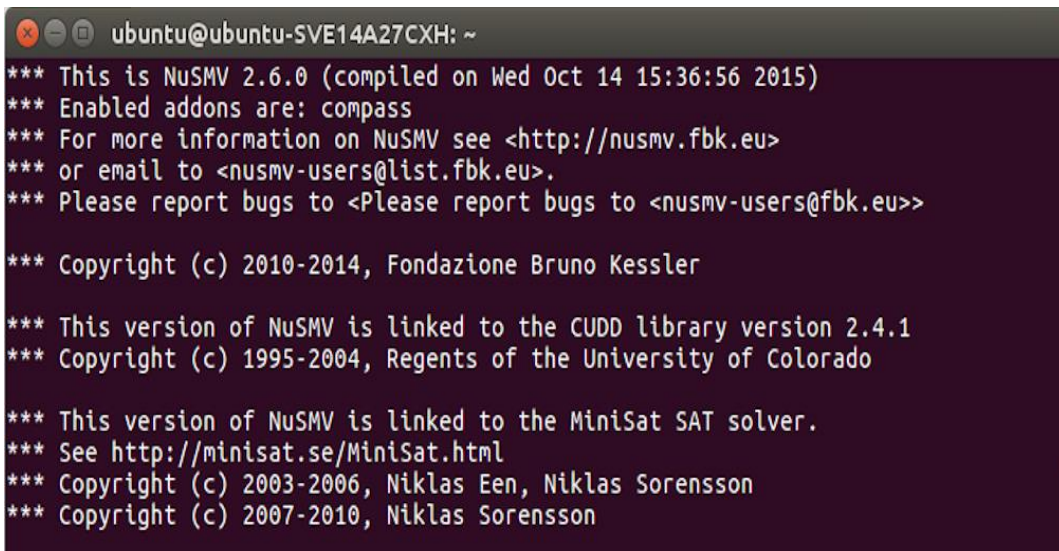
NuSMV website provides the Binary Code, which is the pre-compiled version, and this version can be used for most operating systems and architecture. This Version doesn't need to conduct the above steps, and the installation steps of pre-compiled version are quite simple. The author recommends this installation way, which is more easy and convenient. Table 5-2 shows the installation step of pre-compiled version.

**Table 5-2 Installation Step of NuSMV Binary Code**

Step	Function	Command
1	Unpack the zip-file	# gzip -dc /tmp/NuSMV-2.6.0.tar.gz   tar xf -
2	Run NuSMV	bin/NuSMV(Tab) /<file path >/xxx.smv

If it shows the result is same with Figure 5-5 the after installation and run command in terminal, it means NuSMV installation successful.

**Figure 5-5 NuSMV Installation Result**



```
ubuntu@ubuntu-SVE14A27CXH: ~
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:36:56 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

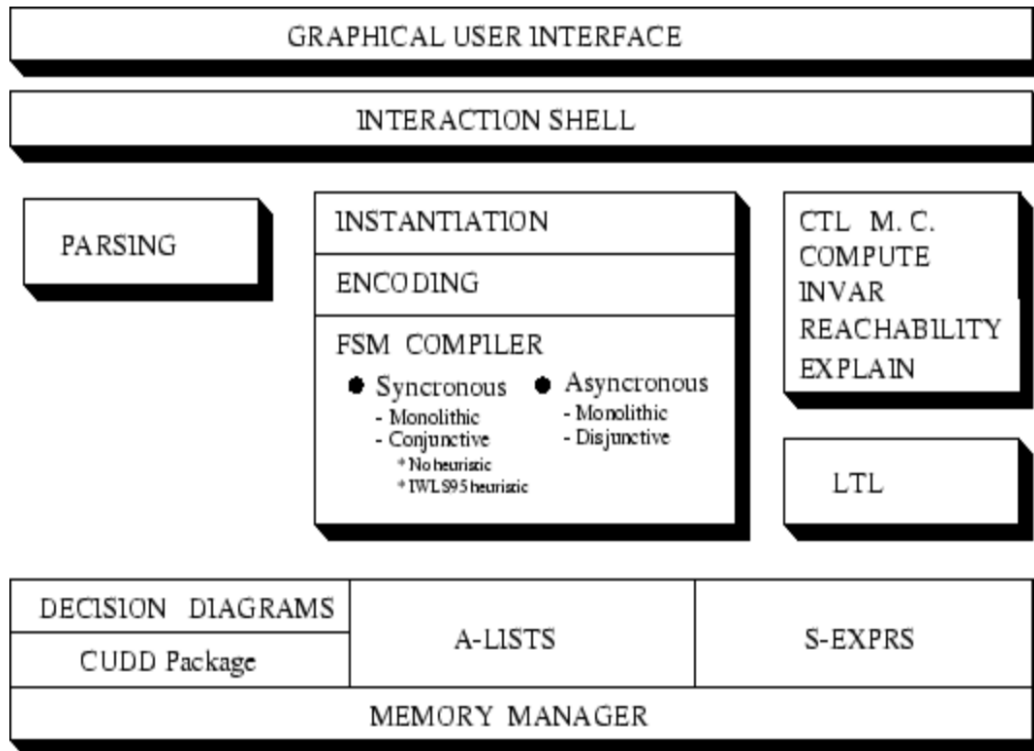
*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson
```

After installation of the NuSMV, it can start to model the position calculation function. NuSMV cannot use model directly, so it need to translate model into the language which NuSMV can recognize and understand. The next part will briefly discuss the input language, SMV language.

### 5.4.2 NuSMV Architecture

NuSMV is completely written in ANSI C and is designed as one open source system. The NuSMV architecture consists of several modules. Different modules implement different functions, and communicates with others by defined interfaces. The architecture of NUSMV shows in Figure 5-6.



**Figure 5-6 Architecture of NUSMV [53]**

- Kernel Module is to provide the lower level function such as dynamic memory allocation, and also provides all the basic Binary Decision Diagram primitives [53].
- Parser Module processes the input file to check the correctness of statement syntactic, and builds the parse tree for representing the internal format of this input file [53].
- Compiler compiles the parsed model into Binary Decision Diagram. Instantiation sub-module is to process the parse tree, and to build a description of the finite state machine to represent the input model. Finite State Machine Compiler sub-module constructs and manipulates the Finite State Machine at the Binary Decision Diagram level, and then conducts all the semantic checks on the input model [53].
- Model Checking module provides several functions, such as CTL model checking, counterexample generation and inspection, invariant checking, and so on [53].

- Interactive shell. By using the interaction shell, the user can access to all of NuSMV functions [53].

### 5.4.3 NuSMV Input Language

NuSMV uses SMV language to describe the Kripke structure and specification. The previous Chapter already presents one NuSMV code example. NuSMV is made up of modules, and each module consist of variables. The common input language shows in Table 5-3 [54].

**Table 5-3 SMV Language Example**

Name	Code	Example
Module define statement	MODULE –Name  VAR --Define variable name and type  ASSIGN – The relationship between states	MODULE main  VAR a: boolean; b: boolean; c:{on, off};  ASSIGN
Variable assignment statement	ASSIGN  init(xx) – Initialize the variable	ASSIGN  init(d):=off
Transition Constraint statement	TRANS xxxx	TRANS (a & !b) -> next(!(a & !b))
Specification statement	SPEC xxxxx	SPEC AG (a -> !b)

**The Kripke structure is used in model checking to represent the behaviour of a finite state system. In NuSMV, it can use SMV language to describe the Kripke structure by create one module or several modules.**

Figure 5-7 show Kripke structure in NuSMV.

**Figure 5-7 Kripke Structure in NuSMV [55]**

```
-- Kripke structure
MODULE
    :
MODULE    main
    :
```

In NuSMV, module consists of three parts, and they are variable, constraint, and CTL or LTL specification. Variables are used to describe each state of the Kripke structure. Constraint are used to describe the transition relationship between each state. Specification is described by CTL or LTL temporal logical and used to describe the requirement.

After all the preparation works finished, it can start to conduct software safety assessment methods and process for position calculation function.

### **5.5 Position Calculation Safety Requirement Elicitation Process**

As mentioned in the previous Chapter, the software safety assessment process has four steps. Due to the larger content of software safety assessment, this research mainly focuses on safety requirement assessment and architecture safety assessment process. So, this case study presents how to apply the safety requirement assessment process and initial DAL allocation process to a practical case.

In safety requirement assessment process, it has four sub-processes. First is to identify the associated hazards. Second, it identifies the software errors or faults caused the hazards. These two steps are to elicit safety requirements. Third, it creates safety requirements specification. Last step is to use NuSMV to verify the safety requirements. The next section discusses the safety requirements elicitation process of FMS position calculation function. It mainly has two tasks, which are to identify hazards and identify causes related to hazards.

### 5.5.1 Identify Hazards

The first step is to identify which hazard related to target software. In this step, it uses the Functional Hazard Assessment.

The case study is about the position calculation function. First, it needs to identify the functional requirement of FMS position calculation, which is the fundament of FHA. The Chapter already discussed the logic of FMS position calculation. In a word, the position calculation is to choose the available and proper navigation source, which can let FMSA to calculation the aircraft position information, and thus indicates the information in PFDs.

Through analysis of the functional requirement, the author found one failure condition related position calculation and shows in Table 5-4 .

**Table 5-4 Failure Condition for Position Calculation**

Failure Condition	Phase	Effects	Classification
The both sides of PFD indicate the wrong navigation and position information	Climb; Curies; Descent; Approach; Landing	1. For aircraft: It significantly reduces the safety margin. 2. For Crew: It will significantly increase in workload to crew. 3. For passenger: No effect.	Hazardous

Due to the calculation failure or other reasons, PDFs will show the wrong position and navigation information to the pilot. This failure condition is applicable during the climb, curies, descent, approach and loading phase. The effects of failure include three parts, which are effect on aircraft, on crew, and on the passengers. For aircraft, it will significantly reduce the safety margin and may lead the aircraft deviate the pre-scheduled route. For crew, crew needs to deselect the FMS as the navigation source and need to use other ways to calculate the position. This will add work to crew. However, there is no effect on the passenger.

The classification of this failure condition is assigned as hazardous. Because of lack of the system architecture, it only can use the conservative DAL assignment process, which means the DAL of this failure condition is same as the DAL of



entire FMS. According to standard [56], the DAL of FMS usually is assigned as the B level. So, the classification is assigned as hazardous.

Appendix A.1 shows some failure conditions related to FMS navigation functions. However, the FMS navigation system has lots of sub-system or sub-functions, which means it is difficult to list all of hazard related to the navigation system. So, case study only focuses on failure condition related the position calculation function. The next section will determine the basic events contribute to this failure condition. This step will use fault tree analysis and failure mode and effects analysis.

### **5.5.2 Identify Safety Requirements**

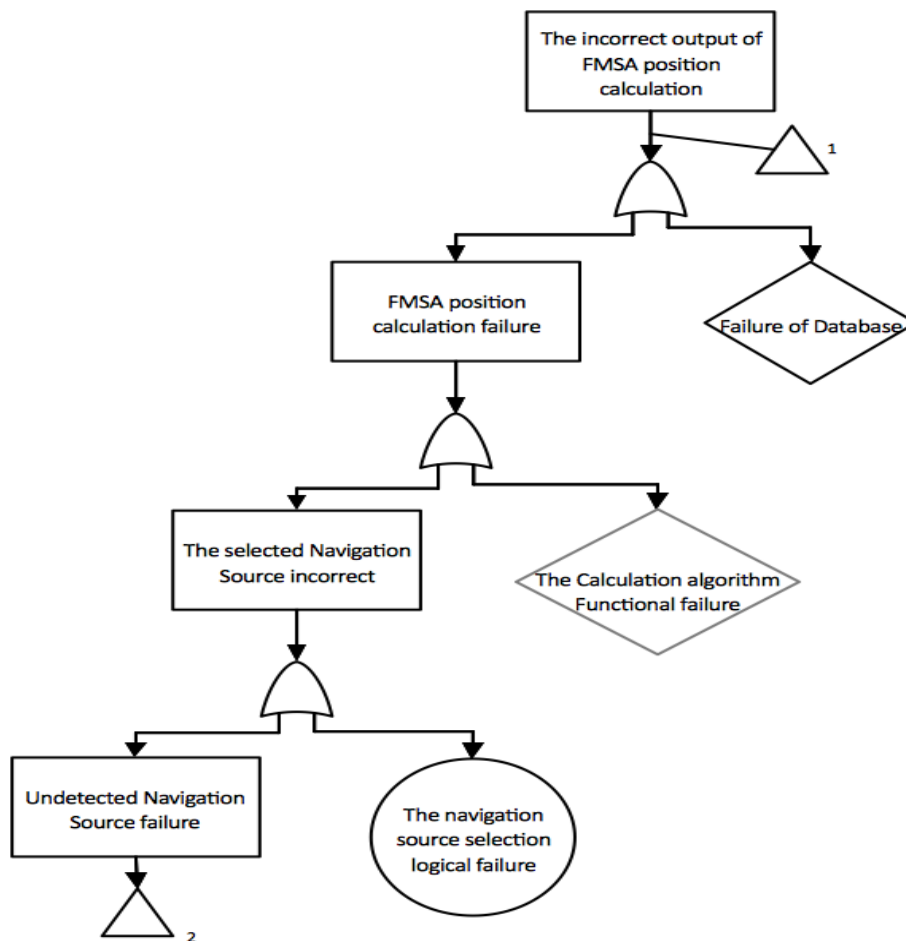
Once the failure condition has been identified, it can start to trace backward to find software faults or error. This step will use fault tree analysis and failure mode and effects analysis. The result of FTA and FMEA will be considered as the safety requirements of the case study.

- **Fault Tree Analysis**

The first step is to use the failure condition “The both sides of PFD indicate the wrong navigation and position information” as the top event of fault tree, and then list all the possible causes according to its functional analysis. The completely decomposing process will present in appendix A.2, and here only discusses the sub-tree which directly related to position calculation function.

The top event of the subtree is “The incorrect of output of FMSA position calculation” and shows in Figure 5-8. The fault tree first splits into “FMSA position calculation failure” and “Failure of Database”, because the incorrect output is caused by position function failed or used the incorrect navigation information to calculate the position. The “failure of database” is out of scope of the case study, and will not be decomposed anymore. So, it is an undeveloped event. In the next level, the “FMSA position calculation failure” can be divided into two events. One is “The selected Navigation Source incorrect” or “The Calculation algorithm functional failure”. However, “The Calculation algorithm functional failure” is not included in this case study, so it is the undeveloped event. The event of “The

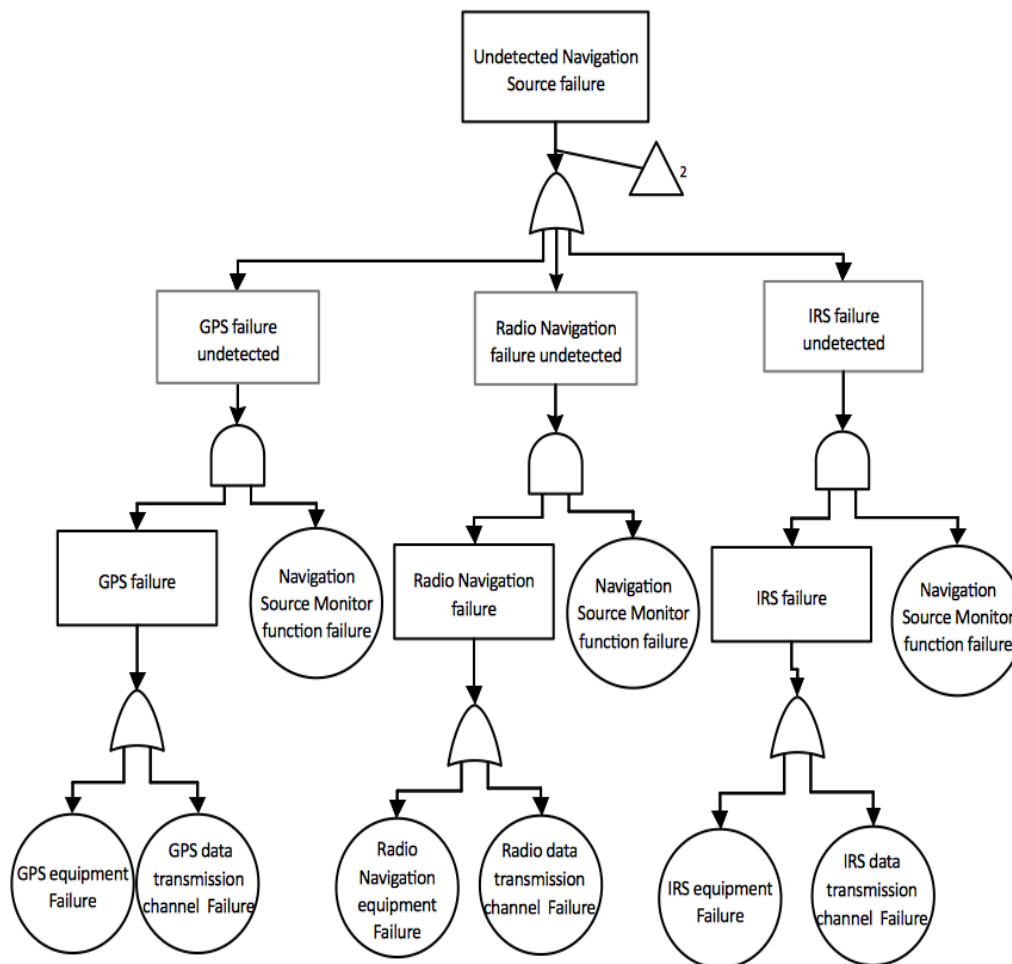
selected Navigation Source incorrect” is caused by selection algorithm error, or it cannot detect the abnormal sensor and still use the abnormal sensor for navigation. “Navigation source selection logical failure” means when GPS is available, but system chooses the Radio Navigation sensor. The “Undetected sensor failure” is related to the failure of monitoring function. It will discuss in the next fault tree.



**Figure 5-8 Fault Tree of “The both sides of PFD indicate the wrong navigation and position information”**

The lower levels of the FTA are shown in Figure 5-9. The position calculation function has three types of navigation sensor, GPS, Radio Navigation, and IRS. If one of navigation sensors failed and undetected, but the system also selects this failed sensor as navigation source, it will cause the “undetected navigation source failure”. At the next level, the “GPS failure undetected” caused by both

GPS failure and monitor function failure happened in the meantime. The monitor function is a software functional block, and it continuously monitors the status and data of each navigation sensor. If the monitor finds any sensor status is abnormal or the sensor sent wrong data, it will rest the sensor and choose the next available navigation sensor automatically.



**Figure 5-9 Subtree of “The both sides of PFD indicate the wrong navigation and position information”**

After finished construction, it needs to conduct the minimum cut set analysis of fault tree. The result of minimum cut set analysis will be regarded as the FFS, which can be considered as the safety requirement of position calculation function. The calculation process of fault tree shows in Appendix A.2, and here only gives the result. The FFS are {The navigation source selection logical failure}, {Navigation Source Monitor function failure, GPS equipment Failure}, {Navigation

Source Monitor function failure, GPS data transmission channel Failure}, {Navigation Source Monitor function failure, Radio Navigation equipment Failure}, {Navigation Source Monitor function failure, Radio Navigation data transmission channel Failure}, {Navigation Source Monitor function failure, IRS equipment Failure}, {Navigation Source Monitor function failure, IRS data transmission channel Failure}.

Through the fault tree analysis and minimum cut set analysis, it can get the following safety requirements which related to “undetected navigation sensor failure”:

1. All Navigation sensor only can be active when sensor status is normal and sensor data correct can be satisfied at the same time. Otherwise, the failed navigation sensor will be turned to rest automatically. For example, if GPS sensor is abnormal or GPS data are wrong, the GPS mode will change to rest automatically.
2. If all navigation sensors are turned to rest mode, the FMS position calculation function will be turned off automatically.

There are safety requirements which related to “navigation source selection logical failure”

1. If GPS is chosen, the rest of navigation source cannot be active.
2. If GPS is failure, the system will choose radio navigation sensor.
3. If GPS and radio navigation sensor are both failed, the system will choose IRS.

As mentioned in Chapter four, these fault trees are the fundamental of DAL assignment process in architecture safety assessment process. However, it is lack of information related to software architecture, so it only can assign the initial DAL of all the FFS member as the same level of the top event. Some FFS members can be degraded if there is enough evidence to provide that both functional independence and item development independence are satisfied. However, the “Navigation Source Monitor function failure” cannot be degraded, because this event is the common mode error of this fault tree. So, it should be

assigned as the same level as the DAL of top level failure condition, which is B level.

- **Failure Mode and Effects Analysis**

The first step in the FMEA is to develop a list of the possible failure modes of position calculation function. In this case study, the author chose to use the basic events of the fault tree as the list of failure modes. Table 5-5 summarizes all basic events from the previous fault tree analysis.

**Table 5-5 Basic Event list of Fault Tree**

Number	Basic Event	Software or Non-Software
1	Data transmission channel of FMSA to Left PFD is incorrect	Non-Software
2	Data transmission channel of FMSA to Right PFD is incorrect	Non-Software
3	The navigation source selection logical failure	Software
4	Navigation Source Monitor function failure	Software
5	GPS equipment Failure	Non-Software
6	GPS data transmission channel Failure	Non-Software
7	Radio navigation equipment Failure	Non-Software
8	Radio navigation data transmission channel Failure	Non-Software
9	IRS equipment Failure	Non-Software
10	IRS data transmission channel Failure	Non-Software

Table 5-6 shows the example for software FMEA “Navigation source selection logical failure”. “Navigation source selection logical failure” will lead to incorrect selection of navigation sensor. The selected navigation sensor and the position calculation algorithm doesn’t match. For example, the position calculation applies the radio navigation sensor data to the GPS position calculation algorithm. The result of this failure mode is incorrect output of FMS position information.

**Table 5-6 FMEA for Navigation Source Selection Logical Failure**

Function: FMS position calculation	
Failure Mode	Effect on System
Navigation source selection logical failure	FMS cannot calculate the aircraft position, and thus FMS cannot output the correct position information.

Table 5-7 shows the example for software FMEA “Navigation Source Monitor function failure” and the rest of FMEA table presents in Appendix A.3. Navigation source selection logical failure will lead the FMS position calculation function to use the incorrect sensor data, such as the GPS sensor already failed, but it keeps sending the wrong data to FMS. However, the monitor doesn’t detect this failure, and then the system still uses the wrong data to estimate the aircraft position. This will lead the position calculation function cannot send the correct aircraft position information to pilots.

**Table 5-7 FMEA for Navigation Source Monitor function failure**

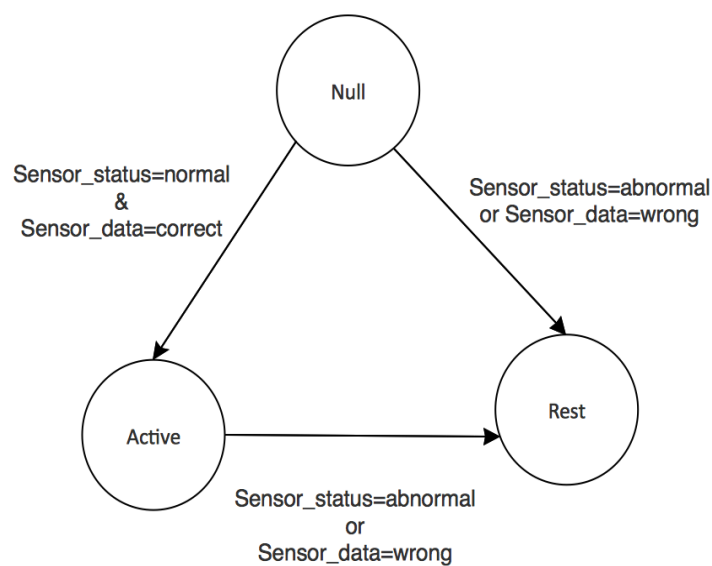
Function: FMS position calculation	
Failure Mode	Effect on System
Navigation Source Monitor Function Failure	Position Calculation function may not use the correct navigation source to calculate. FMS cannot calculate the aircraft position, and thus FMS cannot output the correct position information.

After FTA and FMEA, it can get the safety requirements of FMS position calculation. After summarized and specified, the list of safety requirement shows in appendix A.4.

## 5.6 Formal Modelling

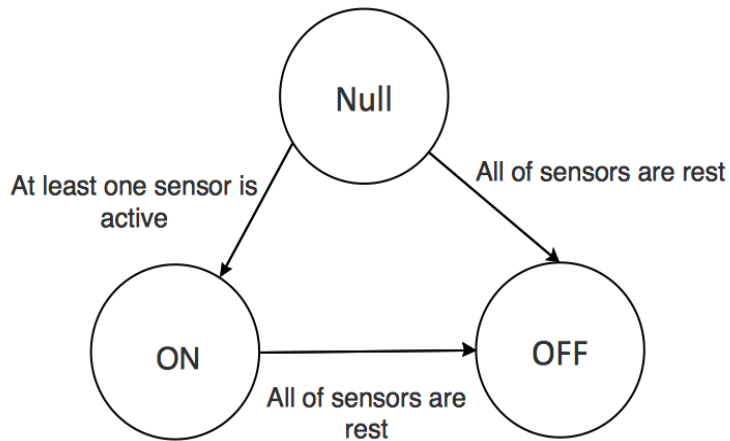
As mentioned before, the case study focuses on monitor function and navigation source selection logic in the position calculation function. If it wants to model the position calculation function, it needs to divide function into two parts, which are sensor model and monitor model.

In sensor model, each of navigation sensor has three states, which are null, active, and rest. The monitor function will monitor the sensor status and sensor data of each sensor. Sensor status has three states, which are null, abnormal and normal. Sensor data have three states, which are null, correct and wrong. The initial state of each sensor is null, and the sensor status and sensor data are both null. Only if both sensor statuses are normal and sensor data is correct are satisfied simultaneously, the sensor can be active. Otherwise, the sensor will be turned to rest. If the sensor state is changed to rest, it cannot change to active or null state any more. So, the state transition graph shows in Figure 5-10.



**Figure 5-10 Sensor State Transition Graph**

The monitor function has been involved in the sensor model, and this monitor model is mainly cares about turning on or off the FMS position calculation function. In monitor model, FMS position calculation function has three states, which are null, on and off. The initial state is null. Only If all the navigation source sensors are into rest state, the FMS position calculation function will be turned to off automatically. If there is at least one navigation sensor in active state, the FMS position calculation function will be turned on. The state transition graph is same as sensor, but the translation relationship is different. Figure 5-11 shows the monitor model.



**Figure 5-11 Monitor Function State Transition Graph**

After modelling, it can start to translate model into NuSMV by using SMV language. The NuSMV code of FMS position calculation model shows in appendix A.6. The next section will show the safety requirement verification result.

## 5.7 Result Analysis

Before verification, it needs to translate safety requirement into specification. And then the author uses NuSMV to verify safety requirement.

After safety requirement elicitation process, it gets several safety requirements. However, it is not specific enough. So, it firstly needs to specify the safety requirements according to function. For example, one of the safety requirement is that navigation mode (such as GPS mode) only can be active when both sensor status is normal and sensor data correct are satisfied at the same time. Otherwise, this navigation sensor will be turned to rest. This requirement can be divided into three categories, GPS mode, radio navigation mode, IRS mode. GPS mode has two input variables, GPS sensor status and GPS sensor data. If both GPS sensor status is normal and GPS sensor data is correct can be both met, the GPS mode will be active. If GPS sensor status is abnormal or GPS sensor data is wrong, the GPS mode will be rest. So, this requirement can be translated into two specifications shows as below, and AG means the specification can be satisfied for all states.



- If both GPS sensor status is normal and GPS sensor data is correct can be both met in the meantime, the GPS mode will be active.
  - Specification1: SPEC AG((gpssensor\_status=normal) & (gpssensor\_data=wrong)-> (gps\_mode=active))
- If GPS sensor status is abnormal or GPS sensor data is wrong, the GPS mode will be rest.
  - Specification2: SPEC AG((gpssensor\_status=abnormal) | (gpssensor\_data=wrong)-> (gps\_mode=rest))

Put these two specifications into model, the result shows in Figure 5-12 and Figure 5-13.

```

ubuntu@ubuntu-SVE14A27CXH: ~
ubuntu@ubuntu-SVE14A27CXH:~$ /home/ubuntu/Desktop/nusmv/NuSMV-2.6.0-Linux/bin/Nu
SMV /home/ubuntu/Desktop/2
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:36:56 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

-- specification AG ((gpssensor_status = normal & gpssensor_data = correct) -> g
ps_mode = active) is true
ubuntu@ubuntu-SVE14A27CXH:~$ █

```

Figure 5-12 NuSMV Result of Specification 1

```

ubuntu@ubuntu-SVE14A27CXH: ~
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

ubuntu@ubuntu-SVE14A27CXH:~$ /home/ubuntu/Desktop/nusmv/NuSMV-2.6.0-Linux/bin/Nu
SMV /home/ubuntu/Desktop/2
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:36:56 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

-- specification AG ((gpssensor_status = abnormal | gpssensor_data = wrong) -> g
ps_mode = rest) is true
ubuntu@ubuntu-SVE14A27CXH:~$ █

```

Figure 5-13 NuSMV Result of Specification 2

Here gives one counterexample of GPS navigation mode. The requirement is that if both the GPS sensor status and GPS sensor data were not satisfied simultaneously, the GPS mode would not be active. Figure 5-14 shows the counterexample.

- If one of the GPS sensor status is normal or GPS sensor data is correct can be satisfied, the GPS mode would be active.
- Specification3:  $AG((gpssensor\_status=normal) \mid (gpssensor\_data=correct) \rightarrow (gps\_mode=active))$

```
-- specification AG ((gpssensor_status = normal | gpssensor_data = correct) -> gps_mode = active) is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  gpssensor_status = null
  gpssensor_data = null
  gps_mode = null
  radiosensor_status = null
  radiosensor_data = null
  radio_mode = null
  irssensor_status = null
  irssensor_data = null
  irs_mode = null
  fms_positioncalculate = null
-> State: 1.2 <-
  gpssensor_status = abnormal
  gpssensor_data = correct
  gps_mode = rest
  radiosensor_status = abnormal
  radio_mode = rest
```

**Figure 5-14 NuSMV Counterexample Result**

The counterexample shows that one state which doesn't satisfy this specification. It can be explained as: when GPS sensor data correct but the GPS sensor status is abnormal, and the GPS mode still be rest. According to the result of fault tree analysis, if the position calculation function wants to choose GPS as the navigation source, the GPS sensor data correct and the GPS sensor status is normal should be satisfied at the same time. So, this specification cannot be satisfied by the model.

Here only gives the GPS mode safety requirements example, and the rest of the safety requirement and verification results respectively shows in appendix A.4 and A.5.

The emphasis of this case study is to show how to apply software safety assessment process and method to one practical software case. Through

understanding the functional logic, it can establish the formal model and elicit the safety requirements. By using NuSMV verification tool, the result can be used to prove the correctness and consistency of safety requirements.

## **5.8 Summary**

This Chapter uses the FMS position calculation as a practical case study to present the flow of the recommended software safety requirement assessment process. By analysing verification result, it proved the correctness of software safety requirements, and the consistency between the formal model and safety requirements. Meanwhile, it proved the feasibility of suggested software safety assessment methods.

This software safety requirement assessment process and proposed software safety assessment method can help engineer to identify the errors or mistakes of the requirement at the early stage of development, which will reduce the workload and saving money.



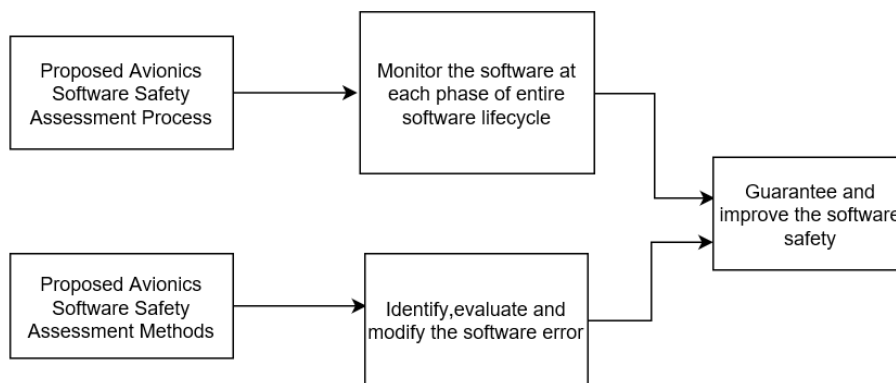
## 6 Conclusion and Further Work

### 6.1 Conclusion

This research aims to develop one comprehensive and systematic avionics software safety assessment process derived from software lifecycle introduced in DO-178C and existing safety assessment process. Meanwhile, the research suggested the recommended safety assessment methods for each step in this software safety assessment process according to its objectives.

The proposed avionics software safety assessment process has the congruent relationship with the DO-178C avionics software lifecycle, which will assist the engineer clearly understand the scope and component of avionics software safety assessment. This process also sets the order of software safety assessment activities and the engineer can follow these activities to monitor the software at each phase of the entire software lifecycle.

The recommended safety assessment methods are the way which engineer used in software safety assessment activities, and aim to identify, evaluate and modify the software error, which guarantees and improves the software can perform as its expected. Figure 6-1 shows how each objective improves the software safety.



**Figure 6-1 The Relationship between Objectives and Achievement**

For the safety requirement assessment process, the research developed a methodology for combination of traditional safety assessment such as FHA, FTA and FMEA, with the emerging method, Formal Verification. This combination

method aimed to provide the comprehensive and correctness assessment for avionics software safety requirements.

For the architecture safety assessment process, the research summarized and compared the existing Development Assurance Level assignment process, and proposed the recommendation allocation process for avionics software according to efficiency and cost.

In case study, the research firstly analysed the FMS and position calculation. Secondly, the research established the formal model of FMS position calculation in NuSMV, and elicited the safety requirements by using traditional safety assessment methods. The formal model verified eleven safety requirements and generated enough evidence to prove the consistency and correctness of requirements. Finally, the results of case study proved the feasibility of this safety requirement assessment process and the combination method.

This research found the combination of the formal method and traditional safety assessment method, especially the Bi-Directional Analysis would be the effective and flexible methods for avionics software safety. Nowadays, there are more and more aviation companies or organizations to use the model-based approach for developing software and conducting safety assessment. This combination safety assessment method can be a part of new model-based approach and be the efficient support for safety assessment.

## **6.2 Further Work**

As the lack of software architecture of FMS position calculation function, the author only assigned the initial DAL to each FFS member according to the classification of top level failure condition, and put forward the degraded suggestions for some FFS members in case study. Due to the lack of software architecture, it was hard to have enough evidence to prove each FFS member has both development independence and item development independence. Therefore, it cannot degrade the DAL of FFS member. However, the author has given the suggestion of DAL degraded.

Model-based approaches have become increasingly mature, and could be applied to every phase in software safety assessment process. For example, the software safety assessment process is based on the software lifecycle model. In proposed software safety assessment process, the software safety requirement assessment process is based on the software formal model.

Currently, this formal model is only used for safety requirement verification. In further work, it can be expanded for each software safety assessment stage to achieve the safety purposes. For example, the model can be used in architecture safety assessment and code safety assessment, which can help to validate the software architecture and the implementation whether satisfies the proposed architecture or not. In order to make the formal model be suitable for the entire software safety assessment process, the formal model needs to be developed at the early stage of software lifecycle. Through each step of software development and safety assessment process, the model can be modified and applied, thereby it ensures the correctness of software design and implementation and the software safety.

Furthermore, software safety engineer wants to use quantitative methods which used in hardware to assess the software safety. Nowadays, there are two quantitative methods for software safety, one is fault density, and the other is fault rate.

Fault density is defined as the ratio of faults in a software to the size of a software, and it has the closed relationship with the number of fault. If one software contains more fault, its fault density will increase [57]. This means more than one fault will occur when software is running, which will decrease the software safety. For most safety critical software, the fault density 1 fault per thousand lines of code is acceptable [58]. But, fault density is difficult to calculate, because the new fault may be introduced when the code change.

The other is quantitative method is the failure rate, and this is the most common measure for hardware. However, to use software failure rate to assess software safety is a difficult issue, and there is even argument in defining software failure rate quantitatively. Unlike hardware maintenance, the software maintenance is

more complex and will undergo during its entire lifecycle. During each maintenance, some new defects may be introduced, and failure rate will be higher. When software failure rate can be estimated as a fixed value, the additional maintenance may be required. This causes the failure rate will be changed again. So, software failure rate is hard to quantifiable.

Software failure is the result of design error or fault that has been introduced in software lifecycle, especially the software development [59]. To solve this problem, the existing software safety standards are to use the approach and concept to limit and assure the software development process. In civil airborne software, the engineers use Development Assurance Level (DAL) to measure the rigor level of software development, and it can limit the error rate occurred in software development process to an acceptable level [10]. In proposed avionics software safety assessment process, the DAL assignment is one of the proposed method for software. DAL assignment accords with the most severity of related system hazard that the software could contribute. In some study, the authors think software failure rate also can be determined based on the severity of hazard, and DAL and failure rate has the corresponding relationship. For example, A level corresponds to the failure rate of  $1 \times 10^{-9}$  per flying hour [58].

Nowadays, there are some arguments about the reliability of software safety quantitative method. Some studies regard that these data can prove whether software is safe, and some studies think the failure rate of software is no meaning, because it doesn't make any mathematical sense. Moreover, the failure rate is kind of technical measure which helps the engineer to express their own opinion about the life characteristics of this software [60].

In my opinion, these methods only can use to show the probability of failure occurrence, and it cannot identify and modify the failure. In the further work, how to combine software safety quantitative method with the qualitative software safety assessment such as FTA and formal method will be ever more important. This new combination will make software safer.



## REFERENCES

- [1] S. Li and . S. Duo, "Safety analysis of software requirements: model and process," in *3rd International Symposium on Aircraft Airworthiness*, 2014.
- [2] A. . J. Kornecki and J. Zalewski, "Aviation Software: Safety and Security.," in *Wiley Encyclopedia of Electrical and Electronics Engineering*, John Wiley & Sons, Inc., 2015, pp. 1-30.
- [3] N. G. Leveson, *Safeware: system safety and computers*, New York: Association for Computing Machinery, 1995.
- [4] The Dutch Safety Board, "Crashed during approach, Boeing 737-800, near Amsterdam Schiphol Airport,," The Dutch Safety Board, Hague, 2010.
- [5] Radio Technical Commission for Aeronautics, DO-178C Software Considerations in Airborne Systems and Equipment Certification, Radio Technical Commission for Aeronautics, 2012.
- [6] Department of Defense, MIL-STD-882D, DEPARTMENT OF DEFENSE STANDARD PRACTICE: SYSTEM SAFETY, Department of Defense, 2000.
- [7] International Organization for Standardization, ISO/IEC 12207 Systems and software engineering – Software life cycle processes, International Organization for Standardization, 2008.
- [8] NASA, *NASA Software Safety Guidebook*, NASA, 2004.
- [9] SAE International, *ARP4761 GUIDELINES AND METHODS FOR CONDUCTING THE SAFETY ASSESSMENT PROCESS ON CIVIL AIRBORNE SYSTEMS AND EQUIPMENT*, SAE International, 1996.
- [10] SAE International, *ARP4754A Guidelines For Development Of Civil Aircraft and Systems*, SAE International, 2010.

- [11] I. Dodd and I. Habli, "Safety certification of airborne software: An empirical study," *Reliability Engineering & System Safety*, vol. 98, no. 1, pp. 7-23, 2012.
- [12] A. J. Kornecki and J. Zalewski, "Software Safety in Aviation".
- [13] D. L. Dvorak, "NASA Study on Flight Software Complexity," NASA, 2001.
- [14] V. Wiels , R. Delmas , P.-L. Garoche, D. Doose , J. Cazin and G. Durrieu , "Formal Verification of Critical Aerospace Software," *Aerospace Lab*, pp. 1-8, 5 2012.
- [15] E. Wong, V. Debroy and A. Restrepo, "The Role of Software in Recent Catastrophic Accidents," *IEEE Reliability Society 2009 Annual Technology Report*, 1 2009.
- [16] A. C. Tribble, S. P. Miller and D. L. Lempia, "Software safety analysis of a flight guidance system," *21st Digital Avionics Systems Conference*, pp. 27-31, 10 2002.
- [17] W. S. Greenwell, J. C. Knight and E. A. Strunk, "Risk-based Classification of Incidents," *Second Workshop on the Investigation and Reporting of Incidents and Accidents*, pp. 39-59, 2003.
- [18] Australian Transport Safety Bureau, "In-flight upset 154km west of Learmonth, WA," Australian Transport Safety Bureau, 2011.
- [19] Airbus, "A Statistical Analysis of Commercial Aviation Accidents 1958-2016," 7 2017.
- [20] L. Sha, "The Complexity Challenge in Modern Avionics Software," 14 8 2006.
- [21] Federal Aviation Administration , "Chapter 10 System Software Safety," in *System Safety Handbook*, Federal Aviation Administration , 2000.

- [22] S. Ravichandran, "IMPORTANCE OF SOFTWARE SYSTEM SAFETY WITH REFERENCE TO CUSTOMER POINT OF VIEW," *International Journal of Computer Aided Engineering and Technology*, pp. 520-526, 12 2012.
- [23] The International System Safety Society, "International System Safety Training Symposium 2014," 8 2014. [Online]. Available: <http://issc2014.system-safety.org/>. [Accessed 8 2017].
- [24] C. Tilghman, M. Zemore and M. Li, "Software Safety Analysis Procedures," 2014.
- [25] Software Engineering Standards Committee of the IEEE Computer Society, IEEE Standard for Software Safety Plans, IEEE Standards Board, 1994.
- [26] M. S. Javadi, A. Nobakht and A. Meskarbashee, "Fault Tree Analysis Approach in Reliability Assessment of Power System," *INTERNATIONAL JOURNAL OF MULTIDISCIPLINARY SCIENCES AND ENGINEERING*, pp. 46-50, 2 9 2011.
- [27] J. G. Glancey, "MEEG 446- Failure Analysis Methods What, Why and How," 2006.
- [28] W. E. Vesely, F. F. Goldberg, N. H. Roberts and D. F. Haasl, *Fault Tree Handbook*, U.S. Nuclear Regulatory Commission, 1981.
- [29] B. Vesely, "Fault Tree Analysis (FTA): Concepts and Applications," NASA.
- [30] M. Rausand, *SYSTEM RELIABILITY THEORY: Models, Statistical Methods, and Applications*, A JOHN WILEY & SONS, INC., P, 2004.
- [31] C. Giannetti, R. S. Ransing, D. T. Gethin, J. Sienz, M. R. Ransing and D. C. Bould, "Product specific process knowledge discovery using co-linearity index and penalty functions to support process FMEA in the steel industry," *CIE 2014-44th International Conference on Computers and Industrial*

*Engineering and IMSS 2014 - 9th International Symposium on Intelligent Manufacturing and Service Systems*, pp. 1-15, 10 2014.

[32] J. Delange and J. Hugues, "Safety Analysis with AADL," Software Engineering Institute , Pittsburgh, 2015.

[33] J. Best, "Failure Mode and Effect Analysis," Oklahoma State University, 2004.

[34] T. Gu, *Formal Methods of Software Development*, Higher Education Press, 2012.

[35] L. Sommerville, *Software Engineering*, Pearson Publishing Ltd, 2015.

[36] S. A. Kripke, "Semantical Analysis of Modal Logic I Normal Modal Propositional Calculi," *Mathematical Logic Quarterly*, pp. 67-96, 1963.

[37] C. Baier and J. P. Katoen, *Principles of Model Checking*, The MIT Press, 2008.

[38] C. Liu, "Aircraft System Safety Analysis Method Research Based on Model Checking," Nanjing University of Aeronautics and Astronautics , 2011.

[39] S. Konur, "A Survey on Temporal Logics," *Frontiers of Computer Science*, 25 4 2011.

[40] National University of Singapore, "CS 5219 - Temporal Logics," National University of Singapore, 2012.

[41] E. M. Clarke and E. A. Emerson, "Design and Synthesis of Synchronization Skeletons Using Branching Time Temporal Logic," *Springer Lecture Notes in Computer Science*, pp. 1-37, 1981.

[42] G. J. Holzmann, "The Model Checker SPIN," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 5 1997.

- [43] NuSMV Development Team, "NuSMV: a new symbolic model checker," [Online]. Available: <http://nusmv.fbk.eu/>. [Accessed 19 10 2017].
- [44] Software Engineering Standards Committee of the IEEE Computer Society, IEEE Recommended Practice for Software Requirements Specifications, The Institute of Electrical and Electronics Engineers, Inc, 1998.
- [45] A. C. Tribble, S. P. Miller and D. L. Le, "Software Safety Analysis of a Flight Guidance System," NASA, 2004.
- [46] L. Chung, "Model Checking," 2014.
- [47] V. Hilderman , "DO-178B Costs Versus Benefits," 2009.
- [48] S. Miller, "Contribution of Flight Systems to Performance-Based Navigation," *AERO Magazine*, vol. 34, 3 12 2012.
- [49] SKYbrary, "Flight Management System," SKYbrary, 3 8 2016. [Online]. Available:  
[https://www.skybrary.aero/index.php/Flight\\_Management\\_System](https://www.skybrary.aero/index.php/Flight_Management_System).  
[Accessed 18 10 2017].
- [50] Aviation Stack Exchange, "Aviation Stack Exchange-What's the difference between FMS and FMC?," 5 2017. [Online]. Available:  
<https://aviation.stackexchange.com/questions/37792/whats-the-difference-between-fms-and-fmc>. [Accessed 18 10 2017].
- [51] H. Aviation, "Collins\_FMS-3000\_Operator\_s\_Guide," 2017. [Online]. Available:  
[http://hatcheraviation.com/uploads/Collins\\_FMS-3000\\_Operator\\_s\\_Guide.pdf](http://hatcheraviation.com/uploads/Collins_FMS-3000_Operator_s_Guide.pdf). [Accessed 15 9 2017].
- [52] ubuntu, "Ubuntu and Debian," ubuntu, [Online]. Available:  
<http://www.ubuntu.com/about/about-ubuntu/ubuntu-and-debian>. [Accessed 10 2017].

- [53] A. Cimatti, E. M. Clarke, F. Giunchiglia and M. Roveri, "NUSMV: A New Symbolic Model Verifier," *Proceeding CAV '99 Proceedings of the 11th International Conference on Computer Aided Verification*, pp. 495-499, July 1999.
- [54] R. Cavada, A. Cimatti, C. A. Jochim, G. Keighren, E. Olivetti, M. Pistore, M. Roveri and A. Tchaltsev, "NuSMV 2.6 User Manual," FBK-irst, Trento, 2010.
- [55] E. M. Clarke, "Model Checking VI," Computer Science Department, Carnegie-Mellon University, Pittsburgh, 2014.
- [56] Airlines Electronic Engineering Committee, ARINC 702A-3 Advanced Flight Management Computer System, ANNAPOLIS: AERONAUTICAL RADIO, INC., 2006.
- [57] N. DiGiuseppe and J. A. Jones, "Fault density, fault types, and spectra-based fault," *Empirical Software Engineering*, pp. 928-967, 8 2015.
- [58] W. Youn and B.-j. Yi, "Software and hardware certification of safety-critical avionic systems:," *Computer Standards & Interfaces*, 20 February 2014.
- [59] J. McDermid and T. Kelly, "Software in Safety Critical Systems: Achievement and Prediction," *Nuclear Future*, vol. 3, 2006.
- [60] N. Singpurwalla, "The Failure Rate of Software: Does It Exist?," *IEEE Transactions on Reliability*, 10 1995.
- [61] RTCA, Inc, DO-236C Minimum Aviation System Performance Standards: Required Navigation Performance for Area Navigation, Washington: RTCA, Inc, 2013.
- [62] K. Jayasri and P. Seetharamaiah, "Study on Software Safety in Safety Critical Computer Systems," *International Journal of Computer Science And Technology*, pp. 163-167, 4 2015.

# APPENDICES

## Appendix A Position Calculation Safety Requirements Assessment Process

### A.1 Functional Hazard Assessment Result

Table A-1 Functional Hazard Assessment

Functional	Failure Condition	Number	Phase	Effects	Classification
Navigation		1			
FMS Navigation	The both sides of PFD indicate the wrong navigation and position information	1-1a	Climb; Cruises; Descent; Approach Landing	1. For aircraft: It leads aircraft deviate from its scheduled flight path, and significantly reduce the safety margin. 2. For Crew: It will significantly increase in workload to crew. 3. For passenger: No effect.	Hazardous
	Unannounced modification of the installed navigation database	1-1b	ALL	1. For aircraft: It significant reduces in safety margins. 2. For Crew: It will significant increase in workload to crew. 3. For passenger: No effect.	Hazardous
	Loss Navigation Function	1-1c	ALL	1. For aircraft: It slightly reduces in safety margins. 2. For Crew: It will slightly increase in workload to crew. 3. For passenger: No effect.	Hazardous

## A.2 Fault Tree Analysis

### A.2.1 Fault Tree Construction Result

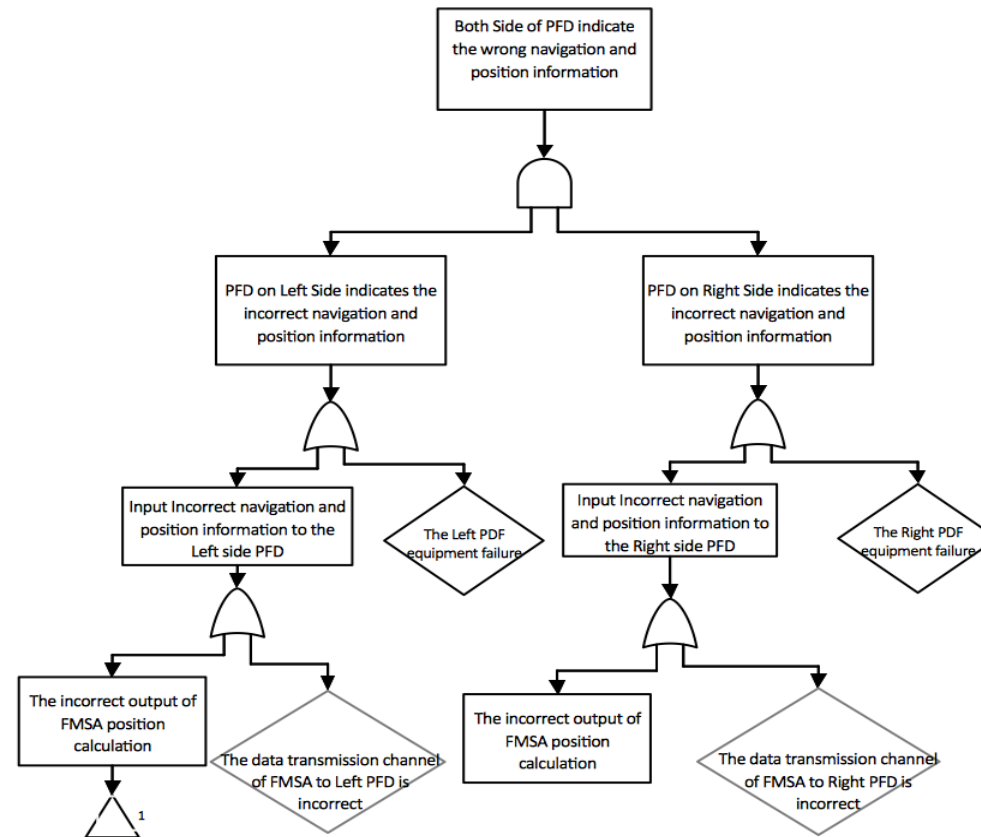


Figure A-1 The Fault Tree of “The both sides of PFD indicate the wrong navigation and position information”





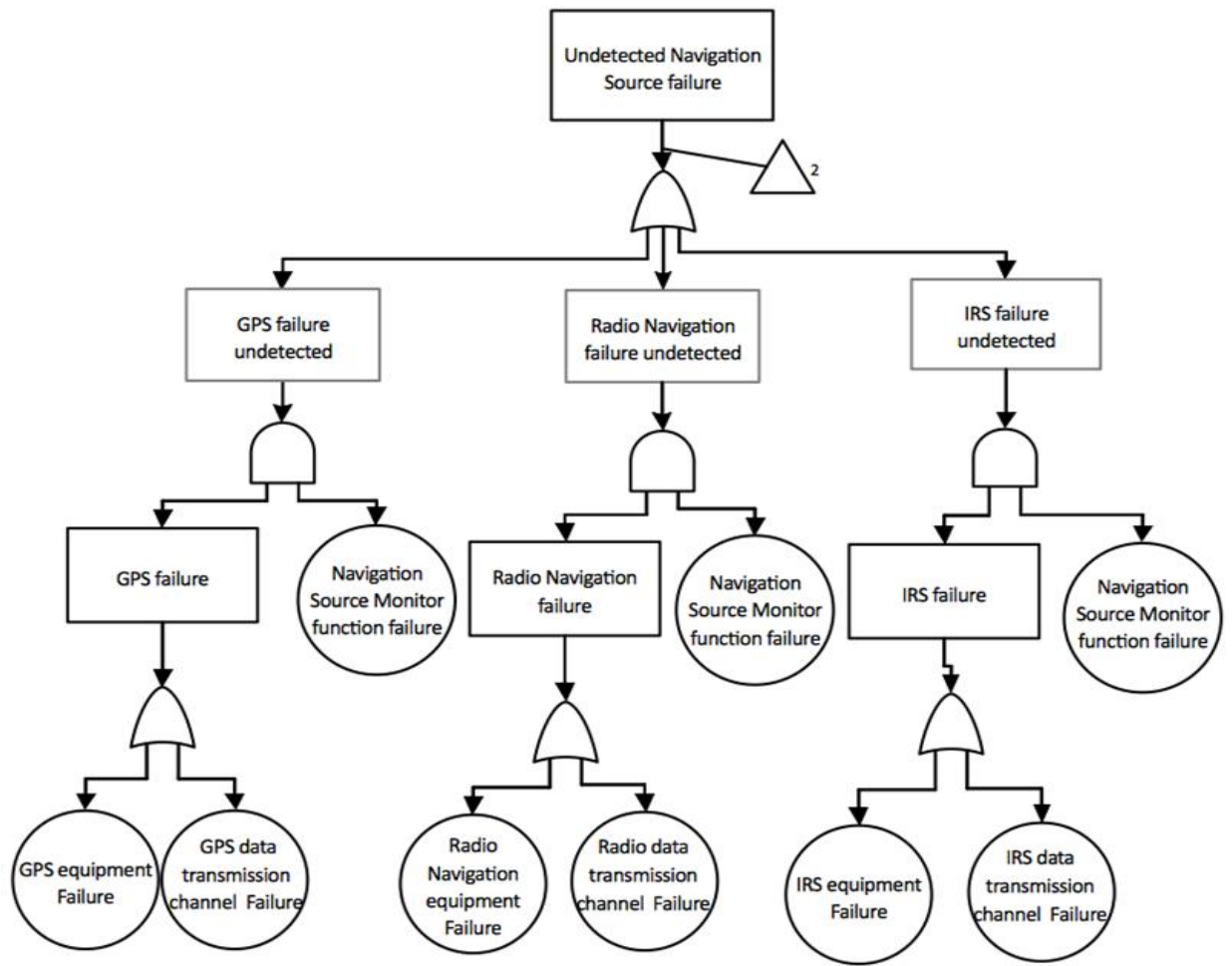
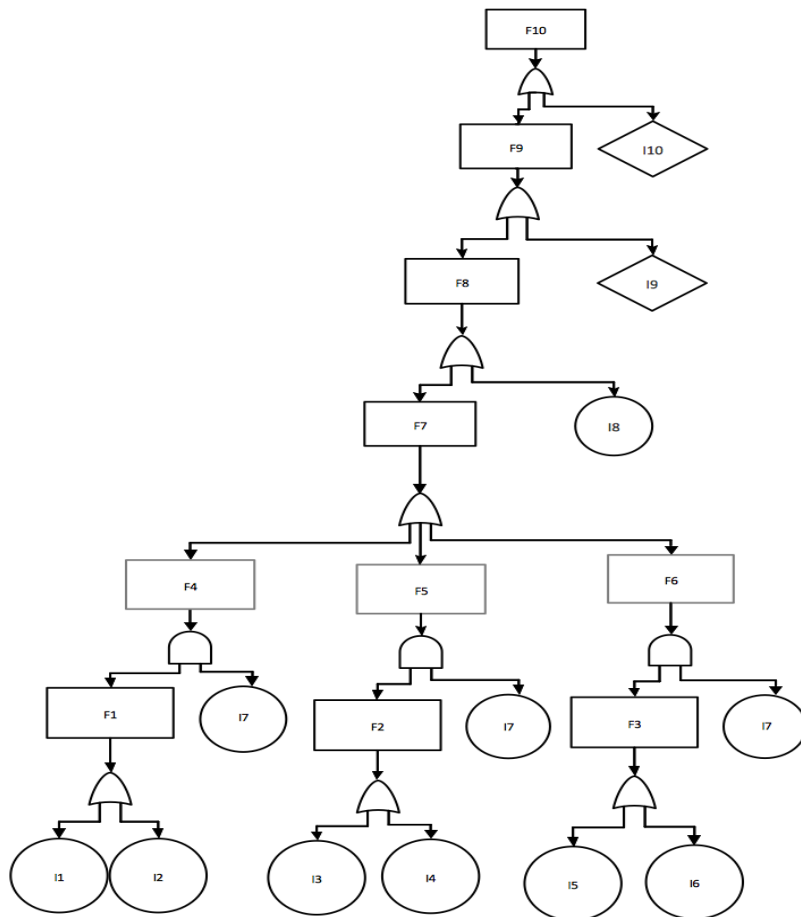


Figure A-3 Subtree of “The both sides of PFD indicate the wrong navigation and position information”-2

**A.2.2 Minimum Cut Set Analysis The minimum cut set starts with the “incorrect output of FMSA position calculation”**



**Figure A-4 Fault Tree Minimum Cut Set Analysis**

In Minimal Cut Set Analysis, the AND gate can be expressed as in “+”; the OR gate can be expressed as “\*”. So, the calculation shows below:

$$\begin{aligned}
 F1 &= I1 + I2 \\
 F2 &= I3 + I4 \\
 F3 &= I5 + I6 \\
 F4 &= F1 * I7 \\
 F5 &= F2 * I7 \\
 F6 &= F3 * I7 \\
 F7 &= F4 + F5 + F6 \\
 F8 &= F7 + I8 \\
 F9 &= F8 + I9 \\
 F10 &= F9 + I10
 \end{aligned}$$

Because I10 and I9 are undeveloped event, so it doesn't need to be considered.

$$\begin{aligned}
 \text{So, } F10 &= F7 + I8 \\
 &= I8 + I7 * (I1 + I2 + I3 + I4 + I5 + I6) \\
 &= I8 + I7 * I1 + I7 * I2 + I7 * I3 + I7 * I4 \\
 &\quad + I7 * I5 + I7 * I6
 \end{aligned}$$

The minimum cut sets are

$$\{I8\}, \{I7, I1\}, \{I7, I2\}, \{I7, I3\}, \{I7, I4\}, \{I7, I5\}, \{I7, I6\}$$

### A.3 Failure Mode and Effects Analysis

**Table A-2 FMEA of Navigation Source Selection Logical Failure**

Function: FMS position calculation	
Failure Mode	Effect on System
Navigation Source Selection Logical Failure	FMS cannot calculate the aircraft position, and thus FMS cannot output the correct position information.

**Table A-3 FMEA of Navigation Source Monitor Function Failure**

Function: FMS position calculation	
Failure Mode	Effect on System
Navigation Source Monitor Function Failure	Position Calculation function may not use the correct navigation source to calculate. FMS cannot calculate the aircraft position, and thus FMS cannot output the correct position information.

**Table A-4 FMEA of GPS Equipment Failure**

Function: FMS position calculation	
Failure Mode	Effect on System
GPS Equipment Failure	Position Calculation function cannot use GPS sensor information to calculate aircraft position. FMS cannot output the correct position information.

**Table A-5 FMEA of GPS Data Transmission Channel Failure**

Function: FMS position calculation	
Failure Mode	Effect on System
GPS Data Transmission Channel Failure	Position Calculation function cannot use GPS sensor information to calculate aircraft position. FMS cannot output the correct position information.

**Table A-6 FMEA of Radio Navigation Equipment Failure**

Function: FMS position calculation	
Failure Mode	Effect on System
Radio Navigation Equipment Failure	Position Calculation function cannot use Radio Navigation sensor information to calculate aircraft position. FMS cannot output the correct position information.

**Table A-7 FMEA of Radio Data Transmission Channel Failure**

Function: FMS position calculation	
Failure Mode	Effect on System
Radio Data Transmission Channel Failure	Position Calculation function cannot use Radio Navigation sensor information to calculate aircraft position. FMS cannot output the correct position information.

**Table A-8 FMEA of IRS Equipment Failure**

Function: FMS position calculation	
Failure Mode	Effect on System
IRS Equipment Failure	Position Calculation function cannot use IRS sensor information to calculate aircraft position. FMS cannot output the correct position information.

**Table A-9 FMEA of IRS Data Transmission Channel Failure**

Function: FMS position calculation	
Failure Mode	Effect on System
IRS Data Transmission Channel Failure	Position Calculation function cannot use IRS sensor information to calculate aircraft position. FMS cannot output the correct position information.

## A.4 Safety Requirement Specification Table

Table A-10 Safety Requirement Specification Table

Number	Safety Requirements	Formal Specification
1	If both GPS sensor status is normal and GPS sensor data is correct can be met simultaneously, the GPS mode will be active.	SPEC AG((gpssensor_status=normal) & (gpssensor_data=correct)-> (gps_mode=active))
2	If GPS sensor status is abnormal or GPS sensor data is wrong, the GPS mode will be rest.	SPEC AG((gpssensor_status=abnormal)   (gpssensor_data=wrong)-> (gps_mode=rest))
3	Counterexample: If GPS sensor status is normal or GPS sensor data is correct can be met, the GPS mode will be active.	SPEC AG((gpssensor_status=normal)   (gpssensor_data=correct)-> (gps_mode=active))
4	Radio Navigation Mode will be active only if GPS mode is rest and both Radio Navigation sensor is normal and sensor data is correct are satisfied simultaneously.	SPEC AG((gps_mode=rest) & (radiosensor_status=normal)&(radio sensor_data=correct)-> (radio_mode=active))
5	Counterexample: If both Radio Navigation sensor status is normal or Radio Navigation sensor data is correct can be meet, the Radio Navigation mode will be active.	SPEC AG((radiosensor_status=normal) (radio sensor_data=correct)-> (radio_mode=active))
6	IRS Mode will be active only if both GPS mode and Radio Navigation mode are rest and both IRS sensor is normal and sensor data is correct are satisfied simultaneously.	SPEC AG((gps_mode=rest) & (radio_mode=rest)&(irssensor_status=normal)&(irssensor_data=correct) -> (irs_mode=active))

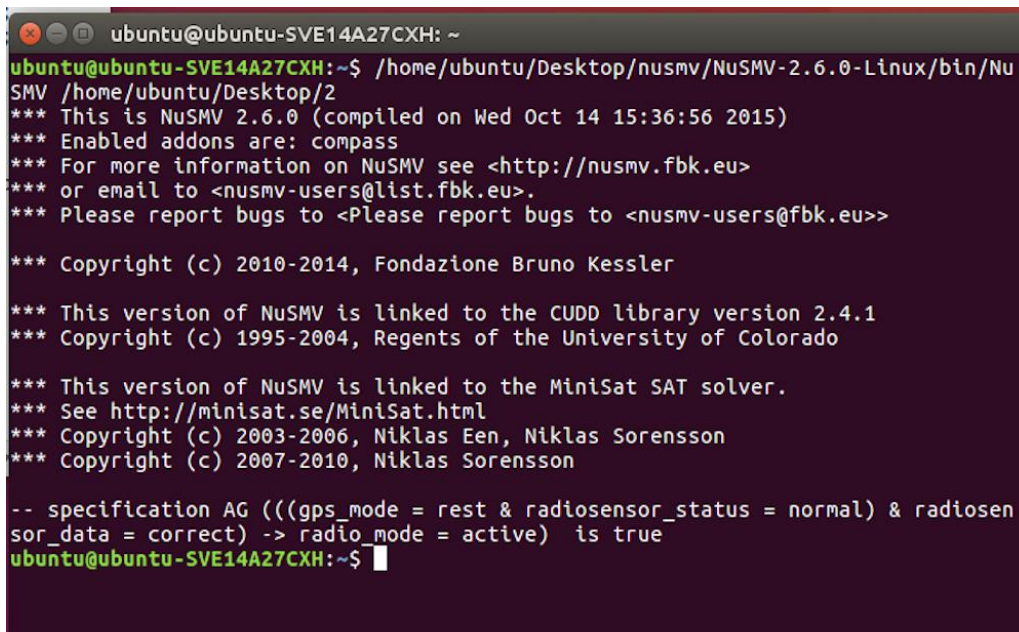
**Table A-11 Safety Requirement Specification Table-Continued**

7	IRS Mode will be turned to rest if IRS sensor is abnormal or IRS sensor data is wrong.	SPEC AG((irrsensor_status=abnormal   irrsensor_data=wrong)&(gps_mode=rest) & (radio_mode=rest)-> (irs_mode=rest))
8	If GPS or Radio Navigation mode is active, the IRS mode will be rest.	SPEC AG((gps_mode=active) (radio_mode=active)-> (irs_mode=rest))
9	If all of sensor are turned to rest, the FMS position calculation function will be turned off.	SPEC AG(gps_mode=rest & radio_mode=rest & irs_mode=rest) -> (fms_positioncalculate=off)
10	If there is at least one sensor is active, the FMS position calculation function will be turned on.	SPEC AG(gps_mode=active   radio_mode=active   irs_mode=active)-> (fms_positioncalculate=on)
11	Counterexample: If GPS mode is active and both Radio Navigation and IRS mode rest, the FMS position calculation function will be turned off.	SPEC AG((gps_mode=rest & radio_mode=rest & irs_mode=active) -> (fms_positioncalculate=off))

## A.5 Safety Requirement Verification Result

Specification 1 to 3 already shows in previous chapter, so here doesn't present again.

1. If both GPS sensor status is normal and GPS sensor data is correct can be met simultaneously, the GPS mode will be active.
2. If GPS sensor status is abnormal or GPS sensor data is wrong, the GPS mode will be rest.
3. Counterexample: If GPS sensor status is normal or GPS sensor data is correct can be met, the GPS mode will be active.
4. Radio Navigation mode will be active only if GPS mode is rest and both Radio Navigation sensor is normal and sensor data is correct are satisfied simultaneously.
  - SPEC AG((gps\_mode=rest) & (radiosensor\_status=normal)&(radiosensor\_data=correct)-> (radio\_mode=active))



```
ubuntu@ubuntu-SVE14A27CXH: ~
ubuntu@ubuntu-SVE14A27CXH:~$ /home/ubuntu/Desktop/nusmv/NuSMV-2.6.0-Linux/bin/NuSMV /home/ubuntu/Desktop/2
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:36:56 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

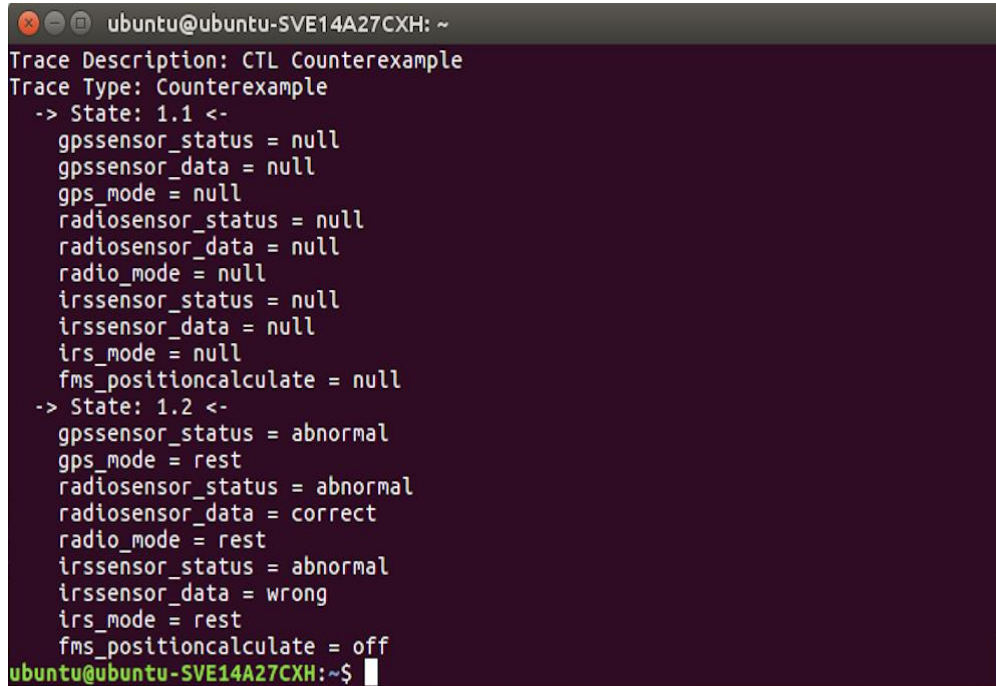
-- specification AG (((gps_mode = rest & radiosensor_status = normal) & radiosensor_data = correct) -> radio_mode = active) is true
ubuntu@ubuntu-SVE14A27CXH:~$
```

Figure A-5 Verification Result of Specification 4



5. Counterexample: If Radio Navigation sensor status is normal or Radio Navigation sensor data is correct can be met, the Radio Navigation mode will be active.

- SPEC AG((radiosensor\_status=normal) | (radiosensor\_data=correct) -> (radio\_mode=active))



```
ubuntu@ubuntu-SVE14A27CXH: ~
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  gpssensor_status = null
  gpssensor_data = null
  gps_mode = null
  radiosensor_status = null
  radiosensor_data = null
  radio_mode = null
  irssensor_status = null
  irssensor_data = null
  irs_mode = null
  fms_positioncalculate = null
-> State: 1.2 <-
  gpssensor_status = abnormal
  gps_mode = rest
  radiosensor_status = abnormal
  radiosensor_data = correct
  radio_mode = rest
  irssensor_status = abnormal
  irssensor_data = wrong
  irs_mode = rest
  fms_positioncalculate = off
ubuntu@ubuntu-SVE14A27CXH:~$
```

Figure A-6 Verification Result of Specification 5

6. IRS Mode will be active only if both GPS mode and Radio Navigation mode are rest and both IRS sensor is normal and sensor data is correct are satisfied simultaneously.

- SPEC AG((gps\_mode=rest) & (radio\_mode=rest)&(irssensor\_status=normal)&(irssensor\_data=correct)-> (irs\_mode=active))

```

ubuntu@ubuntu-SVE14A27CXH: ~
NuSMV terminated by a signal

Aborting batch mode
ubuntu@ubuntu-SVE14A27CXH:~$ /home/ubuntu/Desktop/nusmv/NuSMV-2.6.0-Linux/bin/Nu
SMV /home/ubuntu/Desktop/2
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:36:56 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

-- specification AG (((gps_mode = rest & radio_mode = rest) & irssensor_status
= normal) & irssensor_data = correct) -> irs_mode = active) is true
ubuntu@ubuntu-SVE14A27CXH:~$ █

```

Figure A-7 Verification Result of Specification 6

7. IRS Mode will be turned to rest if IRS sensor is abnormal or IRS sensor data is wrong.

- SPEC AG((irssensor\_status=abnormal | irssensor\_data=wrong)&(gps\_mode=rest) & (radio\_mode=rest)-> (irs\_mode=rest))

```

ubuntu@ubuntu-SVE14A27CXH: ~
ubuntu@ubuntu-SVE14A27CXH:~$ /home/ubuntu/Desktop/nusmv/NuSMV-2.6.0-Linux/bin/Nu
SMV /home/ubuntu/Desktop/2
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:36:56 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

-- specification AG (((irssensor_status = abnormal | irssensor_data = wrong) &
gps_mode = rest) & radio_mode = rest) -> irs_mode = rest) is true
ubuntu@ubuntu-SVE14A27CXH:~$ █

```

Figure A-8 Verification Result of Specification 7

8. If GPS or Radio Navigation mode is active, the IRS mode will be rest.

- SPEC AG((gps\_mode=active)|(radio\_mode=active)-> (irs\_mode=rest))

```

ubuntu@ubuntu-SVE14A27CXH: ~
ubuntu@ubuntu-SVE14A27CXH:~$ /home/ubuntu/Desktop/nusmv/NuSMV-2.6.0-Linux/bin/NuSMV /home/ubuntu/Desktop/2
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:36:56 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

-- specification AG ((gps_mode = active | radio_mode = active) -> irs_mode = rest) is true
ubuntu@ubuntu-SVE14A27CXH:~$ █

```

Figure A-9 Verification Result of Specification 8

9. If all of sensor are turned to rest, the FMS position calculation function will be turned off.
- SPEC AG(gps\_mode=rest & radio\_mode=rest & irs\_mode=rest) -> (fms\_positioncalculate=off)

```

ubuntu@ubuntu-SVE14A27CXH: ~
NuSMV terminated by a signal
Aborting batch mode
ubuntu@ubuntu-SVE14A27CXH:~$ /home/ubuntu/Desktop/nusmv/NuSMV-2.6.0-Linux/bin/NuSMV /home/ubuntu/Desktop/2
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:36:56 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

-- specification (AG ((gps_mode = rest & radio_mode = rest) & irs_mode = rest) -> fms_positioncalculate = off) is true
ubuntu@ubuntu-SVE14A27CXH:~$ █

```

Figure A-10 Verification Result of Specification 9

10. If there is at least one sensor is active, the FMS position calculation function will be turned on.

- SPEC AG(gps\_mode=active | radio\_mode=active | irs\_mode=active) → (fms\_positioncalculate=on)

```
ubuntu@ubuntu-SVE14A27CXH:~$ /home/ubuntu/Desktop/nusmv/NuSMV-2.6.0-Linux/bin/NuSMV /home/ubuntu/Desktop/2
*** This is NuSMV 2.6.0 (compiled on Wed Oct 14 15:36:56 2015)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <Please report bugs to <nusmv-users@fbk.eu>>

*** Copyright (c) 2010-2014, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://minisat.se/MiniSat.html
*** Copyright (c) 2003-2006, Niklas Een, Niklas Sorensson
*** Copyright (c) 2007-2010, Niklas Sorensson

-- specification (AG ((gps_mode = active | radio_mode = active) | irs_mode = active) -> fms_positioncalculate = on) is true
ubuntu@ubuntu-SVE14A27CXH:~$
```

Figure A-11 Verification Result of Specification 10

11. Counterexample: If GPS mode is active and both Radio Navigation and IRS mode rest, the FMS position calculation function will be turned off.

- SPEC AG((gps\_mode=rest & radio\_mode=rest & irs\_mode=active) → (fms\_positioncalculate=off))

```
ubuntu@ubuntu-SVE14A27CXH: ~
-- specification AG (((gps_mode = rest & radio_mode = rest) & irs_mode = active) -> fms_positioncalculate = off) is false
-- as demonstrated by the following execution sequence
Trace Description: CTL Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  gpssensor_status = null
  gpssensor_data = null
  gps_mode = null
  radiosensor_status = null
  radiosensor_data = null
  radio_mode = null
  irssensor_status = null
  irssensor_data = null
  irs_mode = null
  fms_positioncalculate = null
-> State: 1.2 <-
  gpssensor_status = abnormal
  gps_mode = rest
  radiosensor_status = abnormal
  radio_mode = rest
  irs_mode = active
  fms_positioncalculate = on
ubuntu@ubuntu-SVE14A27CXH:~$
```

Figure A-12 Verification Result of Specification 11

## A.6 FMS Position Calculation Function NuSMV Code

```
MODULE main
```

```
VAR
```

```
gpssensor_status:{null,abnormal,normal};
```

```
gpssensor_data:{null,wrong,correct};
```

```
gps_mode:{null,active,rest};
```

```
radiosensor_status:{null,abnormal,normal};
```

```
radiosensor_data:{null,wrong,correct};
```

```
radio_mode:{null,active,rest};
```

```
irssensor_status:{null,abnormal,normal};
```

```
irssensor_data:{null,wrong,correct};
```

```
irs_mode:{null,active,rest};
```

```
fms_positioncalculate: {null,on,off};
```

```
ASSIGN
```

```
init(gps_mode):=null;
```

```
init(gpssensor_status):=null;
```

```
init(gpssensor_data):=null;
```

```
--gps transimition
```

```
TRANS ((gpssensor_status=null) & (gpssensor_data=null))->(gps_mode=null);
```

```
TRANS (gpssensor_status=normal &  
gpssensor_data=correct)->(gps_mode=active);
```

```
TRANS ((gpssensor_status=abnormal) |  
(gpssensor_data=wrong)&(gps_mode=null))->(gps_mode=rest);
```

```
TRANS ((gpsensor_status=abnormal) |  
(gpsensor_data=wrong)&(gps_mode=active))->(gps_mode=rest);
```

ASSIGN

```
init(radio_mode):=null;
```

```
init(radiosensor_status):=null;
```

```
init(radiosensor_data):=null;
```

--radio transition

```
TRANS (gps_mode=active)->(radio_mode=rest);
```

```
TRANS ((gps_mode=rest)&(radio_mode=null)&(radiosensor_status=null) &  
(radiosensor_data=null))->(radio_mode=null);
```

```
TRANS ((radiosensor_status=normal & radiosensor_data=correct &  
gps_mode=rest))->(radio_mode=active);
```

```
TRANS ((radiosensor_status=abnormal |  
radiosensor_data=wrong)&(gps_mode=active))->(radio_mode=rest);
```

```
TRANS (radiosensor_status=abnormal | radiosensor_data=wrong &  
radio_mode=active & gps_mode=rest)->(radio_mode=rest);
```

ASSIGN

```
init(irs_mode):=null;
```

```
init(irssensor_status):=null;
```

```
init(irssensor_data):=null;
```

--irs transition

TRANS

```
((gps_mode=rest)&(radio_mode=rest)&(irs_mode=null)&(irrsensor_status=null)
& (irrsensor_data=null))->(irs_mode=null);
```

```
TRANS ((irrsensor_status=normal & irrsensor_data=correct)&(gps_mode=rest)
& (radio_mode=rest))->(irs_mode=active);
```

TRANS

```
((gps_mode=active)|(radio_mode=active))(((irrsensor_status=abnormal |
irrsensor_data=wrong)))->(irs_mode=rest);
```

ASSIGN

```
init (fms_positioncalculate):=null;
```

--fms positioncalculate

```
TRANS ((gps_mode=null) & (radio_mode=null) &
(irs_mode=null))->(fms_positioncalculate=null);
```

```
TRANS ((gps_mode=active) & (radio_mode=active) &
(irs_mode=active))->(fms_positioncalculate=on);
```

```
TRANS ((gps_mode=active) | (radio_mode=active) |
(irs_mode=active))->(fms_positioncalculate=on);
```

TRANS

```
((gps_mode=rest)&(radio_mode=rest)&(irs_mode=rest))->(fms_positioncalculat
e=off);
```