SAFE SOFTWARE SYSTEM DEVELOPMENT FRAMEWORK FOR MINI UNMANNED AERIAL SYSTEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ÖNDER ALTAN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
AEROSPACE ENGINEERING

SEPTEMBER 2014

Approval of the thesis:

SAFE SOFTWARE SYSTEM DEVELOPMENT FRAMEWORK FOR MINI UNMANNED AERIAL SYSTEMS

submitted by **ÖNDER ALTAN** in partial fulfillment of the requirements for the degree of **Master of Science in Aerospace Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. Ozan Tekinalp
Head of Department, **Aerospace Engineering** _____

Prof. Dr. Nafiz Alemdaroğlu
Supervisor, **Aerospace Engineering Department, METU** _____

Examining Committee Members:

Prof. Dr. Nihan Kesim Çiçekli
Computer Engineering Department, METU _____

Prof. Dr. Nafiz Alemdaroğlu
Aerospace Engineering Department, METU _____

Prof. Dr. Ozan Tekinalp
Aerospace Engineering Department, METU _____

Assoc. Prof. Dr. İlkay Yavrucuk
Aerospace Engineering Department, METU _____

Assist. Prof. Dr. Ali Türker Kutay
Aerospace Engineering Department, METU _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ÖNDER ALTAN

Signature :

ABSTRACT

SAFE SOFTWARE SYSTEM DEVELOPMENT FRAMEWORK FOR MINI UNMANNED AERIAL SYSTEMS

Altan, Önder
M.S., Department of Aerospace Engineering
Supervisor : Prof. Dr. Nafiz Alemdaroğlu

September 2014, 132 pages

Due to the rapid growth of the utilization of Mini Unmanned Aerial Systems (MUAS) in populated areas, system safety concerns regarding these systems are becoming more important more than ever. Reliability and robustness of the software systems (SS), which are embedded within MUASs to handle their autonomy, should be assured by applying safe development methodologies. This thesis introduces a unique and comprehensive software framework for design, implementation and testing of MUAS software systems, which ensures desired software system safety is achieved with reasonable effort by prioritizing and applying software safety (software airworthiness) concept.

The proposed software framework increases software reliability as it simplifies and assures the implementation of fault detection, tolerance and recovery mechanisms, and focuses on software system robustness by identifying failure conditions of the software in MUASs. Besides, as the framework focuses on simple development approach, it tries to reduce efforts undertaken to perform safety analysis and reviews in the development life cycle. In addition to design and implementation methodologies provided by the framework; mission based, full autonomous and simple testing methodology (Assassin Process Method, APM) is introduced in the framework to improve entire software system safety. Moreover, autonomous APM helps small MUAS teams during development phase by providing human readable test verification results

as a test assessment report.

Through this thesis, the framework as well as the philosophy behind why such framework is necessary, important and unique is explained in detail. Finally, all contributions of the idea to safe software development for MUASs are presented through a prototype in which the verification and tests of the intended software system have been performed. In the prototype, a MUAS's software system, which is developed by using the suggested framework for a created case, is embedded into a hardware architecture, and using hardware-in-the-loop (HIL) simulation as a real-time integration environment, system verification process is iterated for safe software system development steps introduced in the framework.

Keywords: Software Airworthiness, Software System Safety, Real Time Operating Systems, Software Reliability, Hardware-in-the-loop Simulation, Safe Software System Design, Software Development Life-Cycle, Mission-based Testing, Assassin Process Method-APM

ÖZ

MİNİ İNSANSIZ HAVA ARAÇLARI İÇİN GÜVENLİ YAZILIM GELİŞTİRME İSKELETİ

Altan, Önder
Yüksek Lisans, Havacılık ve Uzay Mühendisliği Bölümü
Tez Yöneticisi : Prof. Dr. Nafiz Alemdaroğlu

Eylül 2014 , 132 sayfa

Mini İnsansız Uçan Sistemler'in (MİUS), sayılarındaki ve insanlı bölgelerdeki operasyonel kullanımlarının belirgin artışına bağlı olarak, bu sistemlerin güvenliği konusundaki çekinceler, hiç olmadığı kadar önemli bir tartışma konusu haline gelmiştir. Buna bağlı olarak MİUS'lara entegre edilen ve onların otonom kontrol edilmesini sağlayan alt sistemleri olan yazılım sistemleri de, genel sistem güvenliğini düşünülerek ve teminat altına alarak geliştirilmelidir. Bu tez çalışmasında; hedeflenen yazılım sistem güvenliğine, makul bir çaba harcayarak erişilmesini, yazılım sistem güvenliği konseptini öncelikli görüp, uygulayarak garantilemeyi amaçlayan; yazılım tasarım, kodlama ve entegrasyon süreçleri için geliştirilmiş benzersiz ve kapsamlı bir Güvenli Yazılım Geliştirme İskeleti tanıtılmaktadır.

Önerilen Yazılım İskeleti, yazılım güvenilirliğini; yapısındaki hata tanıma, hata toleransı ve sistem geri kazanım mekanizmalarıyla arttırmakta, bunun yanında da tanımlanmış hata durumları için MİUS yazılım sisteminin dayanıklılığını geliştirmeye odaklanmaktadır. Bu tezde, yazılım geliştirme iskeletince sağlanan yazılım tasarım ve kodlama metotlarına ek olarak; tüm MİUS sistem güvenliğini arttıran görev bazlı, tam otonom test yöntemi Suikast İşlem Methodu (SİM) önerilmiştir. Otonom SİM, ufak MİUS geliştirme ekiplerine, sistem güvenliği değerlendirme aşamalarında; kolay okunup anlaşılabilir test doğrulama sonuçları ve değerlendirme raporları almala-

rını sağlayarak, yardımcı olur.

Bu tezde, yazılım iskeletinin MİUS'lar için gerekliliğinin tartışılmasının yanı sıra, böyle bir yapının neden önemli ve benzersiz olduğu da detaylı bir şekilde anlatılmıştır. Son olarak, tezde tanıtılan konseptin, MİUS'lar için Güvenli Yazılım Geliştirmeye olan katkıları; donanım-döngüde simülasyon yöntemiyle kurulan entegrasyon ortamı ve onun üzerinde geliştirilen yazılım sisteminin oluşturulup, doğrulama çalışmaları ve testlerin gerçeklendiği bir örnek durum çalışması üzerinde uygulanarak gözlemlenmiş, iskeletin faydaları ve literatüre olan katkıları detaylarıyla sunulmuştur.

Anahtar Kelimeler: Yazılım Uçuşaelverişliliği, Yazılım Sistem Güvenliği, Gerçek Zamanlı İşletim Sistemi, Yazılım Güvenilirliği, Donanım Döngüde Simülasyon, Güvenlik Durum Çalışması, Güvenli Yazılım Sistem Tasarımı, Yazılım Geliştirme Döngüsü, Görev Tabanlı Test, Suikastçi İşlem Metodu-SİM

To Happiness & Peace

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my supervisor Prof. Nafiz Alemdaroğlu for his endless support and encourage.

My special thanks to Burcu Yılmaz and Fulya Tuncer Cetin for their being not only my colleagues but also perfect friends with undeniable trust. Their advices were the most precious gifts I had ever taken during the thesis process.

Being a lucky person, I would like to express my special thanks to Zeynep Kirecci as she always encourages me to achieve better. Without her endless support, this thesis would not be completed.

I would like to thank Engin Çağlav for sharing his knowledge and experience with me. Without him, this thesis would be incomplete.

I really want to express my gratitude to my colleagues, Sinan Pakkan and Kaan Doğan for their friendly support and technical advice when I felt myself very hopeless.

I wish to state my thanks to my dear friend and colleague Hüseyin Kaval for answering my questions with a great patience.

I also want to thank to my special friends Sule Akdogan and Serkan Naneci for their encouragements and presence even in the jury to support me.

Last but not least, I would like to express my deepest thanks to my mom, who always supports and encourages me, for her trust, understanding and patience. I am very lucky to have such a great mom.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

A/C	Aircraft
AFCS	Automatic Flight Control System
APM	Assassin Process Method
CDR	Critical Design Review
DAP	Data Acquisition Process
DLP	Data Logger Process
FCP	Flight Controller Process
FCS	Flight Control System
FHA	Functional Hazard Analysis
GCP	Ground Control Station Communication Process
MUAS	Mini Unmanned Aerial System
PDR	Preliminary Design Review
PHA	Preliminary Hazard Analysis
PMP	Process Manager Process
SSS	Safe Software System
SC	Safety-Critical
SS	Software System
SLP	Status Logger Process
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle
WDP	Watchdog Process

CHAPTER 1

INTRODUCTION

Mini Unmanned Arial Systems (MUAS) can be defined as a type of small aircraft that does not carry human operator on board and they have been in use for various applications for a few decades now. Nowadays, there is an increasing demand for developing MUASs for both civilian and military applications due to their wide range of usage, e.g., surveillance, reconnaissance, search and rescue, agricultural monitoring and air photography.

The autonomy for MUASs is sustained by using an efficient and reliable Flight Control System (FCS) or autopilot, which handles lots of tasks such as sensor fusion, communications, path planning, trajectory generation, trajectory regulation, cooperative tactics, task allocation and scheduling[2, 5]. As the MUASs are used in the above mentioned fields more frequently; problems related to more complex and real-time control and navigation have to be solved by the system software mostly to achieve precise and effective MUAS.

As the complexity of MUAS's increases, the complexity of the autonomous flight requirements, which are controlled by software sub-system, increases as well. This increasing trend in autonomy together with the drastic growth in available number of MUASs enforces both authorities and industry to question the safety of these autonomous systems. In order to achieve some safety confidence level, safety engineering methodologies applied on manned aerial vehicles become essential and significant for software systems as well[1].

Although there exists software development guidelines, accepted by aviation author-

ities such as DO-178B, MIL-STD-882 etc., having safer aerial systems, and their implications to MUASs do not seem very practical without overburdening the entire MUAS development process. As they are so generic,and they focus on development life cycle instead of being a cook book with instructions, they also support the best engineering practices rather than mandating an architecture or specific methodology. Applying such guidelines to MUAS software development processes seems impractical and inefficient when small aerial system development paradigm has been taken into consideration.

This thesis introduces a systematic and intelligent software framework to achieve a safer software system for MUASs without overburdening the entire software development process. The framework provides an architecture and some methods to design, implement and test the software system, as well as satisfying safety requirements in a cost efficient way. In addition to its contributions to design and implementation phases of the entire development process, unique testing methodology, i.e. – Assassin Process Method (APM), which is specifically proposed within this study, suggests an efficient, successful and innovative software verification feature of the framework to achieve software airworthiness for MUASs.

The concept proof of the framework is realized by applying it to a created safety case for Autonomous Flight Control System (AFCS) onto an onboard computer for a fixed-wing MUAV. Through this case, a prototype implementing the framework is also developed as a proof-of-concept implementation. Hence, not only capabilities but also the efficiency and accuracy of the framework are evaluated at a concrete level. In this manner, after realization of the framework with a MUAS software system prototype, it is proved that the efficiency of the framework and its ease of usability increases software system safety in MUASs and it can definitely be used as a comprehensive tool to reach generic software development milestones, which are design, coding and testing.

In conclusion, generic software development life cycle followed to reach safer software systems for MUASs is iterated in this study and the gaps between the design, implementation and testing milestones of the life cycle is fulfilled with the unique software framework. Hence it is successfully indicated that the framework can achieve

not only the safety requirements but also more robust and reliable software system at the end in a cost, effort and time effective way.

1.1 Background Of The Problem

Mini Unmanned Aerial Systems (MUASs) have wide range of usage areas in both civilian and military applications. In recent years, especially small UAVs have been using due to their robustness, high reliability and portability in warfare operations. Design of FCSs (autopilots), which make a UAS autonomous, is one of the recent research fields in the industry and academia, and there are various efforts to build a MUAS[6].

Due to the technological advances in computer processors, memory and other parts of computer components, software systems in MUAS becomes more complex and dominant increasingly. As complexity and significance of software increases, more emphasis on software systems and their quality is started to be given. Although for manned bigger aerial systems, software quality and software system safety are achieved by following some well known guidelines such as DO-178B etc., however following such guideline for MUASs is not practical due to the lack of resources. Additionally, the commonly followed guideline for civil and military aviation software development which is DO-178B only provides generic software life cycle with milestones to reach but does not supply any requirements, specific descriptions and regulations. As a result, not only lack of resources but also lack of know-how and experience of the small MUAS development teams about DO-178B prevents them from reaching software safety as DO-178B is recognized as “the state of the art”, which means the best engineering practice[24].

On the other hand, considering the enthusiasm for integrating them into civil applications and using them in the civil terrains and also with their growth in population, MUASs have to reach some level of safety and satisfy airworthiness requirements. However, DO-178B or any other guidelines do not provide a direct road map with detailed and specific requirements and methods to achieve a software safety neither for manned aerial systems nor for MUASs. Following these guidelines also becomes

overwhelming as a whole for a small team of MUAS development. Considering the facts mentioned above, while airworthiness practices for large aerial systems may be similar to manned aircraft, it is clear that MUAS require a paradigm shift from the airworthiness practices of manned aircraft[4]. Furthermore an easy, flexible, systematic, autonomous and comprehensive tool, framework or method are required to be developed for the benefit of aeronautics community and to motivate MUAS development teams to stand at the safe zone during MUASs software development process.

1.2 Statement Of The Problem

Due to the rapidly increasing roles of the software in not only manned aerial systems but also MUASs, authorities, developers and academia started to put more emphasis on software quality and safety. Although there are well known software assurance guidelines accepted by authorities such as DO-178B etc., they are not very well applicable to the MUASs because these guidelines are written so generic and their processes are time consuming and time intensive[18] which is definitely not acceptable for MUAS development.

On the other hand caused by the great enthusiasm of aerospace community to develop MUASs for both military and civil applications, many design and development teams effort to have their own MUAS. Under these circumstances it is clear that the population of MUAS on the market and daily life grows rapidly however one of the most significant parts of the MUASs which are software systems are not being investigated and studied very well to reach safer software system in systematic way neither in industry nor in academia. As anticipated for the near future, MUAS are expected to take place in sky more frequently with more complex capabilities and varying mission profiles.

Under these circumstances in order to reach some level of safety maturity for software system of MUAS, the projection of current guidelines to newer one should be performed and a systematic way to reach the milestones provided by the guidelines should be followed without overburdening and destroying the small aerial vehicle design and development paradigm. In this way, it is expected to ensure that even small

development teams of MUASs reach a level of safety for their systems and they can sustain it easily. It is obvious that easy and smart comprehensive framework might help and motivate people, communities or organizations interested in MUAS development.

1.3 Purpose Of The Study

The purpose of this thesis is to introduce a comprehensive software framework which can be used to design, implement and test a safe software system for MUASs in systematic way by considering and satisfying the software system's safety without overburdening the entire development process. In addition to the realization of design, implementation and testing phases of SSS development process for MUAS, the generic process milestones suggested by the well-known guidelines are also expected to be followed in order to have a complete prototype of the framework. A hardware-in-the-loop system is planned to be designed and implemented to show the accuracy of the proposed framework in realistic simulation environment. Moreover, this study aims to discuss whether using such inclusionary tool during the critical software design, coding and testing phases improves the safe software development process for MUASs with feasible effort and limited resources.

1.4 Significance Of The Study

This thesis study introduces a software framework which can be used during the software development phase for MUASs applying safe system development aspects. The generic safe software system development process mandates to reach four main milestones which are software requirements, software design, software coding (implementation and testing), and software integration. The entire gaps between these milestones are completed by developers. Considering these four milestones, the framework is especially useful to accomplish design and coding tasks' requirements in a comprehensive, easy and efficient way. There are four significant contributions of this study to the safe MUAS software development concept.

First the framework allows design, coding and implementation activities to be performed in a systematic way. At the same time, developers are still parallel with the reference guidelines. Meanwhile, gaps between process's milestones are filled by applying the framework in a very easy and efficient way.

Second, this study technically puts most important software safety related definitions on stage and handles problems related with these definitions with one and unique framework. For example, during the software design process, framework's multi-process architecture eliminates single-point failures and increases modularity. As a part of the architecture, fault detection, tolerance and recovery mechanisms improve not only reliability but also robustness of the system. Furthermore, synchronization issues caused by the multi-processing paradigm are eliminated with an easy way. High level implementation of design reduces the complexity caused by low level hardware interactions and complicated software design patterns.

The test method suggested in the framework is definitely the third contribution of the thesis to both academia and industry. It is undeniable that software test process takes a lot of time and consumes considerable effort during software development. Being the most challenging part of the entire development process, testing is given a special attention during the thesis study and Assassin Process Method, APM is developed. Using autonomous, intelligent and easy APM method, all safety requirements can be tested and the test results can be obtained without human interaction. Moreover, for the non-test or non-software people, APM provides analytical test assessment reports. As testing is a very complex part of the entire development process with many kinds of methods, tools, systems and purposes; APM successfully ensures whether software system is safe or not.

Apart from its contributions to design, implementation (coding) and testing processes, this study also integrates safe software development philosophy to MUAS concept, develops a prototype with a successful achievements and proves that even small teams with less resource for MUAS development can reach some level of safety and sustain the gained safety level to end of the life cycle. The result of the thesis shows that engineering principles are still applicable during the safe software development for MUASs process and the comprehensive framework can accomplish safety require-

ments with a reasonable effort.

In conclusion, the software development framework proposed in this thesis provides a comprehensive architecture and methods with easy usability for software design, implementation and testing during safer MUASs development. Using the framework, in a systematic way without spending much effort and dedicating excessive resources to each software design, coding and testing tasks introduced by the safety assurance guidelines can be reached satisfactorily.

1.5 Assumptions

For this study, the following assumptions are made;

- The aerial platform is assumed to be verified.
- Governing mathematical equations used in the literature, during MUAV modeling phase of the platform assumed to be correct.
- All hardware components are assumed to be working properly.
- The safe case is constructed in order to implement the framework to only AFCS subcomponent.
- Test cases are performed by using XPC Target simulation
- Software system safety processes are performed with an effort of satisfying minimum requirements of safe system design

considering the scale of unmanned aerial vehicle.

1.6 Limitations

The aerial vehicle has physical limitation depending on its mission profile. Also, the hardware, which is used during the study, has surely limitations (e.g., CPU capacity, speed, data processing, buffer size).

This thesis proposes a framework to develop safe software system for MUAS and creates a prototype as an end product. The prototype only realizes the requirements which are determined by the thesis and obey the limitations provided in the case definitions.

1.7 Terms And Definitions

Unmanned Aerial Vehicle (UAV) is an aircraft with no pilot on board. UAVs can be remote controlled aircraft (e.g. flown by a pilot at a ground control station) or can fly autonomously based on pre-programmed flight plans or more complex dynamic automation systems.

Mini Unmanned Aerial Vehicle is an unmanned aerial vehicle small enough to be man-portable.

Unmanned Aerial System is a term which is used by U.S Federal Aviation Administartion (FAA) to reflect the fact that complex UAV systems include ground stations and other elements besides the actual air vehicles. Officially, the term 'Unmanned Aerial Vehicle' was changed to 'Unmanned Aircraft System' to reflect the fact that these complex systems include ground stations and other elements besides the actual air vehicles.

Airworthiness is a demonstrated capability of an aircraft (e.g., unmanned Aircraft system) or aircraft subsystem or component (including software) to function satisfactorily when used a maintained within prescribed limits (AR70-62).

Hardware-In-The-Loop Simulation (HIL) is a technique that is used in the development and test of complex real-time embedded systems. HIL simulation provides an effective platform by adding the complexity of the plant under control to the test platform. The complexity of the plant under control is included in test and development by adding a mathematical representation of all related dynamic systems. These mathematical representations are referred to as the "plant simulation". The embedded system to be tested interacts with this plant simulation.

1.8 Thesis Organization

In this chapter, brief introduction about the thesis study is given. Moreover, purpose of the study is presented as developing a comprehensive framework to achieve software system safety for MUASs which accomplishes design, coding and testing processes of the entire development life cycle. In order to achieve goals of the study, in Chapter 2 related studies and works in literature are synthesized. In chapter 3, software airworthiness concept is introduced. Related terms and procedures are explained. Contribution of software system safety into an entire system is discussed. Explanation of the framework's architecture together with its detailed features is written in Chapter 4. Assassin Process Method, APM for testing process is presented and its unique contributions are stated in detail. In Chapter 5, a case for MUAS's AFCS is derived and the framework is applied to the case as a prototype. All the phases of the case are explained, entire system definitions are given and framework implementation steps are expressed. In the mean time the proof of concept is accomplished. In the last Chapter 6, the conclusion of the thesis is declared and the works and results are summarized.

CHAPTER 2

LITERATURE SURVEY

2.1 Synthesize Of Literature Survey

2.1.1 Generic MUAS And Autonomy Concepts

In recent years, increasing demand on using MUAS in the real life for both civilian and military purposes causes a significant growing effort for the researches on autonomy[6]. Together with the great advantages in materials, sensor technology, data processing hardware, propulsion system and software techniques have made a UAV design highly feasible[13, 9, 19].

Owing to the increasing demand on the MUAVs, the major part of the system, which is autonomy, becomes the growing area of research in the aerospace field recently [2]. Autopilot design using an onboard hardware and software together with COTS sensor, data fusion and control algorithms is one of the common accurate ways followed in the field[2]. The major requirement during the autopilot design following the onboard system architecture is that handling the complexity of real-time processes due to the series of concurrent tasks. In order to guarantee all the autopilot tasks complete within predictable time duration and every process exactly perform its mission; autopilot software is implemented in real-time[14].

The development of autonomous UAVs for real-world applications is a challenging area of research in recent years. Issues caused by the critical requirements of the real-time operations on multiple tasks for a successful onboard software design for UASs can be carried out by using RTOS. Hong et al. states that using RTOS has significant

impact on the performance of the autonomous flight. In their study, autopilot software uses Linux as an operating system and RT-Linux as a RTOS in order to compare their real time performances. When the results of the timing performances obtained by the standard Linux and RT-Linux, it is obvious that RT-Linux has better performance than standard one. Another study concludes that real-time operating system (RTOS) satisfies all the requirements to have stable, strictly real-time capable, easy-to-modify and secure embedded real-time application for a UAV[22]. The implementation of control and navigation solutions using the RTOS in the autopilot for UAVs is an efficient way for stable, reliable and robust autopilot systems.

[22] have a PID controller in their autopilot design. The onboard avionics system is composed of an enclosure that is used to contain the onboard hardware, a PC/104 computer that handles the I/O signals and data processing, an inertial measurement unit (IMU), GPS, sonar altimeter that obtains the height form the ground, a telemetry system for data observations on the ground, and electric power system. Moreover, the onboard software is developed under the RTOS called QNX Neutrino with the capabilities of multitasking, threads, rapid context switching, and preemptive scheduling. Using this configuration, successful results of the implementation of advanced and sophisticated control and navigation algorithms for UAV system are reached.

The control algorithm running concurrently on the RTOS based onboard software can be designed by following the classical controller design methodology[15]. In the study, general UAV architecture and autopilot design is explained. In autopilot design, a PID controller is used together with the 3-state Kalman Filter and Extended Kalman Filter for compensation of the low data rate of GPS data. Kahraman (2010) uses PID controller and attitude heading reference system (AHRS) as the navigation mode.

2.1.2 Software System Safety Concept

Apart from the above mentioned engineering solutions and studies in academy which focuses only on realization of autonomy concepts for small UASs, another real-life related paradigm, system safety and assurance, is started to become very important, especially after operations on national flight zones are initiated. It is obvious that due

to the demand for software-controlled systems such that unmanned aerial systems, space crafts, nuclear plants etc. are undeniably higher and their critical software control functionality starts to dominate entire systems. As a fact that safety concerns caused by the safety-critical software are started to be investigated for their effects in case accidents occur. It is obviously critical that any failure in a safety critical component might cause catastrophic results, deaths and injuries[12]. As a result of the safety concerns, to achieve an acceptable level of safety for software systems used in critical applications, software safety engineering must be emphasized early in the requirements definition and system conceptual design process. Safety-significant software must then receive continuous emphasis from management and a continuing integrated engineering analysis and testing process throughout the development and operational lifecycles of the system[1].

In order to develop software systems with a level of confidence, numerous directives, regulations, standards or guidelines such that D0-178B, MIL-STD-498, MIL-STD-882, MIL-STD-2167A, IEEE/EIA-12207, IEC 61508, and U.K . Defenses Standard 0-5 are introduced by authorities. Especially for civil aviation, DO-178B is the most famous one and it is more comprehensive than the others. Although DO-178B provides guidance for the production of software for airborne systems and equipment such that there is a level of confidence in the correct functioning of that software in compliance with airworthiness requirements, it does not discuss specific development methodologies or management activities[10]. Further, it does not provide a complete description of the system life cycle processes, including the system safety assessment and validation processes or aircraft and engine certification processes, nor does it cover operational aspects of software[20]. In common, all these guidelines, standards or regulations introduce a life cycle for safe software system development and they follow some milestones on that life cycle in order to achieve an assurance level. These life cycles generally include the planning process, the software development process (requirements, design, coding and integration) and the integral processes (verification, configuration management, software quality assurance, and certification liaison). DO-178B defines objectives for each of these processes as well as outlining a set of activities for meeting the objectives additionally; the software life-cycle processes and transition criteria between life-cycle processes in a generic sense without specifying

any particular life-cycle model are discussed in it. Finally, it has to be emphasized that DO-178B objectives do not directly deal with safety. Safety is dealt with at the system level via the system safety assessment. DO-178B objectives help to verify the correct implementation of safety-related requirements that flow from the system safety assessment. Like any standard, DO-178B has good points and bad points (and even a few errors) [10].

When software development process is considered under the safety aspects, using mentioned guidelines with traditional approach as done for manned aircrafts and large UAV systems is time consuming, resource intensive so that not productive owing to nature of synthesize, implementation and validation objectives [16]. While airworthiness practices for large UAS may be similar to manned aircraft, it is clear that small UAS require a paradigm shift from the airworthiness practices of manned aircraft [4].

Software development life cycle defined by well-known guidelines with four processes of requirement, design, coding and testing is described briefly in the guidelines and with generic words since they tend to vary substantially between various development methodologies. Especially the testing and verification processes accounting for more than half of the entire life cycle [10] is the bottleneck for the MUAS development. In literature for MUAS developments and prototypes, verification of the autopilot can be done in three major ways. First one is software simulation [23]; next one is the hardware-in-the-loop (HIL simulation) and the last one is real flight test that uses the platform and onboard autopilot together as a whole system [2, 3]. However the purpose of these tests is commonly to ensure that implemented theories are correct for that simulation rather than verifying the entire software product performs its intended function and does not demonstrate any unintended actions.

2.1.3 RealTime Simulation Environment

As a test system, HIL simulation is widely used for autopilot researches. In the study [2], it is indicated that HIL simulation decreases the cost of developing FCS and rapid development of the FCS software. Due to the study, MUAV systems testing are very expensive and involve risk of many crashes prior to the successful development of the autopilot system. During the flight tests of an autopilot, there is a high risk of

failure caused by the autopilot controller responses. In order to avoid these harmful conditions, safer real-time simulations can be used. Rapid prototyping requires real model or plant that is tested using simulated control. [2] proposes two major parts for the HIL simulation architecture which are simulated aircraft model developed using Aerosim blockset and dSpace real time system. The aircraft model is developed using Simulink in Matlab and dSpace is compatible with Simulink. The MPC 555 is used as an embedded controller in the autopilot design and it interfaces with the dSpace. It behaves like a virtual aircraft. The simulation results are expressed on the ground control station (GCS) by transmitting the data via wireless network.

2.2 Conclusion

In literature, numerous studies about MUAS development, their autonomy and experimental prototypes can be found. Especially, as a trendy topic, implementation of autonomous control and flight of the MUAS is studied many times and common results which state software for such autonomous system requires real-time behavior due to the critical importance at the data processing for reliable, robust and successful autopilot system are published. Many control theory implementations for different kinds of hardware-software combinations are discussed with a great enthusiasm. In order to handle real-timing issues caused by the processing loads or hardware/-software constraints or resource limitations, RTOS are investigated and as a result accepted that they provide good performances for such safe real-time requirements for autopilot systems development.

Considering the growing population of MUASs and their increasing tendency to be integrated into the national flight zones, safety of these systems started to be questioned. At that point, spontaneously, traditional standards, directives or guidelines used in the development of manned aircrafts and large UAVs realization such that DO-178B are coming to stage but due to their excessive and time consuming objectives to be reached, applying these processes to a MUAS development seems not practical and assessed inefficient. Moreover, being generic conceptual guidelines with milestones and state-of-art procedures, DO-178B and likely standards are not practiced for MUAS and their feasibility for such systems is suspected.

As software development life cycle with four processes of requirements, design, coding and testing, MUAS development requires a paradigm shift from traditional generic life cycles to small system development life cycles with easy, feasible, systematic and comprehensive approach. However, neither in literature nor in industry developing a safe small unmanned aerial system with safety constraints by following a systematic way is not studied, experimented and discussed satisfactorily. Being a challenging part, testing and verification of the software systems does not being performed under safety approach systematically, instead HIL simulation or flight testing are done to prove that system is realized rather done trying to ensure that system is developed in a safe way.

In conclusion, one by one most of the sub parts of MUASs are being investigated and studied both in academia and industry but safety seems to be underestimated in common. It is crystal clear that as an engineering process, developing MUASs with a safety compliance level requires systematic ways which can efficiently satisfy safe development requirements and without devastating small unmanned aerial system development paradigm.

CHAPTER 3

SOFTWARE SYSTEM SAFETY CONCEPT

In this chapter, a different aspect of UAS development life cycle which is software system safety will be discussed. This term relies on the concept of safety and in general indicates that the airworthy UAS software failures are predictable and preventable. Additionally for an airborne system, failures will not be expected to have hazard effect as they occur. Being a very generic term, safety, embodies specific procedures to follow and terminology to identify recent system status. This special concept of software airworthiness, becomes more important as the aerospace technology proceeds and more complex software-integrated systems start expected to flight in every field including civil areas. Besides, the focused design question changes from "what is designed" to "how was designed" to have safe systems as a whole. Within this chapter, related terminology and concept will be introduced in detail.

3.1 Overview Of Software System Safety Concept

Software airworthiness depends on the works done for satisfying Software System Safety (SSS). Formal definition of the airworthiness by (AR-70-62) is

A Demonstrated capability of an aircraft (e.g, Unmanned Aircraft System) or aircraft subsystem or component (including software) to function satisfactorily when used an maintained within prescribed limits.

Or by (ARP 4754A) is

The condition of an aircraft, aircraft system, or component in which it operates in a

safe manner to accomplish its intended function.

Unmanned aerial systems are composed of several subsystem components all of which has different safety expectations one by one or after their integration. Apart from unmanned aerial systems, all kinds of applications which require special attention as their failures cause death, serious injury or huge economical loss, safety concept should be applied. Hence the software systems safety concept has a deep background coming from real life issues and experiences, theoretical implications and trial-and-error experiments.

Considering the increasing demand for complex autonomous systems, the importance of software and firmware logic continue to play a significant and evolutionary role in the operations and control of systems which need to be validated for an acceptable safety level. Within 25 years, the expectation of engineers to have human control for system component, which can cause hazard at operation has diminished due to the fact that SSs are capable of handling such safety critical components reliably at speeds unmatched by human operator. Besides, reliability and acceptable operation time, and other factors such as increased versatility, higher performance capability, greater efficiency, increased network interoperability, and decreased lifecycle cost increases the usability of software systems into the safety critical systems [1].

Since the software components with high criticality levels integrated into more systems, additional software components which monitors and diagnoses the entire system, are introduced. This fact, cause an extraordinar growth in the software functionality and complexity. Furthermore, as the systems become more depended to software functionality, software specification errors, flaws in designs and algorithms, lack of software system safety requirements and software implementation errors become significantly important for entire system safety.

Another way looking, although software development part is cheaper than that of hardware, agile technological improvements on processors field of art, requirements of the SS can be changed easily and the expected functionality of the software might be enlarge significantly by increasing software components' complexity and cost drastically. Moreover, not only the cost but also the SSS risks might ascend dramatically causing a serious doubt in verification authority.

Considering the UAS within the scope of this study, it is definite that UAS will completely rely on the software that will try to successfully overcome the hard-real time processing's requirements accurately. At that point, it is important to express that embedded systems differ from desktop systems with their quality of service (QoS) aspects. In real-time embedded systems, for example, whole system failure can be easily trigged by just missing a single processing deadline. Moreover, safety and reliability requirements have significant importance on such systems.

Despite the seriousness and severity of safety and reliability requirements, design and implementation processes of those terms are not a part of normal undergraduate and graduate level curriculum. Moreover, most engineers fail to apply correct terms to safety and reliability engineering processes and also they don't even know the correct terms to use[7].

3.1.1 Safety Terms And Generic Concepts

Safety is a system property, a system issue! [1, 7].

In formal definition of (MIL-STD-882C) **safety** is freedom from those conditions that can cause death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment. A **safe system** is one that does not occur too much risk to persons or equipments[7]. The term risk is a key parameter to identify safety of a system.

[7] defines **risk** to be

a combination of the likelihood of an accident and the severity of the potential consequences another way of saying the chance that something bad will happen.

In safety terminology **"something bad"** is named as mishap or accident. **Mishap** is an unplanned event or series of events resulting in death, injury, occupational illness, damage to or loss of equipment or property, or damage to the environment (MIL-STD-882C). In addition to mishap; hazard is also a very important term used in the process of designing a system which considers safety requirements. A **hazard** is

A state or set of conditions of a system (or an object) that, together with other conditions in the environment of the system (or object) will inevitably leads to an accident (loss event).

Failures cause accidents to occur and prevent system to achieve its intended function within its performance constraints. **Errors** on the other hand are static conditions which take place at the product life cycle and are inherent characteristics of the system. In general failure states and errors are different kinds of **faults** which are unsatisfactory system condition or state (Safety critical systems design).

There two other terms most commonly misused or confused which are reliability and **security**. First of all, the common mistake in using **reliability** term is placing it as a synonym of safety. Another saying, **"safety is not reliability"**. Reliability is

a measure of the up time, or availability, of a system – specifically, it is the probability that a computation will successfully complete before the system fails[7].

If a system is defined as reliable it does not fail frequently, but makes no guarantees what happens should it fail. Failing frequently, a system can be assessed as a safe system by failing in a safe way without causing any mishap. On the other hand, a system can have high reliability by running all the time, but it can consistently put people at risk meaning so be reliable but not very safe.

[7] defines **security** as

Security deals with permitting and denying system access to appropriate individuals. A secure system is one that is relatively immune to attempts, intentional or not, to violate the security barriers set into place.

Before starting to design of an aircraft which is intended to be a safe system; **safety-critical** term should also be introduced. During the **life cycle** –all phases of system's life, including design, research, development, test and evaluation, production, deployment (inventory), operations and support, and disposal (MIL-STD-882C) - of UAS defining safety-critical parts of the system has a significant importance on system safety. **Safety-critical**

is a term applied to any condition, event, operation, process or item whose proper recognition, control, performance or tolerance is essential to safe system operation and support (e.g., safety-critical function, safety-critical path, or safety-critical component.)

During the life cycle of a system design, redundant components (e.g., software, hardware) can be used. **Redundancy** is defined by [17] as

Provision of additional functional capability (hardware and associated softwware) to provide at least two means of performing the same task.

As definition endorses redundant components can increase the operation time which means an increase in reliability. However, when defining safety and reliability, it is explained that reliability does not mean that safety or vice a versa. Additionally, being a common way to reduce possible single point failures, redundancy puts extra complexity into fault-tolerant systems which may make systems more vulnerable to additional failure modes that requires extra attention by developers[1]. So that, intuitively redundancy is also not same with safety, it is only one contributing factor.

The entire system can be safe or not. Not the software, not the electronics, not the mechanics can be safe. Stated in this way, safety is a system issue. It is undeniable that each of these components has an impact of entire system's safety as well as their integration with each other does.

3.1.2 Software System Safety: Purpose And Methodology

Software system safety can be achieved by applying well defined milestones during the SSS process by hand shaking the authority. In previous statement, two words are safety-critical for understanding the SSS concept. First one is "process" which identifies SSS as a time taking activity. Second one is "authority" meaning that all actions taken during the life cycle should be assessed by someone else and incidentally he/she/it will have an authorization for approval/rejection.

This SSS process is to reduce likelihood or severity of system hazards exposed by poorly specified, designed, developed, or operated software in safety-critical applications[1].

Moreover technological improvements, changes in systems specifications, additional functionality injections and risk assessment are all major operational fields of SSS process.

3.1.2.1 Authority

Operations of systems (especially the aerial vehicle systems) depend on the authorization of the appropriate authority. Considering the operational field and scope of it, different authorities can enforce the system to obey different kinds of standards, directives, regulations, and regulatory guides.

Authorities determine the safety levels and must-have/do requirements of systems, subsystems or components in order to achieve desired non hazardous operations to complete.

National/international governmental organizations and national/international nongovernmental authorities may provide standards for systems that have different kinds of operational functionalities as well as different kinds of safety and assessment expectations.

Due to the fact that this thesis study relies on the aerial vehicle systems, military/ civilian national/international aerospace/aeronautics authorities are the driving forces for determination of certifying authority.

Considering the focus of this thesis study on software airworthiness; military standards MIL-STD 882D, or NASA-STD-8719-13B, or other governmental standard DO-178B can be accepted as an authority for having certified UAS with airworthy software component.

3.1.2.2 Interdisciplinary Team Work

Software safety (software airworthiness in the scope of this study) is an exact product of a process which is sustained by coordinated teams working together. It is apparent that software, safety and system engineering are vital players of the SSS team[1].

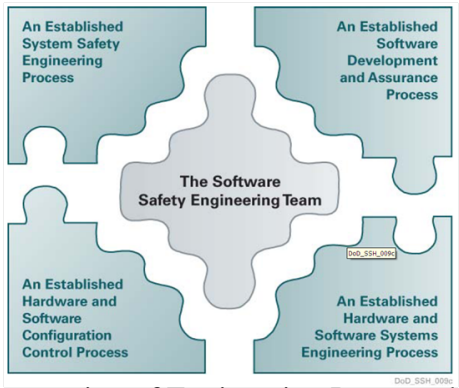


Figure 3.1: Integration of Engineering Personnel and Processes

Neither safety engineers nor software engineers can accomplish an expected safety level by themselves. Each team member must be aware of the importance of each discipline's functionality and their duties, responsibilities and tasks. Moreover, the life cycle of project, commitments, meetings, reviews and analysis process are also the facts that each member of SSS must be aware of Figure 3.1.

Responsibilities of each team member can be determined considering the culture and structure of the organization and duty assignment should be performed depending on the capability and experience of the person by management people. Definitions of responsibilities of different disciplines can be made specific to the organization or predefined instructions can be used as described in [17].

Within the above mentioned organizational structure, Department of Defense, USA states that a successful and credible SSS engineering program will include:

- A defined and established system safety engineering process
- A structured and disciplined software development process
- An established hardware and software systems engineering process
- An established hardware and software configuration control process
- An established software assurance and integrity process for safety-critical software development and testing
- An established software system safety engineering hazard analysis process
- An integrated SSS team responsible for the identification, implementation, and verification of safety-specific requirements in the design and code of the software.

3.2 Accomplishing Software System Safety

Being subsystems of an entire system, software systems are exposed to the safety assessment processes to accomplish software system safety. Even if safety processes very well applied to software systems (SS), the integration of SS with other system components can treat the system safety. Always remembering that fact, not only safety of a component but also the safety of all components together must be exposed to the same safety assessments.

Restating the system safety as an elimination of hazard occurrence risk within a system, determination of hazard conditions and risk together with correlating those conditions with subsystem components and functions can be seen as a starting point of the system safety process. As well as determining hazard conditions for safety, understanding the causal factors for scary "hazard" conditions is also significant for a healthy safety assessment. With this perspective, the causal factors of hazards can be identified as below[1].

- Functional Hazard Causal Factors
- Interface-Related Hazard Causal Factors
- Zonal Hazard Causes
- Data Interfaces
- COTS
- Technology Issues

3.2.1 Effecting Factors of SSS

In earlier sections of the thesis, the motivation of the software safety was explained. In summary, it can be paraphrased that importance of software increases due to the expectations of people, governments or industries. As technological improvements appeared in different fields, autonomous, intelligent and real-time behaving systems are becoming more common so that more software depended. Because of that fact safety questions for software driven systems are being asked more frequently as well as being mandated to operate in a safe way by governmental authorities. Under these

circumstances, during the safety process, safe requirements can be derived by considering the affecting factors of safe software in order to be aware of the possible hazards and to eliminate the risks. [17] categories the factor affecting the software safety as listed below:

Degree Of Control: *The degree of control that the software exercises over safety-critical functions in the system.*

In case software itself is responsible for controlling or monitoring software-critical services, or providing failure prevention services, or taking automatic precautions, it must be paid extra attention and safety resources together with detailed assessments must be ensured.

Complexity: *The complexity of the software system. Greater complexity increases the chances of errors.*

Complexity increases with the increasing number of;

- safety related software requirements for hazards control,
- Subsystems controlled,
- Interacting, parallel executing processes,
- Logical Operations.

Timing Criticality: *timing criticality of hazardous control actions.*

If a software system exceeds the time which is required to prevent hazard situation, then it causes hazard. That time constraint is determined by the requirements of entire system and hazard analysis.

3.2.2 General Rules For Safer Software

Software system safety cannot be reached by itself, another saying it cannot just happen. It is a team work of software engineers, safety engineer and other system related people. The reason why there is much emphasis on "team work" is that, in order to satisfy safety requirements; the strong need for cooperation between software engineers and other disciplined-engineers must be well understood and never be underestimated. Additionally, software system safety is a different aspect of entire system

safety, so that people with safety background must consider the software design processes and methodologies as well as the software developers who are unfamiliar with the safety-critical systems.

The anticipated safe software system can be obtained by applying five rules given by [17]. To have safe software systems;

1. Communicate
2. Have and Follow Good Software Engineering Practices and Procedures
3. Perform Safety and Development Analyses
4. Incorporate Appropriate Software Development Methodologies, Techniques And Design Features
5. Caveat Emptor

3.2.2.1 Incorporate Appropriate Software Development Methodologies, Techniques And Design Features

With many methodologies, techniques and design features, software development is a complete process and considering the safety critical expectations and entire software system requirements, appropriate approaches must be selected and implemented. Although it is not feasible to list all software methodologies, techniques and design features, some available and appropriate of them can be listed as below Figure 3.2 and detailed explanations of them can be obtained from software engineers, people from software profession or software literature survey.

Software Lifecycles	Software Configuration Management	Tool and Operating System Selections	Interface Design/Human Factors
Design Methodologies	Programming for Safety	Coding Checklists, Standards, and Language restrictions	Integrating COTS Software
Design Patterns	Language Selections	Defensive Programming	Refactoring

Figure 3.2: Example To Incorporate Appropriate Software Development Methodologies, Techniques and Design Features

3.2.3 Road Map To Software System Safety

With a strong emphasis on difficulty of software safety process, constructing a safe software system can be broken up multistep plan [7].

1. Identify the hazards
2. Determine the risks
3. Define the safety measures
4. Create safety requirements
5. Create safe design
6. Implement safety
7. Assure the safety process
8. Test, test, test

This roughly broken up multistep plan, lists the general conceptual process steps and can be seen as a complete approach to reach safer software systems.

3.2.3.1 Determine The Risks

There are several factors to determine risks in systems. Before telling about these factors, it should be recalled that determination of risk depends on the system and environment that the system will operate. Coming back to the determination of risks, the factors affecting the risk can be identified as;

- Severity of risk
- Number of times that damage might occur
- Potential for personal injury or environmental damage

Different authorities define risks within their scope of interest and state the severity and occurrence probability of the hazard conditions with different levels implying different safety constraints. Besides, they put different safety-criticality levels (indexes or risk indexes) to the hazard conditions.

Severity Categories Hazard and related mishaps are elementary factors for determination of hazard severity. In addition to these two factors, severities of damage and injury are also significant factors for hazard severity categories.

Within the concept of software airworthiness for UAS, military and non military governmental agencies are in charge. In this thesis MIL-SDT-882 is a base and the severity category for software safety is suggested as in the Table 3.1.

Table 3.1: Mishap Severity Categories. - Suggested by MIL-STD 882

Description	Category	Environmental, Safety, and Health Result Criteria
Catastrophic	I	Could result in death, permanent total disability, loss exceeding $1M, or irreversible severe environmental damage that violates law or regulation.
Critical	II	Could result in permanent partial disability, injuries or occupational illness that may result in hospitalization of at least three personnel, loss exceeding $200K but less than $1M, or reversible environmental damage causing a violation of law or regulation.
Marginal	III	Could result in injury or occupational illness resulting in one or more lost work days(s), loss exceeding $10K but less than $200K, or mitigatible environmental damage without violation of law or regulation where restoration activities can be accomplished.
Negligible	IV	Could result in injury or illness not resulting in a lost work day, loss exceeding $2K but less than $10K, or minimal environmental damage not violating law or regulation.

Probability Categories Identification of probability of occurrence is another significant factor for determination of risk. Being a statistical data, it cannot always be possible to identify probabilistic result from system components. However in aerial systems MIL-STD 882 categorizes is as in the Table 3.2

Hazard Risk Index As a combination of hazard severity and probability concepts, system risk index (for software airworthiness: safety critical index-SCI) can be used to have hazard risk assessment. SCI helps SSS teams to understand same meanings,

Table 3.2: Mishap Probability Levels. Suggested by MIL-STD 882

Description	Level	Specific Individual Item	Fleet or Inventory
Frequent	A	Likely to occur often in the life of an item, with a probability of occurrence greater than 10-1 in that life.	Continuously experienced.
Probable	B	Will occur several times in the life of an item, with a probability of occurrence less than 10-1 but greater than 10-2 in that life.	Will occur frequently.
Occasional	C	Likely to occur sometime in the life of an item, with a probability of occurrence less than 10-2 but greater than 10-3 in that life.	Will occur several times.
Remote	D	Unlikely but possible to occur in the life of an item, with a probability of occurrence less than 10-3 but greater than 10-6 in that life.	Unlikely, but can reasonably be expected to occur.

consider same hazards with same mishaps and evaluate same software components within the same criticality level. Besides, it helps safety people to develop and evaluate software components with high SCI more rigorously.

Under these definitions and concepts risk assessment suggested in MIL-STD-882 can be visualized as in the matrix given in Figure 3.3.

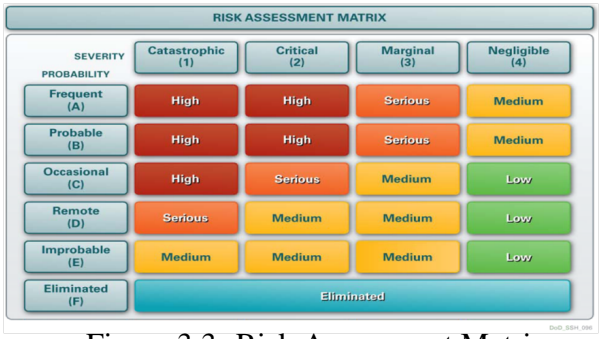
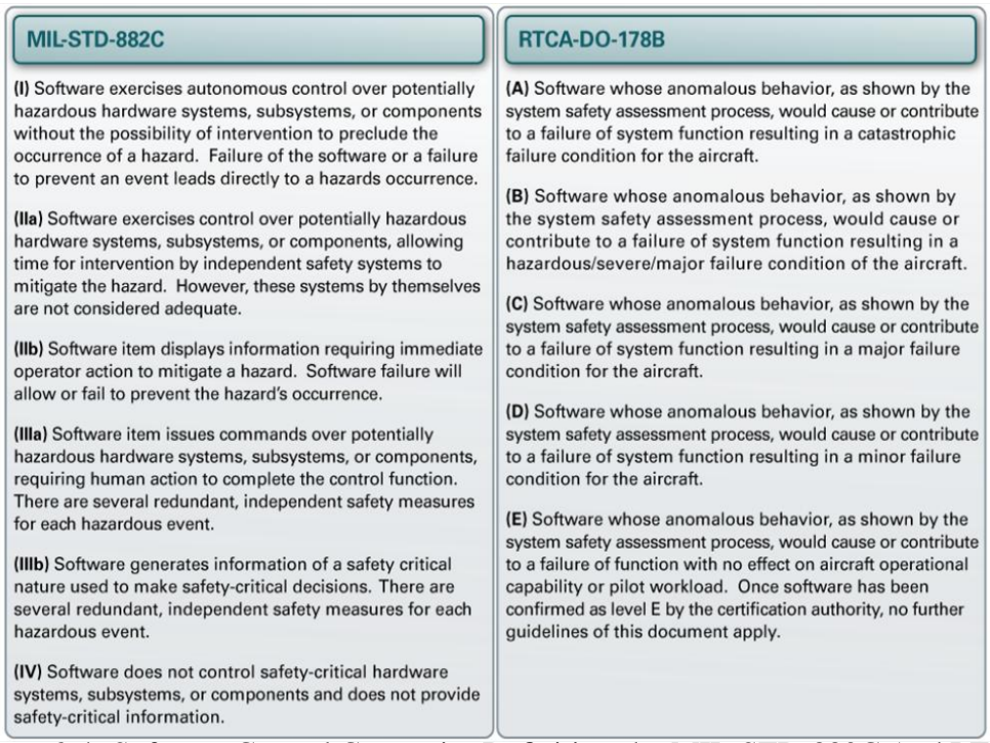


Figure 3.3: Risk Assessment Matri

In addition to risk assessment, identifying Safety Integrity Level (SIL) to subsystem components can help assessment and development of the system by determining level-of-rigor of software in the entire system. There are several SIL-type software assurance approaches. The most used approaches used in aerospace are MIL-STD-882C and RTCA DO-178B. The purpose of these approaches is “to assess the severity

of the systems safety-significant functions and the software's control capability in the context of the software's ability to implement the functions" [1] With the help of software control categories (SCC), safety-significant software functions can be labeled. After labeling software functions with respect to their control categories, during the software design, development, test and verification processes required attention, and assessment can be applied to the software. For a healthy SSS, entire team must review the SCC to meet the objectives of the software safety. The SCC definitions of MIL-STD and RTCA DO-178B are as given Figure 3.4.



MIL-STD-882C	RTCA-DO-178B
(I) Software exercises autonomous control over potentially hazardous hardware systems, subsystems, or components without the possibility of intervention to preclude the occurrence of a hazard. Failure of the software or a failure to prevent an event leads directly to a hazards occurrence.	**(A)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a catastrophic failure condition for the aircraft.
(IIa) Software exercises control over potentially hazardous hardware systems, subsystems, or components, allowing time for intervention by independent safety systems to mitigate the hazard. However, these systems by themselves are not considered adequate.	**(B)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a hazardous/severe/major failure condition of the aircraft.
(IIb) Software item displays information requiring immediate operator action to mitigate a hazard. Software failure will allow or fail to prevent the hazard's occurrence.	**(C)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a major failure condition for the aircraft.
(IIIa) Software item issues commands over potentially hazardous hardware systems, subsystems, or components, requiring human action to complete the control function. There are several redundant, independent safety measures for each hazardous event.	**(D)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of system function resulting in a minor failure condition for the aircraft.
(IIIb) Software generates information of a safety critical nature used to make safety-critical decisions. There are several redundant, independent safety measures for each hazardous event.	**(E)** Software whose anomalous behavior, as shown by the system safety assessment process, would cause or contribute to a failure of function with no effect on aircraft operational capability or pilot workload. Once software has been confirmed as level E by the certification authority, no further guidelines of this document apply.
(IV) Software does not control safety-critical hardware systems, subsystems, or components and does not provide safety-critical information.	

Figure 3.4: Software Control Categories Definitions by MIL-STD-882C And RTCA-DO-178B

CHAPTER 4

ARCHITECTURE OF SAFE SOFTWARE SYSTEM DEVELOPMENT FRAMEWORK

4.1 Introduction

Being complex software controlled systems; MUASs started to increase their existence rate in almost every operational field drastically. As a result, their quality and safety are being questioned more frequently during development process by authorities, end-users, developers and public.

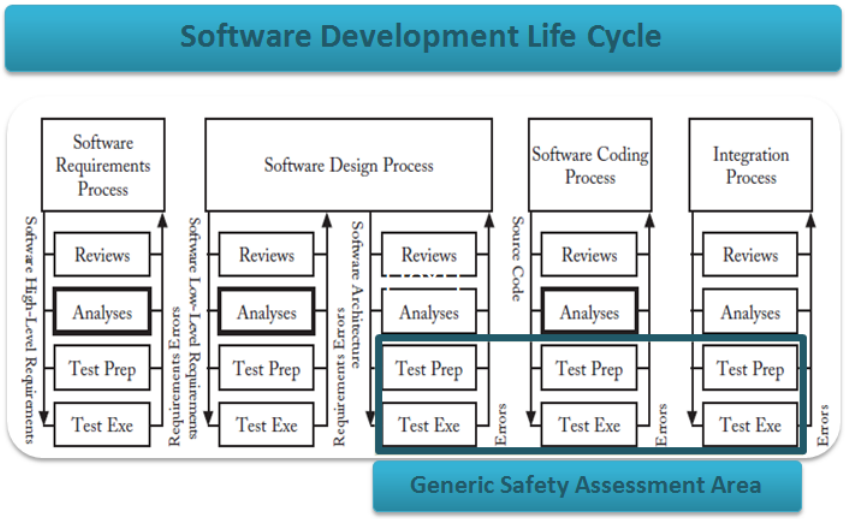


Figure 4.1: Software Development Life Cycle

In order to achieve acceptable safety level of confidence for software system, system development processes are mandated by authorities to follow some guidelines during development life cycle. The strongest point of such guidelines is verification with combination of three tasks of review, analyses and test as shown in Figure 4.1. The

most famous guideline for aviation community, DO-178B, defines these three tasks as [21].

- Review: Provide a qualitative assessment of correctness.
- Analyses: Provide repeatable evidence of correctness
- Test:
  - ✓ Demonstrate that the software satisfy its high level and low level assessments,
  - ✓ Gives confidence,
  - ✓ Acknowledges that test preparation can be as effective as test execution.

When such software system safety concepts are taken into consideration, general tendency is to follow well-known standards, which are widely applied to manned or large UASs mostly by industry as state-of-art approach. However, for MUAS development, they are impractical to be followed due to their being generic software development process guidelines and requiring excessive time and resources which definitely conflicts with the MUASs development paradigm. That conflicting situation between existing safety directives' development life cycle shown in Figure 4.1 and small UAS development life cycle, enforces proposal of an easy and comprehensive solution to reduce risks, achieve safety to a compliance level without overburdening MUASs development philosophy and preserve essence of these guidelines by accurately realizing their minimum objectives through sequential life cycle milestones.

4.1.1 Safe Software System Development Life Cycle Overview

As indicated in Figure 4.1, SSS development life cycle contains four internal processes such that Software Requirement, Software Design, Software Coding (Implementation) and Software Integration (Testing) processes.

With several tasks and documentations to accomplish, these processes enforce developers to fulfill all desired safety assurance level regulations at the milestones which stand at the end of each processes life time.

Although there is not exact to-do-list for task realization or recipe book for safe software system development in the literature, before proceeding to technical details of the SSSDF framework, important and generally anticipated set of milestones' objectives for design, coding and integration processes could be listed as below:

Design process of an entire safe software system development life cycle usually includes three milestones which are:

- Software Preliminary Design Review (PDR)
- Software Critical Design Review (CDR)
- Safety Review or other carrier- or program-specific system safety review

Coding process is expected to achieve below two objectives at the milestone as:

- Software code review or Formal Inspection
- Safety-Review
- Software code review or Formal Inspection
- Safety-Review

Integration process has

- Test readiness reviews
- Safety Verification Tracking Reports

All above mentioned milestone objectives can be achieved by fulfilling some tasks, which are appropriate to realize intended objectives. Additionally, considering to safety criticality level of software system, various combinations of these tasks can be configured to reduce risks and increase safety maturity.

In general, the tasks which are expected to be met by each processes can be listed in Figure 4.2 with red arrows directed from the process blocks. Remembering the fact that these task lists is not the only configuration possible or the best, guidelines has consensus that realizing such set of tasks increases system safety and reduces risks to reach feasible safety level. Hence, under the scope of this thesis they can be seen as the smallest set of logical and required tasks.

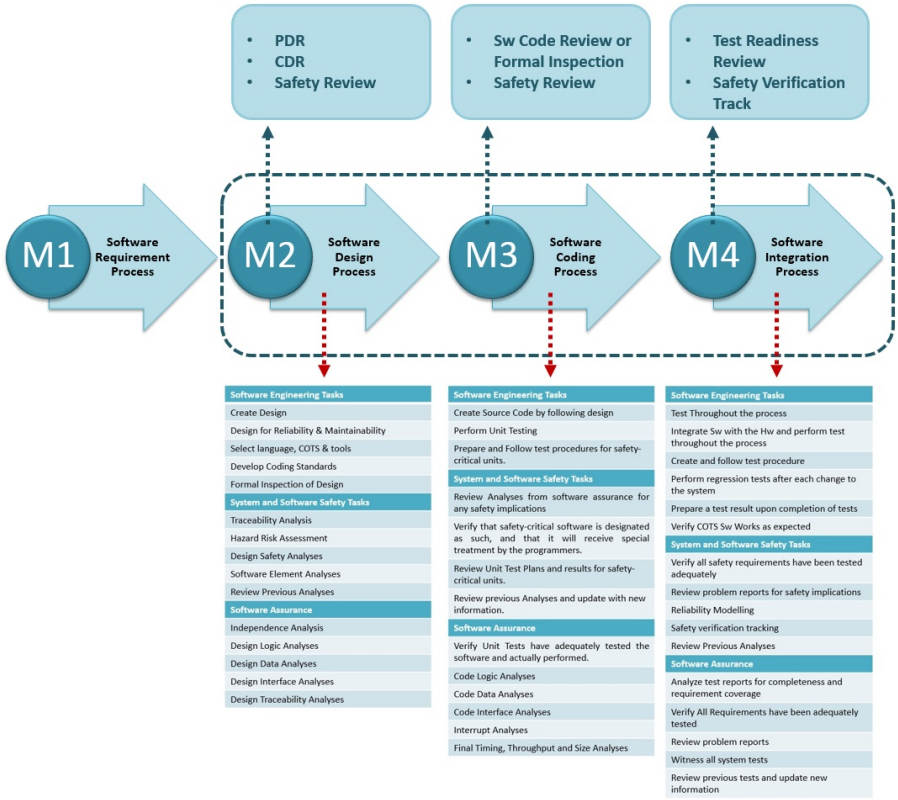


Figure 4.2: Safe Software Development Life-Cyle Milestones And Tasks

4.1.2 Summary

In this thesis study a safe software system (SSS) development framework, which can be used during the design, coding and integration (testing and verification) processes of DO-178B and likely guidelines, is introduced for a MUAS development and its contributions to the small UAS development life cycle is evaluated. Owing to reach a level of confidence for developed software, the SSSDF introduces an architecture, which covers all development processes and provides easy understandable, repeatable and simple methodology, to achieve safe system in an effective way. Furthermore, an intelligent, unique testing methodology, Assassin Process Method –APM, is developed to make testing, verification and assessment objectives of the software development processes easier, sustainable and automatic during small systems design life cycle.

4.2 Safe Software System Development Framework Overview

Considering the software development life cycle mentioned in the previous chapters, the framework proposed in this study tries to achieve design, coding and integration milestones by implementing design, coding, testing and assessment components as shown in Figure 4.3.

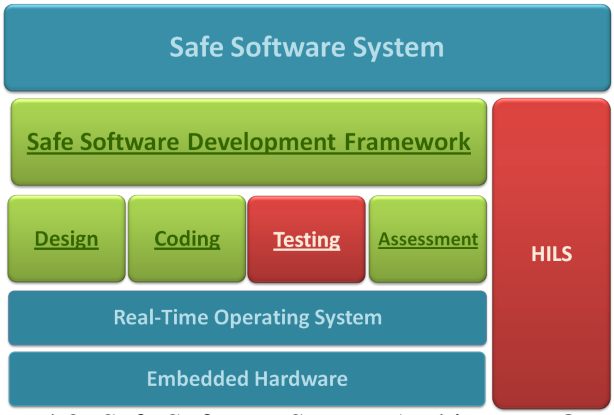


Figure 4.3: Safe Software System Architecture Overview

During MUAS software system development life time, SSSDF is constructed by using micro-kernel-based Real-time Operating Systems such that QNX Neutrino, Integrity etc. on top of a target embedded computer. Due to the modular, separated, flexible and compact architecture of the framework, design and coding objectives can be implemented in simple and effective way. Additionally, logical structure of the design together with power of high level coding and implementation of POSIX1003.1 compliance commands enables development teams to easily fulfill objectives related with process's review and data/structural flow assessments.

The first feature of framework to mention is all development activities except design which are coding and integration are done concurrently on the integration environment. Apart from design, coding and integration tasks, as a very challenging and discouraging activity, testing and its assessments can be performed with comprehensive, dedicated, mission-based and real-time Assassin Process Method(APM). For identified safety-related system and software requirements, test cases are created and erroneous software test scenarios (assassinations of APM) are injected into the entire integration environment by APM itself as indicated in Figure 4.4.

In this study, test cases follow the mission profile of the MUAS and every identified

safety-related requirements are tested at one and complete real-time simulation of MUAS's mission. Being a transparent testing method, APM randomly creates all test cases in real-time during simulation which is equal to the MUAS operation time until all sequential test cases(assassinations) are completed. Then, APM's Software System Safety Assessment GUI generates human understandable test assessment results for all APM's assassination missions with analytical representation. Hence, development teams can easily ensure that whether they tested and verified all safety-related requirements for a MUAS and they are safe for the safety conditions that they defined before real-flight. Therefore, traceability between high level system requirements, low level software requirements, test cases and source code, which are definitely regulations of design guidelines, become achieved in one simulation period of a MUAS.

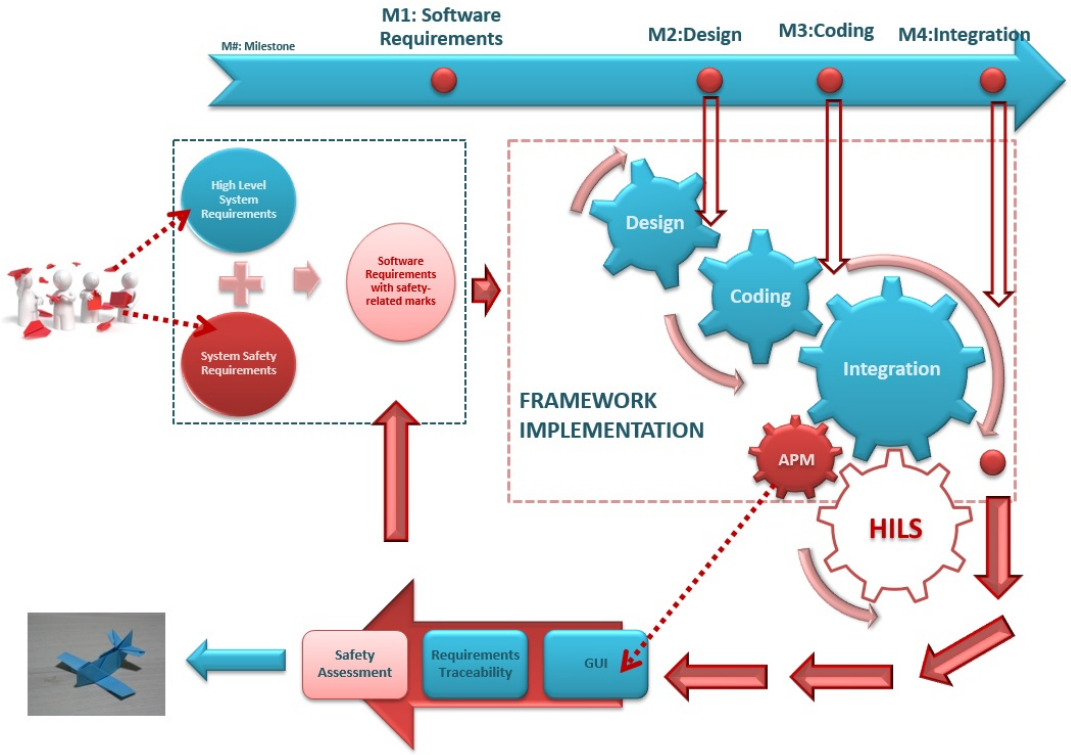


Figure 4.4: Safe Software Development Framework

Last but not least, in this study, propulsive enthusiasm to construct such framework is to test MUAS software system with test cases which are created from system high level requirements and software requirements focusing on safety related actions. Instead of constructing expensive and time taking requirement based testing team using complex tools, simple and comprehensive and safety oriented development paradigm

is generated. On the other hand, it is more than possible to cover every single requirement apart from safety related ones with APM approach because of its very flexible architecture. Meanwhile, although APM tool prioritizes tests of safety-related requirements, it is capable of being a requirement-based testing tool for MUAS's software system as well.

4.3 Framework Architecture

The safe software development framework focuses on design, coding and testing processes and their safety assurance. By that means framework proposes architecture for safe software development, especially for testing and verification. Moreover after-development quantitative assessments of requirements coverage review and assessment tasks are also done by framework automatically.

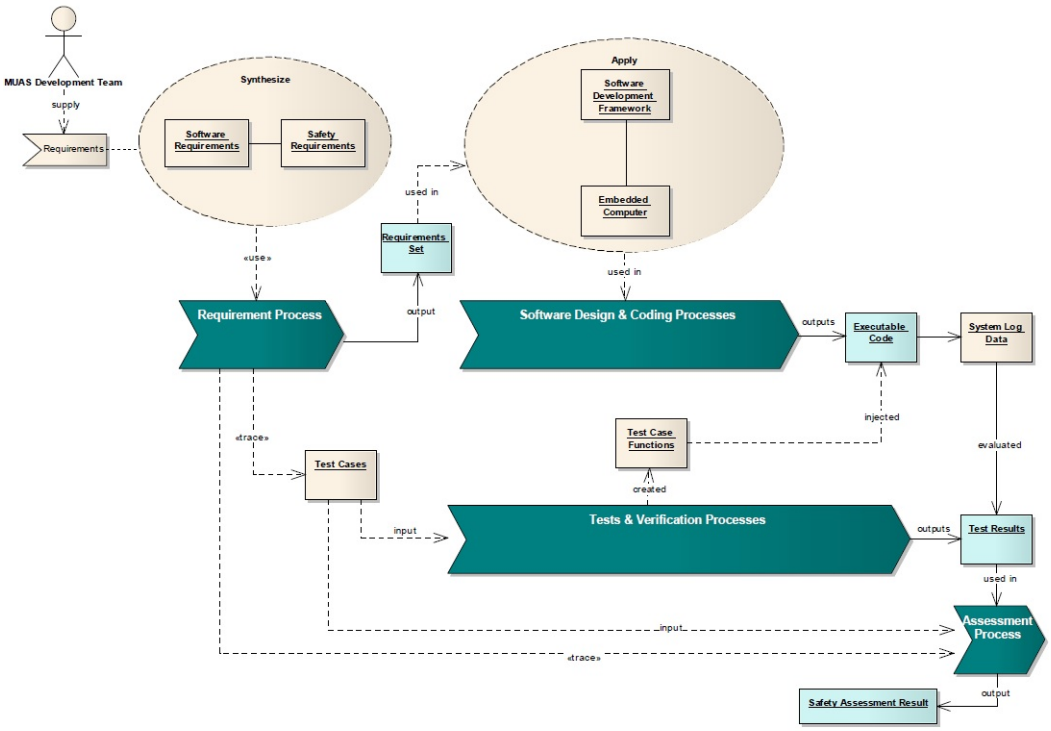


Figure 4.5: Safe Software Development Framework's Initial Architecture

Framework architecture can be easily traced at each development steps. As mostly desired features; elimination of single point failures and increase in reliability of MUAS software system become one of the major benefits of framework caused by its multi-

process implementation. Executable code generated at the end of coding process is tested in the integration environment for test cases derived from the software requirements by using APM automatically. Being an isolated testing process, APM is sustained to create fault condition in a transparent way, just like an assassination, during the real-time simulation of software at system integration environment. Software, which regularly logs it status during runtime with a simple rule-based logging mechanism, is assessed after the completion of APM's Assassination Mission by using GUI (Safe Software Assessment Tool) and all requirements which are traceable to the test cases are tested automatically for all identified failure conditions. As a result, overall percentage of the requirement coverage and corresponding safety percentile of the software system is presented in a human understandable way. Furthermore all these activities with various actions can be performed repeatedly at any time during the development process without spending much time and disturbing the development focus, motivation and workload [Figure - 5.10].

Detailed software framework architecture for each software development process is explained in the following sections and all features as well as contributions of the framework to safe MUASs are presented.

4.4 Framework For Software Design Processes

A typical software system for a MUAS has capabilities of coordinating data acquisition from sensors and interpretation of the data through flight control algorithms and controlling aerial vehicle control surfaces. Additionally software is responsible to keep track of that obtained data and observe system status to sustain system's reliability and robustness. During this thesis study, entire system development with the suggested framework is substantiated considering the mentioned capabilities of the MUAS assuming them as the minimum functionalities.

In order to have safe design, framework is generated for aiming for the below design capabilities which try to eliminate single point failures, achieve traceable design, isolate safety-critical regions from entire software components and equip with fault detection, identification as well as fault tolerance mechanisms.

4.4.1 Design For High Modularity

In this thesis, focuses are given on attaining not only mentioned capabilities but also safe software system development for MUASs. Owing to achieve desired safety level, the framework is constructed on multi-process software design pattern with strict process isolation to minimize risks due to single-point failures and increase software system confidence.

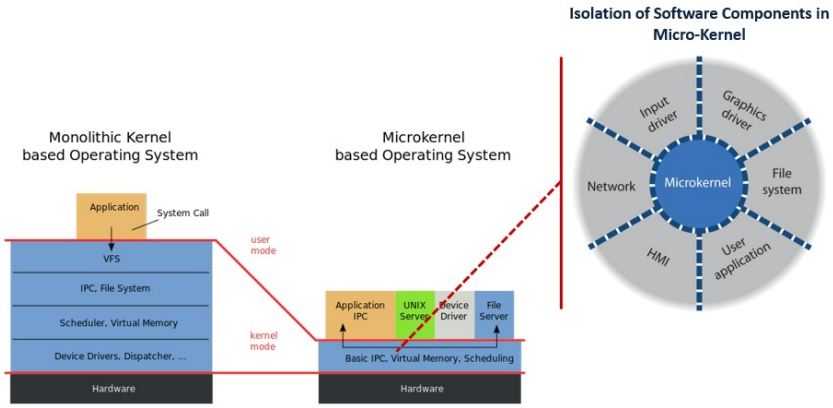


Figure 4.6: Micro-Kernel OS Architecture And Software Applications

In order to reach isolated software components at application level and eliminate risks to crash of entire system, micro-kernel OS [Figure 4.6] is employed as a base layer to the framework rather than monolithic kernel OSs.

Following simplest-is-the-best dialectic, all functionalities of MUAs software are assigned to a single process. Synchronous interoperability of the processes is achieved by coordinator process of data acquisition timer running periodically and message-passing mechanism of micro-kernel based realtime OS [Figure 4.7]. In detail, as message passing mechanism is applied to cooperating processes, execution of sequential components becomes synchronized by blocking/unblocking intended software components during runtime [Figure 4.8].

The SSSDF contains synchronous and asynchronous processes all of which are independent from each other and fully isolated. Considering MUASs, it is clear that directly flight related processes with sensor data interpretation capabilities are members of synchronous process flow and others with indirect/non-flight related functionalities are belonging to the asynchronous flow.

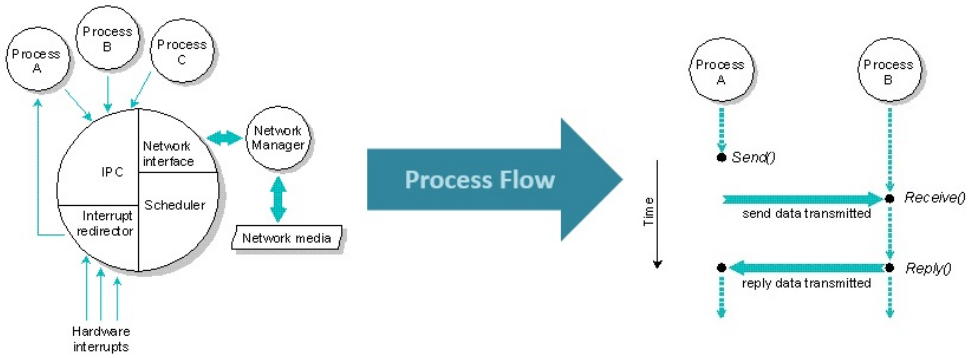


Figure 4.7: Inter Process Communication Overview

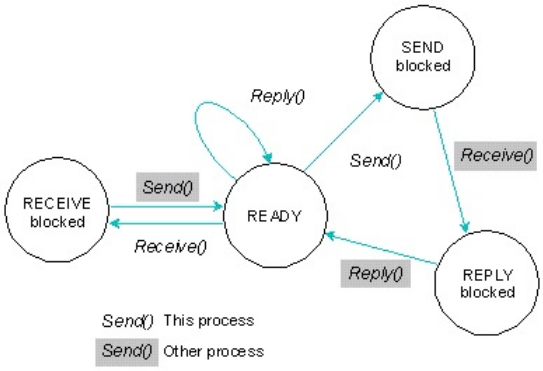


Figure 4.8: IPC Message Passing Mechanism

Due to the nature of control, navigation and guidance algorithms, periodic data processing has crucial importance for a safe flight. This generic and rough discrimination between processes as synchronous or asynchronous makes design of software development easier by abstracting design components due to logical functionalities and constructing structural flow between them in a very flexible and modular way.

4.4.2 Design For Reliability And Maintenance

With essential software capabilities of acquiring sensor data, flight control, continuous system observation, data logging and system safety assurance mechanisms such that fault-detection, fault-identification and recovery, the SSSDF architecture for a MUAS is configured as indicated in Figure 4.9.

Safety conditions can be separated as logical and structural from software design perspective. For logical failures, data computation, computational performance of software system or errornous data flow can be discussed. In case such failures occur,

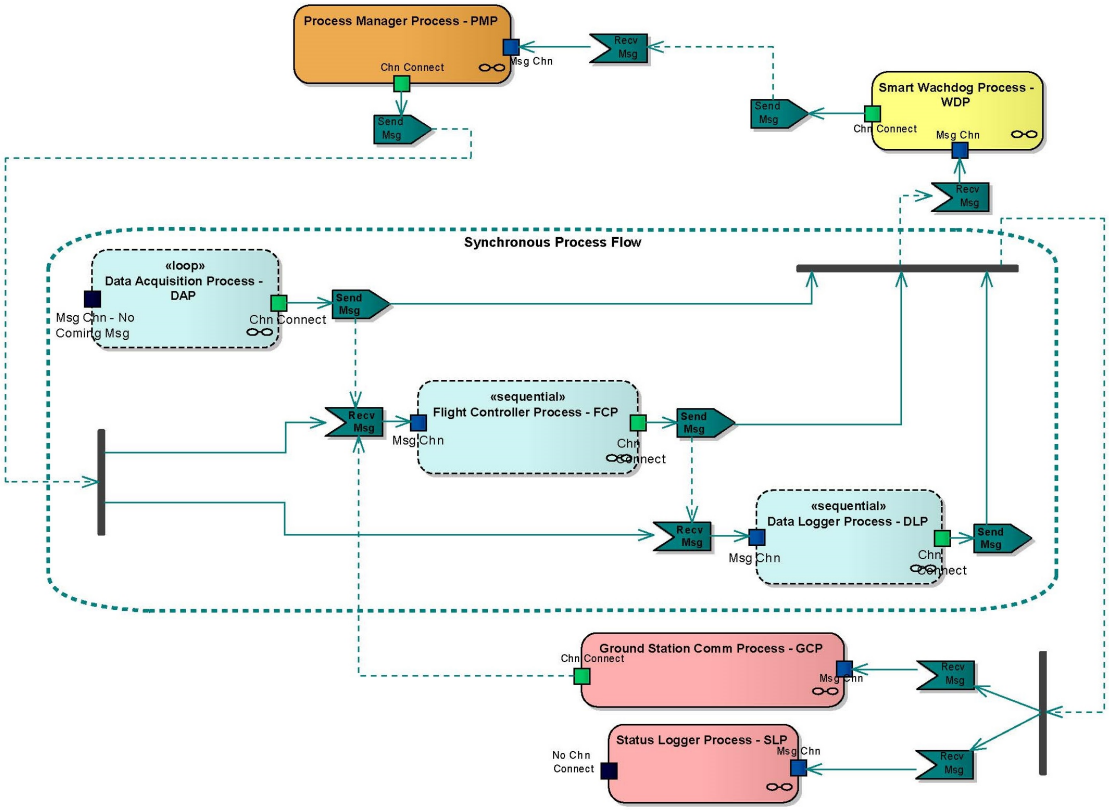


Figure 4.9: SSSDF's Detail Architecture without APM

software is designed to apply fault tolerance mechanisms such that data manipulation, syntactic data generation, tolerance time checking etc. From structural failure perspective, in case software components crash due to non-detected failures or hardware malfunctions etc., immediate recovery of the component is crucial. Framework enables recovery of components and software execution continuity during safety tolerance time to achieve more reliable software.

At that moment, it is important to remember that analyzing all causes of failure and elimination of all of them are essential for safe development. Considering this fact, the significance of the suggested framework comes to stage as effort to sustain SSS for mentioned failure identification and elimination tasks through conventional guidelines is impractical for small systems but with an intelligent framework, safety-related conditions can be grouped and handled in comprehensive and conscious design.

In Figure 4.9, Data Acquisition Process (DAP) is coordinator and periodically waits for sensor data to come. As data acquired, it is passed to next synchronous process of

Flight Controller (FCP) which is responsible for flight control algorithm computation and all flight related logical tasks. Flight related sensor data is processed in FCP and data is used in flight algorithms. The next sequential software component is Data Logging Process (DLP) of recording all raw data coming from sensors and all output data which is generated by FCP at the end of each execution cycle. These three sequential software component, which are DAP, FCP and DLP, works synchronously and periodically. In addition to synchronization between these set of processes; each process periodically send their status to Smart Watchdog Process (WDP) to guarantee continuous checks and send their status to Status Logger Process (SLP) to keep records of all internal events appearing execution time. In case error conditions occur, WDP informs Process Manager Process (PMP) to identify errors and take corresponding safety actions.

In addition, PMP is responsible for system diagnostics and overall system safety. Although asynchronous processes waits for pulse messages from synchronous processes (DAP, FCP and DLP), PMP can become coordinator process by setting proper synchronization timers in case DAP crashes or any other failures. Another process which accomplishes data flow between Ground Control Station and FCS is Ground Control Communication Process (GCP) and it connects link between ground unit and flying unit to establish healthy status information channel.

Under the thesis assumptions which limit the MUAS's software system to perform essential flight operation, the framework modularity is reached as mentioned above using logical and abstract decomposition of MUAS functionalities. In terms of software system safety, above mentioned architecture minimizes the efforts undertaken during design reviews and assessments with framework's nature of logical separation, sequential organization due to intended functionalities and flexible modularity. Moreover, messages between isolated processes can be easily traced during design. Additionally, data flow and internal software activities of all individual processes might be tracked for entire system at each development step. Fault tolerance mechanisms which are injected in to the design can be mapped, modified and observed in a simple way. Furthermore, reliability analyses can be done with less effort by following design flow. As a result, this feature of framework enables any design and implementation modifications to become easier and to have much simpler maintenance

activities at any time in development process.

4.4.3 Design For Traceability

Requirements traceability during software development life cycle is one of the most significant expectations to establish safe software system and requires systematic effort to be achieved.

Development processes governed by the framework enforces logical design and separation of software components regarding to their functionalities. This abstract logical separation philosophy is implemented even to the smallest sub-components of the entire software system. Meanwhile, all level systems requirements can be easily trace on the software components during the design process.

Additionally, logically separated software components act as abstract groups of requirements due to their functionalities and object-oriented architecture of each software component also maps directly to the low-level software requirements. As each software component aims to reach atomic-likely structure, logically isolated components point directly corresponding high level requirements.

In terms of traceability between requirements and design, for all levels of requirements and software units bi-directional traceability can easily be achieved through design process by using activity diagrams. Not only traceability but also the review and assessments of the design process can be performed via simplex approach. Design logical analysis can be performed at any time during design concurrently or at the end of design process as entire software design relies on logical development philosophy.

4.4.4 Design For Safety

Although safety is a very complex and multi-disciplinary concept which requires careful consideration of system components and reasonable effort for tasks realizations, for MUAS development, safety requirements can always be investigated easily through development framework during software life cycle.

Comprehensive development framework always tries to ensure that each software unit is implemented as expected and desired safety cautions are taken to sustain more reliable system. All separated software components have recovery mechanisms and fault tolerance capabilities. Moreover specialized components such that smart watchdog and process manager, can be used for monitoring purposes and can take initiative to prevent failures. In the meanwhile, by using the framework, activities realted to tracking all these safety assurance mechanisms and covering all safety-related requirements in the design are guaranteed. Furthermore, at any time during the development life cycle necessary modifications and functional/structural insertions are more than possible to be realize as design encourages modular paradigm. In the meanwhile, these features of framework provide isolation of safety contaminated components to have fewer side effects in case of failures occur.

Additionally, not only separation of functional software units into isolated components but also logical flows between these components can easily be built. Verification of logical and structural flow's behaviors and elimination of failures are crucial expectations during SSS development. Using the framework in any levels of development, flows can be tracked, modified and updated. Although testing will be discussed in detail later, by using APM testing approach, various tests including flow analysis and performance tests can be done in systematic and time effective way. Last but not least design data analysis which is an important objective to realize at the design process can be evaluated smoothly as design contains minimal hardware interrupts and synchronization depends on the built-in inter process message passing mechanisms. The framework uses hardware interrupts only in software timers for status checks and software component malfunctions monitoring. This architecture of framework makes entire software development process easy to assess in terms of safety.

Considering all the contributing factors discussed above, design architecture which is suggested by the framework minimizes review effort undertaken at the design process and increases conscious design during development life cycle with ease of safety assessment.

4.5 Framework For Software Coding Process

Owing to have more readable source code for easy code reviews and safety assessment, software implementation process relies on more human understandable way of (high level) coding of APIs. By the help of POSIX API provided by RTOS all coding requirements can be accomplished successfully. Apart from high level implementation and POSIX dependability, software coding process aims to establish below gains for safe MUAS software development.

4.5.1 Easy Implementation

Selection of programming language and coding style is one of the coding process requirements. This framework implement C++ and uses OS's high level implementation API. POSIX 1003.1 compliant implementation is strictly followed.

Implementation of modules in design with object oriented development methods makes implementation simpler. As a widely used programming language, C++ is tough almost in every departments of engineering and its semantic is easy to understand.

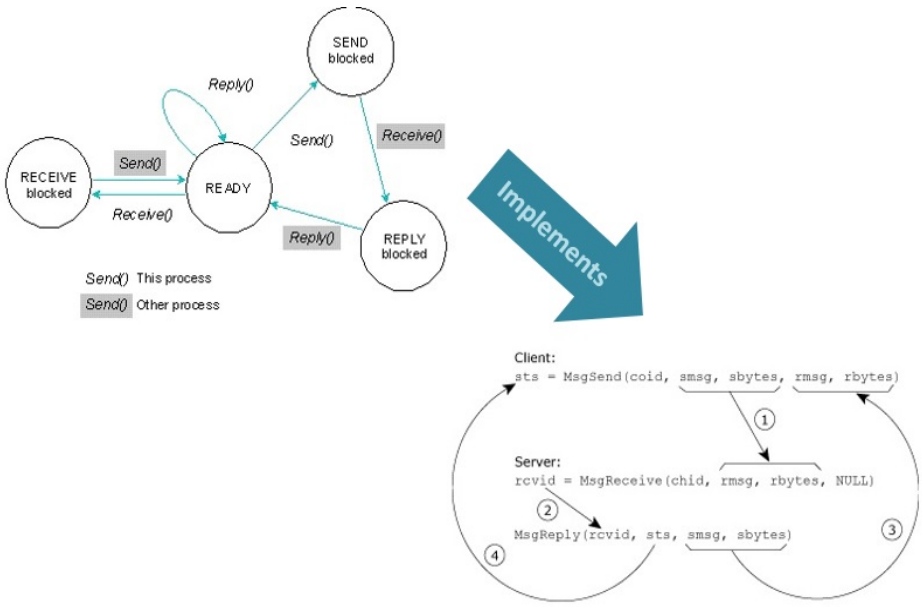


Figure 4.10: IPC Message Passing Implementation

Being one of the biggest drawbacks, implementation of cooperated modular compo-

nents in a synchronous way is handled with message passing paradigm through IPC API which is very abstract and easy to implement as shown in Figure 4.10. Moreover, this message passing methodology makes review of interfaces and data structural flow straightforward.

4.5.2 Code For Easy Software Flow Analysis

Being a compulsory activity, flow analysis is crucial for safe development process. After software design process, implementing the architecture is not enough itself. Analyses of whether software flow between software components is accomplished correctly and verification of each component's interfaces with their corresponding peers have to be performed.

Although the APM testing method is definitely convenient to achieve flow analyses and its contributions will be discussed later on, implementation of framework also makes verification process simpler due to its high level implementation. As software components separated and isolated from each other every single component can be unit tested up to their interfaces. As this framework becomes comprehensive tool for safe development and each development processes are taken into account simultaneously during development life cycle, interface identification, data flow analysis and behavioral analysis of software components can be reached easily by using activity diagrams and IPC messages list together. Moreover implementation of software functionalities in atomic way, software components can be investigated in terms of their behavioral flow by human inspection and can be reviewed smoothly by tracking them from high level system requirements.

4.5.3 Code For Simplicity

Software system complexity is important during the development process because it can affect understandability, readability and maintainability of the code and it has to be kept minimum. In order to decrease complexity of the system, object-oriented paradigm is followed and also software modules are tried to be kept simplex with basic and atomic functionalities.

At that moment, a known paradox which is as the number of modules increases complexity also increases comes to minds and conflicts with the framework's statements. Definitely there is trade-off to achieve less complex software systems with many numbers of components. However using the framework, software modules are separated using abstract logical functionalities define in the high level system requirements and coordination between modules are established using the RTOS IPC's built-in methods. Additionally each component has one connection channels so that they have one communication interface by which decreases complexity. This single-channeled architecture of the software framework ensures that data flow and coordination between modules is controlled by one-and-known connection interface.

Although system has modular architecture, flow between modules is unidirectional and exactly deterministic. This very-known data flow between modules makes implementation more simplistic. By the help of this simple architecture, reviews and analyses can be done straightforwardly.

4.5.4 Code For Easy Testability

Testing is always possible at any time during the development life cycle due to the framework's architecture and APM. Modules can be tested either one by one or as a whole. Especially message passing mechanism between coordinated modules which is encouraged by the framework is a very flexible approach to test any parts of software unit.

Another important coding process objective is to have defensive code which eliminates unidentified software flow and ensures that system behaves correctly during operation. Due to the defensive programming, test side effects can be identified easily and weaknesses of software can be evaluated after APM runs its mission-based assassination scenario.

Software components can easily be tested either to verify of data and design structure or to ensure that system is performing as supposed to do so. Very flexible and intelligent APM tool enables various kinds of tests and analytical test assessment results. The detailed explanation of testing paradigm proposed in the framework is discussed

in the following sections of this chapter.

4.6 Framework For Software Integration Process

Testing is always the most struggling part of an entire development process and it can be very overwhelming if not organized for reasonable objectives. As the framework focuses on satisfying safety-related requirements, suggested usage of the framework is to test every safety–related requirements in random order within a hardware-in-the-loop simulation, which is setup for MUAS by considering its mission profile, in an automatic way. In order to accomplish verification objectives, Assassin Process Method (APM) is developed and used in the framework Figure 5.11.

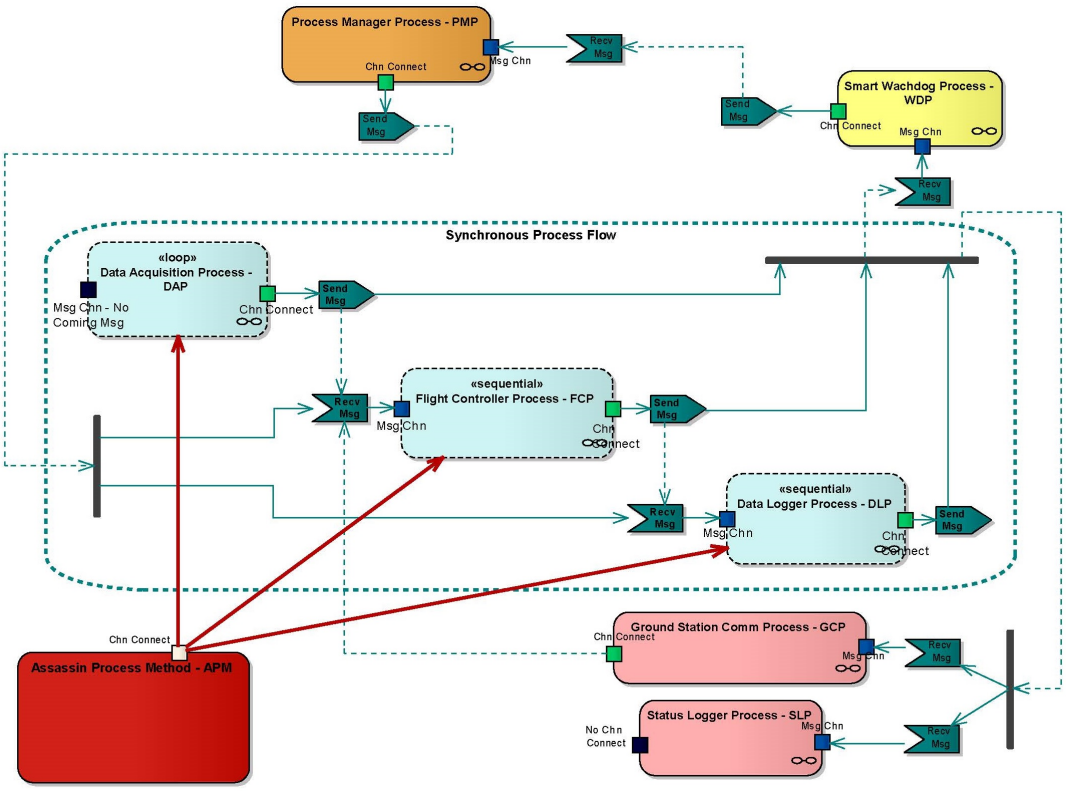


Figure 4.11: SSwD Framework's Detail Architecture With APM Test Approach

Therefore, for each mission-based test scenario, different failure condition are triggered by APM just like an assassination missions of an assassin and possibilities of hazard occurrences which depend on software malfunctions can be tested in a short time interval and without human interruption. Moreover, test results are assessed by

the APM's GUI automatically to provide analytic requirement coverage assessments in human understandable way.

4.6.1 Test For Verification

In order to ensure that all software requirements are satisfied correctly and software executes as desired, verification of the source code within integration environment has to be performed.

Preliminary expectation is to ensure that software system is unit tested carefully and all implemented functionality is working properly as identified in requirements phase. The framework provides an interface to create test cases and all test cases can be run repeatedly in an automatic way at any time of development life cycle by APM. Like a mission-based testing, all requirements are verified throughout assassination scenarios(missions) for created test cases which contains various parameters to check such as fault injections, boundary checks, data flows and behavioral tests. Although Assassin Process(AP) tries to verify software by constructing assassination missions, which focus on testing of safety-related requriments, injected in to HILS environment; each software, system and safety requirements can be tested one by one like a requirement based testing due to the APM's flexiability.

Not only testing software units but also ensuring that all requirements are covered via tests is also very significant to reduce safety risks. Luckily one of the main features of the APM is to provide a comprehensive test coverage assessment at any level of development with its comprehensive, mission-based and repeatable testing capabilities.

4.6.2 Test For Simplicity

Conventional testing requires team working because it consists many kinds of testing objectives including creation of test cases, requirement traceability, test result assessments, re –testing, regression testing, performance testing, integration testing, safety- critical analyses of test results etc. That growing number of activities in test-

ing process can easily make testing a discouraging activity. Even if there is time and budget for these activities, each test types requires deep know-how and experience to be constructed. So that, for MUAS's software development, testing process has to be investigated in a different approach to have a simpler solution.

Focusing especially for safety-critical requirements tests and safety assessments, APM provides very flexible interface to verify software system for different aspects. With APM, starting from design to coding, testing always takes part in the development life cycle and automated testing in the integration environment can be performed easily. Modular architecture of design and message passing IPC approach during development with simple ruled-based system status logging, testing is imposed in all levels of development life time with a minimum effort and provides healthy review period with straightforward test results assessments. This smooth imposition makes not only unit testing but also other kinds of testing feasible and accurate. Moreover using the same programming language during testing and running APM, simplifies the testing due to eliminating dependencies to another test tools or testing paradigm with different expectations, time and experience.

4.6.2.1 Mission-Based Testing

When software safety concept is considered, the common way of verification is requirement-based testing. As very generic and detailed paradigm, requirement-based testing can be applied by many existing tools, methods and approaches. For MUAS development, instead of following strict requirement-based testing, testing for safety-related requirements is performed for a mission profile of MUAS for all safety requirements at the integration environment using HILS. In that perspective, the testing method can be described as mission based safety oriented automatic testing which takes place in the realistic integration environment. Performing this repeatable, one-shot, mission profile and safety-oriented testing, shows that whether system is robust to the failures, has tolerance or reliable for software malfunctions.

This different approach to develop safer MUAS software system testing ensures whether system performs as intended, stays in the tolerance limits, safety mechanisms work properly and different combinations of occurring safety-related issues harms the en-

tire system.

4.6.2.2 Develop And Test Concurrently

As testing is performed in integration environment for real-time simulation using HILS, possible integration problems can be eliminated and early precautions can be taken in case possible system integration problems occur. Being a very complex process, integration becomes less affordable and less problematic by APM tool.

When integration process is taken into consideration, execution of source code in hardware is not the only concern. Additionally, performance issues can be bottle neck for entire process. All such problems can be observed during APM testing period and risks can be reduced. Automated, mission-based execution nature of APM simplifies integration process and eliminates risks caused by unidentified issues.

4.6.3 Test For Short-Time Constraints

Instead of one by one requirement-based testing, software is tested for the indented safety-related requirements as a whole automatically with a mission-based approach. Assassin process which is injected in to the software integration environment creates all test conditions transparently just like an assassination. This automatic assassination scenario can be iterated many times and provides repeatability. Apart from mission based automatic testing, safety related test conditions can be created randomly in the testing process so that various combinations of faults conditions can be tested. This also increases confidence level of software because random occurrence of tests cases simulates real flight conditions more clearly and realistically.

Due to testing is performed for MUAS's mission profile; APM only operates during the mission time. All test cases are created and verified either sequentially or randomly during testing period. That comprehensive approach simplifies time taking testing activities and achieves every desired test objectives.

The significance of the framework is, tests can be applied to developed prototype at any time and it does not require additional time to create test cases or complex system

configuration to have these tests. Whenever software design or coding update/modification takes place, software maturity for safety terms can be automatically assessed by performing safety related tests by APM. This very comprehensive and simple approach suggested by APM ensures whether intended software safety is achieved.

Though safety related requirements tests for a mission profile is performed by APM and it is the strongest part of the method, requirement based testing is also more than possible to be realized. In order to have requirement based testing, instead of creating assassination missions(APM test scenario) based on safety-related requirements for mission-based tests, missions are created considering every single requirements. Meanwhile APM becomes a test tool for requirement-based testing.

4.6.4 Test For Safety

Safety oriented testing suggested by the framework is the strongest part of the entire development life cycle. As testing is very complex process, it requires different kinds of tests like structural and logical tests. APM is definitely capable of performing various kinds of tests as discussed below. The most important feature of the framework is that, in nature, APM tests entire software system at a time for all safety related conditions in the real-time HILS environment and assesses the test results to indicate risks.

Another important task is to ensure that source code is generated as described in the design and all software requirements are included, tested and verified. In other saying, requirements have to be traceable during the design and APM definitely achieves that traceability [Figure 4.12] due to its nature without special effort.

4.6.4.1 Structural And Logical Tests

Data and structural flow tests can be performed by APM. Reliability and robustness of the software system evaluated by verifing system recovery and fault tolerance mechanisms and their efficiency. Additionally interfaces between software components can be tested and data flow through these interfaces can be verified with APM. As this

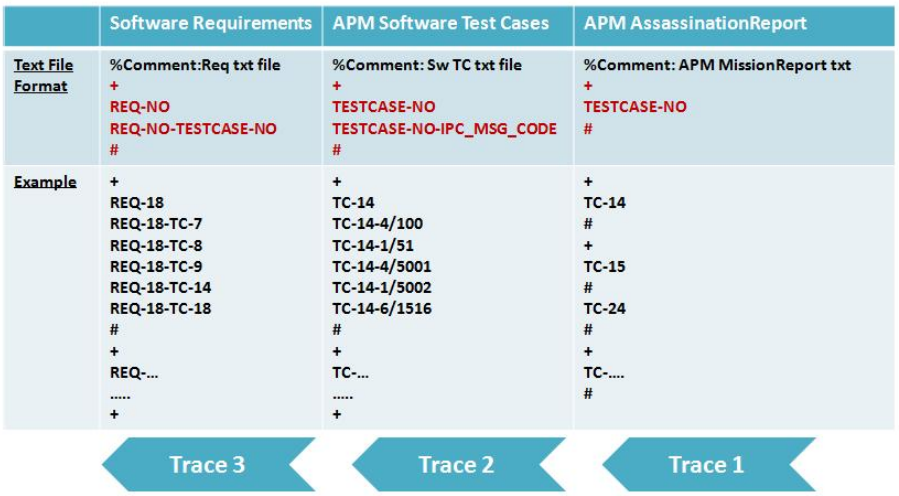


Figure 4.12: APM Traceability Systematic

framework is created to develop MUAS with less resource and effort, logical analysis is very important for development process. So that, data flow or logical components are investigated in more detailed. Using APM, data flow between software components are tested and data validities are checked as a part of data validation activity.

Last and very important feature of the APM is its regression testing capabilities. As random and automatic tests are performed for mission-based test approach by APM periodically for all identified safety-related requirements, each APM execution can also be though as regression testing. Because APM always test software system as a whole and it tests for the last configuration of the software system. At that point it can be easily considered as regression testing which is really important and significant but at the same time underestimated activities of the entire integration process.

4.6.5 Test For Automatic Assessment

With reviews, tests and assessment tasks each development process aims to guarantee that reasonable effort is dedicated and endeavored for safety; additionally conscious actions are taken in order to reach a confidence level.

Apart from implementation of software systems, assessment of the development processes at each step with analytical evaluation has also crucial importance on entire SSS.

The major assessment parameters are traceability, requirements coverage and safety confidence level assessments. Knowing the fact that safety confidence level assessment is very complicated task with many sub tasks and complex parameters, entire system safety can be assessed for identified safety requirements and corresponding software requirements for MUASs.

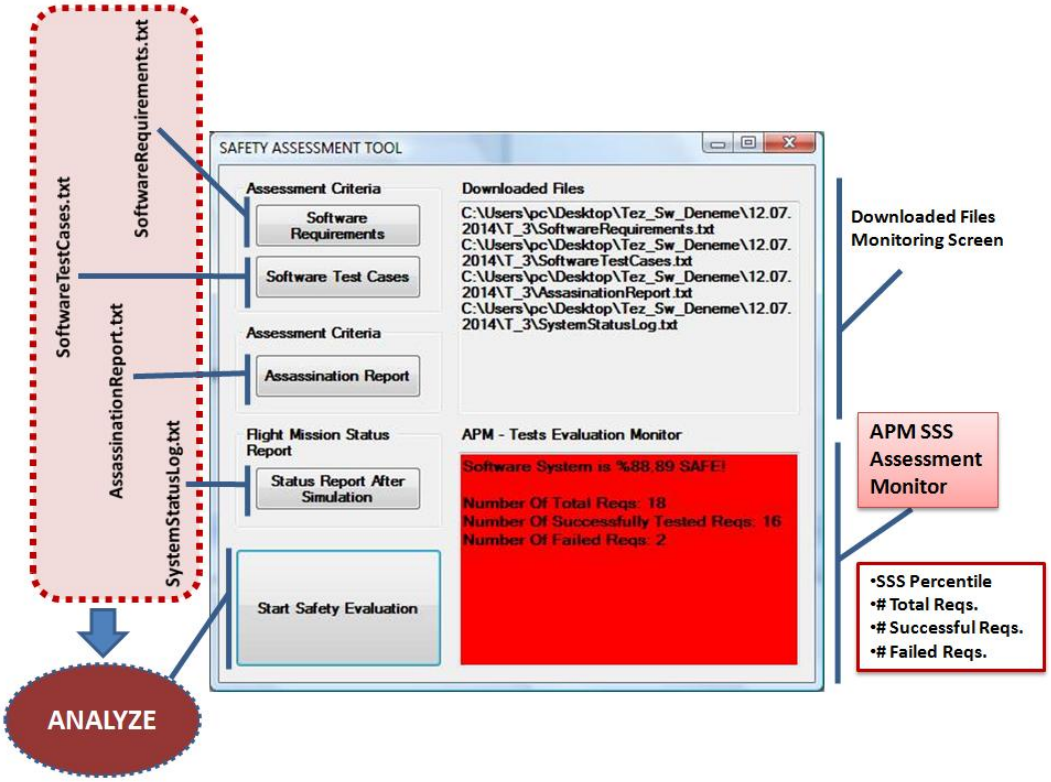


Figure 4.13: APM Assessment Tool's GUI

APM method and its simple GUI automatically handle all safety related assessment tasks. Requirements traceability coverage for corresponding high level system requirements to low level software requirements with safety criticality tags is satisfied by tracking all test cases (assassinations) up to system requirements. As a result, APM returns analytical assessment results which show traceability percentage for safety requirements and number of successfully/unsuccessfully tested items.

Considering the most significant statement about this framework which is software safety for MUAS can be achieved with less resources and effort, APM and its automatic assessment GUI easily identifies, system safety requirements are satisfied to what extend and how confidentially software can perform for safety tasks.

4.6.6 Flexible Test Methodology

Flexible testing means any testing can be performed using the APM method for MUAS software system which is integrated in HILS. Apart from compulsory tests such as unit tests, structural and logical tests, other tests such as performance, data flow, interface and safety-critical region tests can also be realized one-by-one or all together with complete assassination scenario.

Design of APM enables tests of separate software modules due to their isolated implementation. As main purpose of this framework is to discuss whether MUAS software is open to be developed for safety with reasonable effort and time, all tests are not performed separately owing to decrease time and resource needs. With this level of flexibility, APM definitely useful for test and assessment activities and proves its level of confidence together with its accuracy.

CHAPTER 5

MUAS SOFTWARE PROTOTYPING WITH SAFE SOFTWARE SYSTEM DEVELOPMENT FRAMEWORK USING HILS

As software system safety for small UASs is discussed in this thesis, the safe software system development framework (SSSDF) is developed to realize design, coding, integration, review and assessment task. Being small systems with many limitations and time constraints, development of a SSS for MUAS is discussed with a new and unique approach which is uniquely constructed within this thesis. In addition to creation of framework and discussing its contributions to the entire system safety in theory, proof of concept implementation of the framework is applied to an artificially created MUAS development case and evaluation of success of the framework is sustained in a concrete platform as a prototype implementation.

In the previous chapters, focus is given on explaining Safe Software System Development Framework (SSSDF) and discussing its contributions to minimize risks and increase software system confidence as well. In this chapter, more focus will be given on how SSSDF is implemented and how safety process which starts from requirement phase is integrated into the entire development life cycle. Therefore, proof-of-concept prototype of MUAS software, which is developed under the safety regulations, is deviced and assessed.

5.1 Development Case Limitations

Safety case is developed under the below assumptions and limitations;

- MUAS software only capable of data acquisition, flight control algorithm computation and keep logs of corresponding sensor data.
- Safety requirements for MUAS software system are selected to demonstrate capabilities of the framework.
- In order to create a comprehensive case for MUAS software system development, logically coherent requirements are identified and consistent MUAS flight mission profile is constructed.
- Hardware system development is out of this thesis scope; however to achieve hardware in the loop simulation, COTS hardware configuration is selected as explained later in the corresponding section.
- During prototyping, free-in-charge QNX is selected as RTOS which has microkernel architecture.
- This thesis study does not aims to get full compliance to any known guidelines of aeronautics instead it tries to ensure that safe development paradigms can be projected from weighted and complex guidelines to simple and feasible road map which gives confidence with reasonable effort and resource.
- As this study aims to evaluate feasibility of safe software system development for MUAS, effort and resources are always tried to be kept minimum.

5.2 Case Development

The safety case for MUAS development aims to ensure that the proposed framework in the thesis is capable of improving software confidence and achieves generic safe software development objectives in a systematic approach.

Providing MUAS's requirements and drawing the prototype's borders, safety case contains each four software development processes which are requirements, design, coding and integration. Moreover, software and hardware integration is performed in addition to creation of MUAS software model in MATLAB to have hardware-in-the-loop simulation. Remembering the fact that prototype uses HILS as the integration

environment, Assassin Process Method (APM) is also integrated into the HILS to fulfill software verification tasks. Therefore, hardware and software architectures are constructed as a complete MUAS software system prototype which implements a safety case for MUAS using the proposed framework.

5.2.1 Hardware Architecture

Safety case, which is created to demonstrate SSS development life cycle for a MUAS, is constructed on the hardware units as shown in Figure 5.1. In the hardware architecture of the prototype there exist three components which are computation board, real-time simulation computer and host computers.

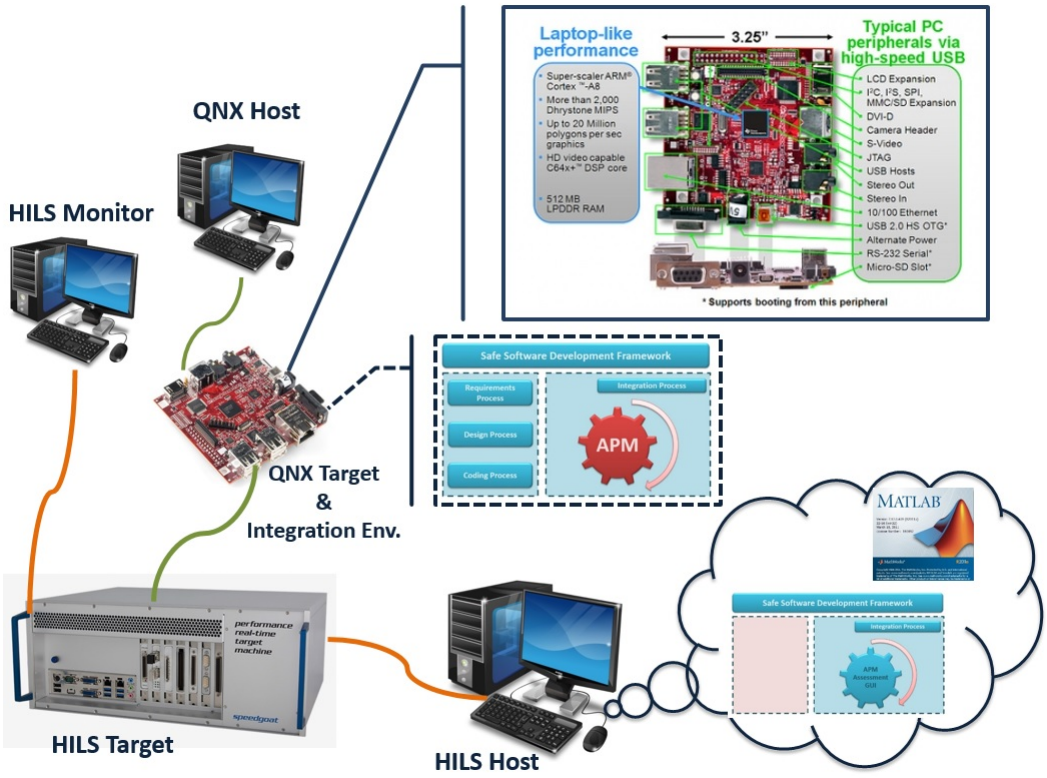


Figure 5.1: Software System Development Hardware Architecture

The computation board can be considered as framework integration hardware or flight computer in aeronautics jargon. In this configuration it is Beagleboard-Xm Rev. C board with ARM Cortex A8 architecture and capable of 1GHz processing in its CPU. QNX Neutrino RTOS and software executable created within the framework are downloaded on that board. At the same time, all APM tests are performed to-

gether with HILS on it. Apart from embedded QNX RTOS on the board, the source code development environment of QNX, which is QNX Momentics 6.5.0, is setup on another host computer and Momentics manages all QNX configuration tasks.

HILS is sustained throughout XPC Target machine that is produced by Speedgoat as indicated in Fig – 1011. This compact real-time computer only contains XPC Target kernel as an OS on it to achieve good real-time performance. It is directly connected to the BeagleBoard-Xm through communication interfaces and real-time data between corresponding components are transferred to have real-time flight simulation. Additionally, there is a host computer for HILS simulation which contains MATLAB Simulink MUAS mathematical dynamic model and simulation in the XPC Target is controlled via this host computer.

The last hardware component is a simple desktop computer which has APM assessment GUI on it. After developing source code and embedding it onto the BeagleBoard-Xm and testing entire system with APM; test results, which are formatted as text files, are downloaded to the APM GUI to generate test assessment results.

It is important to state that after setting up the mentioned hardware configuration once at the beginning, all other development tasks can be handled as a piece of cake without changing any parts of the configuration. In the meanwhile, this architecture makes system development straightforward during the entire life cycle.

5.2.2 Software Architecture

Owing to create a prototype for MUAS software development using the framework, software components which are required to be combined can be listed as shown in Figure 5.2. On top a hardware layer, QNX Neutrino RTOS is built. As explained in detailed why QNX is selected in the previous chapters, developed software application which uses SSSDF is executed and validated. APM approach, which aims to verify software system, might also be considered as another software application in the architecture.

In addition to two software layers mentioned, the logical decomposition of MUAS software can be visualized as in the Figure 5.11. Although software application is

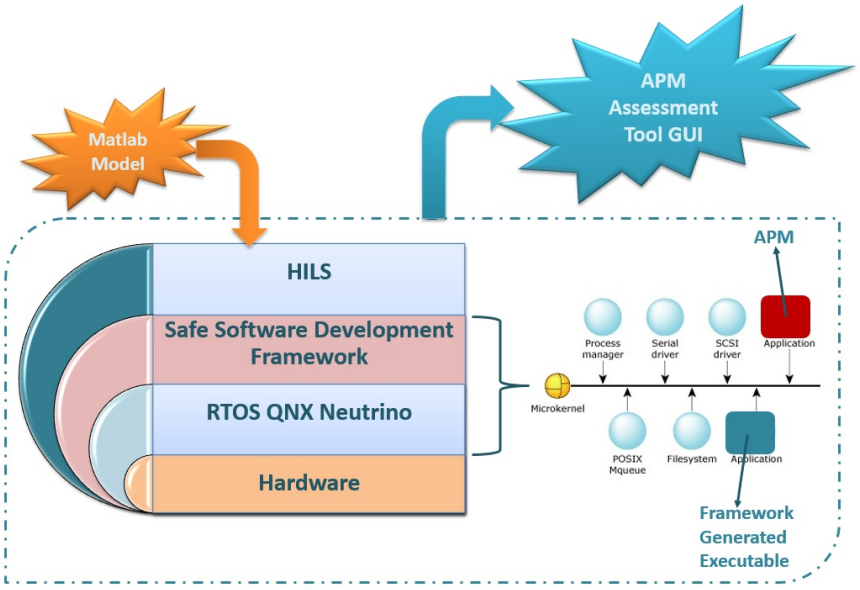


Figure 5.2: Software Architecture

introduced as one component, it consists of various processes all of which are internal software subcomponents of entire MUAS software system. Therefore, prototype consists of QNX RTOS at the lowest software layer and MUAS software application developed using SSSDF on top operating system layer. At the highest layer, HILS combines MUAS application with RTOS and APM's process to achieve complete system.

5.2.3 Applying Safe Software Development Framework To MUAS's Flight Control Software

The software prototype wiht safety case which is created to demonstrate SSS life cycle for Flight Control Software for MUAS development consists of each development process as shown in Figure 5.4. Differing from the previous Figure 4.2, Figure 5.4 also contains Software Requirements Process which aims to integrate Software System Safety Concept into the development life cycle. The reason why entire development process are considered during the thesis is to have much realistic development life cycle and to achieve MUAS software prototype that is developed using the SSSDF. As an initial step of entire development life cycle, system and safety requirements are identified during the Requirements Process, additionally safety hazard and risk analyses are performed to have much confident system. Furthermore, all anal-

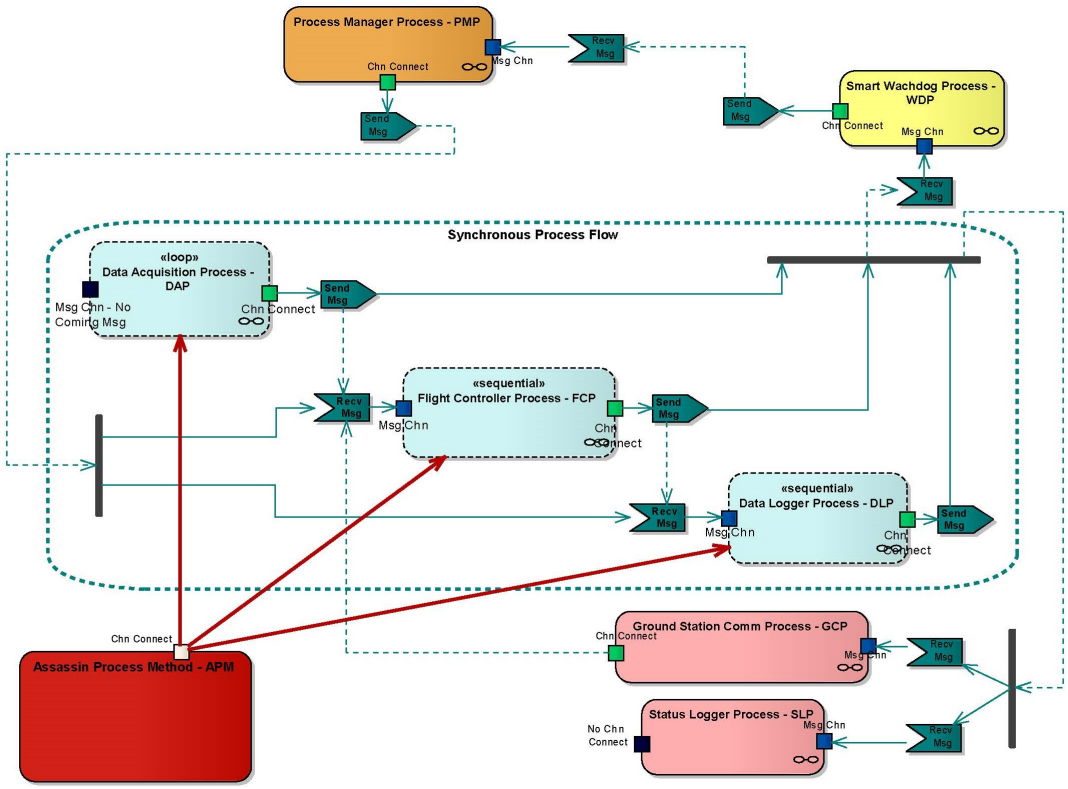


Figure 5.3: Logical View: Software Design Architecture

yses performed in the Requirements Process become the base requirements of the prototype.

After completing Requirement Process, other development processes are also realized by using the framework. All process tasks and milestone's objectives are studied and evaluated.

As every aerial system design starts with Operational Concept Document and High Level System Requirements, this safety case is also started with System Requirements Document [Appendix - A.1] which is also considered as concept description of the MUAS to spend less effort and decrease the number of documentation.

5.2.3.1 Requirements Process

Initial Concept Design Initial Concept Design is an important step to identify system and expectations. Considering the high level system requirements, overview of

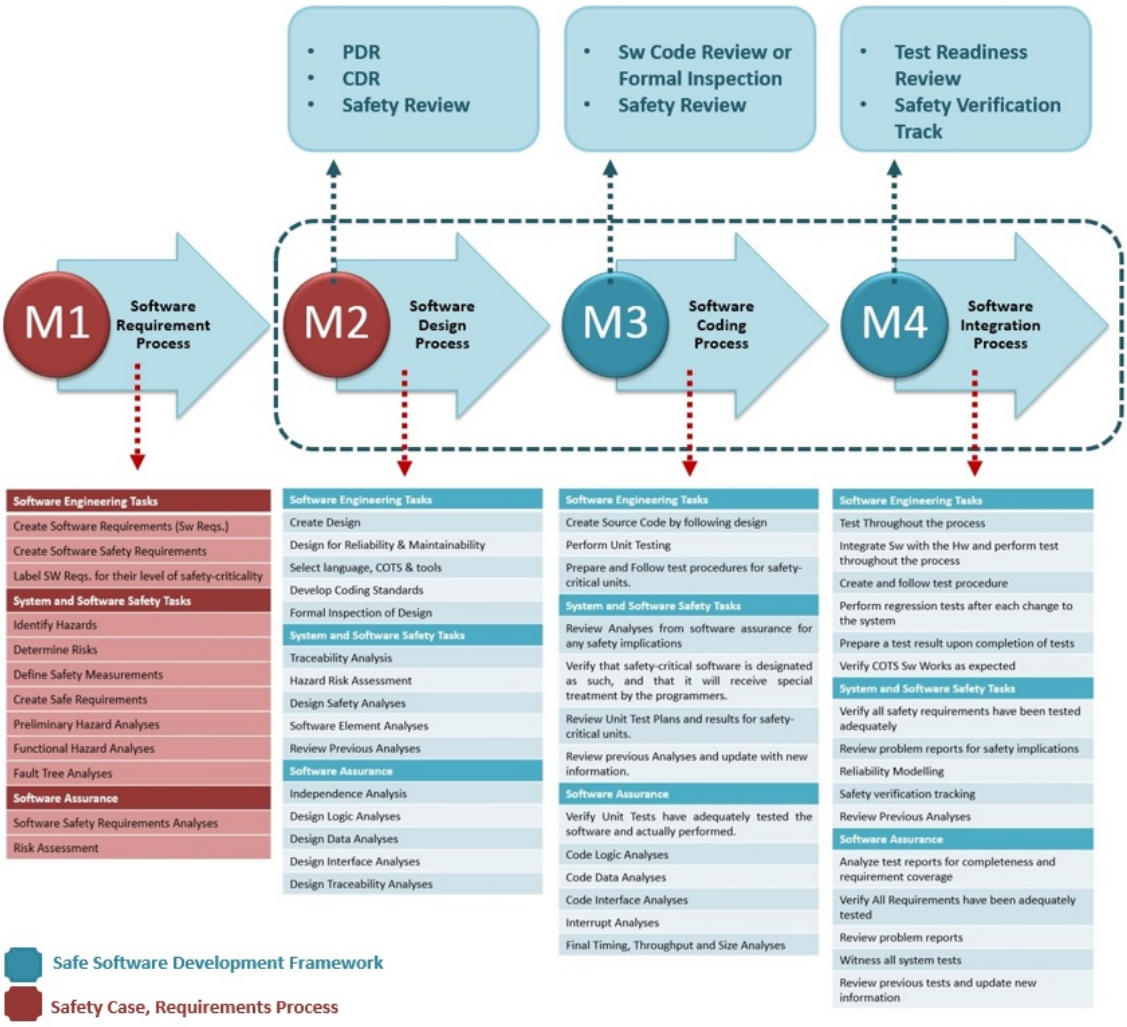


Figure 5.4: Safe Software Development Life Cycle Milestones And Tasks

the software systems and connections between internal software components can be build up from system development perspective as visualized in Figure 5.5. It helps project team to identify what kinds of system requirements can be driven, what kinds of safety requirements can be achieved and what kinds of risks can appear in the software system. The MUAS consists of various system components one of which is Automatic Flight Control System(AFCS) with software module developed to handle all flight control tasks and ensure flight safety. This component is developed as a prototype by following the SSSDF introduced in the thesis. Considering the MUAS operational expectations given in Appendix – A.1, below logical system components are created and logical decomposition of these components with respect to their functionalities are described.

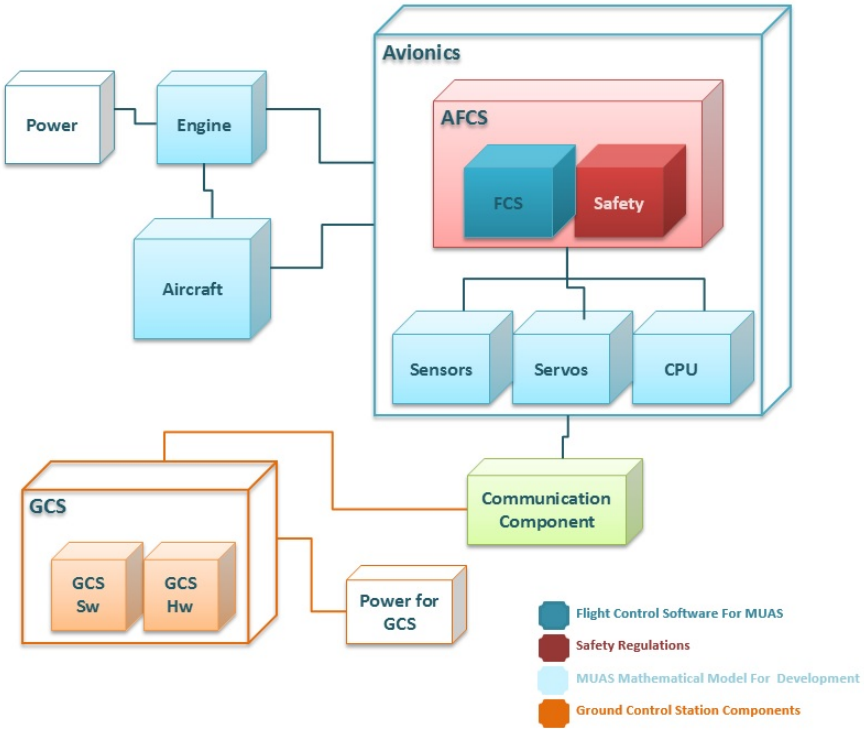


Figure 5.5: Flight Control Software Concept Design Overview

From SSS development perspective, this ICD becomes a base for the logical design approach dominated by the SSSDF and limits the entire system features. As seen from the Figure 5.5, AFCS is developed using SSSDF and it is implemented onto the Beagleboard-Xm as HILS is constructed to achieve realistic system integration environment. Additionally, during prototyping, simplex GCS is created as a monitoring tool and it is used to demostrate communication between AFCS and GCS to demonstrate comprehensive SSS development life cycle using SSSDF.

Functional Hazard Analyses Functional hazard analysis is important due to several reasons to identify hazards and mishaps and also to identify safety-significant functions. So that, comprising FHA in the software development life cycle as early as possible is efficient and also beneficial for safe system design. The benefits of FHA include but not limited to:

- Identification of physical attributes together with system functionality, data requirements and logical structure.
- Identification of interfaces between subsystems

- Identification of hazard and mishap conditions and related functions to them,
- Identification of subsystem severity,
- Identification of safety-significant functionalities.

In order to perform FHA, suggested template given in Table 5.6 is used. Within the scope of this case study, FHA is only performed for the autopilot [Appendix A-2]

NO	Major Component - SubSystem	Functionality	Interfaces	Consequences of Loss of Function	Severity of Consequence	Safety-Significant Functions	Mishaps of Hazards	Safety-Significant SW Modules
1.0	Flight Control System	a)Flight Control b)Autonomous Flight c) Sensor Control	a)A/C Platform b) Engine	a)Loss of A/C	Catastrophic	a,b,	Deaths	a)Navigation SW component b)AFCS SW Control

Figure 5.6: Example To Functional Hazard Analysis Table

Preliminary Hazard Analysis Preliminary Hazard Analysis (PHA) stands in the middle of safety and software engineering analysis and it is performed to identify and prioritize the hazards, their causal factors, severity of the hazard conditions and possible actions to prevent those hazards. Considering the physical and functional system requirements and system's capabilities, PHA is applied, and this assessment process is followed to the end of the entire development life cycle to avoid hazard conditions. In this study the metric used during the PHA is defined as in the Table 5.7.

System: Small Unmanned Aircraft System			Preliminary Hazard List			Analyst: Önder ALTAN Date: 26 Oct 2013
No	Hazard Description	Severity	Causal Factors	Effects and Mishaps	Phase or Mode	Recommended Action
1	Errant Air Vehicle – errors or faults	catastrophic	1)Error in Flight Control SW	Deaths or injury of an innocents or loss of equipment due to the crash	Operational	1) Identify safety-critical SW functions

Figure 5.7: Example To Preliminary Hazard Analysis Table

Being one of the crucial steps for safe design, detailed PHA which is created during the prototyping is presented in the Appendix A Table A.1. After the PHA analysis, considering the hazards and causal factors, safety-critical software components are identified and labeled on the Software Requirements Document [Appendix C.1]. Before determination of the safety-critical software requirements and functions; listing all causal factors is very helpful and accurate to discuss criticality levels of requirements systematically.

No	Mishap Causal Factors
1	Error in Flight Control SW
2	Error in sensor data
3	Error in Navigation SW
4	Mechanical failure in flight control fins
5	User Faults
6	Incorrect user command
7	Failure in avionic processing components /HW
8	Loss of communication between air vehicle and ground system
9	Instant environmental change
10	Failure in parachute
11	Failure in airbag mechanism
12	Improper batteries
13	Not completely charged batteries
14	Error in hardware which causes extra battery consumption
15	Error in battery monitoring
16	User fault during launching

Figure 5.8: Table of Mishap Causal Factors

Apart from listing, creating a traceability matrix for hazards and causal factors becomes very useful document during the determination of safety-critical functions, design process and determination of test cases when applying APM as well(Table 5.7 & Table 5.8).

Risk Assessment and Software Safety Requirement Analyses After performing System Requirements and FHL tasks, software requirements for FCS are established by combining all requirements together. As software requirements are developed, PHA actions and safety-critical regions in the software are labeled on the Software

Requirements document to reach requirement traceability to safety requirements as indicated in the below example Figure 5.9.

All risks and corresponding risk reduction recommendations on the PHA are analyzed and checked in this process. Remembering the fact that software requirements are tested by APM method of SSSDF, they are very crucial for software verification activities and APM assessments.

No	Software Requirements	Sys. Req. No.	Safety-Critical Req.
12.	ASS shall change its flight mode to Fail-Safe when there is no heart beat message coming from GCS for 15 seconds.	4.2.3	S.C
13.	ASS shall notify GCS about mode of the FCS during flight.	4.1.12, 4.2.1	S.C
	Pre-Flight Mode Features		
14.	ASS shall notify GCS that system mode is "Pre-Flight".	4.1.3	
15.	ASS shall be able to receive "Take-off" command from GCS when it is in the Pre-Flight Mode.	4.1.5	S.C
16.	ASS shall initialize FCS at Pre-Flight Mode.	4.1.6	S.C
17.	ASS shall notify GCS when FCS initialization status as OK when initialization is completed without any failure.	4.1.6	S.C
18.	ASS shall notify GCS when FCS initialization status as ERROR when initialization is completed with at least one failure.	4.1.6,	S.C
19.	ASS shall be able to operate a user-defined mission plan written in mission plan file.	4.1.4	
20.	ASS shall be able to store a user-defined mission plan.	4.1.4	S.C

Figure 5.9: Samples From Software Requirements Document

To accomplish Requirements Process's analyses activities, desired functionalities and confidence level of AFCS are evaluated using PHA and FHA. Furthermore, software requirements for the prototype are created and their safety criticalties are identified by the PHA and FHA. Last but not least, feasibility of realization of system requirements are assessed and prototype's development limits are clearly drawn by considering the thesis scope, limitations and motivation.

Summary As Requirements Process in the SSS development life cycle aims to ensure that system is analyzed for its functionalities, performance and safety, and software requirements are identified under these considerations to enable traceability between safety requirements, systematic road is followed to achieve these goals.

First step is to identify system and its requirements. Second one is to evaluate system for its safety and to create safety requirements. Third step is to combine all first two steps to achieve software requirements with their safety-critical correspondence. Last step is to analyze entire requirements for possible risks, risk reduction methods, criticality levels and their overall confidence.

This last step is definitely performed in the PHA phase which also performs system safety analyses and uses FHA, and explained in the PHA document in [Appendix - A.6]. Hence, requirement process is accomplished with the desired analyses and reached the main assessment objectives successfully.

5.2.3.2 Design Process

Design Process is the initial state of the SSSDF and follows the Requirement Process achieved beforehand. Being a starting point of entire framework's life cycle, design process aims to reach objectives listed in the previous Figure 5.4. At that moment it is important to remind that SSSDF tracks the procedure indicated in the Figure 5.10.

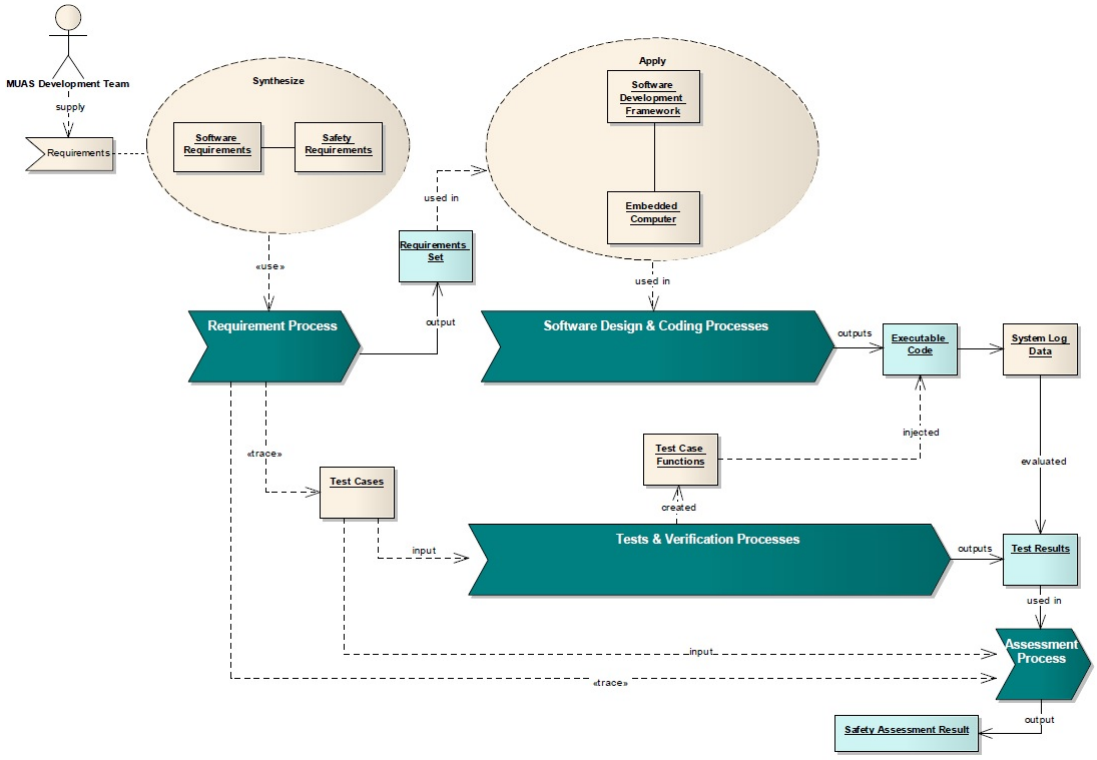


Figure 5.10: Design Architecture Overview

It is also important to remember that entire endeavor undertaken to develop a safe software system for MUAS intends to minimize efforts, documentations, resource supply, demand for specialized labour and extensive time requirements. So that, as it will explained in the proceeding sections of the thesis, development process's tasks are achieved by combining various analyses tools and/or mixing some tasks with the

others.

Apart from Design Process's task, from process's milestone perspective there exists three objectives which are Preliminary Design Review (PDR), Critical Design Review (CDR) and Safety Reviews as shown in Figure 5.4. As mentioned above, in order to have minimal effort, PDR and CDR will be performed together and Safety Reviews will be fulfilled.

Detail Design With Safety Concept Entire design process is initiated from the system concept descriptions and logical decomposition of the system is indicated as in Figure 5.5 and Figure 5.11 respectively. Moreover, the requirements generated in the Requirement Process are strictly tracked and integrated in to the design. Consistent with the SSSDF approach, three actions are considered as the PDR's initial materials. At this moment, in order to accomplish review tasks, abstract functional features of software system is listed and logical design architecture is assessed whether it is capable of fulfilling safe software requirements.

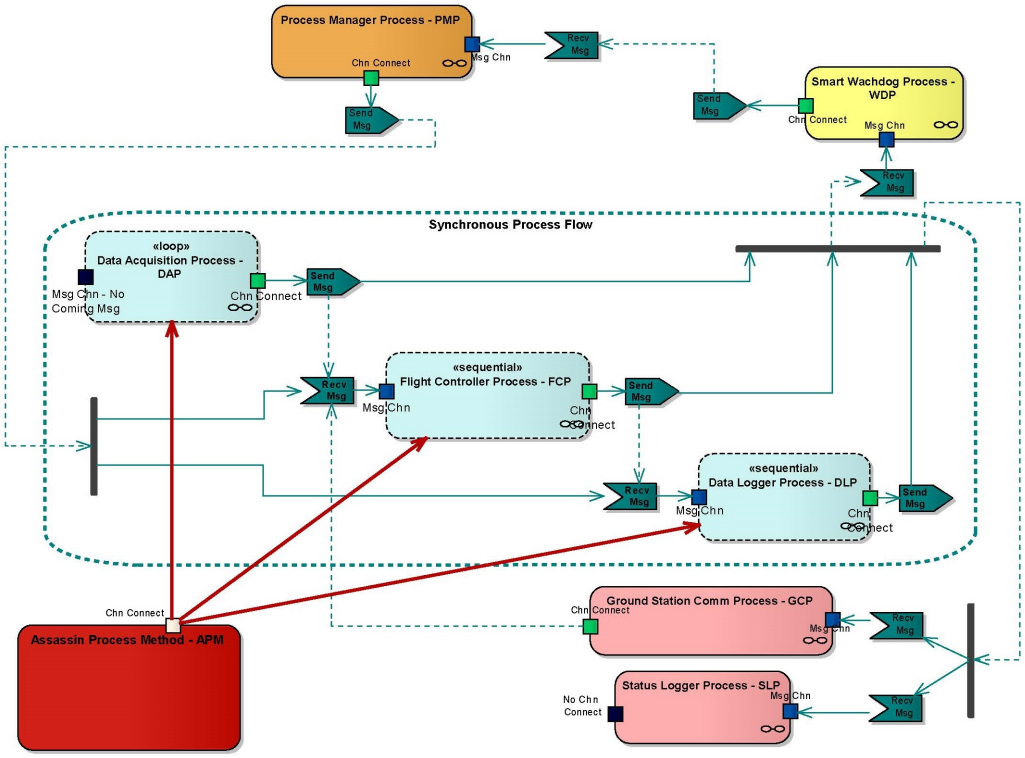


Figure 5.11: Logical Design Architecture

Preliminary logical design architecture schema is drawn in the above Fig – 1017 and detail architectures of each component are shown in [Appendix – A.3, Design components Activity Diagrams]. The significant component indicated in the preliminary design architecture [Figure 5.11] is Assassin Process Method (APM) which is included into the design process due to the nature of SSSDF. Although APM component aims to fulfill verification and system integration tests which are the tasks belonging to the Integration Process, it is included to the software design process from begining of entire software life cycle. Meanwhile very important safety test and assessment tool, APM, suggested by the SSSDF starts to take responsibility to decrease safety risks during the development life cycle.

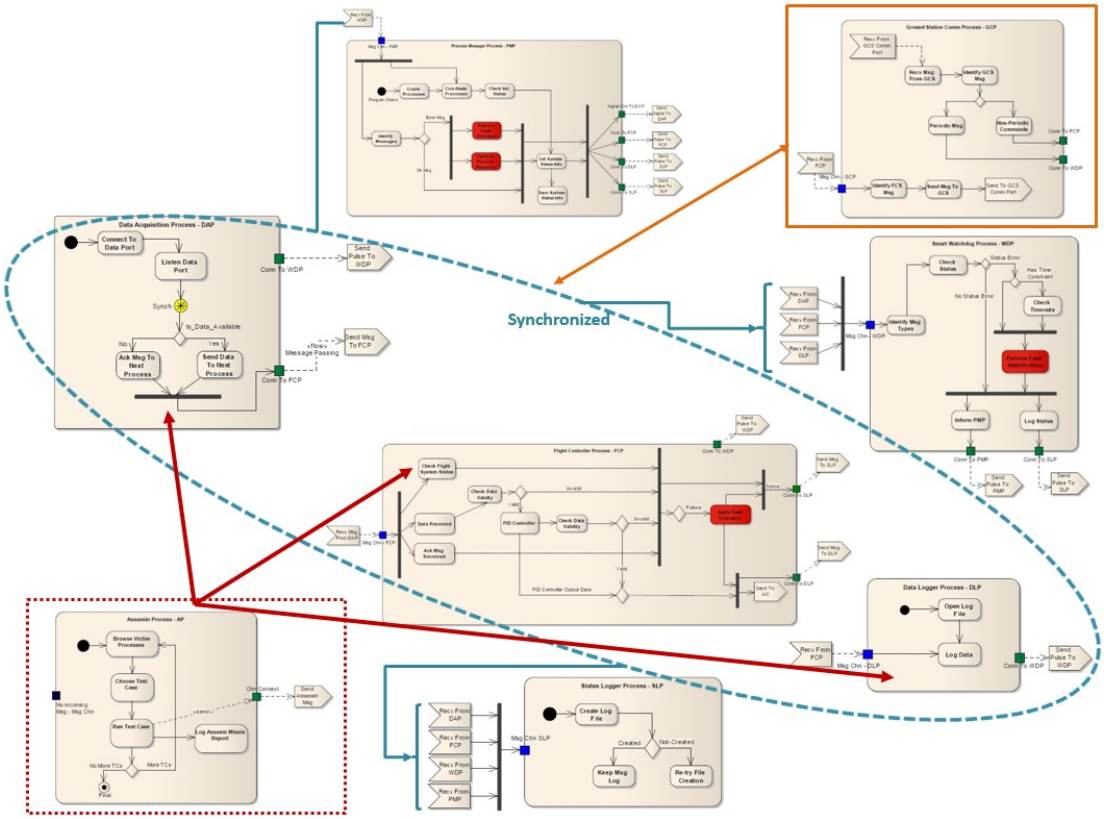


Figure 5.12: Detailed Design Overview Of FCS With SSwDF

Consistent with the Logical Design Architecture in Figure 5.11, detail designs of each component are developed as shown in Figure 5.12. At this point, an activity diagrams which represents work flows of stepwise activities and actions, additionally overall control flow are used for analyze software design. Drawing a parallel with the SSSDF's abstract logical design paradigm, this UML tool is implemented to not

only design tool but also an assessment tool during the development life cycle. Rather than using very specific and complex representation instruments, which requires specialization in computer science and experience, flow-based, activity driven, logical activity diagrams are applied to the design process.

Using the framework's design process methodology, software and safety requirements generated in the previous are realized in the design in Figure 5.12. In the design, each software component, which are indicated as distinct actions (represented as rounded rectangles), are constructed due to their functionalities and they are strived for being atomic. As clearly seen from the design view, flow of every action in the software system can be traced and connections between each component can be tracked easily.

Moreover, safety contaminated regions of the software are colored with red and corresponding safety actions are taken within these component's implementation. It is also important that, every connection to these safety contaminated regions become safety-related parts of the entire system which can also be tracked within the design architecture easily.

Another very significant design activity, which is Inter Process Communication Message Design is performed, and its contribution to the overall process is evaluated. As indicated in the above figure, each software component contains message channels and connections to them. It is also known that entire communication and synchronization are established with this architecture in the framework. In order to accomplish design process tasks and design analyses, these IPC messages are also designed and identified [Figure 5.13]. Not surprisingly and consistent with the framework's philosophy, all messages have logical meaning and direct maps to the software requirements, especially the ones with safety-criticality label.

Applying the above mentioned architecture into the entire design which consists of isolated and logical software components, whole system flow becomes traceable from beginning to end and understandable to the human analyses. Owing to simplify IPC messages and software components (visualized in the activity diagram), sequential behaviors of the system is analyzed with sequence diagrams [Figure 5.14]. Sequence diagrams created in the design process provides straightforward analysis opportuni-

Figure 5.13: IPC Messages Between Software Components

ties. Moreover, safety analyses of the design process can be performed efficiently on that tool by tracking internal messages which has safety-contaminated region interaction. Being a "kill two birds with one stone" activity, sequence diagram clearly indicates safety-critical regions and contaminated areas with in the design. Besides, it emphasizes the safety actions which are mandatory to be taken to achieve safe system [Figure 5.15] by marking the conditional branches in the design.

Inevitable activities in safe design such as assessments and realization of milestone objectives (PDR, CDR, safety assessments and design reviews etc.) are performed using the above discussed design visualization tools. Human inspections are accomplished with a smooth iteration and all safety requirements are evaluated by tracing them on the design. Each safety-critical/non-safety-critical requirement are contained in the design process phases, and flow of entire system is tracked on easy traceable design artifacts which are also needed for safety evaluations.

In this prototype, Appendix-B contains all artifacts to analyze whether system fulfills every requirements and takes corresponding actions. Furthermore, analysis of software design processes are performed by considering framework's safe design actions which encourage modular design with logical abstraction and simple isolated software components decomposition to reduce safety risks and eliminate single point

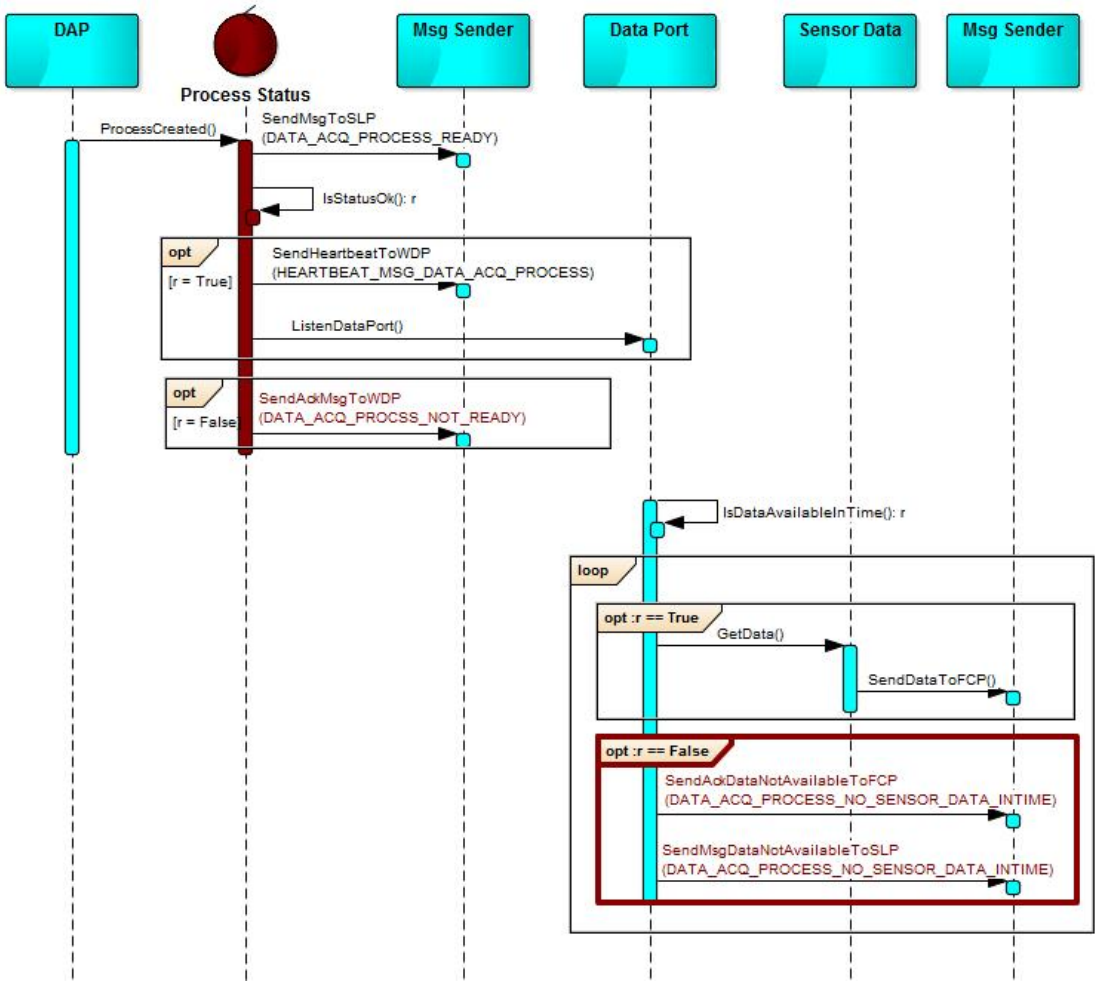


Figure 5.14: Data Acqiusition Process(DAP) Sequence Diagram

failures. Especially, it is shown in the thesis that software system covers all safety requirements and applies fault tolerant mechanisms to prototype system as indicated by IPC messages. By tracking IPC messages on both activity and sequence diagrams; safety-contaminated regions in the design are highlighted and special attention is given to these regions to reduce failure risks.

5.2.3.3 Coding Process

In the framework, coding process is initiated with previous Design Process and relies on the coding standards identified for software development [Appendix-C]. Coding standard consists of various implementation directives to accomplish a safe coding process. Being a best-practices document, coding standards are one of the most im-

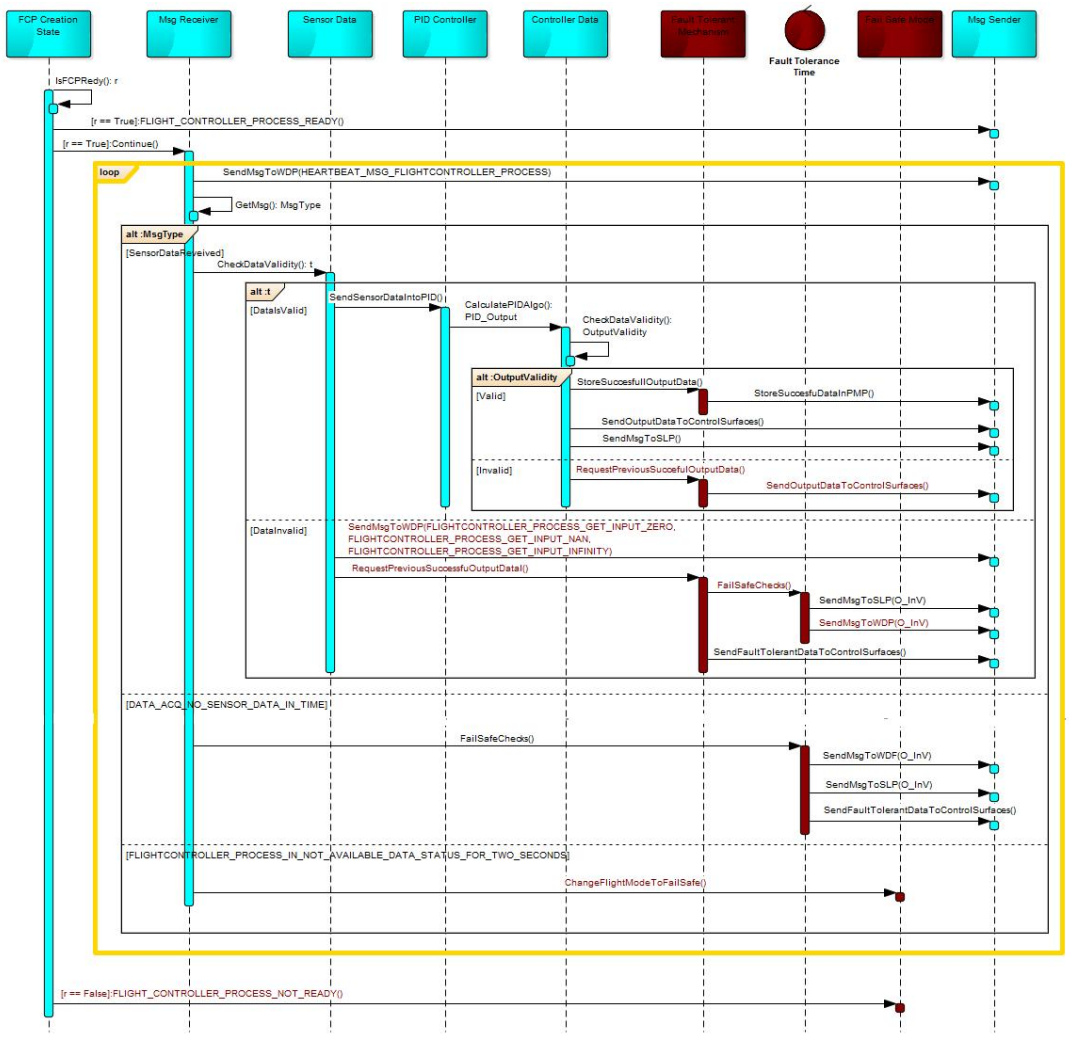


Figure 5.15: Flight Control Process (FCP) Sequence Diagram

portant reviews and formal inspection guideline for coding activities.

Coding process tasks shown in Figure 5.4 expects a software team to create source code which strictly implements design, obeys coding standards and tests the source code.

Similar to design approach of tracking IPC messages between logically decomposed software components; data flow analysis, interrupt analysis and throughput analysis, all of which are coding process's tasks, can also be performed easily.

In the prototype, apart from status and safety messages, data is also transferred by IPC mechanism. So that it is exactly known when sensor data is transferred from which component to others. Considering this fact, sensor data comes at each 5 milliseconds

to DAP in the prototype system and DAP sends data to FCP. Sequential process, FCP, computes corresponding PID controller algorithm's equations and sends its output data to the A/C to control flight control surfaces. After sending data to A/C, both raw sensor data and PID output data are send to DLP to write them into a log file. Meanwhile, periodic and synchronous data acquisition loop is established for the MUAS and data analysis is achieved by tracking IPC messages which carry sensor data and controller outputs.

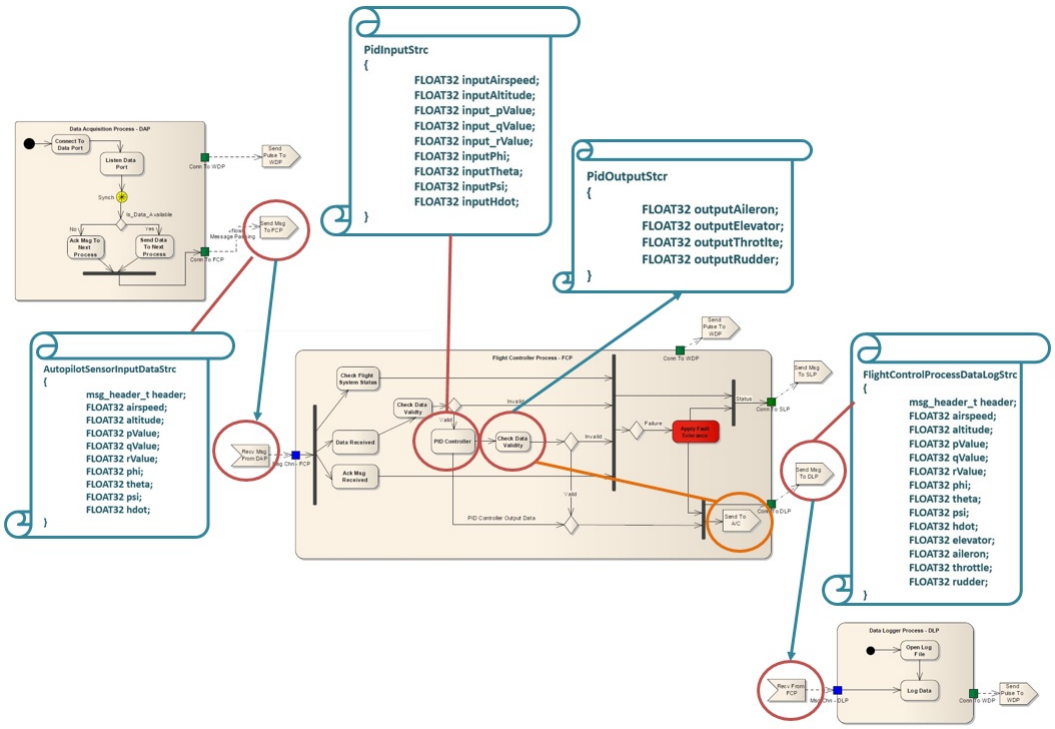


Figure 5.16: Data Flow Analyses For A/C's Sensors Data

For this prototype developed using the SSSDF for MUAS's flight control system, the only data structure, which is carried on message channels, is sensor data shown in Figure 5.16. Additionally, PID algorithm output data is also send to the A/C and is added to the analysis part. Data structures indicated in the above figure are transferred within the message channels and due to the blocking mechanism of IPC one data is transferred at a time to the next software component synchronously. For each transferred sensor data, message channel connections and flow order are reviewed with human eye and ensured that systems works properly. Furthermore, as APM is capable to verify software system with different types of tests, automated APM tests also indicates that software has been implemented accurately under safe development

paradigm.

To perform interface analysis, message channels are traced together with the data flow because they are the natural communication interfaces implemented by RTOS itself. In other words, by tracking messages on the communication channels, data and components interface flow analysis can be performed together. On the other hand, internal class architecture of the software components can be analyzed simply due to the components' atomic nature and fully isolated separation. As each software component only contains an intended class hierarchy with strict object-oriented design approach of separate header and implementation files, interface analyses can be inspected by human-eyes smoothly. Moreover, safety related messages and data structures, which are declared to be used for same functionality, are implemented as shared libraries to quarantine the safety regions and decrease risks caused by implementation failures of interfaces.

Only vulnerable review activity is code peer reviews during this process. Conflicting with the MUAS development paradigm, code review can definitely seem an exhausting activity. However, due to the enthusiasm of the framework, if it is strictly mandated by an outsider to have code peer reviews, it is surely possible to achieve as framework forces coders to follow coding standards and to apply very simple, high level, OOP coding style in their source codes. Relying on the high level implementation, simplex architecture of the framework for logical design and human understandable coding makes code review process easier.

In order to fulfill analysis and review tasks and also process's milestone objectives, which are Software Code Review/Formal Inspection and safety reviews, the APM method in the framework is also integrated into the Coding Process as done in design process. The reason of this early integration is that APM tests and evaluates safety of the software system as coding process proceeds not only unit tests but also for safety requirements. By the meantime, clearly during coding process, APM is used for unit testing as it can execute a test at a time. In addition to the APM, design process analysis tools that are activity and sequence diagrams are also major in the the coding process. As mentioned above paragraphs, nested analyses activities performed instantaneously during the design and coding processes are followed on the activity

and sequence diagrams generated for the software system, and safety of the system is automatically obtained from APM as will be discussed in detail in the next chapter.

From safety perspective, the framework ensures that even if coding process has weaknesses, system safety still becomes protected by APM. In other words, for all safety requirements and probability of failure risks, APM tests source code and evaluates how the robust software system is achieved. It is shown in the prototype that coding process can iteratively implies safety and reduces risks automatically as framework suggested.

5.2.3.4 Integration Process And Software System Prototype

Being a last process of the entire software development life cycle, integration process is definitely the most important process before release of software system as a product. Integration process consists of various review and analysis tasks to ensure system is ready to be delivered. In that manner, what is expected from the integration process is to prove that the software is capable to satisfy all software requirements and it reaches the desired level of confidence for the identified safety regulation.

In the previous chapters and in each development phase, APM takes part and contributes to test, analyze and assess system safety. In chapter 4, significance of APM and why it is useful for the safe software development are explained in detail. This section not only represents the APM's tests and tests results but also APM itself is evaluated for its performance and practical effectiveness through the prototype. In order to have successful evaluation, APM is discussed for testing and review perspectives.

HILS As An Integration Environment

By using the SSSDF, efforts undertaken to achieve a desired level of confidence is minimized as the framework incorporates integration process's tasks into the entire development life cycle with less effort and simplex way. In other words, although the integration process is considered as another process in the life cycle, it starts from

very being to end of it. The smart APM approach introduced in the framework is applied to the life cycle, and it handles almost every tasks of the integration process itself automatically.

Indicated in the hardware architecture section, hardware integration environment of SSSDF consists of QNX board as a target computer and hardware-in-the-loop simulation setup for MUAS model and real-time operations. Entire design, coding and test activities are completely integrated to this HILS setup. Moreover, APM applies assassinations to HILS to test and verify that system is safe. In the prototype, during development, HILS and APM together accomplished both creations of test cases and verifications of software requirements automatically and provided human understandable verification assessment reports. Considering that fact, in the proceeding section, integration means integration of SSSDF generated source code, APM tool, Beagleboard-XM and XPC-Target to device a complete prototype.

In the prototype, coding process starts with creation of PMP, DAP and FCP in the given order after the design process. Later, APM is implemented to realize SSSDF. Due to the same coding approach of APM, implementing APM is nothing more implementing DLP, which is the fourth software component implemented. Being an isolated and transparent software component, APM coding is a piece of cake and initial state of the integration process. Afterwards, WDP and GCP are coded and integrated into the system, and all software components are realized.

Integration Process's Test Approach & APM

Considering the DO-178B and similar software assurance guidelines in integration process, software test cases are created for software requirements which are identified in requirements phase. Up to now verification and test approach seems identical to the well known guidelines and their approaches. However, in this study, APM takes charges and all test cases are combined automatically executed by APM to verify the software and assess whether systems successfully perform as described in the requirements. Just like an assassination to the software system, created test cases which are organized considering for the MUAS mission profile are executed either

sequentially or randomly. Furthermore, APM tests entire requirements, especially the ones with safety-related label, to ensure system safety is achieved and software behaves as indented.

The software requirements, which are identified in the requirements process in the prototype development life cycle, have a safety-critical label to indicate the risk of corresponding requirement. Within this study, almost every software requirement is marked as safety-critical meaning that system has to be tested and verified for all requirements by APM to ensure that it is confident and safe. Therefore, after APM integration, assassination scenarios are implemented in to the APM's code block as shown in Figure - 5.17 to assure that all SC requirements are covered.

```
// PREFLIGHT:: TAKE-OFF SUCCESSFULNESS TEST
int TCF_1(UdpConnection *udpVictim, INT32 coidVictim)
{
    // Send Correct Mission Plan
    sendTakeOffCommand[0] = 1616.0; //Packet Header
    sendTakeOffCommand[1] = 300.0; // Mission Altitude Value
    sendTakeOffCommand[2] = 24.0; // Mission Airspeed Value
    resultUdpSend  = udpVictim->SendUdpData(sendTakeOffCommand, 24);

    sleep(3);

    // Initiate software to apply pre-flight checks even if it has checked it before
    MsgSendPulse(coidVictim, -1, (PULSE_CODE_FLIGHT_CONTROLLER), FLIGHTCONTROLLER_PROCESS_BEFORETAKEOFF_CHECKS_DURATION);

    sleep(3);

    // Take-Off Cmd
    sendTakeOffCommand[0] = 1515.0; //Packet Header
    sendTakeOffCommand[1] = 2.0; // Take-off Cmd
    sendTakeOffCommand[2] = 1516.0; //Packet Control
    resultUdpSend  = udpVictim->SendUdpData(sendTakeOffCommand, 24);

    return resultUdpSend;
}
```

Figure 5.17: APM Code Segment - Example To Test Case Creation

Although APM approach enables both random and sequential execution, in this prototype, random execution is applied. The reason why random execution of assassinations is preferred is, randomized testing provides system developer to assess whether random creation of failures has different impact on system safety. In this way, in addition to the sequential requirement based testing paradigm suggested by DO-178B and other guidelines, various combinations of fail conditions are tested, and system robustness is analyzed much confidentially for these failure conditions.

APM Mission contains many Test Case Functions (TCF, as in Figure - 5.17) with a numeric identifier such as TCP_# (exp: TCP_1). TCFs can definitely create test conditions for several software requirements. Moreover, at assassination time, each TCP_# is called in random order and executed in the mission.

In the prototype, many APM Missions are executed during the development to test internal development phases. To illustrate APM and its capabilities, three missions are selected as representatives in this chapter. In the first mission scenario, a subset of safety-critical requirements is focused and APM is executed to verify them. Drawing a parallel with the framework's philosophy, it is indicated that APM is an efficient tool to test, verify and assess the selected set of software components and requirements. Therefore, during the development time, as software components are coded sequentially, APM can be applied for only coded functionality, and can test their safety implications. At that point, before expressing the APM missions' executions and their assessment results, a summary of what has been done in SSSDF phases together with APM tests can be illustrated through the real system components and structural flow [For SW Reqs. 32] as shown in the figures below [Figures 5.18, 5.19]

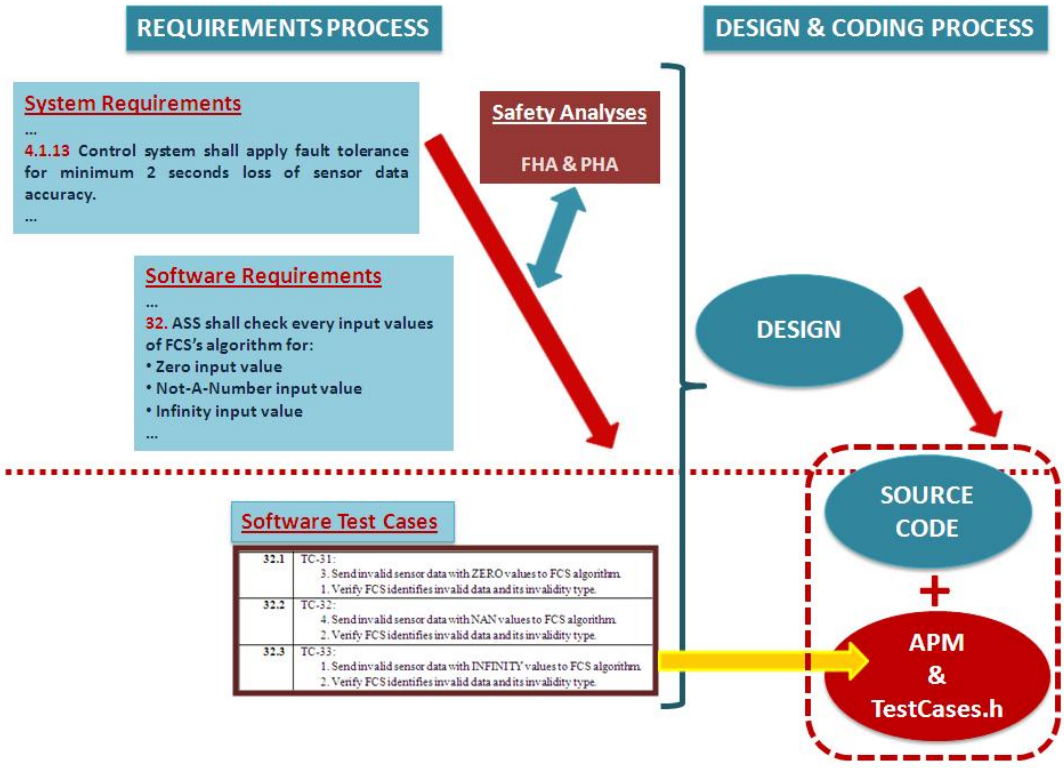


Figure 5.18: Prototype Development Process with APM Tool In HILS

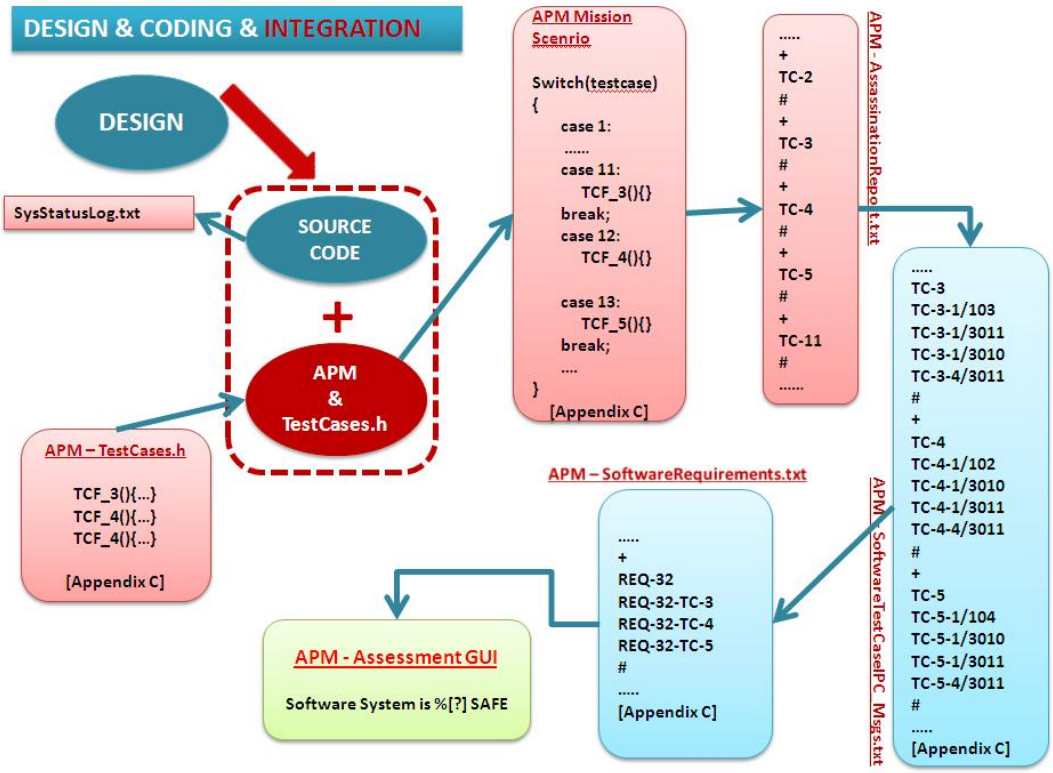


Figure 5.19: Prototype Development With APM Tool In HILS - Cont.

APM Mission - 1

The automatic APM's assassination scenarios, which are created for the prototype's safety critical requirements, are listed in Mission Scenario – 1 in Appendix C explicitly. The result of the APM Mission -1 is also indicated in Figure 5.20

In the above assessment figure, it is shown that, for 18 requirements to be tested, 16 requirements are successfully verified, and remaining 2 requirements are failed. These failed test cases are corresponded with the Sw Reqs.-32, which is related to the sensor data validity during FCS calculations, have safety-critical label. During the APM execution, For NAN and ZERO values, system failed to change its status to Fail-Safe. So that APM Assessment Tool returned the corresponding requirements as Failed-To-Be-Verified. After that assessment, corresponding code segment is changed and system is tested by APM one more time as development is proceeded.

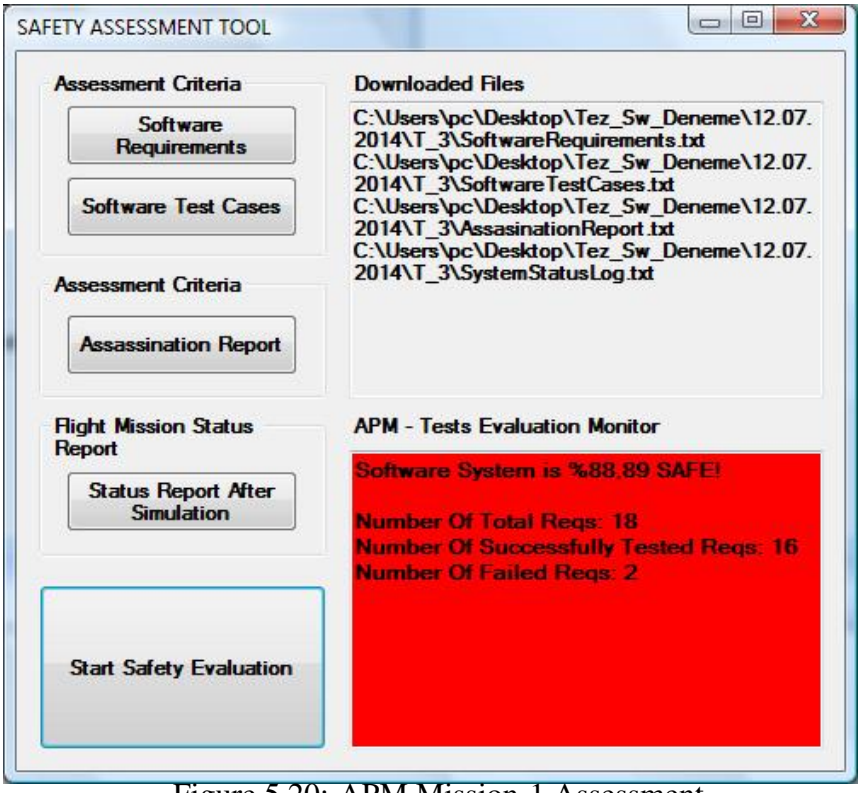


Figure 5.20: APM Mission-1 Assessment

APM Mission - 2

As software development proceeds, additional functionalities are coded and more requirements are expected to be covered successfully by APM. In addition to the 18 requirements implemented and tested above, in this Mission - #2, 26 requirements are tested. Furthermore, in the previous mission -#1, 2 requirements were failed in the verification tests and the corresponding code changes had been performed to correct them, meaning that these 2 requirements are expected to be verified successfully at this mission. When APM was applied to the 26 requirements (with additional 8 requirements to the Mission - #1), the result is pretty charming and is indicated in the Figure 5.21.

As shown in the assessment report of APM Mission - #2 in the above figure, all 26 requirements are tested and all of them are verified successfully.

Apart from verifying all requirements successfully, the cool and significant benefit of the APM, which is re-testing all previously tested requirements together with additional ones, can be observed easily. This re-testing (regression test) of the entire software system also assures that subsequent requirements have no failure effect on

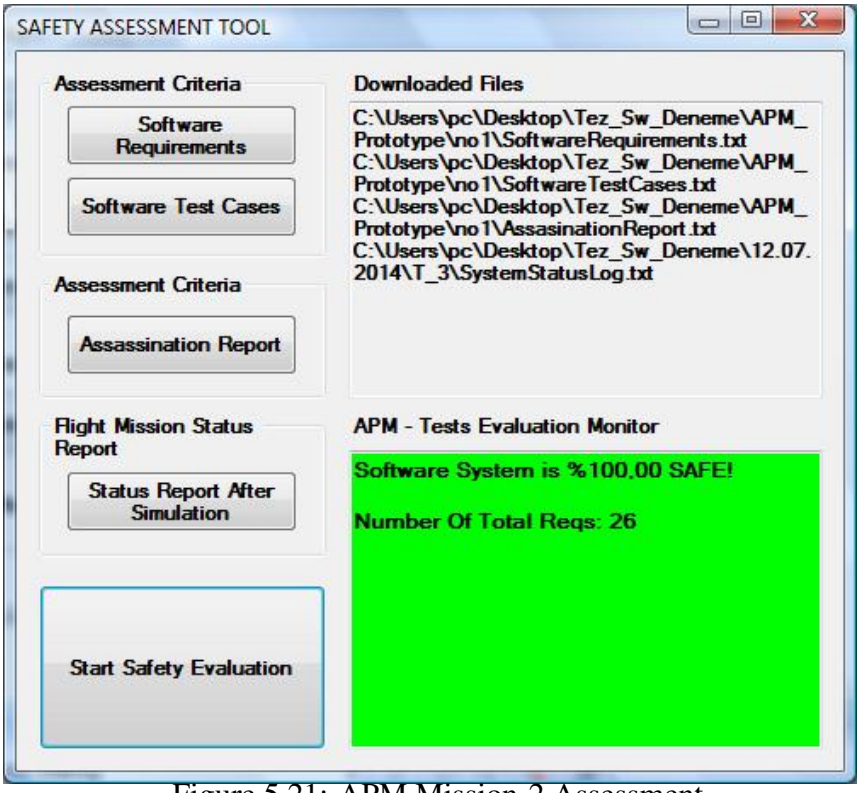


Figure 5.21: APM Mission-2 Assessment

the system and do not create side-effects on the previous healthy software application. Furthermore, iterative testing approach applied by the APM proves that verification tasks in the development life cycle can be achieved easily and accurately at any time during the process. Therefore, as SSSDF aims to device a simple development paradigm for safe MUASs, this iterative and comprehensive mission-based APM approach can be considered as a handy tool to have more confident software systems.

APM Mission - 3 and 4

As the prototype aims to develop a flight control software for a MUAS, flight control algorithm (controller) is implemented into the MUAS's software system. The implemented controller is designed to perform altitude, speed and bank-angle hold operations for a MUAS.

In missions 3 and 4, impact of the safe software development is assessed from another perspective to discuss its importance. By changing the controllers (controller with different performance characteristics), reliability of the MUAS and the impact of the

SSSDF to the entire system safety (considering reliability as an only parameter) is evaluated.

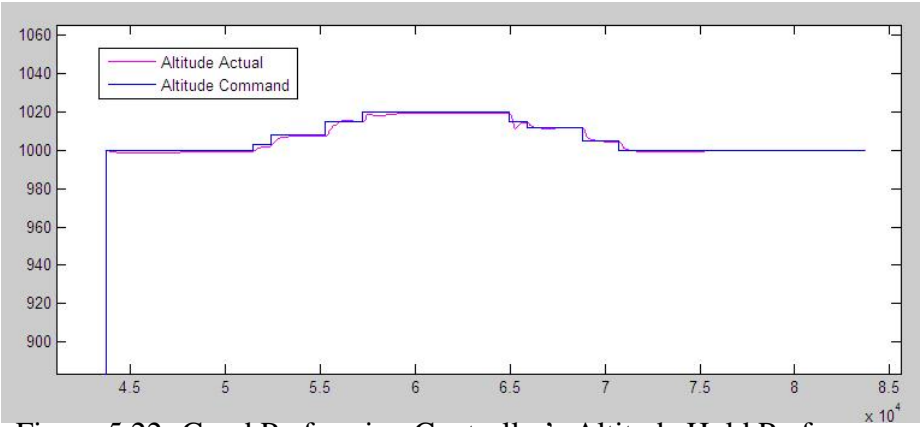


Figure 5.22: Good Performing Controller's Altitude Hold Performance

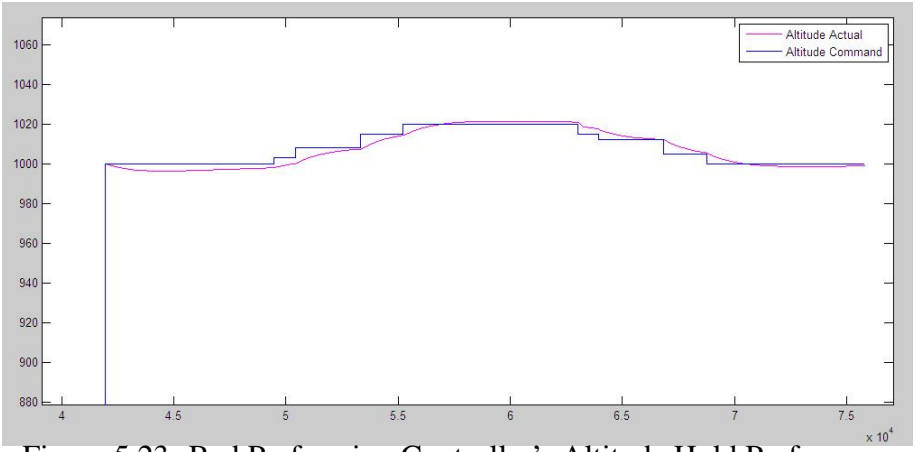


Figure 5.23: Bad Performing Controller's Altitude Hold Performance

In first two of the figures, different flight controller algorithm (controllers) implementations are analyzed for their altitude hold performances. Considering both of them, it can be seem that, controller implemented in the Fig -5.22 performs better with respect to the controller illustrated in Fig -5.23. It is obvious that, for control-people, the controller in the Fig -5.23 is a bad controller to implement when compared to the controller in Fig -5.22. So, let say "good controller" for the better performing controller in Fig -5.22 and "Bad controller" for the other one. Both APM scenario terminate with "kill engine" command.

In Mission - 3

Illustrating the algorihms which perform altitude hold operation, and trying to achieve %100 airborne MUAS for a flight mission, the focus in this experiment is given to the

effects of SSS implementation by using SSSDF and APM. So that, in order to identify SSSDF's impact to the entire system, some parts of the good controller's source code was changed to put software system vulnurable to some failure conditions. In another saying, the code segments, which enable recovery mechanism, was commented out and system kept open to these types of failures. Besides, both APM scenario terminate with "kill engine" command to demostrate direct flight termination.

In fig -5.24, the response of the bad controller in the software system in which APM executes to test, can be seen. It is clear that, the performance of the bad controller is awful and can not be realized in a real MUAS.

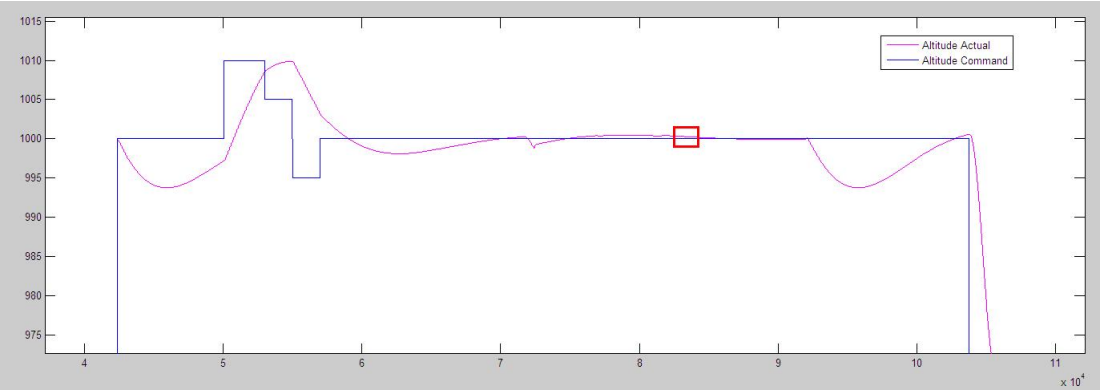


Figure 5.24: APM Applied to "Bad Controller"

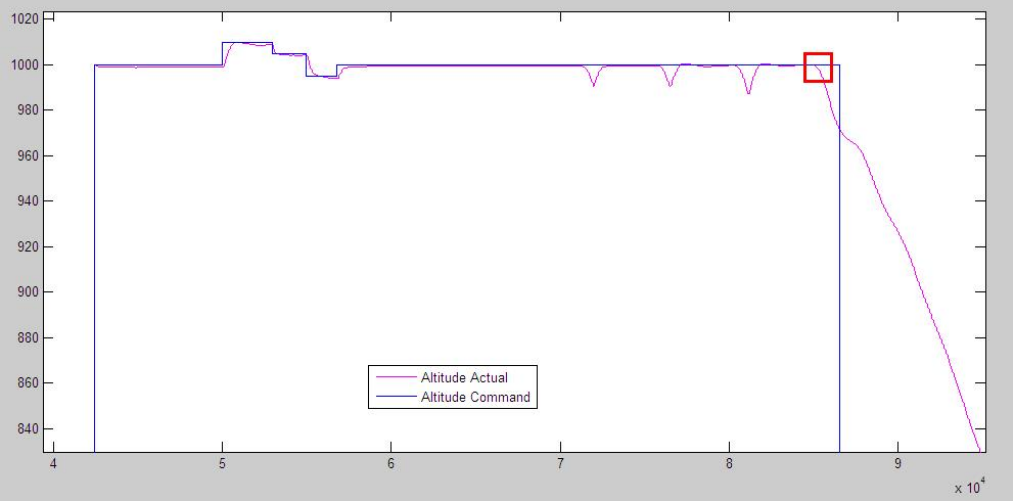


Figure 5.25: APM Applied to "Good Controller"

However, from "safety" perspective, MUAS was still airborne and did not crashed. Moreover, the APM assessment for this software system is "%100 Safe" as expected. In this way, especially considering the civil purpose MUASs , this characterisitic

of the system is safe (when realiability of the system is evaluated as an only safety parameter).

In Mission - 4

On the other hand, In Fig -5.25, good controller-implemented software system is tested with APM and corresponding assessment result was given as %72,22. In the figure, what can be inferred is system performs better than the previous controller from algorithmic and perfomance perspective, however it fails to complete its mission and crashes when an error condition is triggered by APM. The red squares marked on the figures represents the instant times when system was crashed and continued to operate for good-controller and bad-controller implemented systems respectively. Therefore, although "good controller" has a better perfomance than the bad one, it is not safe and has to be assured before start its flight operations.

Although this is not a controlled experiment, what is emphasized is to show that, as well as the control algorithm itself, the way it is implemeted also has a great significance on the system safety. For this experiment, reliability of the system during its flight operation is taken into accout as an only safety parameter. Under this circumstances, system is proved that SSSDF and APM could ensure system safety level for different systems. Hence, remembering the fact that, for civil purposes, MUAS can not crash in the populated areas and has to be assured that it will never crash. So that, this experiment demonstrates that, SSSDF and APM helps developers to improve confidence level of a MUAS's software system by following an easy, systematic and comprehensive approach.

5.2.4 Assessment Using SSSDF's APM Assessment Tool

Test and verification activities performed by APM, which require time, effort, resource and experience, are mandatory to achieve a safe and airworthy MUAS. As indicated in previous sections, APM handled verification tasks and responded with assessment results for tested software requirements. Furthermore, traceability of the requirements is achieved, additionally tests and requirements coverage are satisfied as indicated in the APM Assessment GUI's result monitor [Figure 5.21].

This characteristic of APM makes SSSDF intelligent, easy, comprehensive and safe tool to develop safe MUAS. In this way, as each APM execution performed, system is always tested repeatedly for all requirements, so that developers might always feel themselves comfortable during development and at release times. In this way, APM acts similar to the regression test performer.

Considering the above facts, it is clear that reviewing only source text files which are supplied to APM Assessment GUI, software system safety can be analyzed for entire flow, requirements and safety mechanisms. Moreover, human readable APM Assessment results such as "%80 SAFE!" respresents system safety status clearly and developers can use the tool confidently without gaining extensive expertise about SSS test methodologies.

CHAPTER 6

CONCLUSION

In this thesis Safe Software System Development Framework (SSSDF) for MUAS development is introduced to achieve safer software systems and SSSDF is applied to hardware with HILS setup to create a prototype, which demonstrates whether application of this framework is accurate and efficient to accomplish safety goals. Moreover, for software tests and verifications, a unique, automated and intelligent tool, Assassin Process Method (APM) is introduced and used to improve software integration process efficiency.

Considering the existing airborne SSS guidelines, which discuss and handle safety concepts with generic, multi-stage and state-of-art approaches, SSS development for mini unmanned aerial vehicles becomes impractical to be applied by using these guidelines due to their extensive resource needs which conflicts with the MUAS development paradigm. Although these guidelines are successfully processed during the software development life cycle for bigger aerial systems, tight resource limitations such as time, budget, work force, experience etc. for MUAS development and also non-MUAS specific nature of these guidelines, enforce creation of practical SSS framework for MUASs. In the meantime, SSSDF can be considered as a projection of known guidelines about software systems safety to MUASs, to specifically generate more efficient, practical, comprehensive, motivating, safety-oriented, easy and accurate development process. In this study, simplicity and efficiency of the SSSDF is shown clearly after SSSDF is successfully implemented to a deviced MUAS software system prototype.

SSSDF simplifies design and coding processes of MUAS's software development life

cycle as it provides smooth and straightforward methodologies which apply logical, abstract and flow-based development steps to decrease complications and increase understandability. This distinguishing nature of the SSSDF contributes to elimination of risks which might be caused by the complex architectural design and coding. Furthermore, safety contaminated regions in the design can be easily identified and traced by human eyes as SSSDF's approach encourages system developers to periodically review the software's flow to cover safety-related requirements. Although SSSDF does not suggest strict fault-identification and fault-tolerant algorithms to be applied, it is perfectly capable to integrate corresponding mechanisms into itself to achieve much confident software systems. Focusing on the reliability of the MUAS's software systems, SSSDF proves that it can handle single-point failures and isolate safety contaminated regions from entire system. Additionally, isolated software components recover themselves when failure conditions occur. Together with increased reliability and reduced risks of single point failures, SSSDF improves robustness of the entire software system as it spreads these practice to entire system and includes safety notion in each development step.

Verification, test and assessment activities performed during the SSS realization cycle are the most overwhelming, time taking and costly operations not only for MUASs but also for bigger manned/unmanned systems. The SSSDF proposed in this study handles these difficult tasks by effective and practical solutions by inventing a unique Assassin Process Method (APM) as a tool to reach previously mentioned safety goals. Being a comprehensive, user friendly, easy and automatic tool, APM accomplishes its objectives easily due to its mission-based, real-time, integrated-to-system and flow-oriented test and verification approaches. Moreover, human readable (understandable) safety assessments which are generated by the APM's Assessment GUI increases usability and accuracy of APM for small MUAS development teams. With this nature of APM, software test assessments including requirement coverage and traceability analysis can be performed with "a Blink".

During prototyping, SSSDF is strictly followed to generate MUAS's SS with safety objectives that are identified for the entire system. The success and accuracy of the framework is investigated using the prototype for whether SSSDF can achieve SSS by using the framework and by spending reasonable effort and time. Furthermore,

prototype software is tested automatically in HILS environment using APM for software requirements to increase confidence of the system and to reduce the identified software system risks. Satisfactory APM test results and assessment reports are accomplished as SSSDF claims to provide by APM Assessment GUI. Hence, SSSDF and its APM approach are evaluated for being an accurate, comprehensive and practical tool to realize safe software system development for MUASs.

Considering all the features and contributions of the SSSDF and its APM approach, additional evaluations can be performed for APM in the future to improve their confidence levels. As APM injects additional process into the software system's integration environment, it is expected to have a performance impact on the entire software system due to its CPU utilization and memory usage. In this study, this challenge is neglected as system executions with APM performs without causing any unexpected situations and malfunctions. However, from software engineering perspective, APM's exact perfomance impact has to be assessed, and so that has to be remarked as a futurework. On the other hand, as critical systems are tested for their executions under stress loads, this APM concept can also be considered as a kind of stress testing. Restating the fact that, APM tests, which are performed in this thesis study and prototyping, did not cause an obvious inaccuracy for system's executions and performance. Although APM's process is not expected to have a significant perfomance impact on the entire system, its effects have to be evaluated in the future studies to take every system's effecting parameters under control.

In conclusion, SSSDF framework has successfully applied SSS development philosophy to small unmanned aerial system with its specific methodologies and tools. Design, coding and integration activities of SSS development are all handled by comprehensive SSSDF with minimum effort proving that it is practical, efficient and accurate to reduce risks and achieve safer software system. Moreover, the unique APM approach contributes safety of an entire software system at any development phase with its easy usability, automatic operability, and mission-based safety-critical requirements' verifications at real-time hardware integration environment. Human understandable APM assessments for software requirements' tests increases motivation of developers to include safety concept into their development process as SSSDF and APM handles safe development objectives with systematic, straightforward and easy

instructions. Drawing a parallel with small unmanned system development paradigm, SSSDF approach suggested in the thesis is perfectly capable of handling safe software development tasks and increases entire system safety satisfactorily with less resource consumption.

REFERENCES

[1] Joint software systems safety engineering handbook, 2010.

[2] M. A. Al-Jarrah, S. Adiiansyah, Z. Marji, and M. Chowdhury. Autonomous aerial vehicles, guidance, control and signal processing platform. In *Systems, Signals and Devices (SSD), 2011 8th International Multi-Conference on*, pages 1–17. IEEE, 2011.

[3] A. Al-Radaideh, M. Al-Jarrah, A. Jhemi, and R. Dhaouadi. Arf60 aus-uav modeling, system identification, guidance and control: Validation through hardware in the loop simulation. In *Mechatronics and its Applications, 2009. ISMA'09. 6th International Symposium on*, pages 1–11. IEEE, 2009.

[4] D. A. Burke. System level airworthiness tool: A comprehensive approach to small unmanned aircraft system airworthiness. 2010.

[5] Z. Cakir. Development of aa uav testbed, 2011.

[6] R. S. Christiansen. *Design of an autopilot for small unmanned aerial vehicles*. PhD thesis, Brigham Young University, 2004.

[7] B. P. Douglass. *Doing hard time: developing real-time systems with UML, objects, frameworks, and patterns*, volume 1. Addison-Wesley Professional, 1999.

[8] P. Douglass. Safety-critical systems design. *Electronic engineering*, 70(862):45–6, 1998.

[9] J. Elston, B. Argrow, and E. Frew. A distributed avionics package for small uavs. *AIAA Infotech@ Aerospace*, pages 733–742, 2005.

[10] T. K. Ferrell et al. The avionics handbook ed. cary r. spitzer boca raton, crc press llc. 2001. 2001.

[11] W. J. Fitzpatrick. Unmanned aircraft system software airworthiness, 2012.

[12] L. D. Gowen, J. S. Collofello, and F. W. Calliss. Preliminary hazard analysis for safety-critical software systems. In *Computers and Communications, 1992. Conference Proceedings., Eleventh Annual International Phoenix Conference on*, pages 501–508. IEEE, 1992.

[13] J. S. Jang and C. Tomlin. Design and implementation of a low cost, hierarchical and modular avionics architecture for the dragonfly uavs. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, 2002.

[14] D. Jung Soon Jang Liccardo. small uav automation using mems. *Aerospace and Electronic Systems Magazine, IEEE*, 2007.

[15] D. B. Kingston. *Implementation issues of real-time trajectory generation on small uavs*. PhD thesis, Citeseer, 2004.

[16] R. Loh, Y. Bian, and T. Roe. Uavs in civil airspace: safety requirements. *Aerospace and Electronic Systems Magazine, IEEE*, 24(1):5–17, 2009.

[17] NASA. Sofwtare safety guidebook, 2004.

[18] Y. C. Paw. *Synthesis and validation of flight control for UAV*. PhD thesis, UNIVERSITY OF MINNESOTA, 2009.

[19] M. Quigley, B. Barber, S. Griffiths, and M. A. Goodrich. Towards real-world searching with fixed-wing mini-uavs. Technical report, DTIC Document, 2005.

[20] T. Schultz. Meeting do-178b software verification guidelines with coverity integrity center. *Coverity, Inc., San Francisco*, 588, 2009.

[21] A. Singh. Rtca do-178b (eurocae ed 12-b). `http://www.slideshare.net/iankits/do178bed12b-presentation`, 2011.

[22] Y.-R. Tang and Y. Li. The software architecture of a reconfigurable real-time onboard control system for a small uav helicopter. In *Ubiquitous Robots and Ambient Intelligence (URAI), 2011 8th International Conference on*, pages 228–233. IEEE, 2011.

[23] H. ul Azad, D. V. Lazic, and W. Shahid. Fpga based longitudinal and lateral controller implementation for a small uav.

[24] F. Yan, J. Yang, and P. Wang. Study of safety design of avionics software in civil aviation. In *Intelligent System Design and Engineering Application (ISDEA), 2010 International Conference on*, volume 2, pages 425–429. IEEE, 2010.

APPENDIX A

REQUIREMENTS PROCESS

A.1 System Description Document (Initial System Requirements)

MINI UNMANNED AERIAL SYSTEM (System Specifications Document)

System Components

(a) Unmanned Aerial Vehicle(UAV): It should be complete in all aspects of electronics, sensors, platform, control mechanism, on-board camera as a payload, trans-receivers , on-board batteries and packing case etc.

(b) Ground Control Station: It should be complete in all aspects of hardware and software to launch, navigate, trans-receive data, monitor UAV and GCS current state, land and to have video from on-board camera as well as to complete mission.

Physical Characteristics

(a) It should be man portable and light weight but also be sturdy not to be damaged easily.

(b) It should have sufficient safety features for operators, crew and equipment's in the vicinity as well as in the environments which have possible civilization interaction.

(c) It should be able to sustain minor adverse weather conditions such that light wind, low humidity, light drizzle, etc. and recover safety.

(d) It should be packed in a way that environmental adverse effects can be eliminated.

Operational Characteristics

(a) It should have the capability to take-off and land without any need to runway.

(b) It should have the capability to flight autonomously with the mission that is pre-programmed in GCS and the GCS undertakes the autonomous operations.

(c) Unmanned Aerial System should have the capability to communicate through wireless communication between UAV and GCS.

(d) Each subsystem's flight related parameters should be able to observe through GCS.

(e) All on-board systems should be powered from rechargeable batteries to satisfy desired endurance.

(f) It should have autonomous return home, etc. provisions to satisfy safety/control.

(g) It should have the capability of transmitting stable, real-time, continuous videos and still images from payload to GCS within its operational range.

(h) It should have the some sort of caution/advisory mechanism together with fail safe mechanism.

Operation and Maintenance

(a) **Ease of Operation and Maintenance:** It should be easy and simple to operate. All components should be modular in nature with connectors for interchange ability.

(b) **Self Diagnostic Check:** It should be capable of automatic hardware and software self-checks every time before operation.

(c) **Ruggedness:** It should be sturdy and robust for everyday handling and usage by soldiers.

A.2 Autopilot System Technical Requirements

AUTOPILOT SYSTEM TECHNICAL REQUIREMENTS DOCUMENT

1 Air Vehicle Operational Conditions

1.1 Autopilot System shall be able to control the air vehicle under the following conditions.

Altitude	Min. 300m AGL (Above Ground Level)
	Max. launch at 2500m AMSL (Above Mean Sea Level)
Range	Min. 5 km
Endurance	Min. 30 minutes
Speed Range	45 -130 km/h speed range
Wind Speed	Up to 25km/h
Take-off Type	Hand Launched
Landing Type	Belly Landing
Power Supply	• Re-chargeable on-board battery • Battery with 12-18V dc, Max 1A

Figure A.1: Table Of Specifications

2 Ground Equipment & Data Link

2.1 Items of Ground Equipment and Data Link

2.1.1 Control/Telemetry Data Link Equipment for 2.4 GHz frequency range shall be supplied.

2.1.2 Data Link

2.1.2.1 Control / Telemetry range

- The control/telemetry communication range shall not be less than 5 Kilometers.

2.1.2.2 Control / Telemetry Transmission

- Control Telemetry Link shall be duplex and shall include the following information at minimum:

  Uplink

  a) Mission Plan
  b) Flight Control Commands including velocity, altitude and heading commands.

c) Hearth Beat Message

Downlink

a) Vehicle Status Information including mission plan status, engine on/off status.
b) System Flight Status (System OK, System Failure etc.)
c) System Telemetry
d) Hearth Beat Message

2.2 Ground Control Software Capabilities

2.2.1 Ground Control Software operator interface shall offer language customization.

2.2.2 The Ground Control Software shall provide means to create a mission plan to perform flight.

2.2.3 The Ground Control Software shall monitor mission plan status.

2.2.4 The Ground Control Software shall provide means to reconfigure mission plan during flight.

2.2.5 In real time, the Ground Control Software shall monitor the following on the map/graph displayed at Ground Control Software at minimum:

a) Geographical position of the Aerial Vehicle, Target and the Ground Control Software.
b) The slant range difference between the Ground Control Software and the Aerial Vehicle.
c) The payload sight (dashed on the map)

2.2.6 The Ground Control Software shall monitor following information at minimum:

a) Vehicle status including sensors status and data link connected status, engine status, and operational limits exceeding status.
b) Aerial Platform Altitude AMSL
c) Aerial Platform Altitude AGL
d) Navigation Solution (A/C attitude and position)
e) AGL Information (0-150m range at minimum)
f) Sensor Telemetry

2.2.7 The GCS shall calculate and monitor the bearing angle in degrees.

2.2.8 The Ground Control Software shall record telemetry data.

2.2.9 Recorded telemetry data shall be replayed upon request by the operator at the Ground Control Software.

2.2.10 GCS software shall represent AC orientation (pitch and roll) using the artificial horizon.

3 Environment

3.1 The Control / Telemetry data monitoring rate on the GCS shall not be less than 1Hz.

4 Autonomy

4.1 Autonomous Operations

4.1.1 Control system accuracy / attitude accuracy shall be accomplished by dedicated control algorithm in real-time.

4.1.2 Autopilot system shall perform AFCS algorithms to control A/C.

4.1.3 Autopilot system shall control A/C starting from it is powered to A/C lands and unpowered.

4.1.4 In autonomous operation, the aerial vehicle shall only perform autonomous flight for the GCS created and sent mission plan.

4.1.5 Autopilot shall flight A/C after it receives “Take-off” command from GCS.

4.1.6 Autopilot system shall enable A/C to take-off if system works properly for during confident time.

4.1.7 Autopilot system shall be ready flight at maximum 2 minutes.

4.1.8 In autonomous operation mode, upon a single command at the ground control Software, the mission shall be aborted and the Aerial Vehicle shall perform “Landing” operation.

4.1.9 In autonomous operation mode, upon a single command at the GCS, the A/C shall abort its ongoing mission and start executing the new mission transmitted from the GCS through the data link.

4.1.10 In autonomous mode, altitude at waypoint should be reconfigurable from the Ground Control Software.

4.1.11 In autonomous mode, airspeed at waypoint should be reconfigurable from the Ground Control Software.

4.1.12 Autonomous system shall periodically notify GCS about its status as “OK” or “Failure”.

4.1.13 Control system shall apply fault tolerance for minimum 2 seconds loss of sensor data accuracy.

4.1.14 Control system shall have Emergency Landing Mode when system has non-tolerable failures.

4.1.15 Autopilot system shall record system status information in a log file.

4.1.16 Autopilot system shall record sensor and control data in a log file.

4.2 Autonomy In Link Loss Conditions

4.2.1 In case of link loss system shall notify GCS.

4.2.2 In all modes, upon link loss for a user specified time, the aerial vehicle shall have the capability to perform autonomous landing to the take-off location ("back to home") or to another user specified landing point. SYSTEM should perform autonomous landing.

4.2.3 In case of GPS signal loss, SYSTEM should follow Emergency Landing procedure.

5 Development

5.1 Authorized operator should be able to change gain values/fine tune parameters form Ground Control Software in case of a change on the platform.

A.3 System Safety Assessment

A.3.1 Preliminary Hazard List

Hazard No	Hazard Name	Hazard Causal Factor
1	Errant Air Vehicle	1) Error in Flight Control SW 2) Error in sensors 3) Error in servos 4) Error in Navigation SW 5) Mechanical failure in flight control fins 6) Maintenance faults 7) Intentionally send improper user commands – user faults 8) Failure in avionic processing components/HW 9) Loss of communication
2	Errant Flight on the Flight Path	1) Error in Flight Control SW 2) Error in sensor 3) Error in servos 4) Error in Navigation SW 5) Mechanical failure in flight control fins 6) Intentionally send improper user commands – user faults 7) Failure in avionic processing components/HW 8) Failure to receive user command 9) Loss of communication 10) Operational/Environmental limits exceed (strong wind etc.)
3	Exceeding operational flight range	1) Error in Flight Control SW 2) Error in sensor 3) Error in servos 4) Error in Navigation SW 5) Failure to monitor A/C position in GCS 6) Failure to receive user command 7) Operational/Environmental limits exceeds 8) Loss of communication

4	Errant Take-off	1) User fault during launching 2) Error in Flight Control SW 3) Error in sensors 4) Error in servos 5) Error in Navigation SW 6) Mechanical failure in flight control fins 7) Intentionally send improper user commands – user faults 8) Failure in avionic processing components /HW 9) Operational/Environmental limits exceed (strong wind etc.)
5	Flight without mission plan	1) Inadvertent erase mission plan during flight 2) During re-planning erase the mission plan and fail to install new plan 3) Failure reading memory segment that keeps mission plans 4) Termination of process which reads/writes mission plan 5) Intentionally send improper user commands – user faults 6) Failure to monitor the completion of installing mission plan 7) Failure to check missing mission plan 8) Loss of communication
6	Failure to re-configure mission plan	1) During re-planning erase the mission plan and fail to install new plan 2) Termination of process which reads/writes mission plan 3) Intentionally send improper user commands – user faults 4) Loss of communication
7	Failure to process “go back home” command	1) Loss of communication 2) Failure to receive/identify command message 3) Failure to send command from GCS 4) Error in SW implementing communication protocol

8	Errant Landing	1) Error in Flight Control SW 2) Error in sensors 3) Error in servos 4) Error in Navigation SW 5) Mechanical failure in flight control fins 6) Intentionally send improper user commands – user faults 7) Failure in avionic processing components /HW 8) Failure in parachute 9) Failure in airbag mechanism 10) Operational/Environmental limits exceed (strong wind etc.) 11) Loss of communication 12) Failure to receive user command 13) Failure to send command from GCS 14) Failure to monitor A/C position in GCS 15) Inadvertent erase mission plan during flight 16) Missing landing mission plan
9	Failure to store take-off position	1) Inadvertent erase mission plan during flight 2) Error in sensors 3) Error in Navigation SW 4) Failure in avionic processing components/HW
10	Exhaust Battery	1) Improper batteries 2) Not completely charged batteries 3) Error in hardware which causes extra battery consumption 4) Error in battery monitoring
11	Loss of communication between autopilot and GCS	1) Failure in heartbeat messages 2) Error in data link HW 3) Error in SW implementing communication protocol 4) Failure in entire SW components (autopilot and GCS) 5) Failure in OS to handle high load SW operations

12	Failure to monitor autopilot flight mode status	1) Loss of communication 2) Failure in GCS 3) Error in SW implementing communication protocol 4) Failure in the autopilot status sending process 5) Failure in autopilot to detect its status
13	Inadvertent Take-off mode on	1) Loss of communication 2) Intentionally send improper user commands – user faults 3) Error in SW implementing communication protocol
14	Inadvertent Landing	1) Loss of communication 2) Intentionally send improper user commands – user faults 3) Error in Flight Control SW 4) Error in sensors 5) Error in SW implementing communication protocol
15	Inadvertent Emergency kill engine mode on	1) Intentionally send improper user commands – user faults 2) Error in Flight Control SW 3) Error in Navigation SW 4) Error in sensors 5) Error in SW implementing communication protocol
16	Failure to activation of kill engine mode off	1) Loss of communication 2) Intentionally send improper user commands – user faults 3) Error in Flight Control SW 4) Error in SW implementing communication protocol
17	Failure to monitor for flight mode changes	1) Loss of communication 2) Failure in GCS 3) Error in SW implementing communication protocol

18	Processing inadvertent velocity, altitude and heading commands at autopilot	1) Error in SW implementing communication protocol 2) Fail to check data appropriateness 3) Errors in GCS implementation for numeric data command inserts 1) Intentionally send improper user commands – user faults
19	Failure to process user velocity, altitude and heading commands	1) Loss of communication 2) Error in SW implementing communication protocol 3) Intentionally send improper user commands – user faults
20	Failure to send heartbeat message	1) Loss of communication 2) Error in SW implementing communication protocol
21	Failure to receive heartbeat message	1) Loss of communication 2) Error in SW implementing communication protocol
22	Failure to modify gain values parameters	1) Loss of communication 2) Fail to check data appropriateness 3) Failure at autopilot memory to save changes 4) Error in SW implementing communication protocol 5) Failure in GCS
23	Failure to receive heartbeat message	1) Loss of communication 2) Fail to check data appropriateness 3) Intentionally send improper user commands – user faults 4) Ability to change at flight time

A.4 Functional Hazard Analysis

NO	Major Component - SubSystem	Functionality	Interfaces	Consequences of Loss of Function	Severity of Consequence	Safety-Significant Functions	Mishaps of Hazards	Safety-Significant SW Modules
1.0	Autopilot System	a) Flight Control b) Autonomous Flight c) Engine Control d) Control Logic e) Flight Stability f) Data acquisition g)Computational Processing Unit h) Process User Commands i)Flight Data Logging j)Mission operations	a) A/C Platform b) Engine c) on-board Power Supply d)Communication Component e)Sensors f)Processing Unit (HW) g) Servos	a)Loss of A/C – A/C crash b) Dead/Injury c)Environmental Damage	Catastrophic	a,b,c,d,e,f,g,h ,j	Deaths or injury of an innocents or loss of equipment due to the crash	a)Navigation SW component b)AFCS SW Control c)Operating system d)Sw collecting sensor data e)Sw Creating Servo Signal f)Communication Protocol g)Determination of use cases h)State transitions
1.1	Navigation SW	a)Applying Navigation Algorithm using sensor data b)Provide A/C	a)Sensors b)AFCS c)Communication Component	a)Loss of navigation b)Loss of control c)Loss of A/C	Catastrophic	a,b,c	Deaths or injury of an innocents or loss of equipment	a) Algorithm implementation b)Sensor data collection

Figure A.2: Functional Hazard Analysis

NO	Major Component - SubSystem	Functionality	Interfaces	Consequences of Loss of Function	Severity of Consequence	Safety-Significant Functions	Mishaps of Hazards	Safety-Significant SW Modules
		attitude data through telemetry link c)Data acquisition from other system components		d)Dead/Injury e)Environmental Damage			due to the crash	c)NAV Output data generation d)Data flow between functions e)Identification of sensor states f)Data validation/ erroneous data check
1.2	AFCS SW	a)Applying Control Algorithm using sensor and 1.1 data b)Provides data to servos to control A/C fins c)Stabilizes the A/C during autonomous flight	a)Nav Sw b)Servos c) Communication Component	a)Loss of control b)Loss of A/C c)Dead/Injury d)Environmental Damage	Catastrophic	a, b, c,	Deaths or injury of an innocents or loss of equipment due to the crash	a) Algorithm implementation b)Output data generation for servos c)Input data collection from Nav d)Data flow between functions e)Data validation/ erroneous data check

Figure A.3: Functional Hazard Analysis - Continue

NO	Major Component - SubSystem	Functionality	Interfaces	Consequences of Loss of Function	Severity of Consequence	Safety-Significant Functions	Mishaps of Hazards	Safety-Significant SW Modules
1.3	Mission Control SW	a)Mission plan create, modify, delete b)Provide heartbeat SW messages to GCS c)Controls data flow between autopilot SW components d)Identifies autopilot states/modes e)Apply mission related procedures (safety procedures, fault tolerant etc.) f)Processing user commands						GCS: a)Mission plan send, modify and delete functions b)User validation functions c)Heartbeat implementing function d)Functions displaying autopilot states/modes e)Functions creating cautions, warning to user f)User command sending functions Autopilot: a)Mission plan receive functions b)Heartbeat implementing function c)Interfaces between SW components d)Autopilot states functions e)Sending telemetry

Figure A.4: Functional Hazard Analysis - Continue

NO	Major Component - SubSystem	Functionality	Interfaces	Consequences of Loss of Function	Severity of Consequence	Safety-Significant Functions	Mishaps of Hazards	Safety-Significant SW Modules
								functions f)Functions processing user commands
2.0	Sensors (HW)	a) Collects data for Nav and AFCS algorithms b)Identifying A/C current state	a)Nav SW b)Processing Unit (Hardware)	a)Loss of navigation b)Loss of control c)Loss of A/C d)Dead/Injury e)Environmental Damage f)Loss of user awareness for current A/C status	Catastrophic	a,b	Deaths or injury of an innocents or loss of equipment due to the crash	a)Nav Sw b)AFCS Sw
3.0	Processing Unit (HW&SW)	a)Sensor data acquisition b)Nav&AFCS algorithm computation c)Real-time data processing d)Data logging	a)Nav SW b)AFCS SW c)Sensors and Servos (Hardware)	a)Loss of sensor data acquisition b)Loss of navigation c)Loss of control d)Loss of A/C e)Dead/Injury	Catastrophic	a,b,c,e,	Deaths or injury of an innocents or loss of equipment due to the crash	a)Data acquisition b)Algorithm implementation c)Real-time data processing d)Comm protocol implementation

Figure A.5: Functional Hazard Analysis - Continue

NO	Major Component - SubSystem	Functionality	Interfaces	Consequences of Loss of Function	Severity of Consequence	Safety-Significant Functions	Mishaps of Hazards	Safety-Significant SW Modules
		e)Communication SW implementation to receive data from GCS and to send data to GCS		f)Environmental Damage				
4.0	Servos (HW)	a)Run the flight control fins	a)AFCS SW b)Processing Unit (Hardware)	a)Loss of control b)Loss of A/C c)Dead/Injury d)Environmental Damage	Catastrophic	a	Deaths or injury of an innocents or loss of equipment due to the crash	a)Data generation for servos to run fins correctly.

Figure A.6: Functional Hazard Analysis - Continue

A.5 Preliminary Hazard Analysis (PHA)

System: Small Unmanned Aircraft System			Preliminary Hazard List			Analyst: Önder Altan Date:14 July 2014
No	Hazard Description	Severity	Causal Factors	Effects and Mishaps	Phase or Mode	Recommended Action
1	Errant Air Vehicle	catastrophic	1) Error in Flight Control SW 2) Error in sensors 3) Error in servos 4) Error in Navigation SW 5) Mechanical failure in flight control fins 6) Maintenance faults 7) Intentionally send improper user commands – user faults 8) Failure in avionic processing components/HW 9) Loss of communication	Deaths or injury of an innocents or loss of equipment due to the crash	Operational	1,4,7,9) Identify safety-critical SW functions 2,3.a) Determine the error detection time 2,3.b) Determine the exposure time 2,3.c) Alarm the error on the monitoring system 2,3.d) Use reliable sensors 2,3.e) checksum the sensor data 3,5,6) Maintenance and pre-flight checks for mechanical component shall be performed by trained users 7.a) Identify hazardous user commands 7.b) For hazardous user commands, determine cautions 8.a) Specify effecting factors of hardware failures 8.b) Apply authority's hardware quality assurance procedure 8.c)Apply self-diagnostic method for HW 9) Determine scenarios for flight

Figure A.7: Functional Hazard Analysis - Continue

System: Small Unmanned Aircraft System			Preliminary Hazard List			Analyst: Önder Altan Date:14 July 2014
						control and apply safety assurance procedures for functions operate these scenarios.
2	Errant Flight on the Flight Path	catastrophic	1) Error in Flight Control SW 2) Error in sensor 3) Error in servos 4) Error in Navigation SW 5) Mechanical failure in flight control fins 6) Intentionally send improper user commands – user faults 7) Failure in avionic processing components/HW 8) Failure to receive user command 9) Loss of communication 10) Operational/Environmental limits exceed (strong wind etc.)	Deaths or injury of an innocents or loss of equipment due to the crash	Operational	1,4,7,9) Identify safety critical SW functions 1) Determine the exposure time 2,3.a) Determine the error detection time 2,3.b) Determine the exposure time 2,3.c) Alarm the error on the monitoring system 2,3.d) Use reliable sensors 2,3.e) checksum the sensor data 4.a) Identify the mishaps cause by different sensor errors 4.b) Identify tolerance time for navigation error 2,3,5) Maintenance and pre-flight checks for mechanical component shall be performed by trained users 6.a) Identify hazardous user commands 6.b) For hazardous user commands, determine cautions

Figure A.8: Functional Hazard Analysis - Continue

System: Small Unmanned Aircraft System	Preliminary Hazard List					Analyst: Önder Altan Date:14 July 2014
						6.c) Validate flight path correctness before operation 7.a) Specify effecting factors of hardware failures 7.b) Apply authority's hardware quality assurance procedure 8.a)Identify mishaps/failures 8.b)Identify scenario to prevent 8.a mishaps 9.a) Determine scenarios for flight control and apply safety assurance procedures for functions operate these scenarios 9.b) Validate the flight pathis the one used in one of the scenarios 10.Before flight check weather conditions
3	Errant Take-off	catastrophic	1) User fault during launching 2) Error in Flight Control SW 3) Error in sensors 4) Error in servos 5) Error in Navigation SW 6) Mechanical failure in flight control fins 7) Intentionally send	Deaths or injury of an innocents or loss of equipment due to the crash	Operational	2,5,7) Identify safety critical SW functions 1.a) Identify a take-off procedure 1.b) Identify a take-off checklist 1.c) Educate user for proper take-off procedure 2,4,6) Create a pre-take-off test procedure to ensure the flight

Figure A.9: Functional Hazard Analysis - Continue

System: Small Unmanned Aircraft System			Preliminary Hazard List				Analyst: Önder Altan Date:14 July 2014
			improper user commands – user faults 8) Failure in avionic processing components /HW 9) Operational/Environmental limits exceed (strong wind etc.)				control commands work properly 3.a) Identify sensor data error conditions 3.b) Alarm user for sensor data error 3.c) Do not let user to take-off with sensor data error 5.a) Alarm user for nav errors 5.b) Do not let user to take-off with nav error 7.a) Educate users for take-off commands 7.b) Determine cautions for engine power on 8) Validate that HW works properly before take-off 9.a)Determine a procedure for pre-takeoff which identifies environmental appropriateness 9.b)Before flight check weather conditions 9.c)During flight observe weather conditions
4	Errant Landing	catastrophic	1) Error in Flight Control SW 2) Error in sensors	Deaths or injury of an innocents or loss of equipment due to the	Operational		1,4,5,11,12,13,14,15,16) Identify safety critical SW functions

Figure A.10: Functional Hazard Analysis - Continue

System: Small Unmanned Aircraft System			Preliminary Hazard List			Analyst: Önder Altan Date:14 July 2014
			3) Error in servos 4) Error in Navigation SW 5) Mechanical failure in flight control fins 6) Intentionally send improper user commands – user faults 7) Failure in avionic processing components / HW 8) Failure in parachute 9) Failure in airbag mechanism 10) Operational/Enviromental limits exceed (strong wind etc.) 11) Loss of communication 12) Failure to receive user command 13) Failure to send command from GCS 14) Failure to monitor A/C position in GCS 15) Inadvertent erase mission plan during flight 16) Missing landing mission plan	crash		1.a) Determine the exposure time 1.b) Determine the tolerance time 1.c) Determine landing start/ends conditions 1.d) Engine power off commands time for ground touch shall be determined 1.e) Who will impose Engine power off command during landing (user/flight control SW) shall be determined 2,4.a) Determine the error detection time 2,4.b) Determine the exposure time 2,4.c) Alarm the error on the monitoring system 2,4.d) Use reliable sensors 2,4.e) checksum the sensor data 2,4.f) For invisible landings validate that errors for sensor and nav are eliminated 4) Maintenance and pre-flight checks for mechanical component shall be performed by trained users 6.a) Determine a landing procedure for user 6.b) Hazardous user commands

Figure A.11: Functional Hazard Analysis - Continue

System: Small Unmanned Aircraft System	Preliminary Hazard List			Analyst: Önder Altan Date:14 July 2014
				shall be eliminated during landing 6.c) Identify Engine power off command time for user 7.a) Specify effecting factors of hardware failures 7.b) Apply authority's hardware quality assurance procedure 8.a) Parachute usage procedure shall be determined 8.b) Validate that parachute mechanics works properly 8.c) Validate that parachute eliminates the hazard 9.a) Airbag usage procedure shall be determined 9.b) Validate that airbag mechanics works properly 9.c) Validate that airbag eliminates the hazard 10) Determine a procedure for landing which identifies environmental appropriateness and scenarios 11) Determine scenarios for flight control and apply safety assurance procedures for functions operate these scenarios

Figure A.12: Functional Hazard Analysis - Continue

APPENDIX B

DESIGN PROCESS

B.1 Initial Concept Design

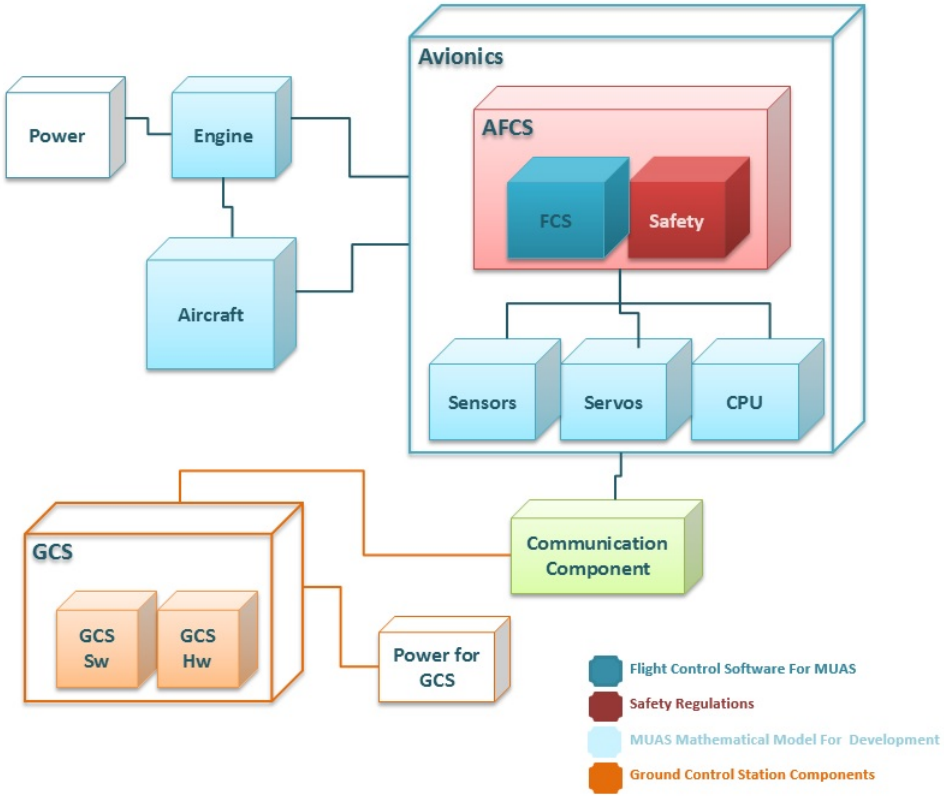


Figure B.1: Initial Concept Design

B.2 Detail Design

B.2.1 Detail Design Overview - Activity Diagram

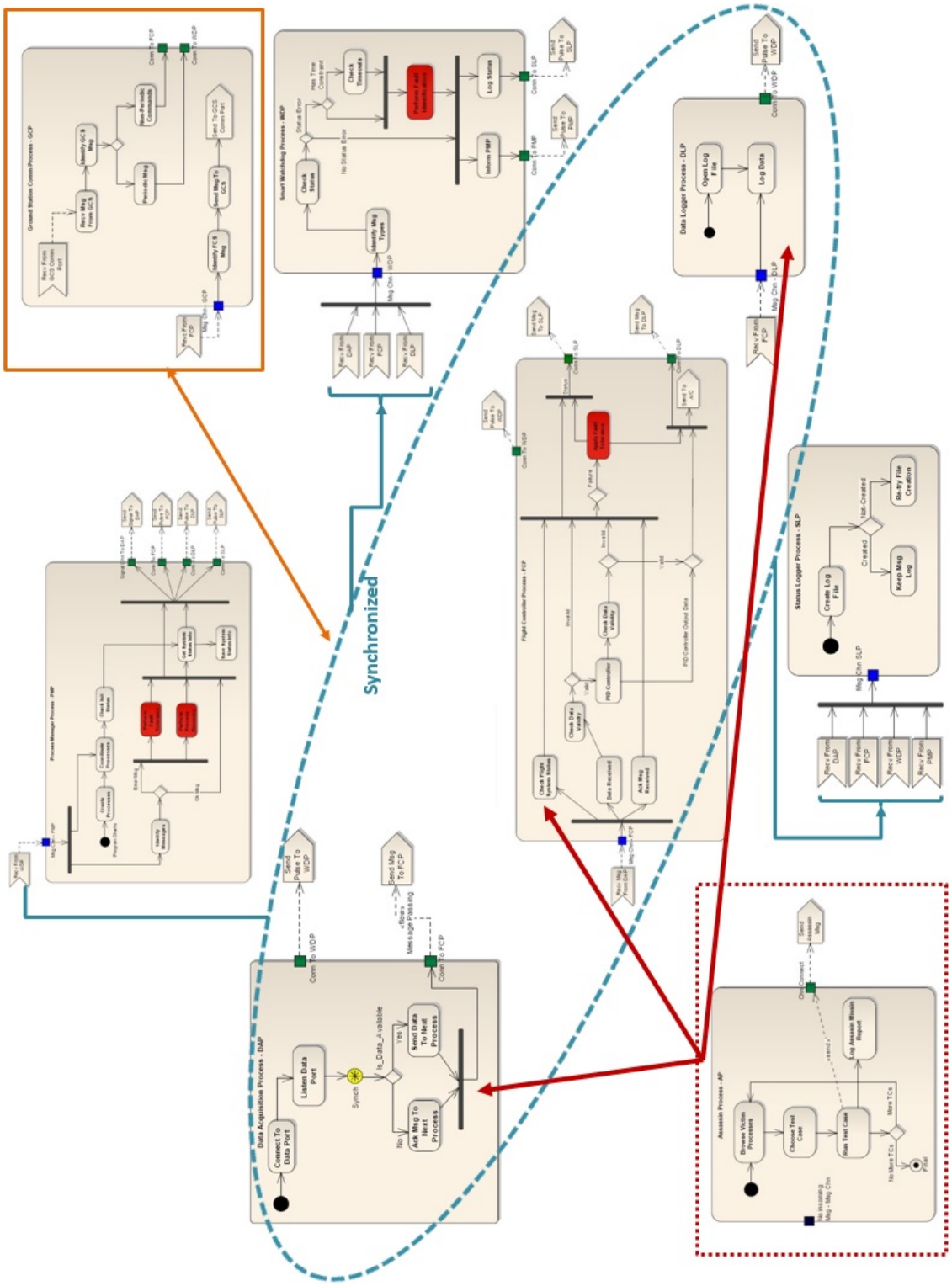


Figure B.2: Detail Design Activity Diagram

B.2.2 Data Acquisition Process - DAP

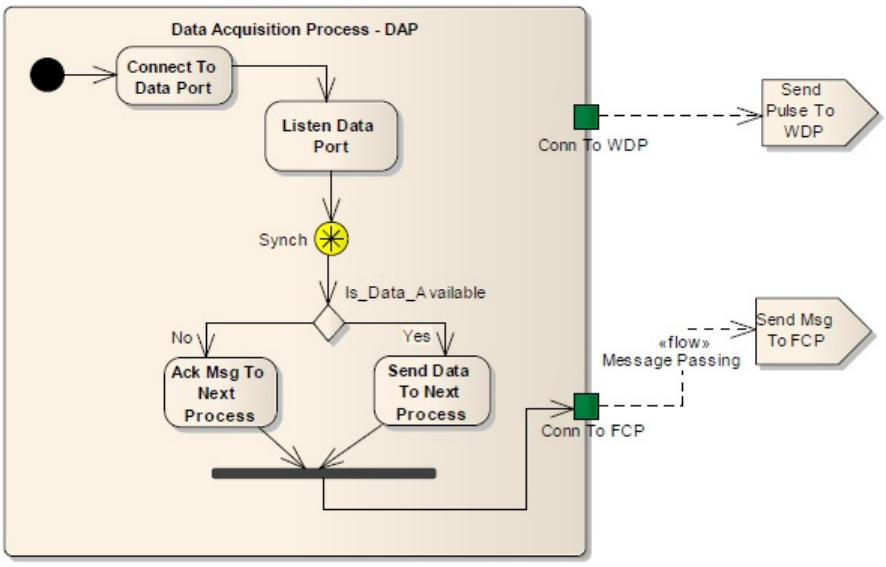


Figure B.3: Data Acquisition Process - DAP

B.2.3 Flight Controller Process - FCP

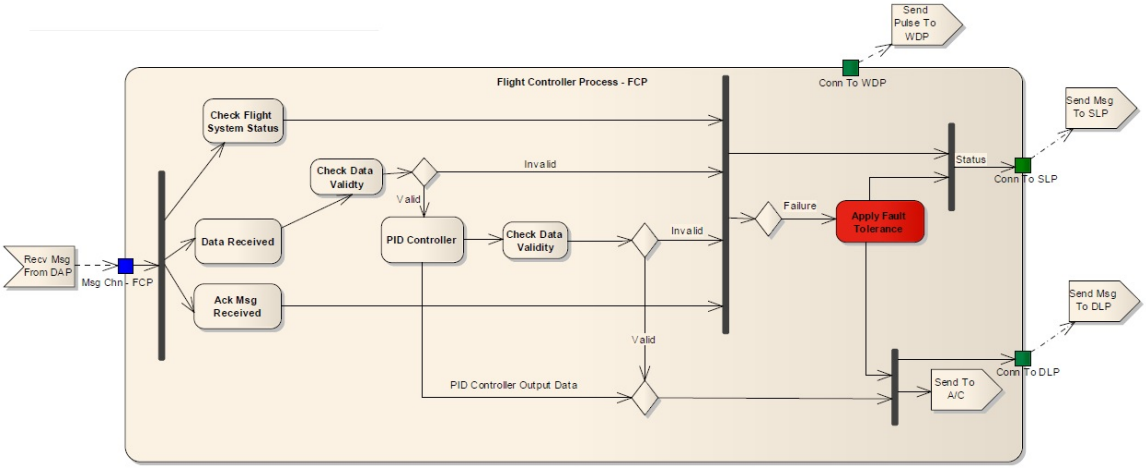


Figure B.4: Flight Controller Process - FCP

B.2.4 Data Logger Process - DLP

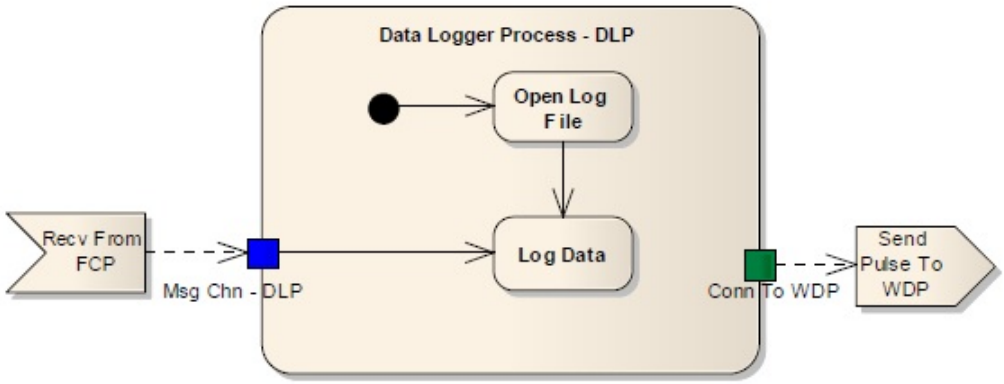


Figure B.5: Data Logger Process - DLP

B.2.5 Smart Watchdog Process - WDP

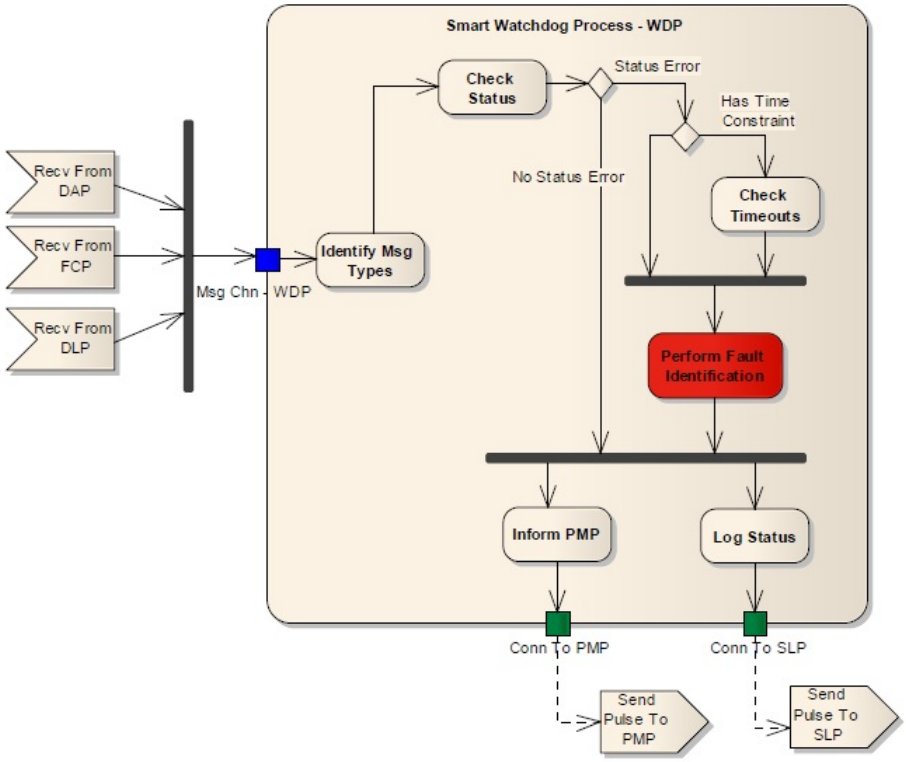


Figure B.6: Smart Watchdog Process - WDP

B.2.6 Process Manager Process - PMP

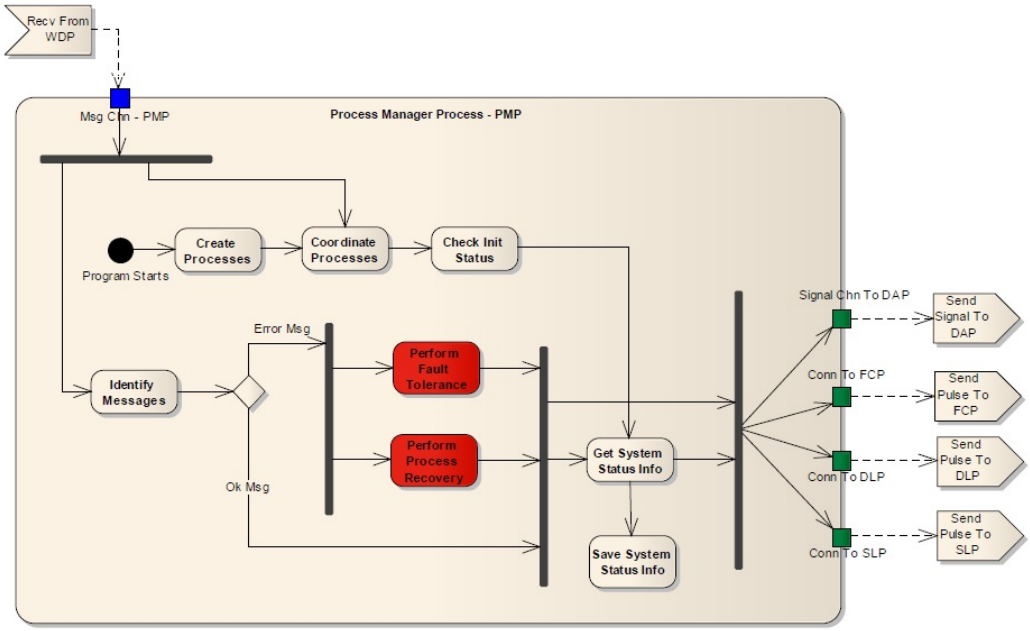


Figure B.7: Process Manager Process - PMP

B.2.7 Status Logger Process - SLP

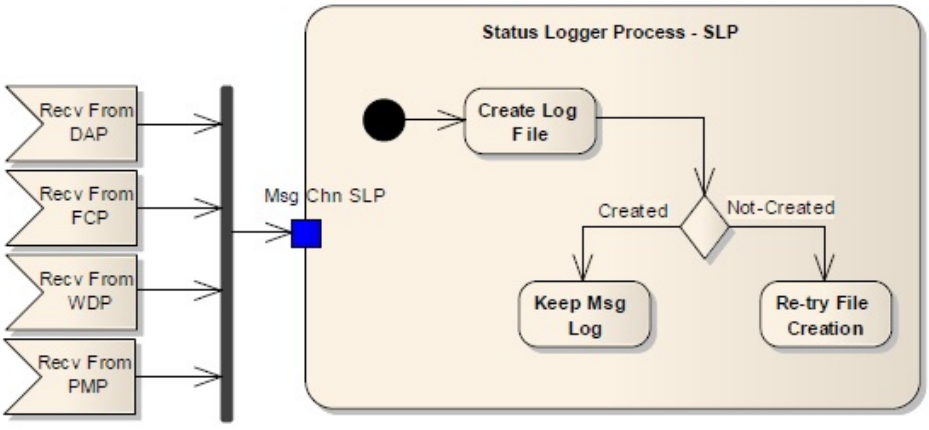


Figure B.8: Status Logger Process - SLP

B.2.8 GCS Communication Process - GCP

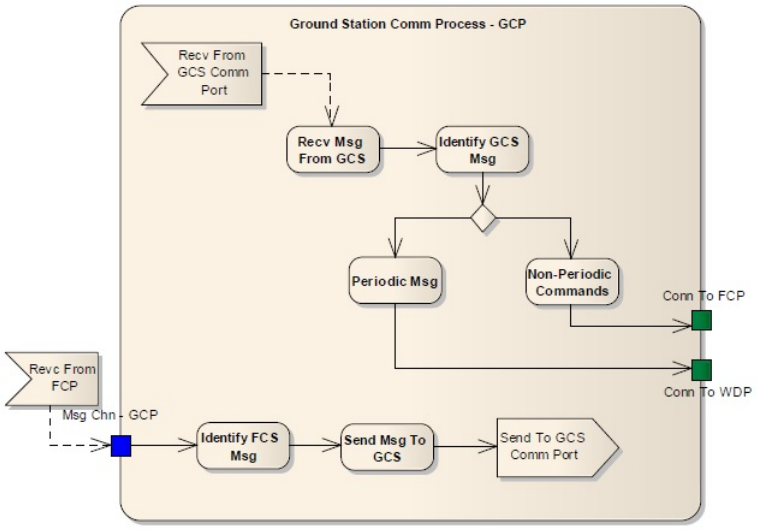


Figure B.9: GCS Communication Process - GCP

B.3 Assassin Process Method- APM

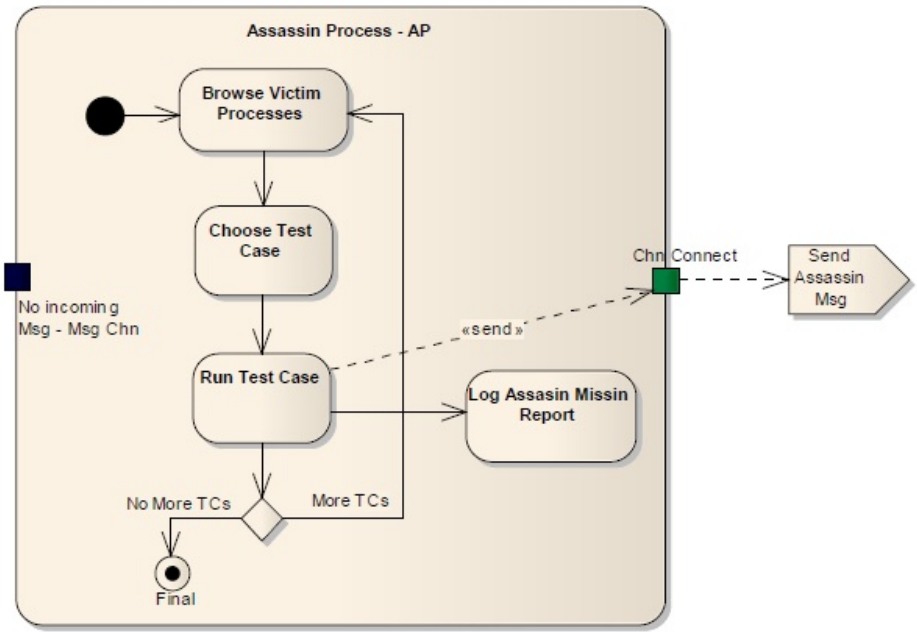


Figure B.10: Assassin Process Method - APM

APPENDIX C

CODING & INTEGRATION PROCESS

C.1 Software Requirements & Test Cases

No	Software Requirements	Sys. Req. No.	Safety-Critical
	DEFINITIONS AND TERMS		
	Autopilot Software System, ASS: ASS is the software which handles all flight related operation on a MUAS automatically by software itself. **Aircraft, A/C:** Aircraft is the intended small aerial vehicle for which the ASS is developed during the MUAS development process. **Flight Control Software, FCS:** FCS is the subcomponent of ASS which handles flight control algorithm computations and algorithm related operations.		
	AUTOPILOT SOFTWARE SYSTEM (ASS) FLIGHT MODES		
1.	ASS shall have the following flight modes during the operation. • Pre-Flight • Cruise • Landing	1.1, 4.1.3	
2.	Pre-Flight Mode is the first mode of software system when it is powered and A/C is on the ground.	4.13	
3.	Cruise Mode is the second mode after Pre-Flight when A/C is flying autonomously.	4.1.3	
4.	Landing Mode is the mode when A/C quits cruise mode and performs landing operation.	4.1.3	
	ASS FLIGHT SAFETY MODES		
5.	• Fail-Safe • Emergency Landing	4.1.12, 4.1.13, 4.1.14	S.C
6.	Fail-Safe Mode is a flight mode when system detects failure/failures in the system which might cause A/C to crash, loss, damage, injury etc. and tries to prevent/minimize effect/effects of failure/failures.	4.1.12, 4.1.13, 4.1.14	S.C
7.	Emergency Landing is a mode when A/C starts to land due to the failure conditions which occur during flight.	4.1.12	S.C
	ASS FEATURES		
8.	ASS shall start its operation at Pre-Flight mode when system power is given for the first time.	4.1.3	
8.1	TC-1: 1. Verify system mode status is Pre-Flight Mode when the power to the system is given first time.		
9.	ASS shall operate at least 30 minutes.	1.1	
9.1	TC-2: 1. Verify that ASS executes at least 30 minutes without halting after system is powered.		
10.	ASS shall be ready at minimum 2 minutes to operate.	4.1.7	
10.1	TC-3: Verify that ASS initialization process and Pre-Flight duration take maximum 2 minutes. 1. Power on system. 2. Start time count from power on to beginning of READY_TO_TAKEOFF status. 3. Check time duration is less than 2 minutes.		

Figure C.1: Test Cases & Software Requirements, Part-1

No	Software Requirements	Sys. Req. No.	Safety-Critical
11.	ASS shall receive heart beat message from GCS at 1Hz.	3.1	S.C
11.1	TC-4: Verify that ASS receive heartbeat message from GCS at 1 HZ. 1. After power on, wait for 30 seconds and count the number of received heartbeat messages. 2. Verify that ASS received 30 heartbeat messages		
12.	ASS shall change its flight mode to Fail-Safe when there is no heart beat message coming from GCS for 15 seconds.	4.2.3	S.C
12.1	TC-5: Verify that ASS changes its safety mode to Fail-Safe when sequential heartbeat messages are missing for 15 seconds. 1. Close connection between GCS and ASS for 15 second and verify that ASS safety mode changes Fail-Safe.		
13.	ASS shall notify GCS about flight mode of the FCS during flight.	4.1.12, 4.2.1	S.C
13.1	TC-6: Verify ASS notifies GCS about ASS's flight status. 1. Verify ASS sends Pre-Flight mode status to GCS after power on. 2. Verify ASS sends Cruise mode status when Take-Off command received. 3. Verify ASS sends Landing Mode status when Land command is received.		
14.	ASS shall notify GCS about safety mode of the FCS during flight.	4.1.12, 4.2.1	S.C
14.1	TC-7: 1. Verify ASS sends Fail-Safe mode status when is in Fail-Safe mode. 2. Verify ASS send Emergency Landing mode status when system is in Fail-Safe Mode for 2 seconds and in Emergency Landing mode.		
	Pre-Flight Mode Features		
15.	ASS shall notify GCS that system mode is "Pre-Flight".	4.1.3	S.C
	TC-8: 1. Verify ASS sends Pre-Flight mode status when is in Pre-Flight mode.		
16.	ASS shall be able to apply "Take-off" command received from GCS when it is in the Pre-Flight Mode.	4.1.5	S.C
16.1	TC-9: 2. Verify system is not in Pre-Flight mode. 3. Send take-off command to FCS 4. Verify FCS does not apply Take-off Command.		
16.2	TC-10: 1. Verify system is in Pre-Flight mode. 2. Send take-off command to FCS 3. Verify FCS applies Take-Off Command.		
17.	ASS shall initialize FCS at Pre-Flight Mode.	4.1.6	
17.1	TC-11: 1. Power on the system.		

Figure C.2: Test Cases & Software Requirements, Part-2

No	Software Requirements	Sys. Req. No.	Safety-Critical
	2. Count for 15 seconds. 3. Check Pre-Flight Initialization Status is OK.		
17.2	TC-12: 1. Verify that FCS valid mission plan. 2. Prevent sensor data arrivals to FCS. 3. Initiate Pre-Flight check operations. 4. Verify FCS Pre-Flight Mode is Failure.		
17.3	TC-13: 1. Verify that FCS valid mission plan. 2. Send invalid data to FCS. 3. Initiate Pre-Flight check operations. 4. Verify FCS Pre-Flight Mode is Failure.		
18.	ASS shall notify GCS when FCS initialization status as OK when initialization is completed without any failure.	4.1.6	S.C
18.1	TC-14: 1. Verify FCS generates Pre-Flight initialization status message when initialization is OK.		
19.	ASS shall notify GCS when FCS initialization status as ERROR when initialization is completed with at least one failure.	4.1.6,	S.C
19.1	TC-15: 1. Verify FCS generates Pre-Flight initialization status message when initialization has failed.		
20.	ASS shall be able to operate a user-defined mission plan written in mission plan file.	4.1.4	
20.1	TC-16: 1. Send Mission Plan to FCS. 2. Verify FCS operates at altitude and speed defined on mission plan.		
21.	ASS shall be able to store a user-defined mission plan.	4.1.4	S.C
21.1	TC-17: 1. Send Mission Plan to FCS. 2. Verify FCS saved the mission plan.		
22.	ASS shall permit flight operation only a mission plan is stored on the ASS.	4.1.4	S.C
22.1	TC-18: 1. After power on and Pre-Flight initialization status is ok, verify that there is no mission plan saved in FCS. 2. Send Take-Off command. 3. Verify FCS denied take-off.		
22.2	TC-19: 1. After power on and Pre-Flight initialization status is ok, send invalid mission plan to FCS.		

Figure C.3: Test Cases & Software Requirements, Part-3

No	Software Requirements	Sys. Req. No.	Safety-Critical
	2. Send Take-Off command. 3. Verify FCS denied take-off.		
22.3	TC-20: 1. After power on and Pre-Flight initialization status is ok, send valid mission plan to FCS. 2. Send Take-Off command. 3. Verify FCS accepted take-off command and FCS flight mode is cruise.		
23.	ASS shall notify GCS that operation is not permitted without any mission plan on ASS.	**4.1.4**	**S.C**
23.1	TC-21: 1. Verify that FCS send notification in case Take-Off command is rejected due to the improper mission plan status.		
24.	ASS shall fly at altitude defined at mission plan.	**4.1.4, 4.1.9, 4.1.10**	**S.C**
24.1	TC-22: 1. Verify that A/C flights at the altitude defined in mission plan.		
25.	ASS shall fly at speed defined at mission plan.	**4.1.4, 4.1.9, 4.1.11**	**S.C**
25.1	TC-23: 1- Verify that A/C flights with the speed defined in mission plan.		
26.	ASS shall check sensor data format and validity for 30 seconds in the Pre-Flight mode and store the status of it.	**4.1.4, 4.1.6**	**S.C**
26.1	TC-24: 1. Power on the system. 2. Count for 30 seconds. 3. Verify FCS denied receives valid data from sensors. 4. Verify that Pre-Flight data validity status is OK.		
26.2	TC-25: 1. Change FCS's status to power on status. 2. Send invalid data for 30 seconds. 3. Count for 30 seconds. 4. Verify that Pre-Flight data validity status is Error.		
27.	ASS shall only permit flight operation when FCS initialization and data format/validity checks are performed and GCS user "Take-Off" command is received.	**4.1.6, 4.1.5**	**S.C**
27.1	TC-26: 1. Put FCS into power on state. 2. Send valid mission plan.		

Figure C.4: Test Cases & Software Requirements, Part-4

No	Software Requirements	Sys. Req. No.	Safety-Critical
	3. Enable valid sensor data to arrive FCS. 4. Send Take-off command. 5. Verify that FCS is in Cruise Mode.		
	Cruise Mode Features		
28.	ASS shall notify GCS that system mode is "Cruise".	**4.1.3**	**S.C**
28.1	TC-27: 1. Verify that FCS send Cruise notification to GCS.		
29.	ASS shall perform FCS algorithms to control A/C's fins.	**4.1.1, 4.1.2**	**S.C**
29.1	TC-28: 1. Verify that FCS algorithm is implemented in to the ASS.		
30.	ASS shall receive sensor data at 200 Hz from corresponding hardware components to use it in FCS algorithm.	**4.1.1, 4.1.2**	**S.C**
30.1	TC-29: 1. Put FCS into power on state.		
31.	ASS shall check sensor data format and validity before using it in FCS.	**4.1.6, 4.1.13**	**S.C**
31.1	TC-30: 1. Send invalid sensor data to FCs algorithm. 2. Verify FCS identifies invalid data.		
32.	ASS shall check every input values of FCS's algorithm for • zero input value • Not-A-Number input value (NAN) • Infinity input value	**4.1.13**	**S.C**
32.1	TC-31: 3. Send invalid sensor data with ZERO values to FCS algorithm. 1. Verify FCS identifies invalid data and its invalidity type.		
32.2	TC-32: 4. Send invalid sensor data with NAN values to FCS algorithm. 2. Verify FCS identifies invalid data and its invalidity type.		
32.3	TC-33: 1. Send invalid sensor data with INFINITY values to FCS algorithm. 2. Verify FCS identifies invalid data and its invalidity type.		
33.	ASS shall check FCS's output data validity.	**4.1.13**	**S.C**
33.1	TC-34: 1. Send invalid sensor data to FCs algorithm.		

Figure C.5: Test Cases & Software Requirements, Part-5

No	Software Requirements	Sys. Req. No.	Safety-Critical
	2. Verify FCS identifies invalid data. 3. Verify that FCs algorithm output data is fault tolerant data.		
33.2	TC-35: 1. Send valid sensor data to FCs algorithm. 2. Check output data for its validity.		
34.	ASS shall check every output data of FCS for • zero input value • Not-A-Number input value • Infinity input value	4.1.13	S.C
34.1	TC-36: 1. Send invalid sensor data with ZERO values to FCS algorithm. 2. Verify FCS identifies invalid data and its invalidity type.		
34.2	TC-37: 1. Send invalid sensor data with NAN values to FCS algorithm. 2. Verify FCS identifies invalid data and its invalidity type.		
34.3	TC-38: 1. Send invalid sensor data with INFINITY values to FCS algorithm. 2. Verify FCS identifies invalid data and its invalidity type.		
35.	ASS shall use default FCS control gain values when system initialization.	4.1.6	S.C
35.1	TC-39: 1. Verify that FCS control gains do not contain zero values for Proportional, Integral and Derivative coefficients at the same time.		
36.	ASS shall be reliable for 5 second in case loss or invalid sensor data for FCS as an input.	4.1.13, 4.1.41	S.C
36.1	TC-40: 1. Change FCS data status to SENSOR_DATA_LOSS for 5 seconds. 2. Verify FCS identified SENSOR_DATA_LOSS condition. 3. Verify FCS changed its safety status to Fail-Safe.		
36.2	TC-41: 1. Send invalid ZERO valued data to FCS for 5 seconds. 2. Verify FCS identified SENSOR_DATA_LOSS condition. 3. Verify FCS changed its safety status to Fail-Safe.		
36.3	TC-42: 1. Send invalid ZERO valued data to FCS for 5 seconds.		

Figure C.6: Test Cases & Software Requirements, Part-6

No	Software Requirements	Sys. Req. No.	Safety-Critical
	2. Verify FCS identified INVALID_NAN_VALUE condition. 3. Verify FCS changed its safety status to Fail-Safe.		
36.4	TC-43: 1. Send invalid ZERO valued data to FCS for 5 seconds. 2. Verify FCS identified INVALID_INFINITY_VALUE condition. 3. Verify FCS changed its safety status to Fail-Safe.		
37.	ASS shall achieve reliability of FCS when no sensor data is available or data is invalid by using the last correct sensor data as an input to FCS.	4.1.13	S.C
37.1	TC-44: 1. Supply invalid data to FCS for 5 seconds. 2. Verify that FCS output data is updated with the last successful data during 5 second. 3. Verify FCS safety status changed to Fail-Safe.		
38.	ASS shall notify GCS when FCS has a failure in validity of FCS algorithm input and output data.	4.1.12	S.C
38.1	TC-45: 1. Change FCS safety status to Fail-Safe. 2. Verify FCS notifies GCS for Fail-Safe Status Msg.		
39.	ASS shall change its flight mode to Fail-Safe when sensor data to FCS is not available due to the data loss or invalid conditions.	4.1.13	S.C
39.1	TC-46: 1. Change FCS data status to SENSOR_DATA_LOSS for 5 seconds. 2. Verify FCS identified SENSOR_DATA_LOSS condition. 3. Verify FCS changed its safety status to Fail-Safe.		
39.2	TC-47: 1. Send invalid ZERO valued data to FCS for 5 seconds. 2. Verify FCS identified SENSOR_DATA_LOSS condition. 3. Verify FCS changed its safety status to Fail-Safe.		
39.3	TC-48: 1. Send invalid ZERO valued data to FCS for 5 seconds. 2. Verify FCS identified INVALID_NAN_VALUE condition. 3. Verify FCS changed its safety status to Fail-Safe.		
39.4	TC-49: 1. Send invalid ZERO valued data to FCS for 5 seconds. 2. Verify FCS identified INVALID_INFINITY_VALUE condition.		

Figure C.7: Test Cases & Software Requirements, Part-7

No	Software Requirements	Sys. Req. No.	Safety-Critical
	3. Verify FCS changed its safety status to Fail-Safe.		
40.	ASS shall generate error message to GCS when FCS output data generated for A/C's control surfaces is invalid. .	4.1.12	S.C
40.1	TC-50: 1. Create invalid FCS's algorithm output data condition. 2. Verify FCS notifies GCS when FCS's algorithm generates invalid data.		
41.	ASS shall change its flight mode to Fail-Safe when FCS algorithm's output data is not available.	4.1.13	S.C
41.1	TC-51: 1. Create invalid FCS's algorithm output data condition. 2. Verify FCS changes its Safety Status to Fail-Safe.		
42.	ASS shall log the sensor raw data to a log file during flight mission.	4.1.16	
42.1	TC-52: 1. Start ASS. 2. Save sensor data to log file. 3. At the end of the flight, count number of data samples and verify that its equal to the #executiontime*200		
43.	ASS shall log the FCS output data to a log file.	4.1.16	
43.1	TC-53: 1. Start ASS. 2. Save sensor data to log file. 3. At the end of the flight, count number of data samples and verify that its equal to the #executiontime*200		
44.	ASS shall log erroneous software conditions into a software status log file.	4.1.15	
44.1	TC-54: 1. Verify that for every error condition, corresponding error message is logged in the log file.		
45.	ASS shall be able to receive "Land" command from GCS.	4.1.3	S.C
45.1	TC-55: 1. TC-Send Land command to ASS. 2. Verify FCS flight mode status changed to "Landing".		
	Landing Mode Features		
46.	ASS shall apply landing procedure.	4.1.3, 4.1.8	
46.1	TC-56: 1. Verify FCS flight mode status changed to "Landing".		
47.	ASS shall notify GCS that system mode is "Landing".	4.1.12	S.C
47.1	TC-57: 1. Verify FCS notifies GSC as Flight mode is "Landing".		

Figure C.8: Test Cases & Software Requirements, Part-8

No	Software Requirements	Sys. Req. No.	Safety-Critical
	Safety Mode Features		
48.	ASS shall notify GCS when system changes its safety mode to Fail-Safe.	4.1.12	S.C
48.1	TC-58: 2. Create Fail-Safe conditions. 3. Verify FCS changes its safety status to Fail-Safe.		
49.	ASS shall only perform Emergency Landing after system stays in Fail-Safe mode for minimum 2 second.	4.1.13, 4.1.14	S.C
49.1	TC-59: 1. Change ASS safety status to Fail-Safe. 2. Count for 2 seconds. 3. Verify safety status changed to Emergency Landing.		
50.	ASS shall operate Emergency Landing procedure when FCS crash occurs for more than 5 seconds.	4.1.14	S.C
50.1	TC-60: 1. Crash FCS two software components at the same time. 2. Verify that FCS cannot recover itself for 5 seconds. 3. Verify safety status changed to Emergency Landing.		
51.	ASS shall stops engine in emergency landing mode.	4.1.14	S.C
51.1	TC-61: 1. Change safety status to Emergency landing. 2. Verify FCS throttle command is zero.		
52.	ASS shall notify GCS when system changes its safety mode to Fail-Safe Landing.	4.1.12	S.C
52.1	TC-62: 1. Change safety status to Emergency landing. 2. Verify FCS notifies GCS about Emergency Landing Mode status.		

Figure C.9: Test Cases & Software Requirements, Part-9

C.2 APM Assessment Text Files

C.2.1 APM Software Requirements Text File

Start: SoftwareRequirements.txt	Cont. - 1	Cont. - 2
REQ-8	+	+
REQ-8-TC-15	REQ-25	REQ-38
#	REQ-25-TC-15	REQ-38-TC-2
+	REQ-25-TC-17	REQ-38-TC-3
REQ-10	#	REQ-38-TC-4
REQ-10-TC-1	+	#
#	REQ-26	+
+	REQ-26-TC-1	REQ-43
REQ-11	#	REQ-43-TC-15
REQ-11-TC-2	+	#
#	REQ-27	+
+	REQ-27-TC-1	REQ-47
REQ-16	#	REQ-47-TC-2
REQ-16-TC-15	+	REQ-47-TC-3
#	REQ-31	REQ-47-TC-4
+	REQ-31-TC-3	REQ-47-TC-11
REQ-17	REQ-31-TC-4	REQ-47-TC-12
REQ-17-TC-15	REQ-31-TC-5	REQ-47-TC-13
#	#	#
+	+	+
REQ-20	REQ-35	REQ-48
REQ-20-TC-1	REQ-35-TC-3	REQ-48-TC-6
#	REQ-35-TC-4	#
+	REQ-35-TC-5	+
REQ-21	REQ-35-TC-11	REQ-51
REQ-21-TC-1	#	REQ-51-TC-6
REQ-21-TC-15	+	#
REQ-21-TC-18	REQ-36	
#	REQ-36-TC-11	
	#	

Figure C.10: APM Software Requirements Text File

C.2.2 APM Software Test Cases Text File

Start: APM_SoftwareTestCases.txt	Cont. -1	Cont. -2	Cont. -3
+	+	+	+
% PREFLIGHT TEST: TAKE-OFF	TC-8	% PREFLIGHT	TC-4
COMMAND WITHOUT	TC-8-1/104	TEST: REJECT	TC-4-1/102
% SUCCESFULL INITIALIZATION	TC-8-1/51	TAKE-OFF	TC-4-1/3010
STATUS	TC-8-4/5001	% DUE TO	TC-4-1/3011
TC-15	TC-8-1/5002	MISSING PLAN	TC-4-4/3011
TC-15-6/1516	TC-8-4/121	TC-1	#
TC-15-1/1516	TC-8-6/1516	TC-1-6/1616	+
TC-15-1/511	TC-8-1/1516	TC-1-1/1616	TC-5
#	TC-8-1/511	TC-1-1/1618	TC-5-1/104
+	#	TC-1-4/100	TC-5-1/3010
% PREFLIGHT TEST: INVALID	+	TC-1-1/50	TC-5-1/3011
MISSION PLAN	TC-9	TC-1-4/5001	TC-5-4/3011
TC-17	TC-9-1/103	TC-1-1/5002	#
TC-17-6/1616	TC-9-1/51	TC-1-6/1516	+
TC-17-6/1617	TC-9-4/5001	TC-1-1/1516	TC-11
TC-17-1/1619	TC-9-1/5002	TC-1-1/510	TC-11-4/30
#	TC-9-4/121	TC-1-1/3002	TC-11-1/101
+	TC-9-6/1516	TC-1-4/3002	TC-11-4/5
TC-14	TC-9-1/1516	#	#
TC-14-4/100	TC-9-1/511	+	+
TC-14-1/51	#	%% CRUISE:	TC-12
TC-14-4/5001	+	MISSIG	TC-12-4/31
TC-14-1/5002	TC-18	HEARTBEAT &	TC-12-4/6
TC-14-6/1516	TC-18-1/50	FAIL-SAFE	TC-12-1/3011
TC-14-1/1516	TC-18-4/5001	% MODE CHANGE	TC-12-4/3011
TC-14-1/511	TC-18-1/5002	TC-2	#
#	TC-18-6/1616	TC-2-1/100	+
+	TC-18-6/1617	TC-2-1/3011	TC-13
TC-7	TC-18-1/1619	TC-2-4/3011	TC-13-4/32
TC-7-1/102	TC-18-6/1516	#	TC-13-1/32
TC-7-1/51	TC-18-1/1516	+	TC-13-4/7
TC-7-4/5001	TC-18-1/511	% CRUISE:	#
TC-7-1/5002	#	INVALID NAN	+
TC-7-4/121		DATA INPUT TO	TC-6
TC-7-6/1516		FCS	TC-6-1/2501
TC-7-1/1516		TC-3	TC-6-1/3010
TC-7-1/511		TC-3-1/103	TC-6-4/3010
#		TC-3-1/3010	TC-6-1/3004
		TC-3-1/3011	TC-6-4/3004
		TC-3-4/3011	#
		#	

Figure C.11: APM Software Test Cases Text File

C.2.3 APM Assassination Mission Report Text File

Start: AssassinationReport.txt	Cont. - 1
+ TC-15 # + TC-17 # + TC-14 # + TC-7 # + TC-8 # + TC-9 # + TC-18 # + TC-1 #	+ TC-2 # + TC-3 # + TC-4 # + TC-5 # + TC-11 # + TC-12 # + TC-13 # + TC-6 #

Figure C.12: APM Assassination Mission Report Text File