

POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Aerospaziale



Tesi di Laurea Magistrale

Application of Systems Engineering processes, methods and tools to the ECS system (Environmental Control System for UAV aircraft) and integration with parametric cost estimates

Relatori

Prof.ssa Nicole Viola
Ing. Marco Fioriti
Prof. Paolo Maggiore

Candidato

Alessandro Chierchia

Tutor aziendali

Dott.ssa Elena Valfrè
Ing. Gianni Mancuso
Ing. Fabrizio Grasso

15 Ottobre 2019

CONTENTS

Introduction.....	Errore. Il segnalibro non è definito.
1. Systems Engineering	5
1.1 The Systems Engineering Approach	6
1.2 The V-Diagram.....	7
1.3 Costs in Systems Engineering	7
2. The ECS in UAVs	9
2.1 Unmanned Aerial Vehicles (UAVs).....	9
2.2 UAVs classification.....	10
2.3 The Environmental Control System	11
2.4 The Air Cycle Machine	11
2.5 The Vapor Cycle Machine	13
3. Functional analysis	15
3.1 Model Based Systems Engineering (MBSE)	15
3.2 Systems Modeling Language (SysML).....	15
3.3 SysML Diagrams.....	16
3.4 The ECS Model.....	17
4. Performance Analysis.....	29
4.1 Thermal Flow Analysis	29
4.2 Vapor Cycle Simulation	31
4.3 Air Cycle Simulation.....	37
5. Cost Analysis.....	41
5.1 Design to Cost	41
5.2 Project Definition	43
5.3 Cost Methodology	44
5.4 Cost Estimate.....	50
5.5 Hardware Cost.....	52
5.6 Software Cost	53
5.7 Software Cost Estimate Methodologies	55
5.8 Estimation Technique.....	64
6. Cost Estimate Applied to the ECS for UAVs.....	66
6.1 Scheduling.....	66
6.2 Product Breakdown Structures.....	67

6.3 Model Inputs for Hardware72

6.4 Transformation to a COTS base model.....74

6.5 Model Inputs for Software77

6.6 Results80

6.7 Other methods applications81

7. Conclusions.....96

FIGURE INDEX

Figure 1. Scheme of the System Engineering Approach	4
Figure 2. System of Systems composition	5
Figure 3. V-Diagram	7
Figure 4. Life Cycle Cost diagram.....	8
Figure 5. MQ-1 Predator UAV	9
Figure 6. Air Cycle Machine schematization.....	12
Figure 7. Vapor Cycle schematization.....	13
Figure 8. Relation between UML and SysML	16
Figure 9. SysML diagrams hierarchy.....	16
Figure 10. Processes iterations.....	19
Figure 11. Use Case diagram for the ECS under analysis	20
Figure 12. Activity Diagram related to the Provide Air Conditioning Use Case	22
Figure 13. Sequence Diagram related to the Provide Air Conditioning Use Case	23
Figure 14. Internal Block Diagram related to the Provide Air Conditioning Use Case	24
Figure 15. Statechart Diagram for the Provide Air Conditioning Use Case.....	25
Figure 16. Control Panel	26
Figure 17. Activity Diagram related to the Provide Air Conditioning Use Case White Box View ..	26
Figure 18. Sequence Diagrams of the Provide Air Conditioning Use Case White Box View	27
Figure 19. Statechart Diagram White Box View	28
Figure 20. Wall Temperature in function of B and C	30
Figure 21. Mission profile in terms of altitude	30
Figure 22. Mission profile in terms of Mach	31
Figure 23. Vapor Cycle architecture	32
Figure 24. Thermodynamic cycle for the Vapor Cycle in the hot case	32
Figure 25. Thermal flow in the avionic bay in the hot case.....	33
Figure 26. Thermal flow produced detail.....	33
Figure 27. Recovery temperature and external temperature in the hot case	34
Figure 28. Temperature in the avionic bay for the Vapor Cycle in the hot case	34
Figure 29. Thermodynamic cycle of the Vapor Cycle in the cold case	35
Figure 30. Thermal flow in the avionic bay in the cold case for Vapor Cycle.....	35
Figure 31. Thermal flow detail in the avionic bay for the cold case for Vapor Cycle.....	36
Figure 32. Recovery temperature and external temperature in the cold case	36
Figure 33. Temperature in the avionic bay for the Vapor Cycle in the cold case.....	37
Figure 34. Air Cycle architecture.....	37
Figure 35. Thermal flow in the avionic bay in the hot case for Air Cycle	38
Figure 36. Thermal flow detail in the avionic bay for the hot case for Air Cycle	38
Figure 37. Temperature in the avionic bay for the Air Cycle in the hot case	39
Figure 38. Thermal flow in the avionic bay in the cold case for Air Cycle.....	39
Figure 39. Thermal flow detail in the avionic bay in the cold case for Air Cycle.....	40
Figure 40. Temperature in the avionic bay for the Air Cycle in the cold case	40
Figure 41. Project triangle.....	41
Figure 42. Cost estimation phases.....	42

Figure 43. Inflation trend through the years	45
Figure 44. Cost estimation methodologies related to the Program Life Cycle	46
Figure 45. Parametric Methodology phases ref[3].....	47
Figure 46. Parametric cost estimation process.....	48
Figure 47. Labor Learning curve	50
Figure 48. Relation between SLOC and FP in terms of production	54
Figure 49. Learning curve for software estimate	62
Figure 50. Rayleigh curve.....	62
Figure 51. Worksheet Sets	66
Figure 52. Phase Sets	67
Figure 53. Parametric Cost Estimation tool library	68
Figure 54. Product Breakdown Structure for the Vapor Cycle.....	70
Figure 55. Product Breakdown Structure for the Air Cycle	71
Figure 56. Inputs for the Hardware Components.....	72
Figure 57. Hardware specifications for structure.....	73
Figure 58. Hardware specifications for electronics	74
Figure 59. Transition to the COTS components model	75
Figure 60. COTS components inputs.....	75
Figure 61. Labor Learning Curve	76
Figure 62. Use of the outputs of the first model as inputs	77
Figure 63. Software inputs	77
Figure 64. COSMIC methodology process.....	78
Figure 65. Acquisition of the COSMIC Function Points from the Sequence Diagrams	79

TABLES INDEX

Table 1. System Requirements.....	18
Table 2. Traceability Matrix	21
Table 3. SLOC/FP conversion ratio	54
Table 4. Values of a and b depending on the model type	56
Table 5. COCOMO cost drivers	57
Table 6. COCOMO II cost drivers.....	59
Table 7. COSYSMO cost drivers.....	60
Table 8. RE/FP ratio basing on the model type	65
Table 9. Vapor Cycle COSMIC Function Points.....	79
Table 10. Ai Cycle COSMIC Function Points.....	80
Table 11. Vapor Cycle results obtained in the Parametric Cost Estimation tool.....	80
Table 12. Air Cycle results obtained in the Parametric Cost Estimation tool	81
Table 13. SLOC estimations	82
Table 14. Values of a and b depending on the model type for the ECS	82
Table 15. COCOMO cost drivers for Vapor Cycle	87
Table 16. COCOMO cost drivers for Air Cycle	88
Table 17. COCOMO results.....	89
Table 18. COCOMO II cost drivers for Vapor Cycle.....	89
Table 19. COCOMO II cost drivers for Air Cycle	90
Table 20. COCOMO II results	91
Table 21. COSYSMO cost drivers for Vapor Cycle.....	92
Table 22. COSYSMO cost drivers for Air Cycle	93
Table 23. COSYSMO size drivers for Vapor Cycle.....	94
Table 24. COSYSMO size drivers for Air Cycle.....	94
Table 25. COSYSMO results.....	94
Table 26. Results comparison	94

INTRODUCTION

The present work illustrates a case of application of processes, methodologies and tools of Systems Engineering in the life cycle of an Environmental Control System (ECS) on an Unmanned Aircraft Vehicle (UAV) considering two technical solutions:

- Vapor Cycle Machine;
- Air Cycle Machine.

It has been developed in collaboration with Leonardo Aircraft Division. In particular, the different phases of the work have been followed adopting the company methodologies/tools and validated by experts of “Engineering System & Configuration Management”, “Aircraft Systems” and “Independent and Parametric Costing”.

The results obtained in this work are not related to any of Leonardo Aircraft Division projects/programs.

The main task is to show the process of development of the system through:

- Functional Analysis supported by means of IBM Ration Rhapsody® applying the IBM Harmony® methodology;
- Performance Analysis supported by means of Siemens Simcentre Amesim®;
- Cost Estimate supported by means of PRICE TruePlanning®.

In addition, a study on methodologies related to parametric software cost estimation is presented, since nowadays systems need software that controls its functionalities. From the analysis of the state of art (COSMIC Function Points, COCOMO, COCOMO II, COSYSMO, etc.), a comparison between some selected parametric software cost estimation methods has been implemented.

In particular, this document focuses on the aspects related to costs. In fact, the cost has not always been one of the main targets considered in the first phases of the project, but as years go by costs are gaining more relevance, in order to obtain higher profits maintaining a competitive cost on the market. In fact, if the development/production costs result to be excessive, the project will be modified in order to reduce the costs, but this will cause an increase of the development cost and a delay in the delivery of the products.

Instead, a Design to Cost methodology will provide a cost estimation since the first phases of the project fixing a target cost. This methodology will focus on a parametric cost estimation of the whole lifecycle of the project by using a commercial tool. In order to calibrate the automatic parametric estimation, two Leonardo Aircraft Division specialists have been interviewed and a Delphi approach has been adopted to obtain an estimate of the number of Source Lines of Codes required by the software.

The project development needs a multidisciplinary approach, since results must be a compromise to optimize all the aspects of the project.

The whole process is of iterative and multidisciplinary nature, since all the disciplines will exchange data with the others. In fact, as the project gains maturity, always more information are

available and, so, it is possible to return to previous steps to refine the model and to go in more detailed and interdisciplinary analysis.

In addition to the disciplines presented in this work (Functional, Performance and Cost Analysis), another target of the analysis is the Safety Assessment, since the reliability of a system will influence all the others aspects of the project. For example, the level of safety of a system will influence the capabilities of the system regarding the functionalities provided, the level of performance and the operative costs.

A Safety Analysis of the system has been covered in parallel in the thesis work “Model based approach of an environmental control system for an unmanned air vehicle”.

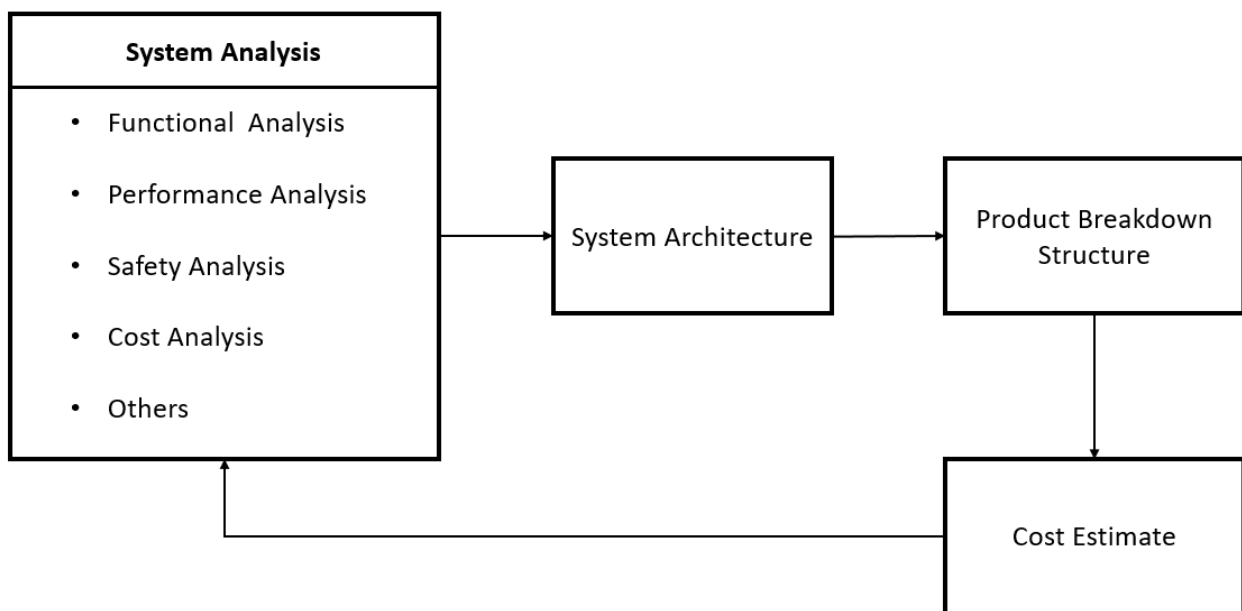


Figure 1. Scheme of the System Engineering Approach

1. SYSTEMS ENGINEERING

A system is an ensemble of independent elements with variable complexity that must provide the achievement of a specific target performing a specific functionality, which cannot be achieved by the single elements. Its complexity comes from relations and interfaces between the elements of the system.

The elements that form the system can be of different types, such as hardware, software, human, documentation, etc.

The interconnections and the interfaces between every element give the system a higher value.

A system can be defined and evaluated in terms of complexity, defined by the number of components and their interactions, and in terms of performances.

Moreover, a system can be part of a bigger ensemble called *System of Systems (SoS)*, where:

- every system works independently from other systems;
- every system has a different life cycle;
- the system requirements depends on System of Systems;
- adding an element bring to an higher complexity of the system;
- the System of System evolves continuously, as the technological level goes on.

Therefore, a System of System is an ensemble of systems called sub-systems that cooperate and exchange data to achieve a common target.

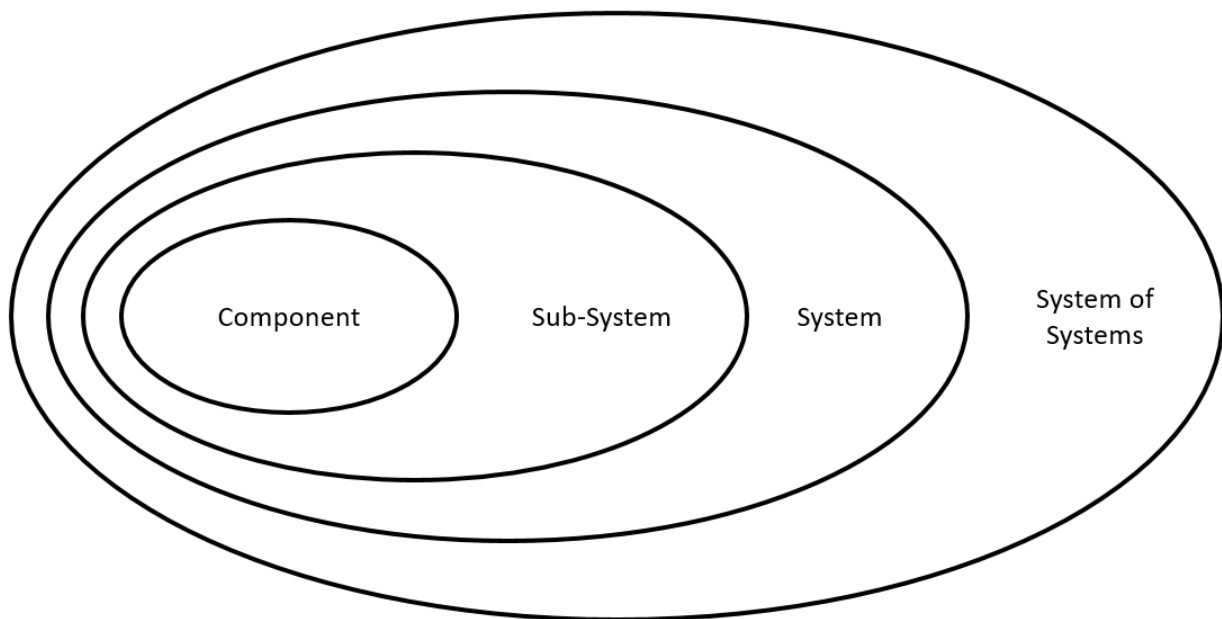


Figure 2. System of Systems composition

1.1 The Systems Engineering Approach

Systems Engineering is an interdisciplinary approach that allows the realization of a system, focusing on:

- customer needs;
- functionality required by the system in the first phase of design;
- requirement documentation;
- architecture definition;
- system validation.

Therefore, System Engineering integrates various disciplines to create a process that goes from the ideation to the realization of the system.

In addition, it considers the design process from an economical point of view, with the target to provide a product that satisfies the customer with the best profit margin.

Systems Engineering is based on the ability to face and resolve a problem, before seeing it from a technical point of view. In fact, in a first time a problem is seen from a global point of view, to identify and to handle relations and dependencies between the parts of the entire system.

Through this process, it is possible to predict and to control the interactions between the components that bring to the global behavior. So many qualities are needed, such as specialist competence in every field, large understanding of the disciplines, and skills in system approaching, open mind and attitude to deepening.

The International Council on Systems Engineering (INCOSE) synthesized the fundamental activities of the system engineering in:

- understand the problem;
- define and handle requirements;
- investigate different solutions;
- define and handle interfaces;
- testing and supporting the system;
- tracing progress compared to the schedule.

To understand Systems Engineering, it is possible to see it as a method to improve system performances and efficiency. It is possible to apply this method to any system at any complexity level.

In addition, this method leads to a reduction in costs among the benefits. In fact, the entire process goes through an iterative analysis that allows obtaining a product optimized in terms of costs, performance, risks and time.

1.2 The V-Diagram

This iterative process can be clearly explained introducing the *V-Diagram*. This diagram is strongly based on Systems Engineering, and shows how to design a system from the conceptual design phase to the disposal phase.

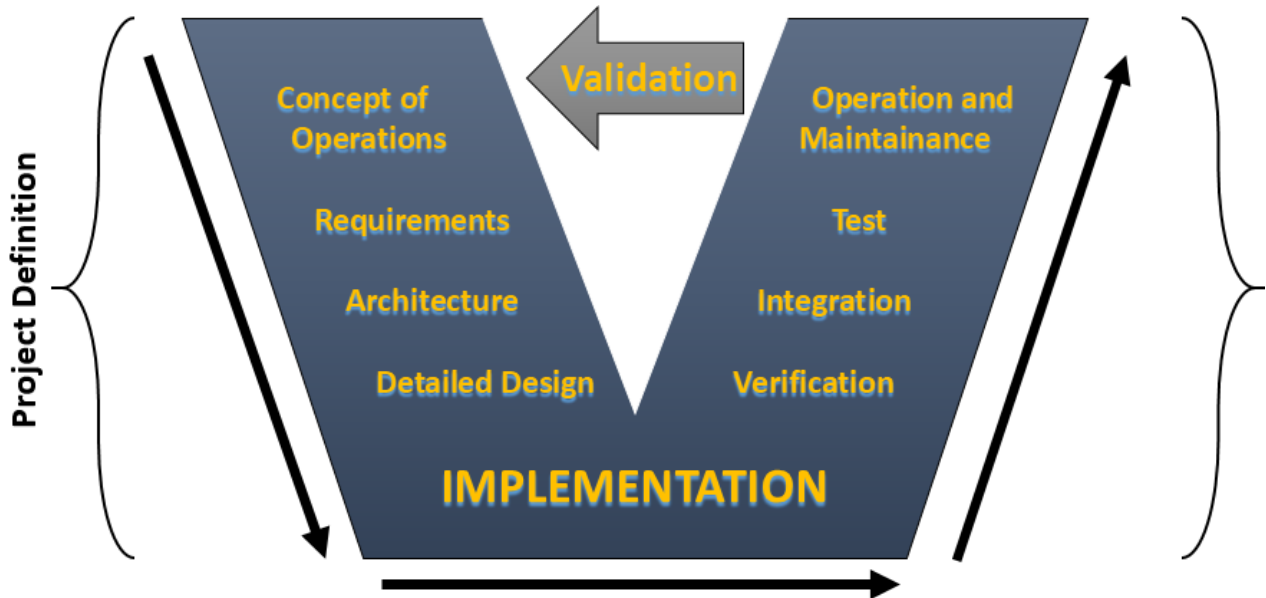


Figure 3. V-Diagram

As shown in Fig. 3, the process begins on the left with a design phase that starts with a focus on the entire system and goes to the single components at an advanced level, passing through a top down approach. On the right, the production aspects related to the design phases are shown, and the horizontal lines going from a side to the other correspond to the necessity to verify and to validate requirements and needs fulfillment.

The rounded lines remark the necessity to iterate every process during every phase, in order to refine the system.

1.3 Costs in Systems Engineering

So, the main target of Systems Engineering is a product that respect customer requirements optimizing the design process and available resources.

As shown in Fig. 4, the *Cost Incurred* curve related to costs actually incurred increases from the preliminary phases to the last one, converging to the *Life Cycle Cost Committed* curve related to the expected cost obtained by means of analysis. In this curve, it is possible to see that the cost is completely defined in first phases of life cycle, with few changing in the last phases. Consequently, the *Ease to Change* curve decreases, due to the difficulty related to reducing costs without changing drastically the system.

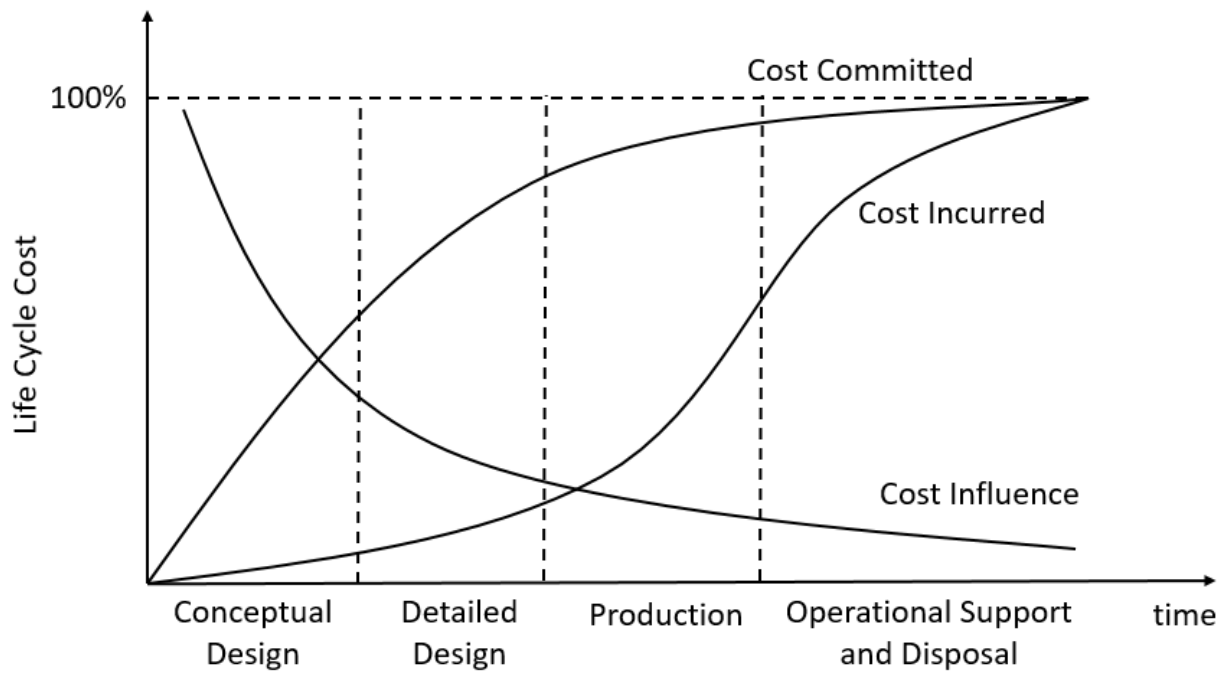


Figure 4. Life Cycle Cost diagram

2. THE ECS IN UAVS

2.1 Unmanned Aerial Vehicles (UAVs)

An *Unmanned Aerial Vehicle* (UAV) is an aircraft providing the achievement of a mission without any pilot or operator on board.

This kind of aircraft is often equipped with an autopilot that provides autonomous operations during the mission, while a *Control Station* (CS) allows remote control, when requested.

Also, UAVs have to send data to the CS about the target of the mission and about the aircraft status.

Obviously, it needs to carry a payload to achieve its mission and all the various systems needed to support the mission, likely common aircraft, but without the need of providing human life support. This take to smaller and lighter aircrafts, reducing consumption.

The UAVs have advantages and disadvantages compared with manned aircraft, depending on the type of mission the aircraft must accomplish. In fact, the type of mission brings to a first classification for these vehicles.



Figure 5. MQ-1 Predator UAV ref.[1]

2.2 UAVs classification

UAVs are strongly related to missions which are dull, dirty and dangerous (DDD):

- For example, missions related on extended surveillance, with a great amount of hours spent watching without relief, can lead to loss of concentration distraction, resulting in a loss of mission efficiency. An UAV equipped with scanner or cameras can perform this kind of mission with max efficiency during the whole time of the mission;
- Also, operating in a contaminated area for surveillance purpose or operating with toxic chemicals on board is dangerous and unnecessary for the crew, so an UAV is a better choice for this kind of mission, requiring only a decontamination of the aircraft the end of the mission;
- Again, an UAV is smaller than a manned aircraft (due to the absence of a crew) and can avoid enemy defenses easier than other aircraft. Also, the loss of both the mission and the UAV is not a risk for the personnel operating at the CS.

An UAV is a better choice also for cover missions, during which a stealth profile is needed due to hide the aircraft presence to the target.

UAVs are used for research and development of new aircraft, too. Producing a scale model of a new aircraft allows the developer to do flight test without putting crew members in danger.

Due to its smaller size, an UAV can operate in inhabited areas with reduced noise and consumption.

An UAV can be classified based on their dimensions, operative altitude and endurance:

- *High Altitude Long Endurance (HALE)* are UAVs operating above 15000 m of altitude with an endurance of 24 hours. They are used for extended surveillance;
- *Medium Altitude Long Endurance (MALE)* are those UAVs operating between 500 and 15000 m with an endurance of 24 hours. They are used for extended surveillance, but with a reduced range compared with HALE;
- *Medium Range or Tactical UAV (TUAV)*, with an operative range of 100-300 km. They are smaller and simpler than other UAVs and are used by land and naval force for military purpose;
- *Close Range UAV*, that operates in a range of 100 km and are characterized by a long list of application in different field;
- *Mini UAV (MUAV)* are UAVs with a mass that is less than 20 kg. These can be hand-launched and have a range of 30 km;
- *Micro UAV (MAV)*, characterized by a wingspan of less than 150 mm. It is used for operations in urban areas and operates at low speed;
- *Nano Air Vehicles (NAV)*, vehicles with the size of a seed used for short range surveillance and for radar interference;
- *Remotely Piloted Helicopters (RPH)* are a category of UAVs characterized by the presence of a rotary wing. These vehicles can perform vertical take-off and landing and are less susceptible to air turbulences at low speed.

Therefore, as we can see, UAVs led to a great advantage when the human action is not always required or when the mission can put in danger the crew.

2.3 The Environmental Control System

The *Environmental Control System (ECS)* of an aircraft is a system that provides avionic cooling and contamination detection, and, in case of manned aircraft, it provides air supply, thermal control and cabin pressurization for the crew and passengers.

The present analysis focuses on UAVs, so the human presence is not contemplated, and the ECS must essentially provide cooling to the avionic bay.

The target of this system is to fulfill these operations in different conditions, since the aircraft operates at different altitude and temperature, so that avionic components can work correctly during the entire duration of the mission.

It is necessary to define an operative temperature range for the avionic components, in order to size the ECS and keep an acceptable temperature in the avionic bay.

Following the System Engineering way to think, once the requirements are met, it is possible to think about the possible configurations of the system.

In the present case, an analysis about two configurations has been done:

- *Air Cycle Machine;*
- *Vapor Cycle Machine.*

These two configurations represent the main architecture for an Environmental Control System in aircraft applications and they differ for the type of components and the cycle type, since the first one is an open loop cycle, while the second one is a closed loop cycle.

2.4 The Air Cycle Machine

The Air Cycle Machine uses bleed air from the engine. This is an open loop cycle, where the bleed air is cooled and is sent to the avionic bay, and then it exhausts into external air.

As shown in Fig. 6, the bleed from the engine goes through a lamination valve that reduces the air pressure. The input conditions depend on the engine and on the compressor stage where the bleed begins. After the valve, there is a *Precooler Unit*, a heat exchanger that cools the bleed air thanks to external air, increasing the efficiency of the entire system, due to the lower temperature entering into the compressor, reducing its work. Before the Precooler, a bypass could split the mass flow, regulated by a controller and a valve, so that an average temperature is achieved downstream.

External air mass flow is provided by means of an inlet, that uses ram air during flight phases and a fan during ground operations.

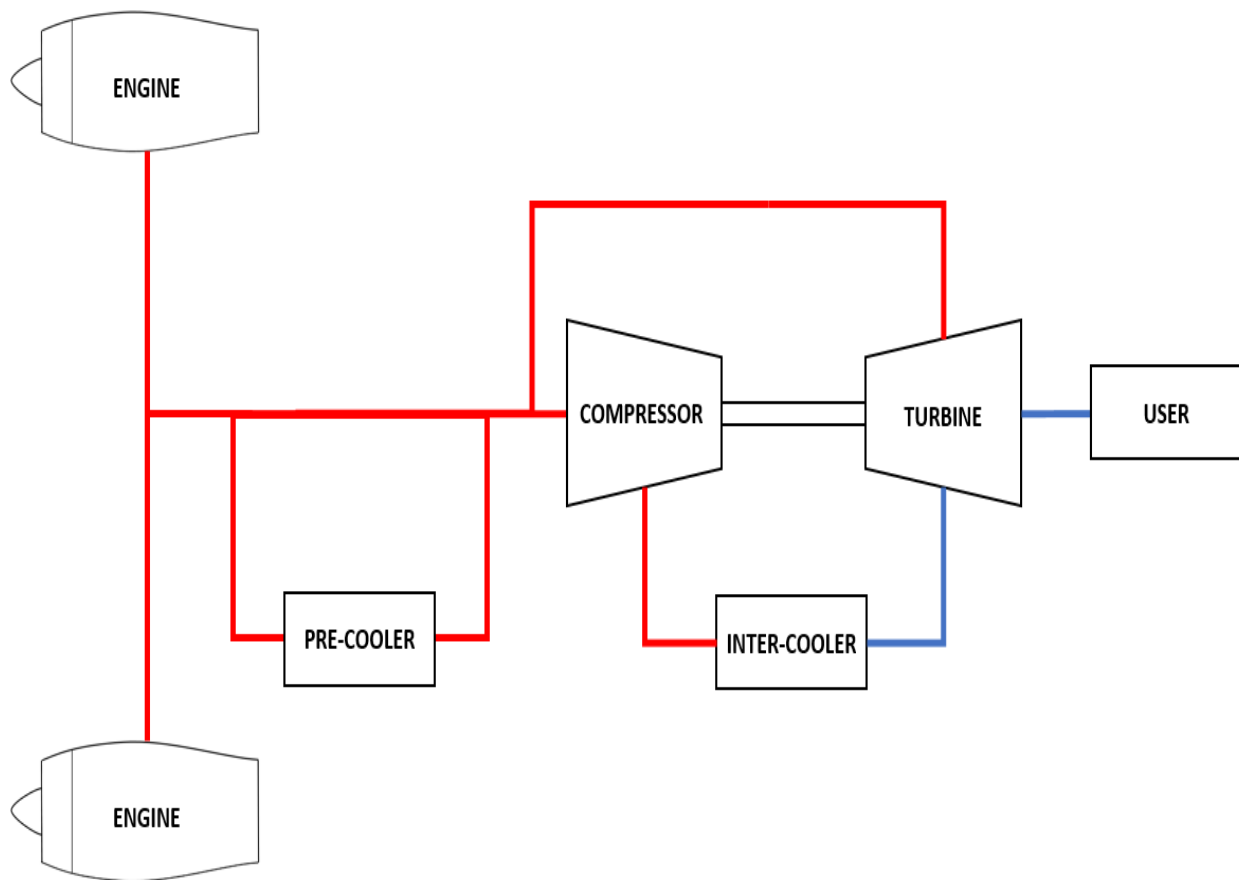


Figure 6. Air Cycle Machine schematization

Downstream another bypass is possible, so that part of the mass flow goes downstream this second bypass, and the rest goes through the *Cold Air Unit (CAU)*. A compressor (previously anticipated) and a turbine essentially compose the CAU: the first one, obviously, compress the airflow with a consequent temperature rise, while the turbine expand the airflow with a decrease of temperature. A shaft connect the two components, so that the work extracted by the turbine is transmitted to spin the compressor and the inlet fan. An *Intercooler*, another heat exchanger like the Precooler, is situated between the compressor and the turbine. Downstream, the two mass flow converge and the total, at average temperature, is sent to the avionic bay for cooling and then it exhausts in external air.

Moreover, the air cooled by the Air Cycle Machine needs to be dehumidified, in order to protect avionic components from damages related to water presence.

The advantages of this solution are its simplicity, its reliability and its lower weight compared with a Vapor Cycle Machine, but it needs great flow of bleed air from the engine, decreasing the aircraft performances.

In addition, it requires external inlet for the heat exchangers, increasing the drag component.

2.5 The Vapor Cycle Machine

The Vapor Cycle Machine is a closed loop cycle, based on heat exchanged during phase changing of a circulating liquid refrigerant. The refrigerant flows by means of a compressor.

The cycle is shown in Fig⁷. When the *compressor* is switched on, it starts delivering pressure and making flow. The refrigerant is initially in liquid form and, by means of an *expansion valve*, its pressure is dropped before going into the evaporator, so that its boiling point is decreased. In the *evaporator*, the refrigerant is exposed to the high temperature air coming from the avionic bay.

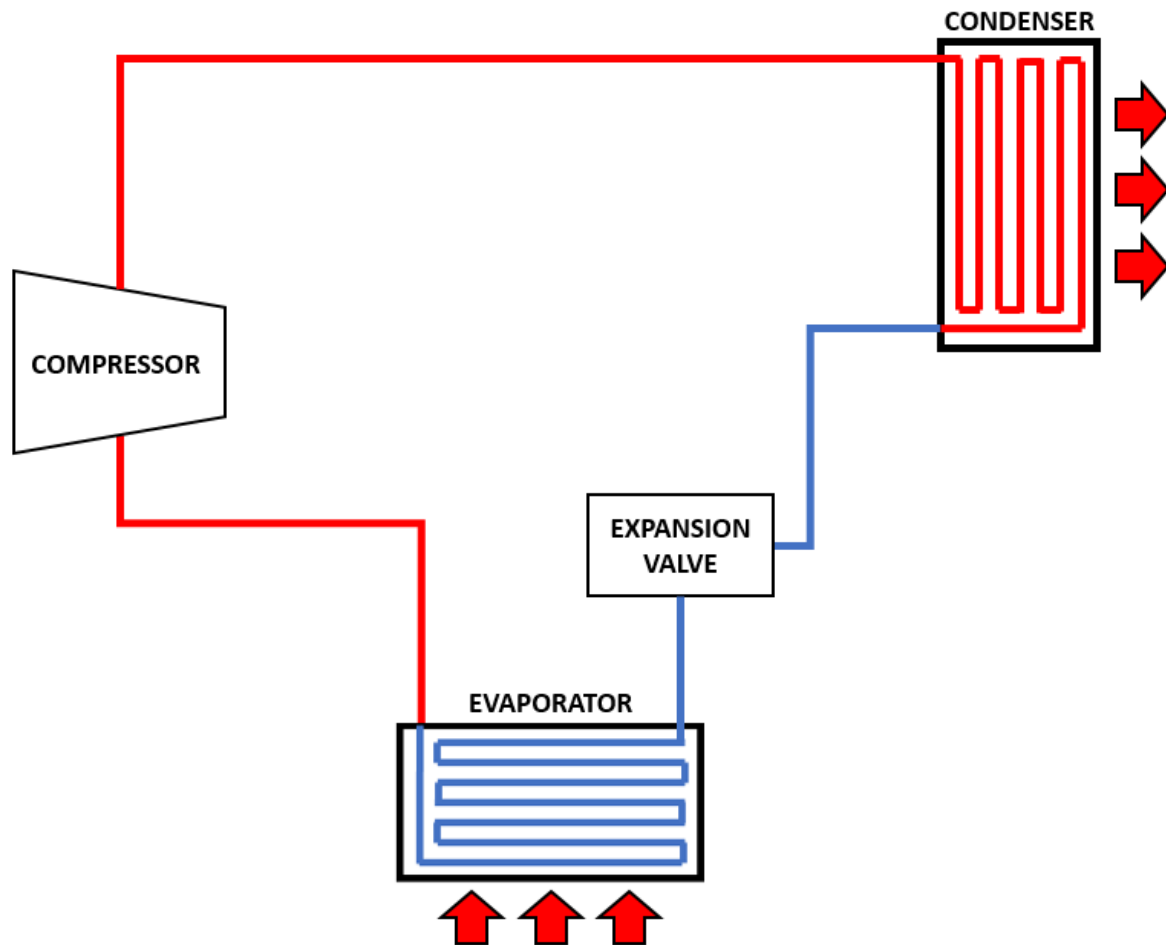


Figure 7. Vapor Cycle schematization

A fan makes the air from the avionic bay flow with a constant mass flow, making an heat exchange by forced convection. This loop is closed, so that the bay receives always the same air. The heat transferred to the refrigerant makes it to change into vapor phase. So, the low-pressure vapor is sent to the *compressor* that compress and rise temperature. This high pressure and high temperature flow enters the *condenser*, where heat exchange is imposed with a forced convection of external air. The mass flow of external air is provided by means of ram air during flight operations and of a fan during ground operations. Therefore, because of this heat transfer, the refrigerant returns to liquid phase and the cycle continues.

In addition, downstream of the condenser, there could be a *receiver dryer* acting as reservoir for the refrigerant. This is used when it is very hot, because in this condition more refrigerant is required by the system. This also act as a filter for foreign particles in the system and captures water that could combine with refrigerant forming an acid, or it could form ice and block the flow.

Vapor Cycle is characterized by higher performances compared with Air Cycle, but, due to the refrigerant presence, it operates in shorter temperature range. Obviously, this cycle depends strongly on the selection of the fluids involved in the heat exchange, as well as on the characterization on the components of the cycle.

3. FUNCTIONAL ANALYSIS

To understand how a system have to be designed, it is useful to think about what the operations required by the system have to be and how it can do these operations, in terms of both what to do and when to do it.

3.1 Model Based Systems Engineering (MBSE)

Model Based Systems Engineering (MBSE) is a methodology related to Systems Engineering. It is based on creating and exploiting conceptual models that incorporates both data and behavior of the system, as first step of systems design.

A model is a simplified version of a system (or of a phenomenon, of a relationship etc.) with the objective of facilitate understanding, examine “*what if scenarios*” and to explain, control and predict events.

These models can also be simulated so that it is possible to explore, update, communicate system aspects to stakeholders and overcome the gap between systems requirements and the output of the simulation. Also, they allow to evaluate design alternatives and validate results, particularly useful when the object of the study is a complex system, where a result is needed before testing it. In fact, due to a wrong system design, an unexpected behavior or malfunction could occur, causing loss of money or damages on human beings.

Obviously, models are never a perfect representation of the real system, but can provide knowledge and guide in system implementation with the lowest impact on the costs and with a faster feedback.

Using MBSE tools and model standards, it is possible to design a coherent model of the system that respects the specification and the requirements in the best way.

3.2 Systems Modeling Language (SysML)

A model is realized by means of a certain language having its own proprieties, forms and meaning. Examples of language used for MBSE models are *Unified Modeling Language (UML)* and *Systems Modelling Language (SysML)*.

The first one is a language born to improve software and it evolved to be applied in different fields. In fact, SysML is one of this evolutions implemented in order to focus UML potentiality on systems model design.

SysML uses and extends some concepts and elements of UML, adding new functionality (i.e. new diagrams, new stereotypes). By means of SysML is possible to analyze, design, verify and validate the complex systems, including hardware, software, data, data exchanges and personnel involved.

The outputs of these models are system architecture, and its components, specifications and activity so that it is possible to simulate system’s behavior.

As shown in the Fig. 8, we can see that SysML is mostly part of UML and adds to it some new stereotypes and diagrams.

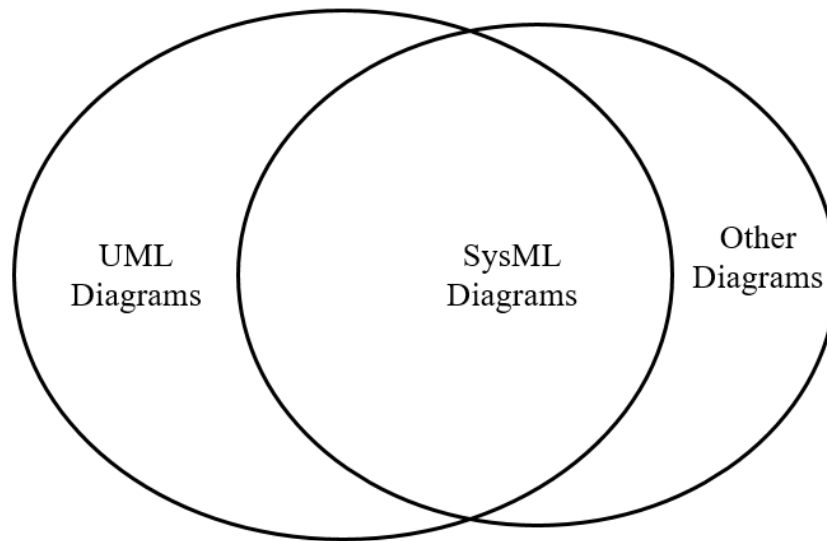


Figure 8. Relation between UML and SysML

3.3 SysML Diagrams

In the SysML architecture, it is possible to implement different types of diagrams, each giving different information about the system. As shown in the following figure, it is possible to see the diagrams and how they are divided in structural diagrams and behavior diagrams.

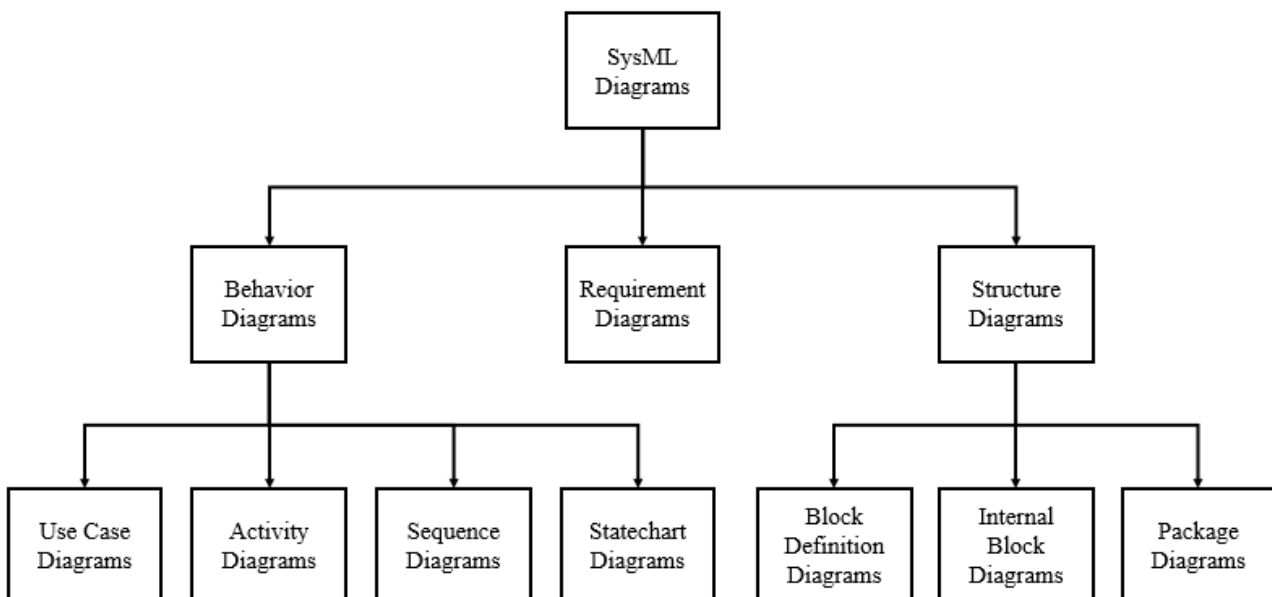


Figure 9. SysML diagrams hierarchy

The first one is the *Requirement Diagram* that gives information about what the system's requirements are and the relations between them and the Use Case Diagram. The requirements are allocated in the Requirements Table, where it is possible to see the requirements and their description and in the traceability matrix, where requirements and Use Case are related. A requirement can also appear on other diagrams to show its relationship to other modeling elements.

The Use Case diagram describes the usage of a system by its actors to achieve a goal, representing the macro functionality of the system and the interactions of the system with the actors to accomplish these tasks.

The *Activity Diagram* shows the activity flow related to a specific Use Cases. In this diagram, it is possible to have information about how the system accomplish every function in terms of choices, data exchanges and sequence of action.

The *Sequence Diagram* shows the same information of the Activity Diagram in terms of messages and data exchanged between the system, its component and the actors. These are shown by means of a control flow between system and actors for a given scenario of a certain Use Case and with time increasing vertically.

The *State Machine Diagram* describes the possible state of the system and can simulate the transition from one state to another by means of trigger events. It can be used to show a specific behavior of the system in terms of events, messages and operations.

The *Block Definition Diagram* shows the blocks forming the system and describes the dependences and associations between them and the environment.

The *Internal Block Diagram* describes the internal structure of a block and its interconnections with environment by means of ports. These are interconnections that specify the type of interconnection between the blocks.

The *Parametric Diagram* contains the equations related to the system behavior.

3.4 The ECS Model

Using the SysML diagrams exposed in the previous paragraph, it is possible to apply the Systems Engineering approach and the methodology presented in Chapter 3 to develop a Functional Analysis of the system.

First, it is necessary to study the system to identify the macro functionality that the system has to perform and the mission that it must accomplish. According with the client and the stakeholder, the inputs of the system are defined in the form of requirements that are all the actions, the performances and the conditions that describe system operations. The process is iterative and has to be validate at every step.

In the following table, it is possible to see the list of the requirements from which the model definition starts.

ID	Name	Specification
SR001	SR001 - Air Conditioning Starting	The ECS shall autonomously assure air conditioning to the avionics/electronics equipment, allocated in the avionic bay, from start up to shut down
SR001.1	SR001.1 - Air Conditioning During Ground Operation	The ECS shall autonomously assure air conditioning to the avionics/electronics equipment, allocated in the avionic bay, during ground operations
SR002	SR002 - Air Filtering Measurement	The ECS shall assure air filtered to avionics equipment, allocated in the avionic bay, by monitoring the pressure difference between the inlet and outlet air flow in order to protect the equipment from fine dust or water
SR002.1	SR002.1 - Pressure Value Comparison	The ECS shall measure information about pressure difference and compare it with a threshold value
SR002.2	SR002.2 - Filter Clogged Verification	When the measured pressure difference is higher than TBD the ECS shall send the value of pressure difference (filter clogged) to Central Maintenance system
SR003	SR003 - Bay Monitoring	The ECS shall monitor the avionic bay temperatures
SR004	SR004 - Over Temperature Or Under Temperature Alert	The ECS shall provide an alert to Utility Management System when avionic bay temperature is out of range TBD
SR004.1	SR004.1 - Autonomous Air Inlet Area Control	In case an over temperature or under temperature is detected in the avionic bay, the ECS shall autonomously increase or decrease air inlet area
SR004.2	SR004.2 - Over Temperature Or Under Temperature Condition - Power Off	In case an over temperature or under temperature is detected in the avionic bay, the ECS shall be powered off by Utility Management System
SR005	SR005 - ECS Health Status Monitoring	The ECS shall monitor its health status, from start up to shut down
SR005.1	SR005.1 - ECS Health Status Information	The ECS shall send information about its health status to the Utility management system and Central Maintenance System
SR005.2	SR005.2 - ECS working fluid Over Or Under Temperature Information	The ECS shall send information about the working fluid over temperature or under temperature events to the Utility Management System and Central Maintenance System.
SR006	SR006 - ECS Start Up Condition	The ECS shall start when powered on by electrical system
SR006.1	SR006.1 - IBIT	At start up the ECS shall provide IBIT to Central Maintenance System
SR007	SR007 - MBIT Performing	The ECS shall perform MBIT when requested by Central Maintenance System
SR007.1	SR007.1 - MBIT Results	The ECS shall send MBIT result to Central Maintenance System
SR008	SR008 - Shut Down	The ECS shall shut down when electrical system stops providing electrical power
SR009	SR009 - Bay Temperature	The ECS shall protect avionics/electronics equipment, allocated in the avionic bay, within the following avionic bays temperature range: from -20°C to 50°C
SR010	SR010 - Filter	The ECS shall provide air filtered to avionics/electronics equipment, allocated in the avionic bay, in order to protect the equipment from fine dust or water

Table 1. System Requirements

Requirements must be satisfactory and not in contradiction between themselves. In the present work, Leonardo Spa provided the requirement of the system and a study of these has been done with the help of a specialist to refine them.

As shown, the requirements are related to performance or related to the functionality and every of them must be verifiable, congruent, not contradictory and univocal.

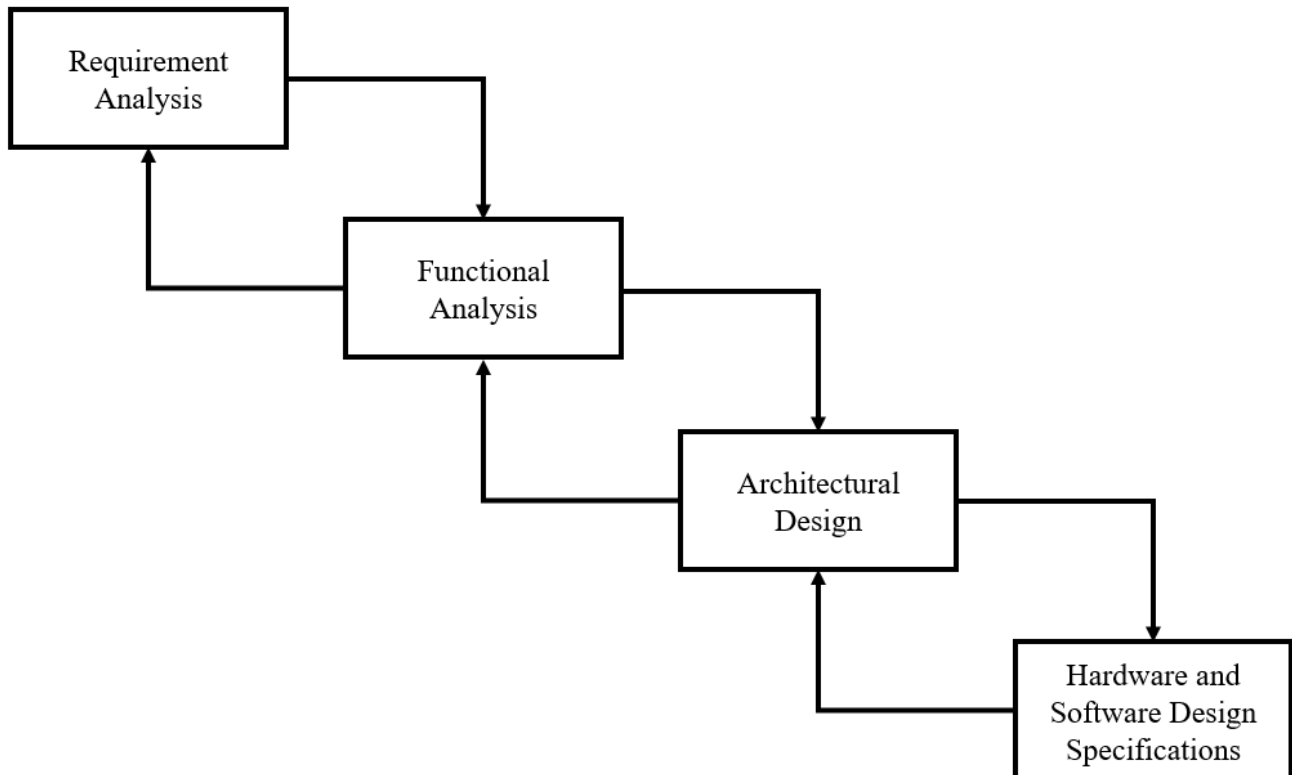


Figure 10. Processes iterations

The previous figure shows the Harmony methodology. It is possible to see that the next step in the system design is to analyze the functionality of the system. The arrows show the iterative nature of this process, due to the necessity to refine the model and to obtain the best results.

So, it is possible to define the Use Case diagram that describes a specific functionality of the system and its interaction with the actors. An actor could be a person, a hardware component, the environment or a software component. The relations between them are of Association type and they represent the interfaces of the systems and the users of the system.

The different Use Cases derive from the requirements previously defined and they are divided in:

- Monitor its health status;
- Provide air conditioning;
- Monitor status of air filter;
- Provide maintenance;
- Start up;

- Shut down.

As obtained by the requirements, the followings are the actors that exchange information with the Use Cases:

- Utility Management System;
- Central Maintenance System;
- Avionic Bay;
- Electrical System;
- Environment.

In the Figure below the Use case chosen for the ECS model are shown related to the respective actors.

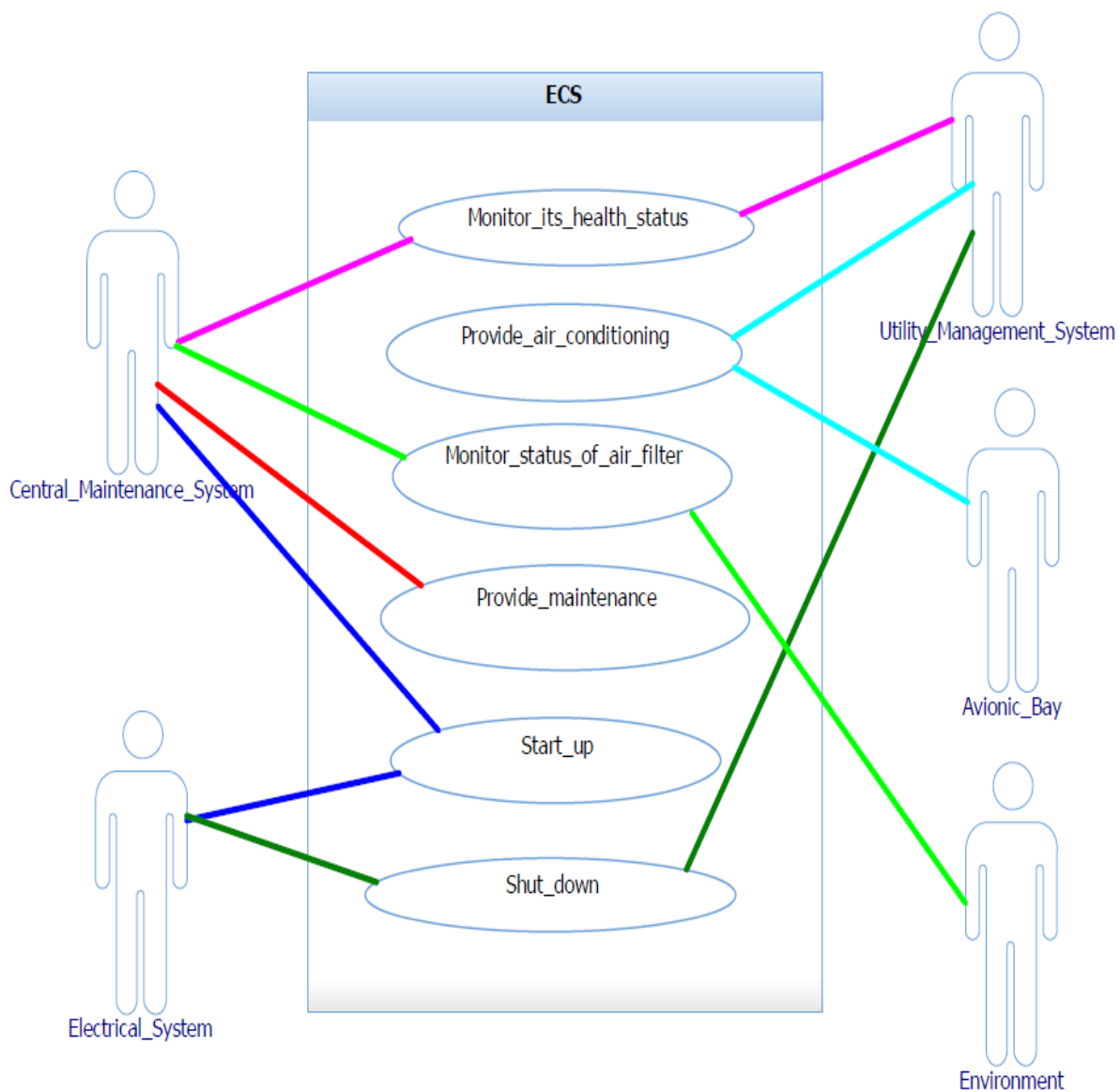


Figure 11. Use Case diagram for the ECS under analysis

	Shut down	Start up	Provide Maintenance	Monitor Status of Air filter	Monitor its Health Status	Provide Air Conditioning
SR005 - ECS Health Status Monitoring					X	
SR004.3 - Over Temperature Or Under Temperature Condition - Power Off	X					
SR005.2 - ECS Over Or Under Temperature Information					X	
SR004.2 - Over Temperature Or Under Temperature Condition - Air Intake Area						X
SR005.1 - ECS Health Status Information					X	
SR006 - ECS Start Up Condition		X				
SR006.1 – IBIT		X				
SR007 - MBIT Performing			X			
SR001 - Air Conditioning Starting						X
SR001.1 - Air Conditioning During Ground Operation						X
SR010 – Filter				X		
SR002 - Air Filtering Measurement				X		
SR002.1 - Pressure Value Comparison				X		
SR002.2 - Filter Clogged Verification				X		
SR003 - Bay Monitoring						X
SR004 - Over Temperature Or Under Temperature Alert						X
SR004.1 - Autonomous Air Inlet Area Control						X
SR007.1 - MBIT Results			X			
SR008 - Shut Down	X					
SR009 - Bay Temperature						X

Table 2. Traceability Matrix

Since Use Cases come from the requirements, it is possible to associate every Use Case with the respective requirement using a relation of Trace type. The traceability matrix is another instrument of the SysML implemented in the present model and it shows these relations between Use Cases and requirements. Every requirement can be associated to only one Use Case, but every Use Case can be associated to more than one Use Case.

In the previously page, it is possible to see this matrix.

At this point, it is possible to start a Black Box analysis, because the architecture is not defined yet. Therefore, the next step is the Black Box Activity Diagram defined for every Use Case. Starting from the definition of the Use Case, the control flow is schematized using action blocks that show, obviously, the actions that the system fulfills. The Actor Pins show when an actor interact with the system and how it does it (input or output). It is also possible to introduce a decision node that makes a choice between what action to do. The figure below shows an example of Activity Diagram for the “Provide Air Conditioning” use case.

The system measure the temperature bay and then it compare the current temperature with the threshold ones. Then the system perform a choice: if the temperature is in the correct range, the system continues to send air-conditioned to the bay, but if it is not so, the system must stop and send an alert.

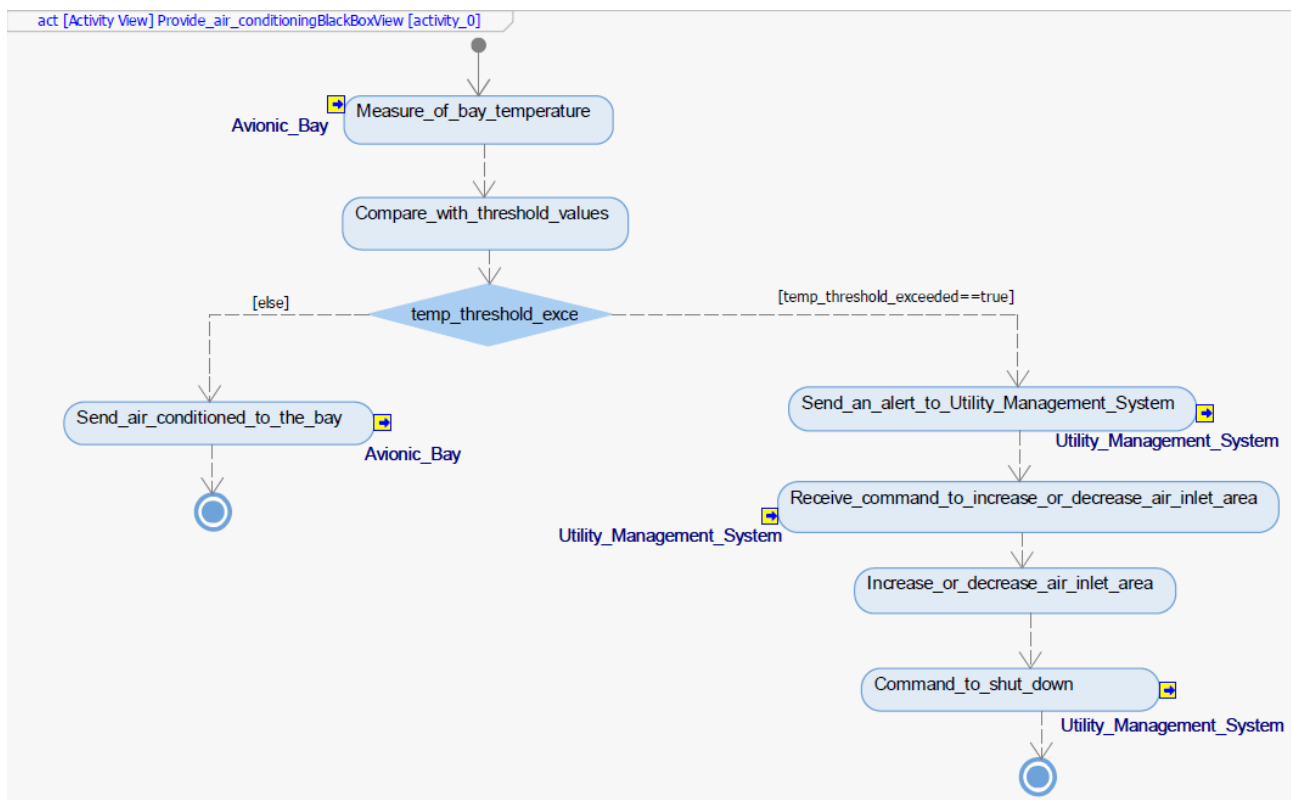


Figure 12. Activity Diagram related to the Provide Air Conditioning Use Case

The next step is the Sequence Diagram, which allows the identification of events and messages intervening between the system and the actors during the accomplishment of the actions showed in the Activity Diagram. In fact, the Sequence Diagram is strongly related to the Activity and can be generated starting from it.

Again, an example of the diagram is reported in Fig 13.

The examined case is the “Provide Air Conditioning” again. It is evident the relation between this Diagram and the Activity Diagram. The elements in the two diagrams are the same, but in different forms. The system and the actors are represented by the blocks, while the arrows show event and messages and between who they are done.

Moreover, in this case, the Sequence Diagram is splitted up in more than one, due to the different possible scenarios the system views when acting.



Figure 13. Sequence Diagram related to the Provide Air Conditioning Use Case

Next to be done is the Internal Block Diagram, which shows the operations that the system performs, ports and interfaces used to interact with the actors. Again, this diagram can be obtained starting from the Activity Diagram. In the Fig. 14 is showed the Internal Block Diagram for the “Provide Air Conditioning” use case.

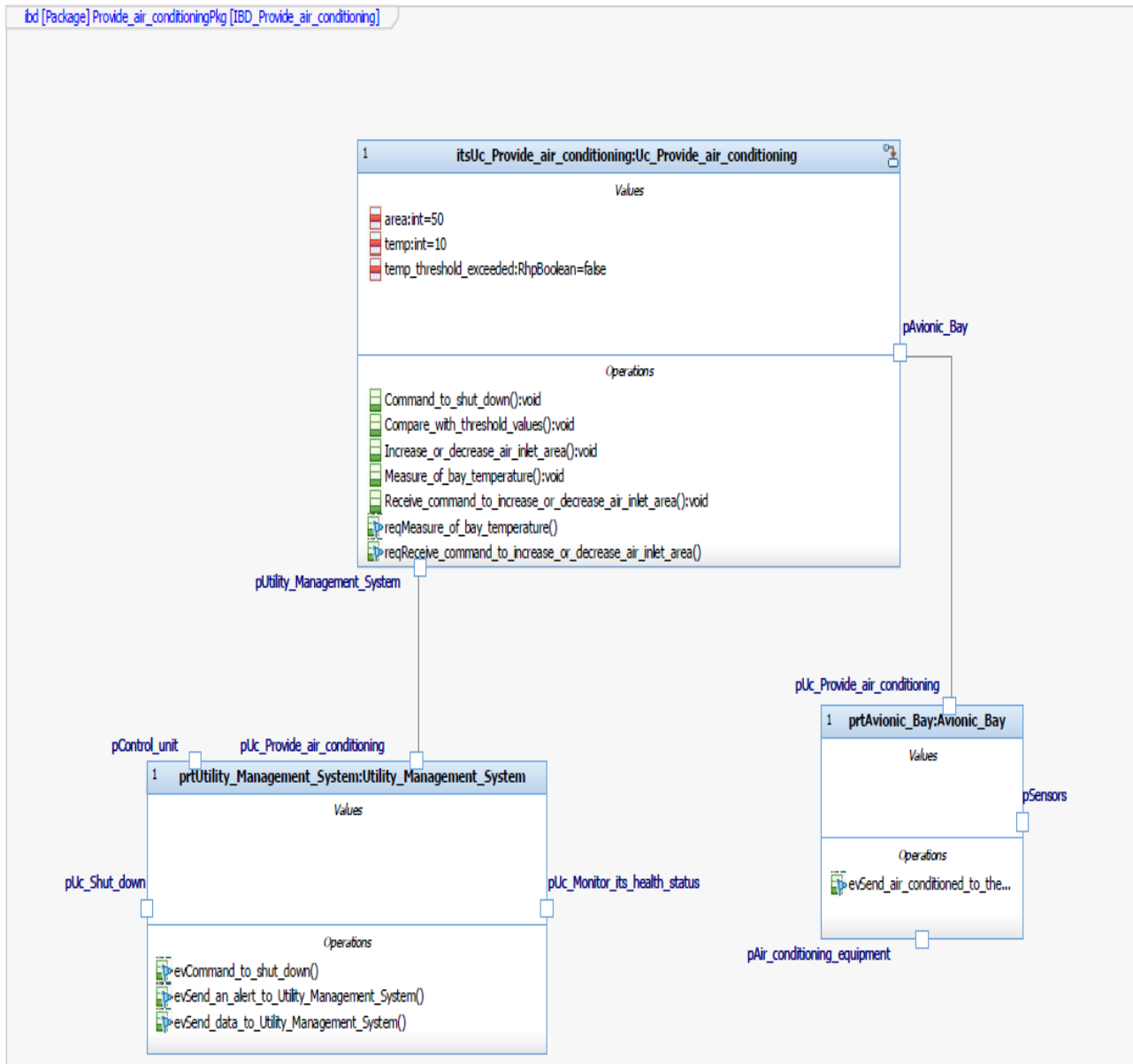


Figure 14. Internal Block Diagram related to the Provide Air Conditioning Use Case

In the end, there are the Statechart Diagrams: they convert the action flow of the use cases in states. The transition from one state to another is regulated by means of the events generated by the system or external actors.

In Fig 15 the “Provide Air Conditioning” Statechart is shown. The diagram start with a default state, then the system goes through the different states according with the trigger events. The Statechart finishes with a Termination State.

Also this diagram is similar to the Activity Diagram, presenting a similar form and showing the states related to the actions the system performs.

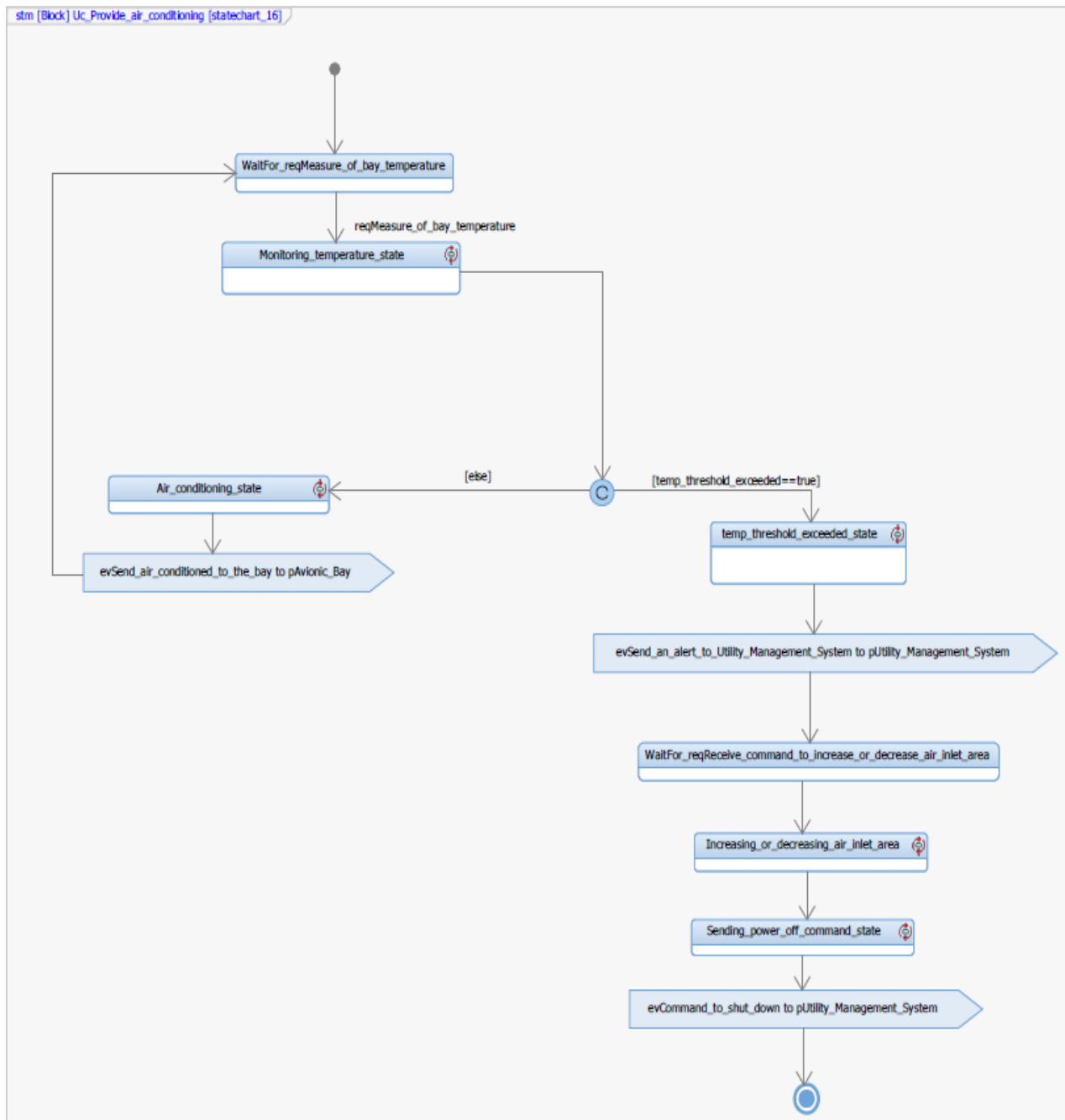


Figure 15. Statechart Diagram for the Provide Air Conditioning Use Case

The importance of this diagram is not only related to the information it gives, but also to the fact that it allows a simulation of the system.

In fact, the model can be simulated by mean of this diagram, using the actions as trigger. To help this simulation, it is possible to generate a control panel, as shown in Fig. 16.

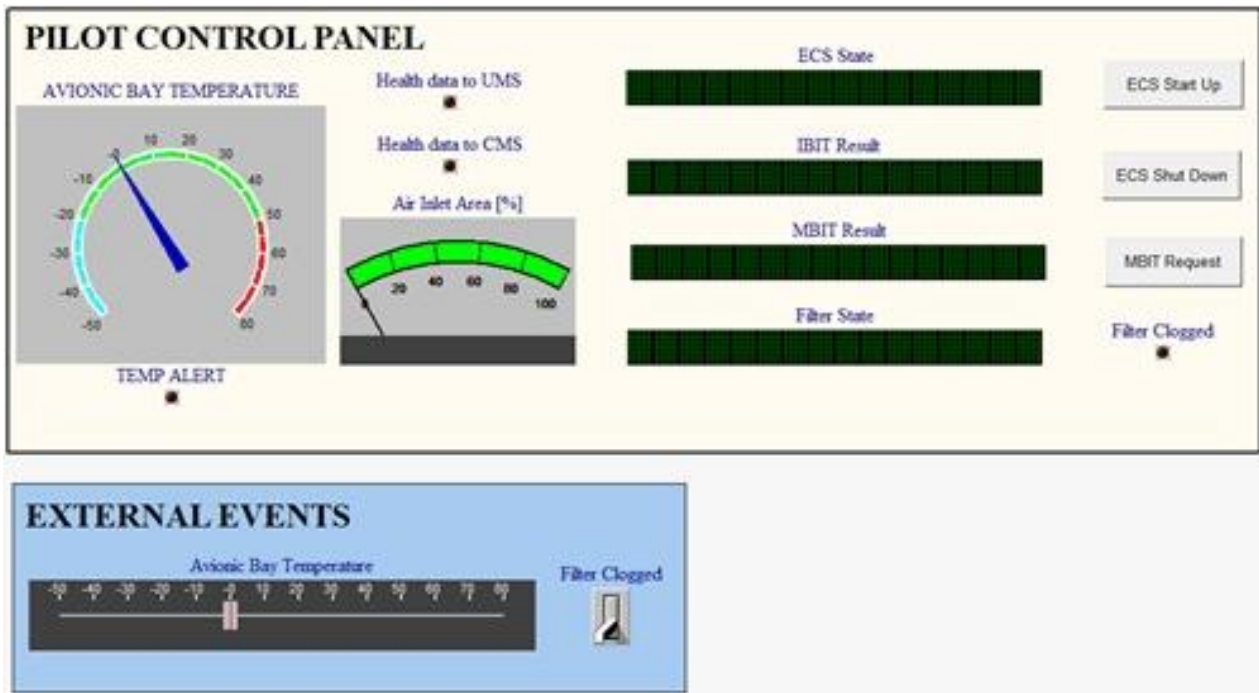


Figure 16. Control Panel

The one made until now is a Black Box View, which is disengaged from system architecture. But a White Box View is also possible, which recall the diagrams seen in the Black Box View relating them with the components of the system. The example of White Box View diagrams for the “Provide Air Conditioning” use case are reported in the following figures.

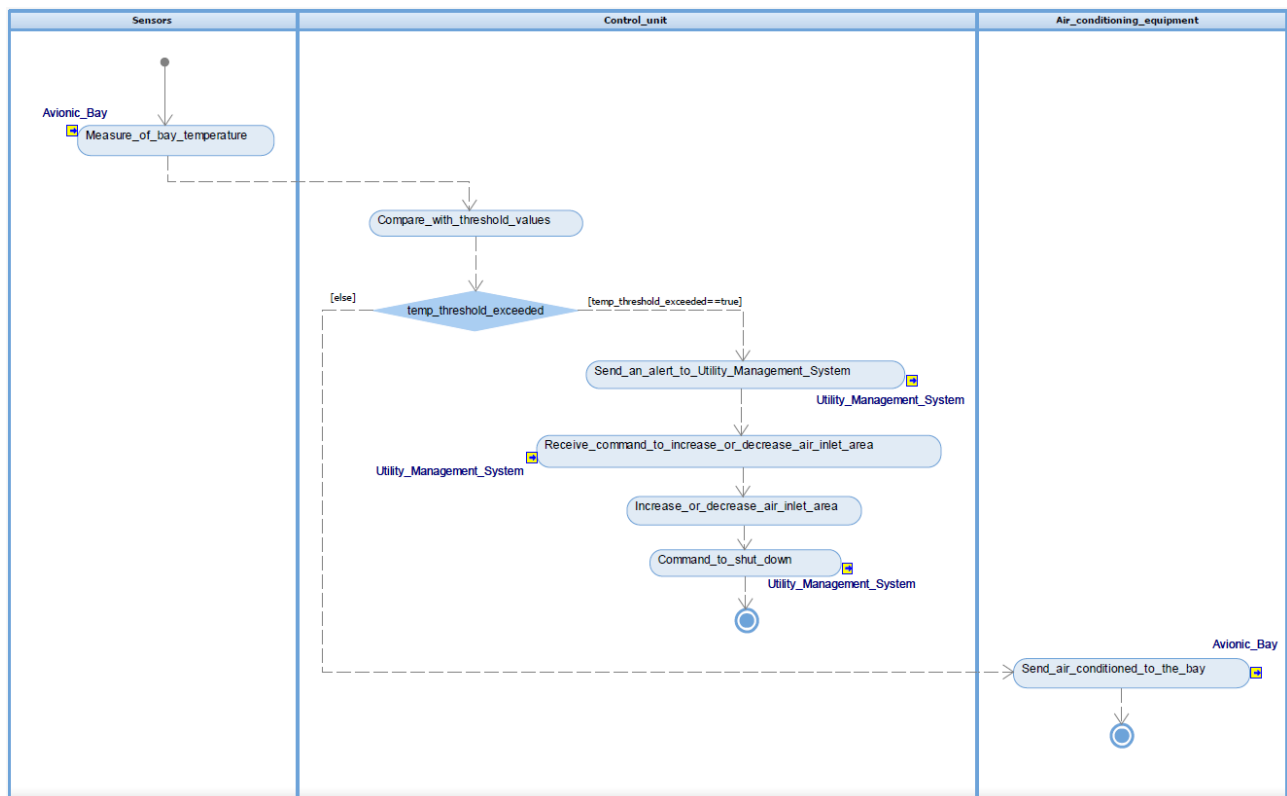


Figure 17. Activity Diagram related to the Provide Air Conditioning Use Case White Box View

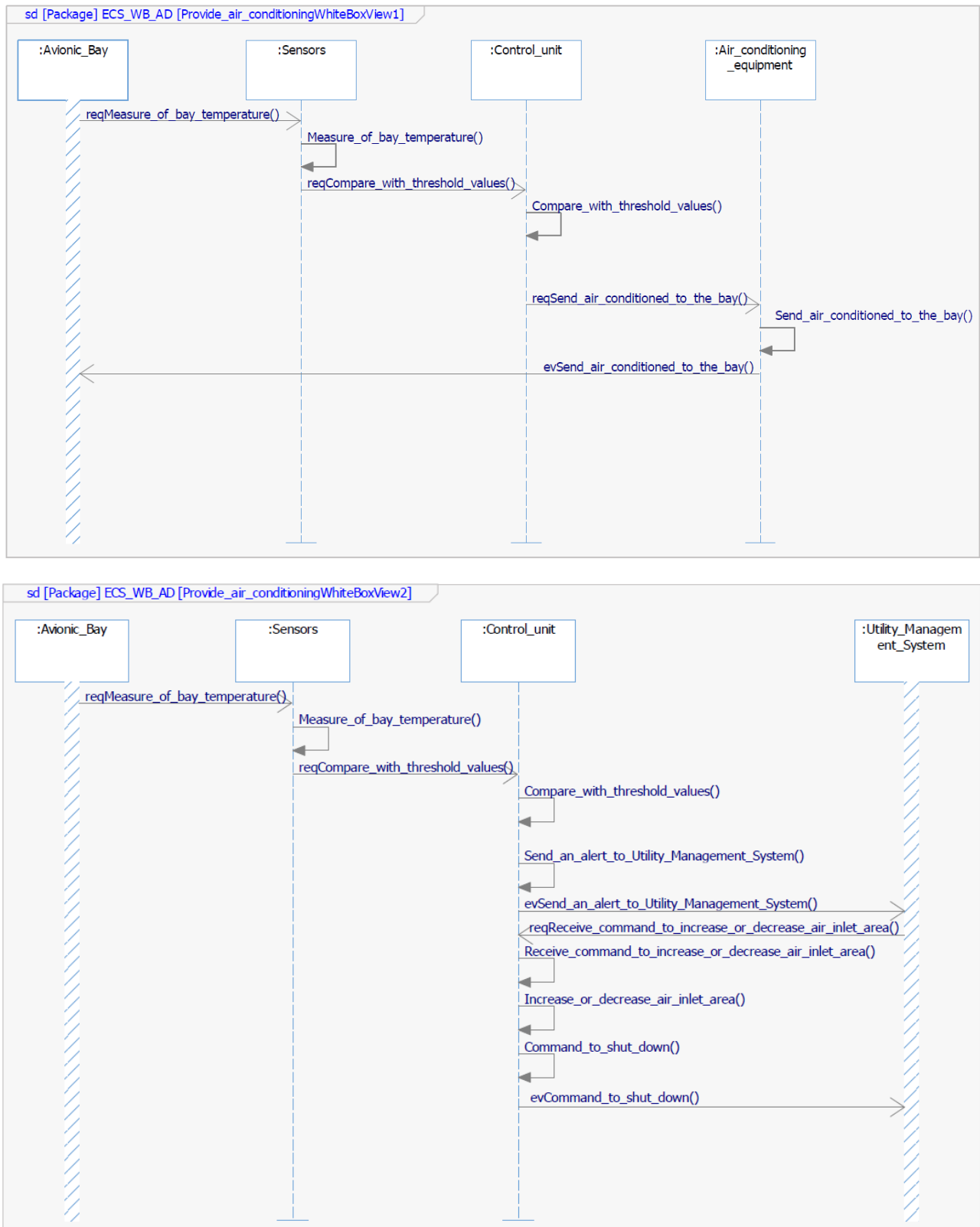


Figure 18. Sequence Diagrams of the Provide Air Conditioning Use Case White Box View

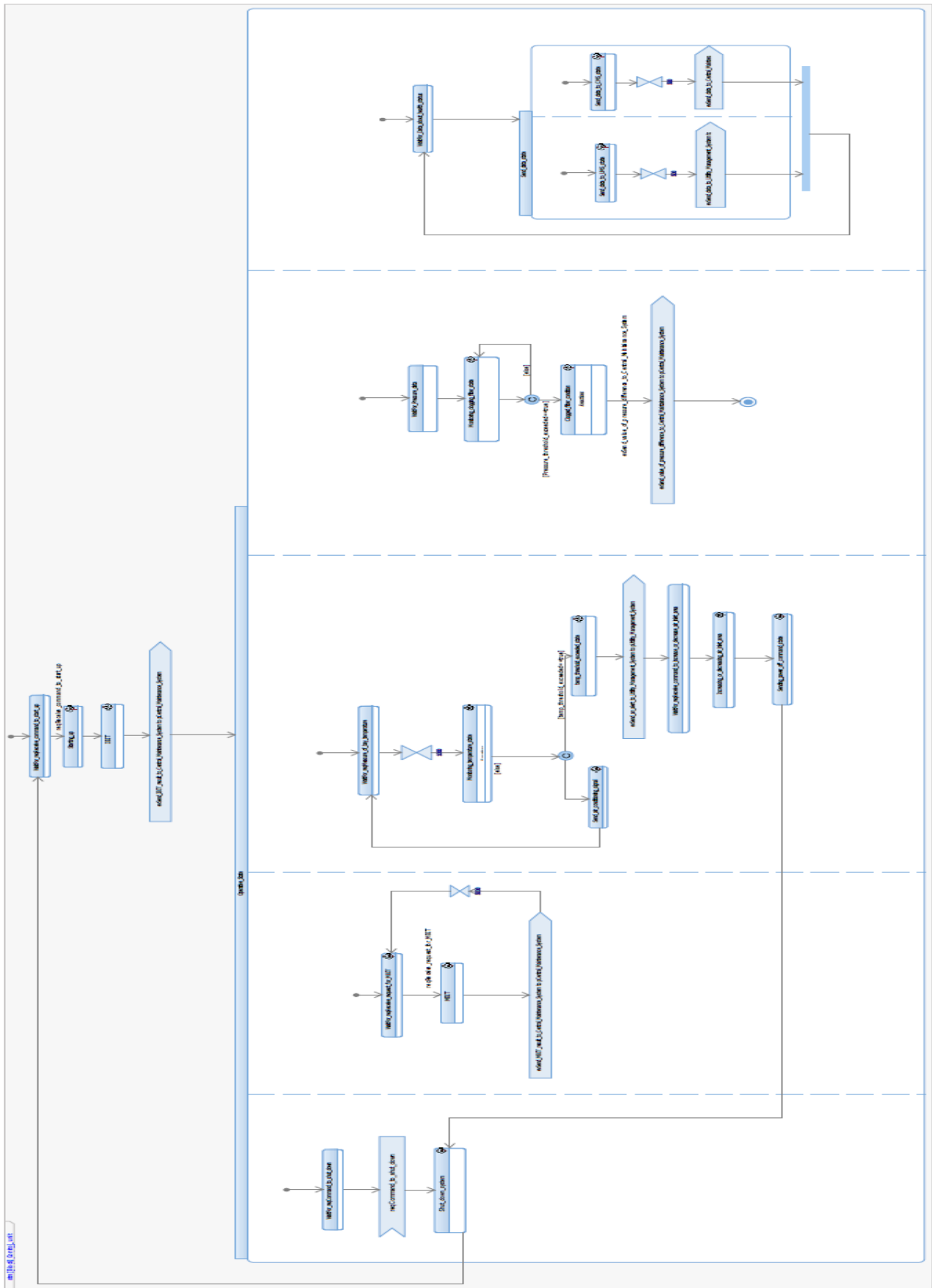


Figure 19. Statechart Diagram White Box View

4. PERFORMANCE ANALYSIS

4.1 Thermal Flow Analysis

At this point, it is possible to proceed with performance analysis. Once how the system work has been defined, it has to be defined how it does it.

The first step of this analysis is to define the operative range of the system: the ECS for UAVs is responsible for monitoring the temperature of the avionic bay. The components are designed to operate in a specific temperature range without performance degradation. From the requirements, it has been fixed a range that goes from -20°C to 50°C.

The system will be subject to thermal flows generated from different sources. The first heat flow comes from the avionics operating in the bay that dissipate energy producing heat.

In addition, the system is subject to a thermal flow dependent from the external temperature. The wall temperature is an important factor in this heat exchange and can be calculated as

$$T_w = T_{rec} = T_0 \left[1 + r \left(\frac{\gamma - 1}{2} \right) M^2 \right]$$

where T_w is the wall temperature, T_{rec} is the recovery temperature, T_0 is the room temperature, M is the Mach number, r is the recovery factor and γ is the specific heat ratio.

The heat exchanged through the skin can be calculated from the following relation

$$h_0(T_0 - T_w) = -G \frac{A_p}{A} + U(T_w - T_c) + \frac{1}{2} \sigma e_w (T_w^4 - T_u^4) + \frac{1}{2} \sigma e_w (T_w^4 - T_l^4)$$

where $h_0(T_0 - T_w)$ Is the heat exchanged by the skin, $G \frac{A_p}{A}$ is the heat exchanged by solar radiation, $U(T_w - T_c)$ is heat exchanged by skin and the interior of the aircraft, $\frac{1}{2} \sigma e_w (T_w^4 - T_u^4)$ is the heat exchanged by radiation from the upper part of the aircraft and $\frac{1}{2} \sigma e_w (T_w^4 - T_l^4)$ by the lower part.

At this point, it is possible to calculate the wall temperature through some steps

$$T_w = B - CT_w^4$$

Where

$$B = \frac{h_0 T_0 + G \frac{A_p}{A} + U T_c - \frac{1}{2} \sigma e_w T_u^4}{U + h_0}$$

$$C = \frac{\sigma e_w}{U + h_0}$$

Once the values of B and C are known, it is possible to obtain T_w from the following figure.

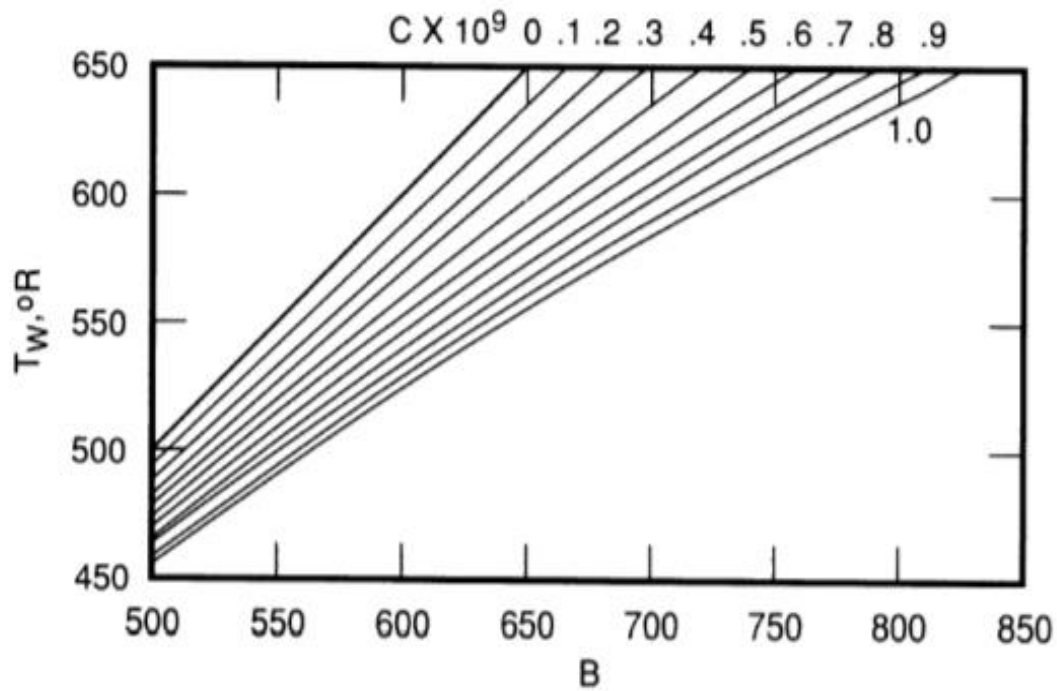


Figure 20. Wall Temperature in function of B and C

In addition, it is necessary a definition of the mission profile: the aircraft will be subject to different environments due to the altitude variations. These temperature variations will affect the skin temperature during the system operations, afflicting the heat flow in the avionic bay. The mission profile is showed in the Fig. 21.

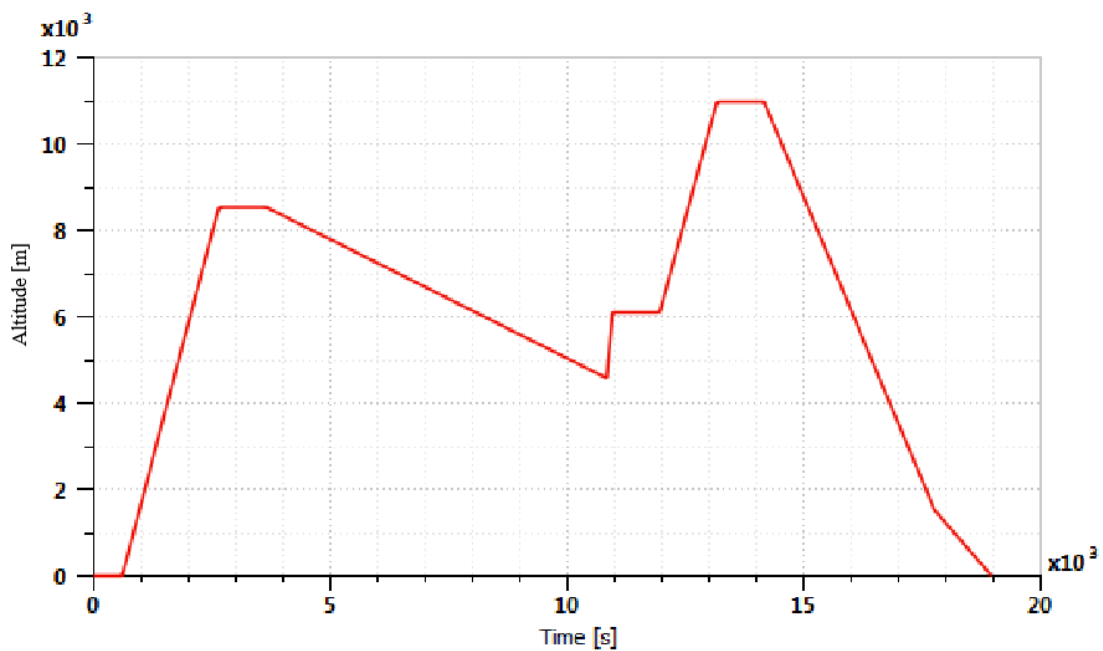


Figure 21. Mission profile in terms of altitude

As shown previously, the wall temperature will depend also on the Mach number, so the mission profile has to be defined also in terms of Mach number. The Mach profile is shown in Fig. 22.

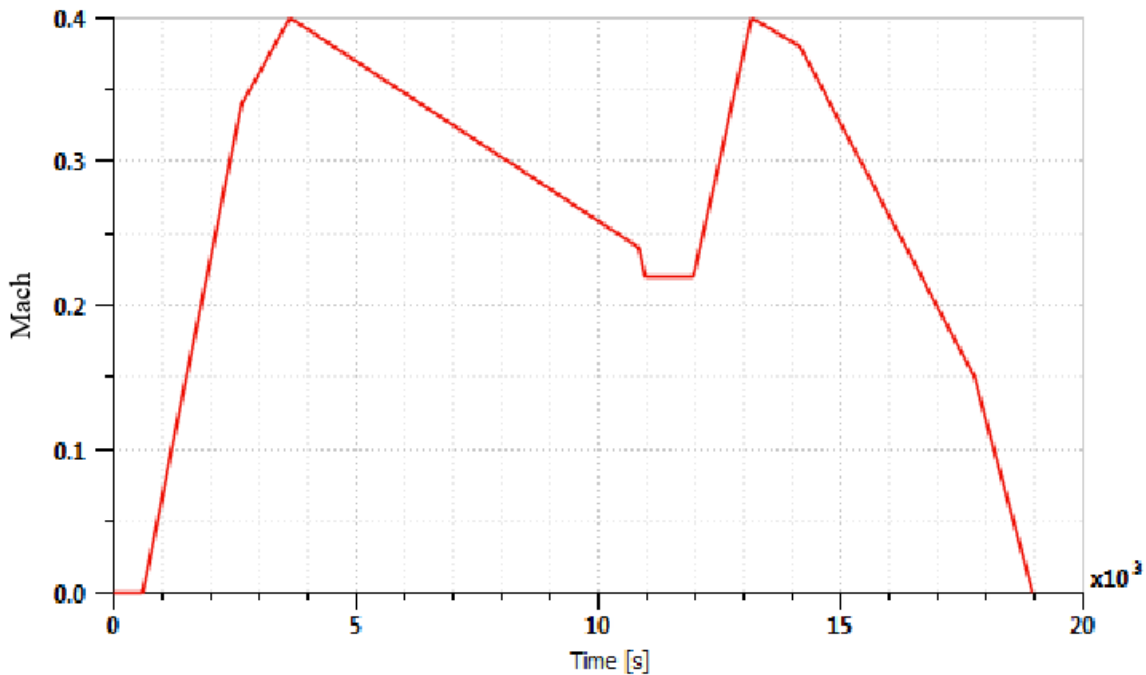


Figure 22. Mission profile in terms of Mach

Moreover, the environment temperature will not be the same every day and in every moment. Therefore, it has to be considered a variation of the external temperature due to climatic changes. It is possible to consider two different cases to study the validity of the model:

- there is the hot case, which considers a temperature of 50°C at sea level;
- then there is the cold case that considers a temperature of -20°C.

4.2 Vapor Cycle Simulation

With the support of a simulation software, it is possible to build a model of the system and to validate its efficiency.

Therefore, it is necessary to build the analytic equivalent of the system. In particular, a model of the avionic bay is built, considering the heat flows previously cited. First of all the heat dissipated by the avionics that starts from a lower value during ground operations and rise after take-off. Then there is the heat exchanged by means of convection and irradiation.

The ECS must be considered in both the considered configurations. The vapor cycle consist of the condenser, the compressor, the evaporator and the expansion valve. These components must be sized due to obtain the wished performance.

The system and avionic bay model for the Vapor Cycle are shown in Fig. 23.

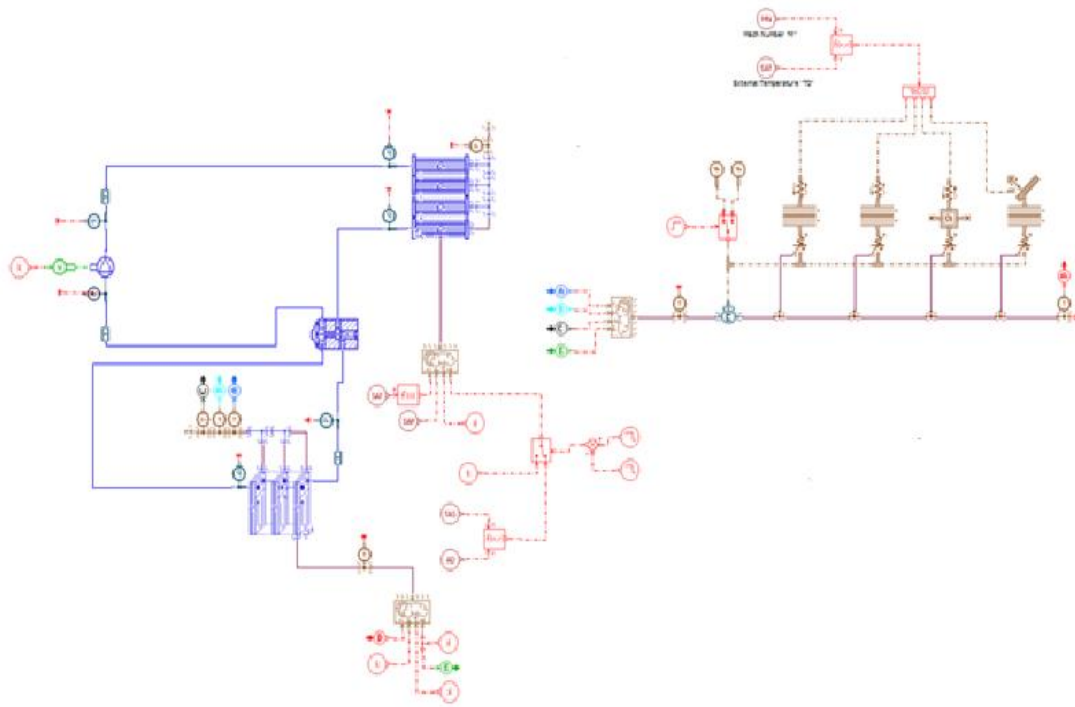


Figure 23. Vapor Cycle architecture

At this point, the model can be simulated, starting from the hot case. For the Vapor Cycle, it is important to see the thermodynamic cycle, which must follow the transformations that have been told in the previous chapter. First, there is an heat exchange between the evaporator and the avionics bay, which gives a temperature rise of the refrigerant at almost constant pressure. Then the pressure and the temperature rise as the refrigerant goes through the compressor. Then it arrives in the condenser where the refrigerant is cooled at constant temperature and it completes the cycle in the expansion valve, where temperature and pressure fall down. The correspondent cycle is reported in the following figure.

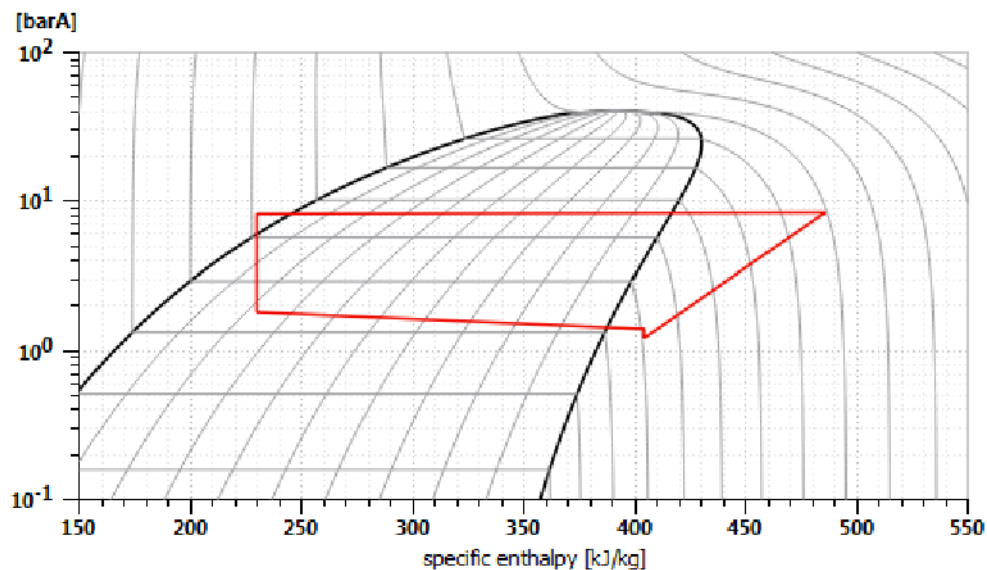


Figure 24. Thermodynamic cycle for the Vapor Cycle in the hot case

The thermal flow produced by the avionic is shown in the following figure. During ground operations, only few components are operative, so the thermal flow is lower than the thermal flow during flight operation.

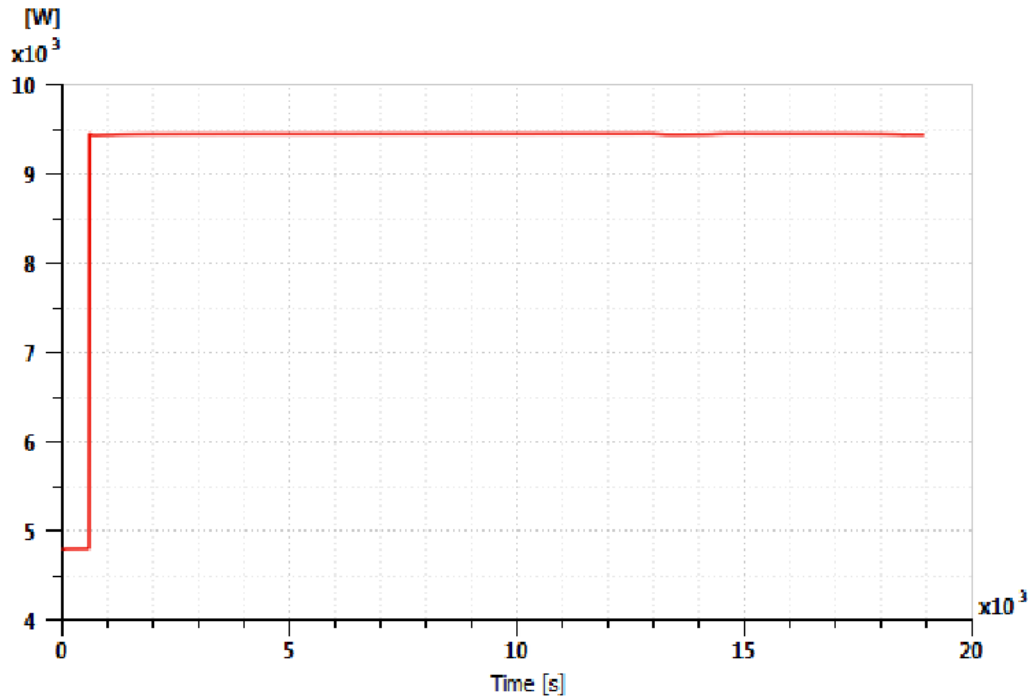


Figure 25. Thermal flow in the avionic bay in the hot case

A zoom on the heat flow in the avionic bay shows that it is not constant, but it changes due to the recovery temperature variations that are negligible compared to the one produced by avionics.

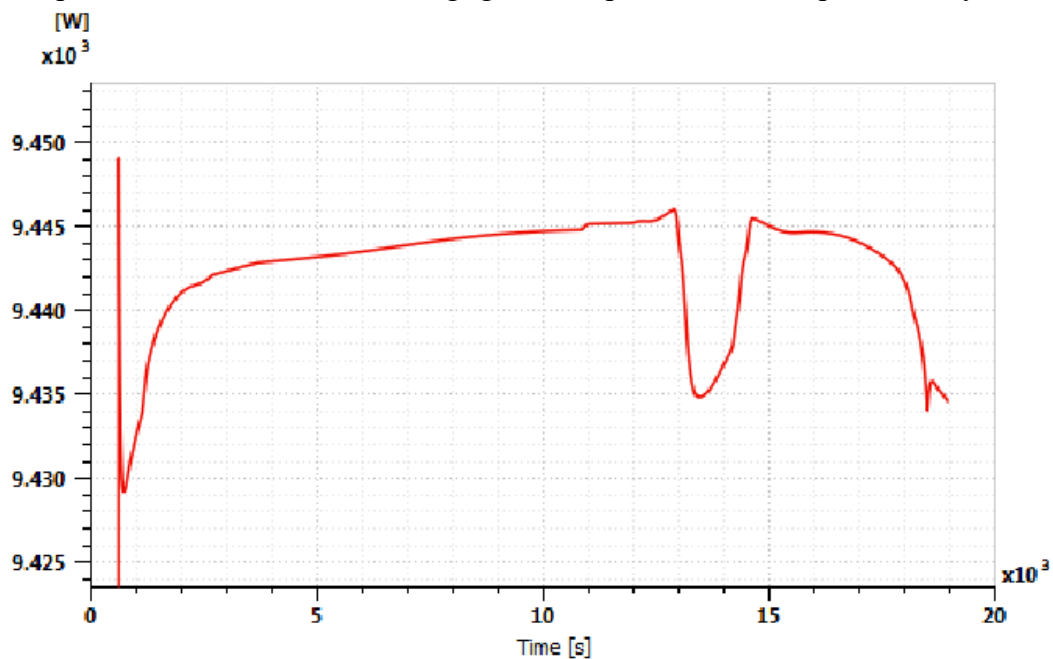


Figure 26. Thermal flow produced detail

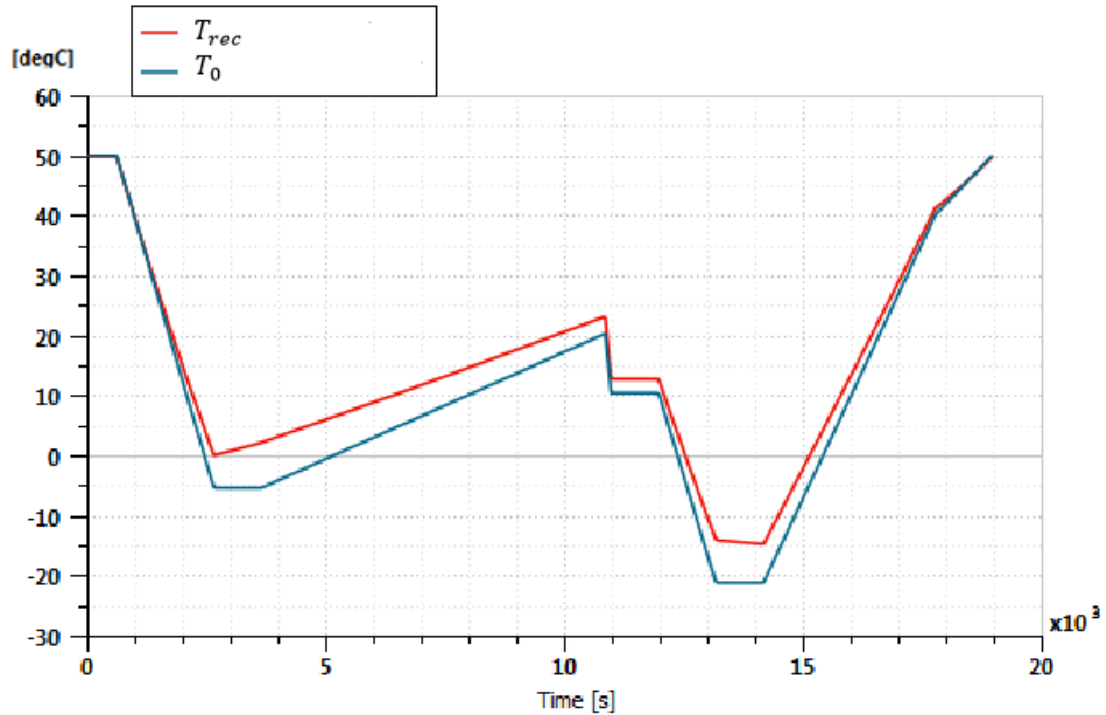


Figure 27. Recovery temperature and external temperature in the hot case

The temperature in the avionic bay during the mission is shown in Fig. 28. The temperature never exceeds the limits of the imposed range, so it works well.

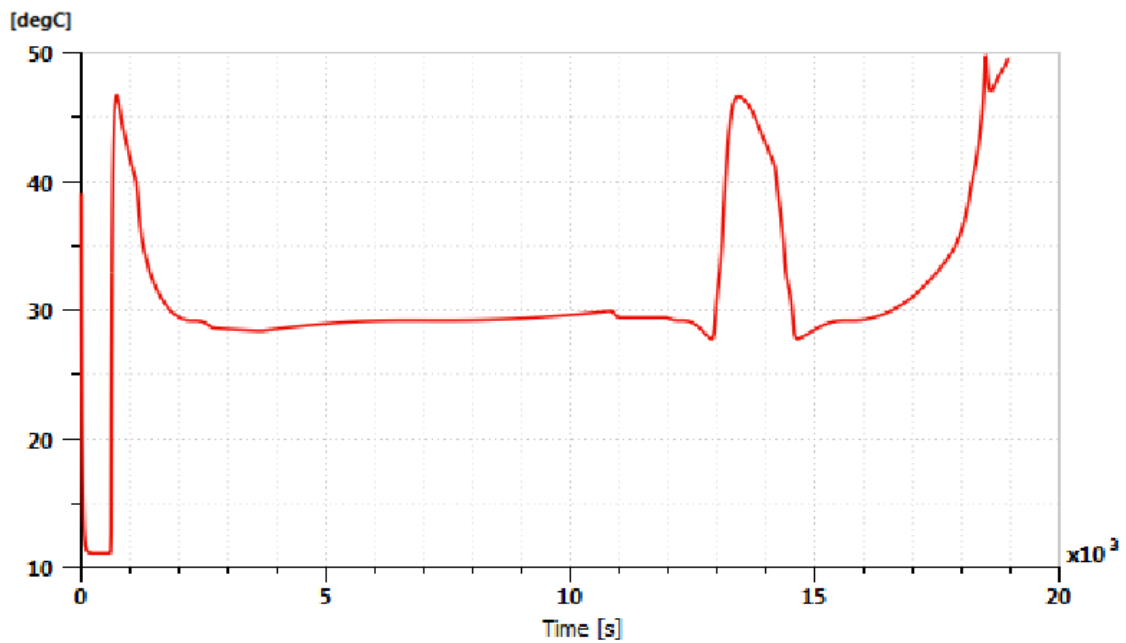


Figure 28. Temperature in the avionic bay for the Vapor Cycle in the hot case

Once validated the hot case, it is the turn of the cold case with a temperature of -20°C at sea level. The thermodynamic cycle changes, but it maintains the same form, following the type of transformations, which it is subject, as it is possible to see in Fig.29.

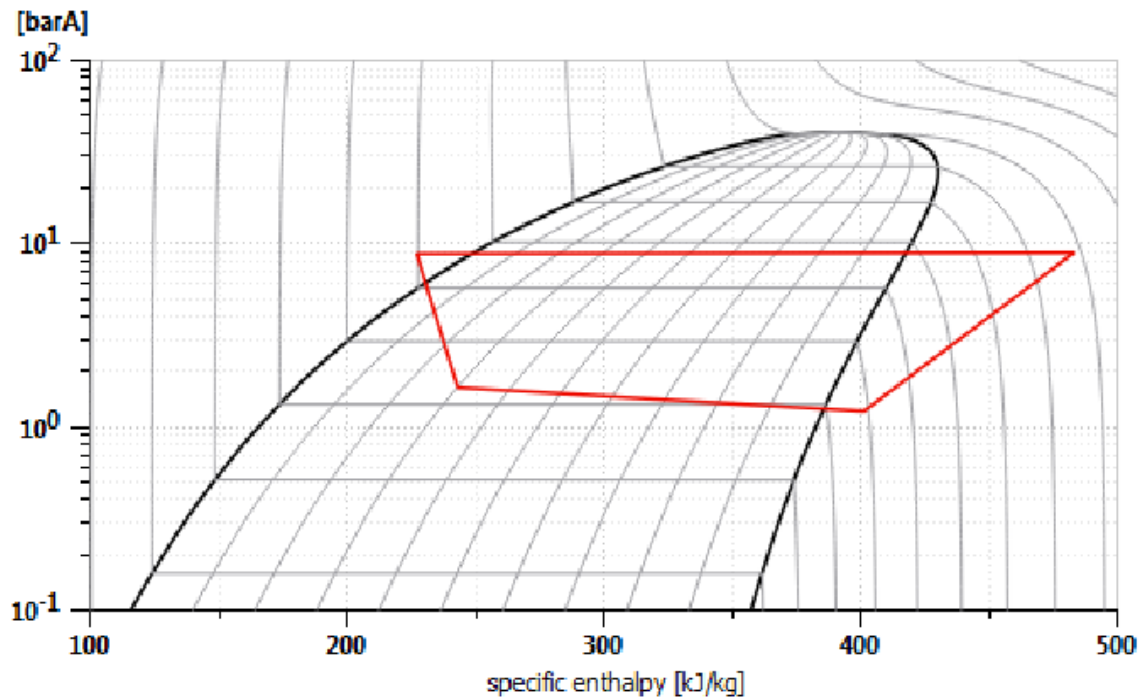


Figure 29. Thermodynamic cycle of the Vapor Cycle in the cold case

The heat flow in the avionic bay is similar to the previous one, since the avionics produce the most of it, but, with a zoom, it is possible to see the differences due to recovery temperature variations.

In the following figures the heat flow, its zoom, the recovery and external temperature are reported.

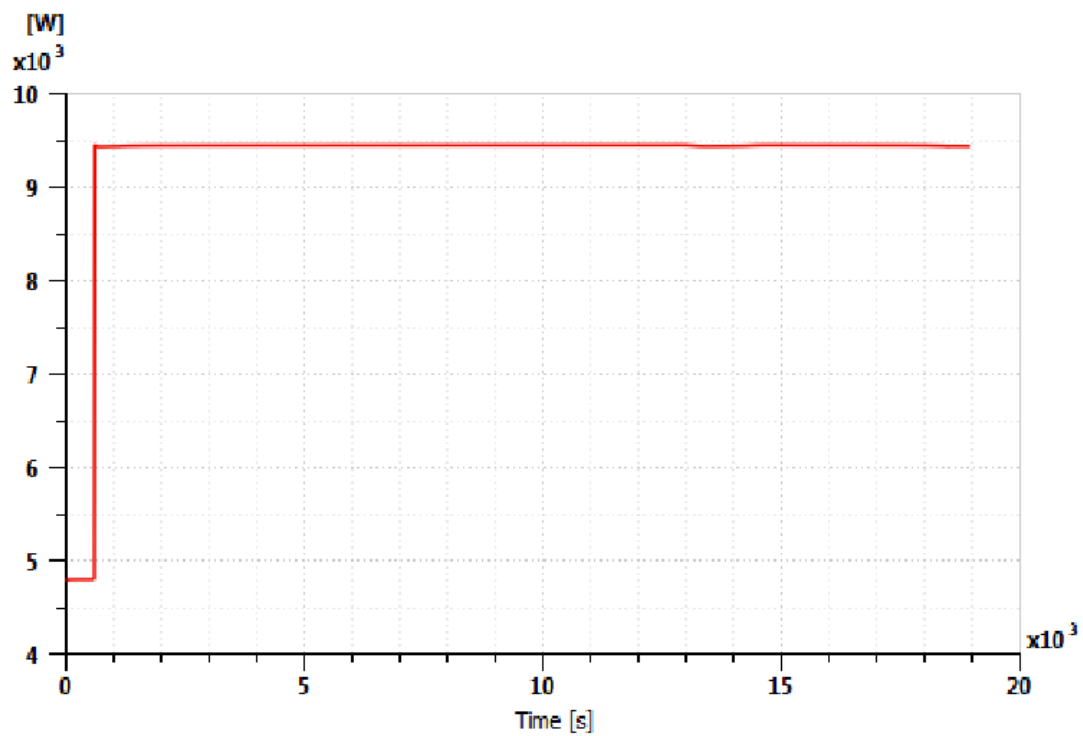


Figure 30. Thermal flow in the avionic bay in the cold case for Vapor Cycle

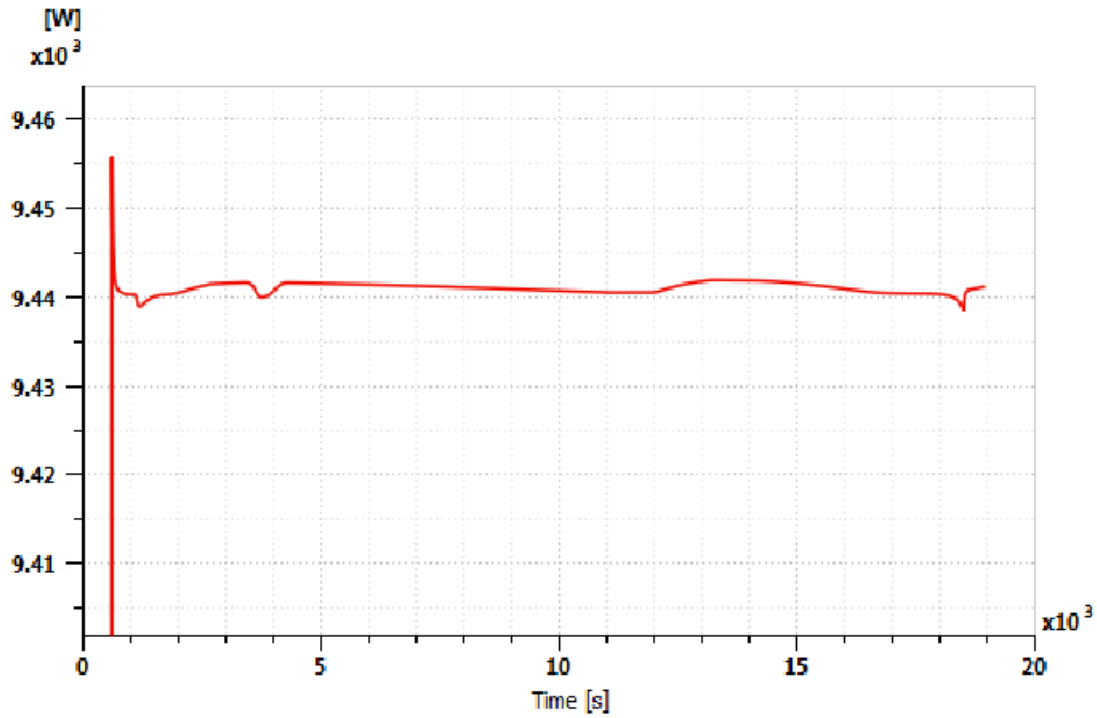


Figure 31. Thermal flow detail in the avionic bay for the cold case for Vapor Cycle

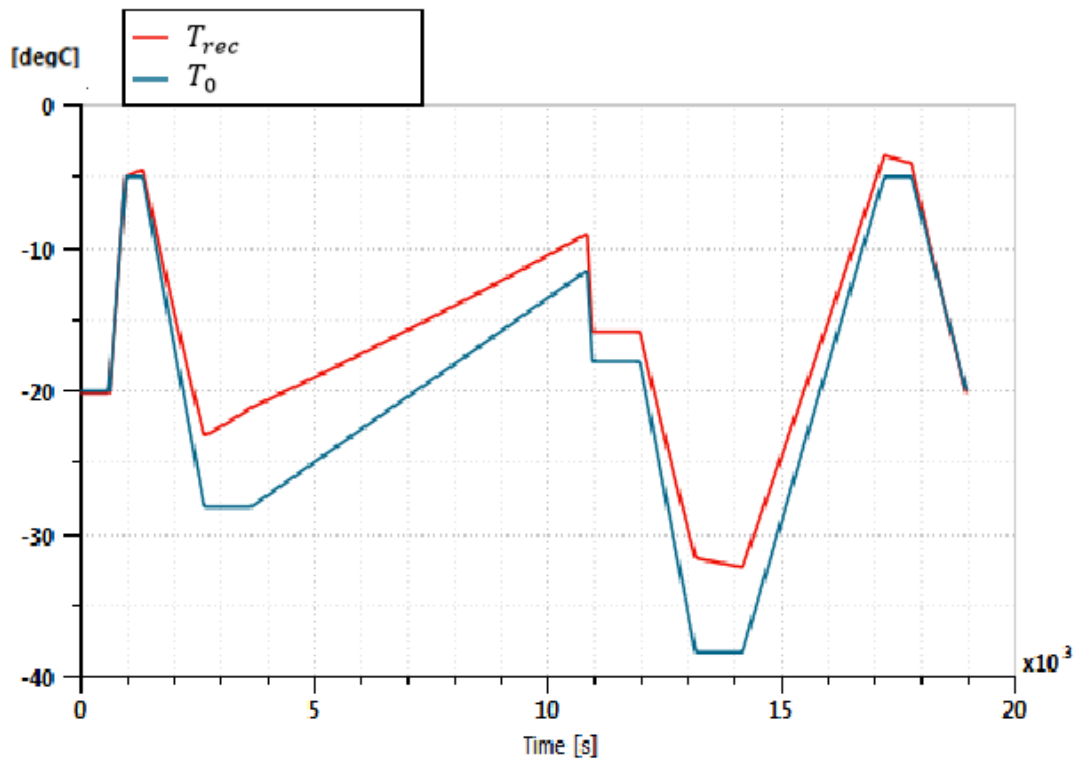


Figure 32. Recovery temperature and external temperature in the cold case

Again, the temperature in the avionic bay stays in the temperature range given by the requirements, as shown in Fig. 33. The system is then capable to maintain the right temperature during every phase of the mission in the whole range of external temperature that goes from the cold case to the hot case.

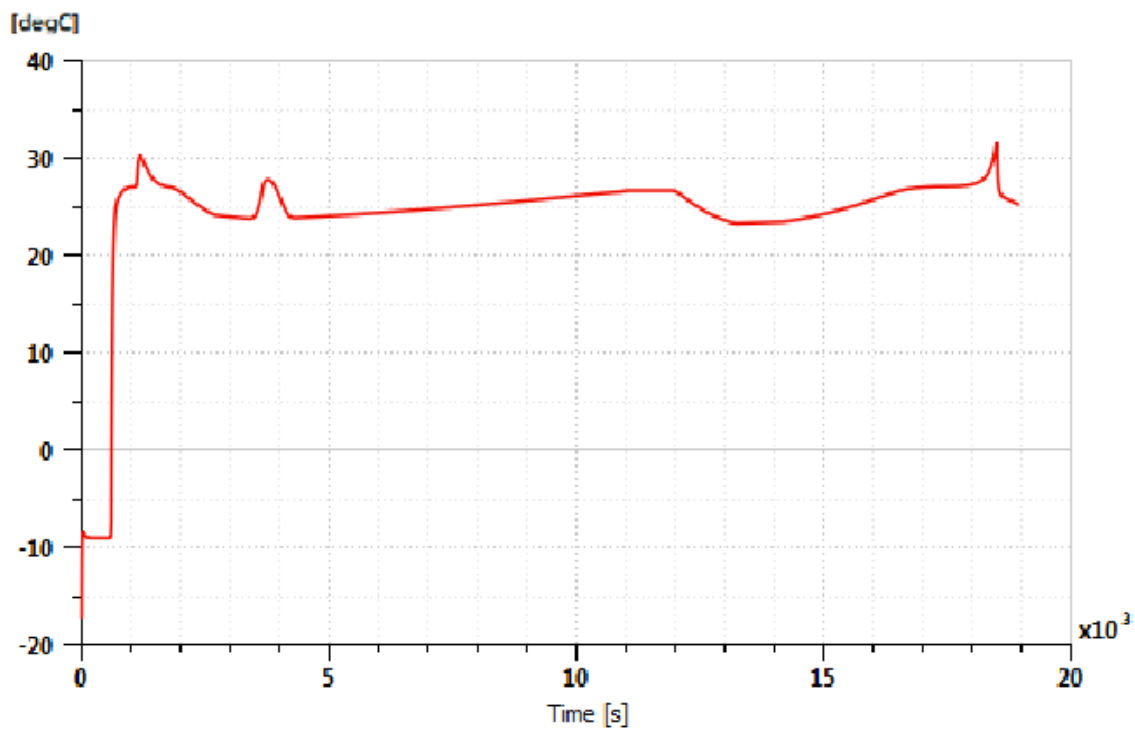


Figure 33. Temperature in the avionic bay for the Vapor Cycle in the cold case

4.3 Air Cycle Simulation

Now, it is time to build the Air Cycle model that is shown in the following figure.

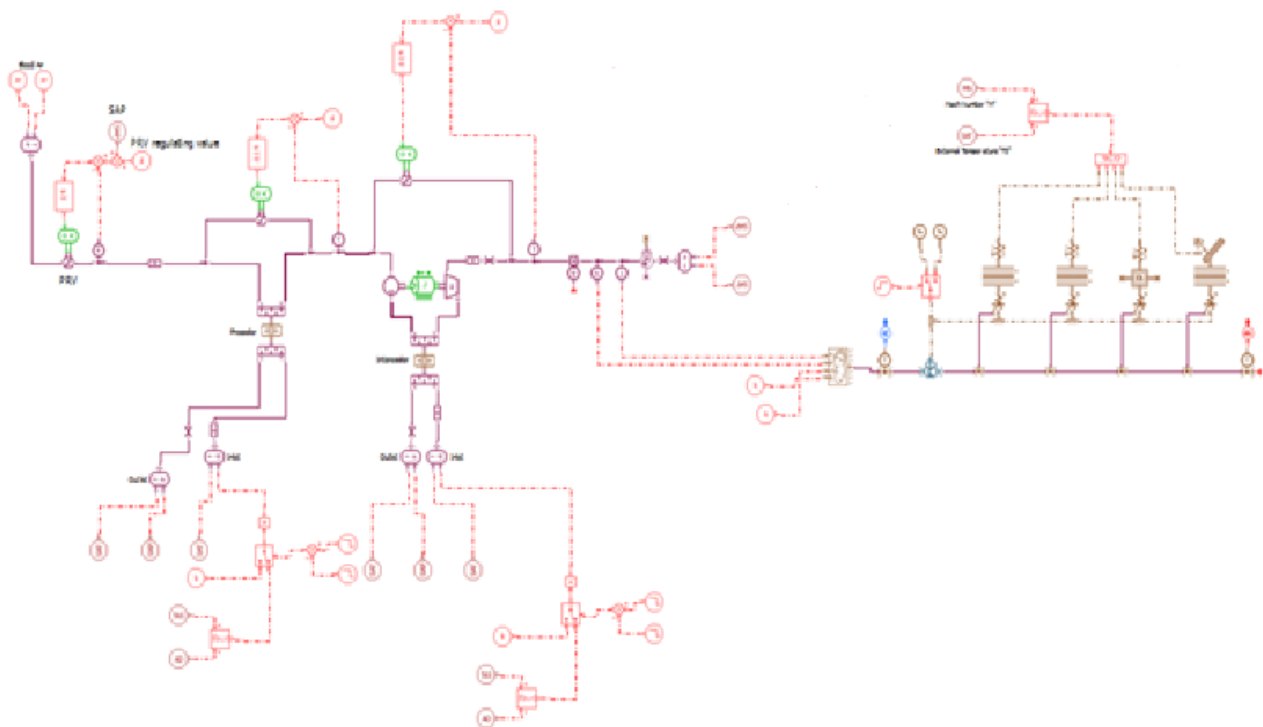


Figure 34. Air Cycle architecture

Considering the hot case, the heat flow in the avionic bay is similar to the previous ones, but, with the zoom, the variations due to the different type of cooling system are visible. The external and recovery temperature are the same of the previous cases.

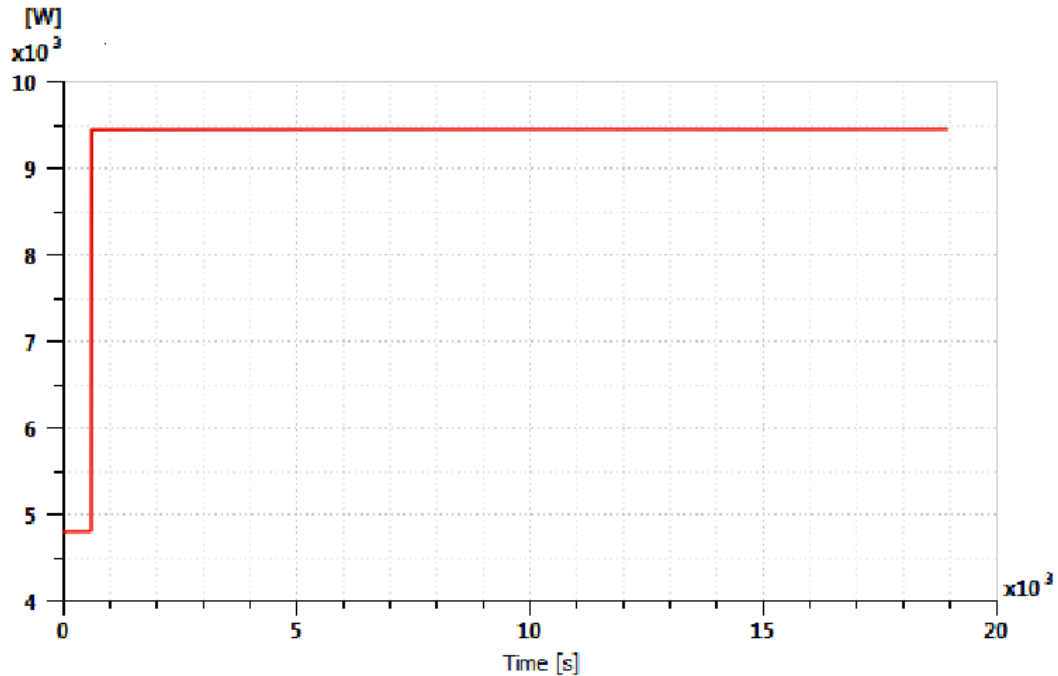


Figure 35. Thermal flow in the avionic bay in the hot case for Air Cycle

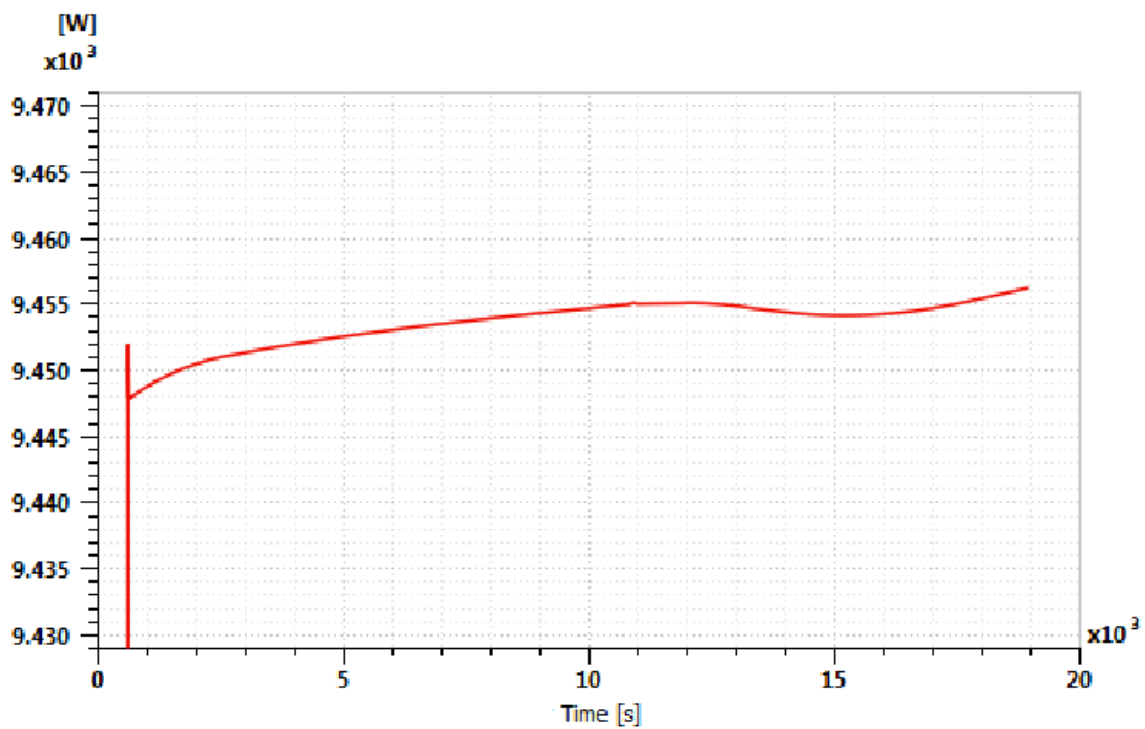


Figure 36. Thermal flow detail in the avionic bay for the hot case for Air Cycle

In this case, the temperature stays again in the given temperature range. Compared with the Vapor Cycle, the bay temperature remains more constant, because the Air Cycle system has three PID controller, one on the engines bleed valve and two at the bypass valves.

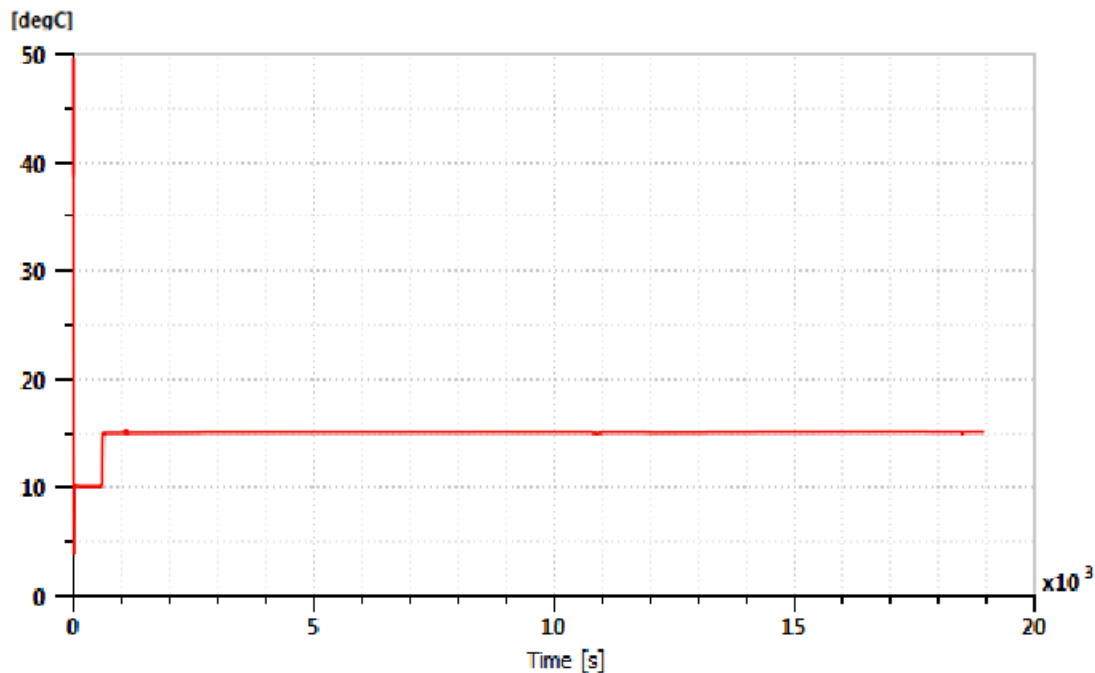


Figure 37. Temperature in the avionic bay for the Air Cycle in the hot case

In the cold case, again, there is a similar heat flow in the avionic bay with few differences due to the recovery temperature. The temperature in the bay stays again in the given range, so the system is well sized.

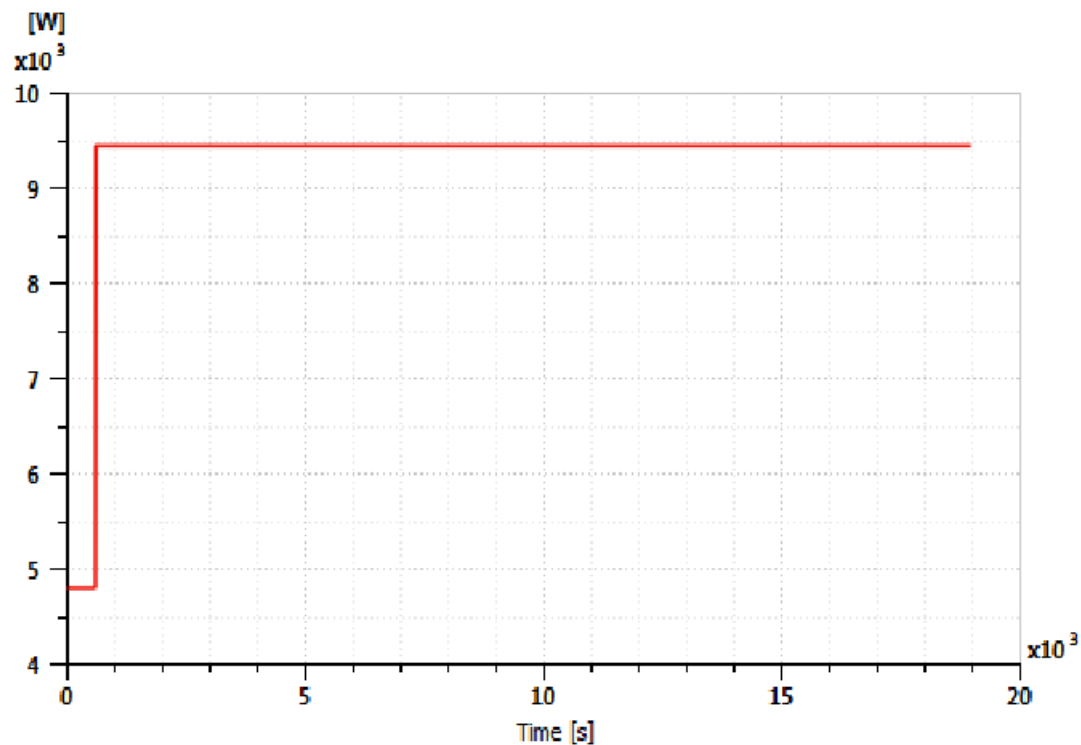


Figure 38. Thermal flow in the avionic bay in the cold case for Air Cycle

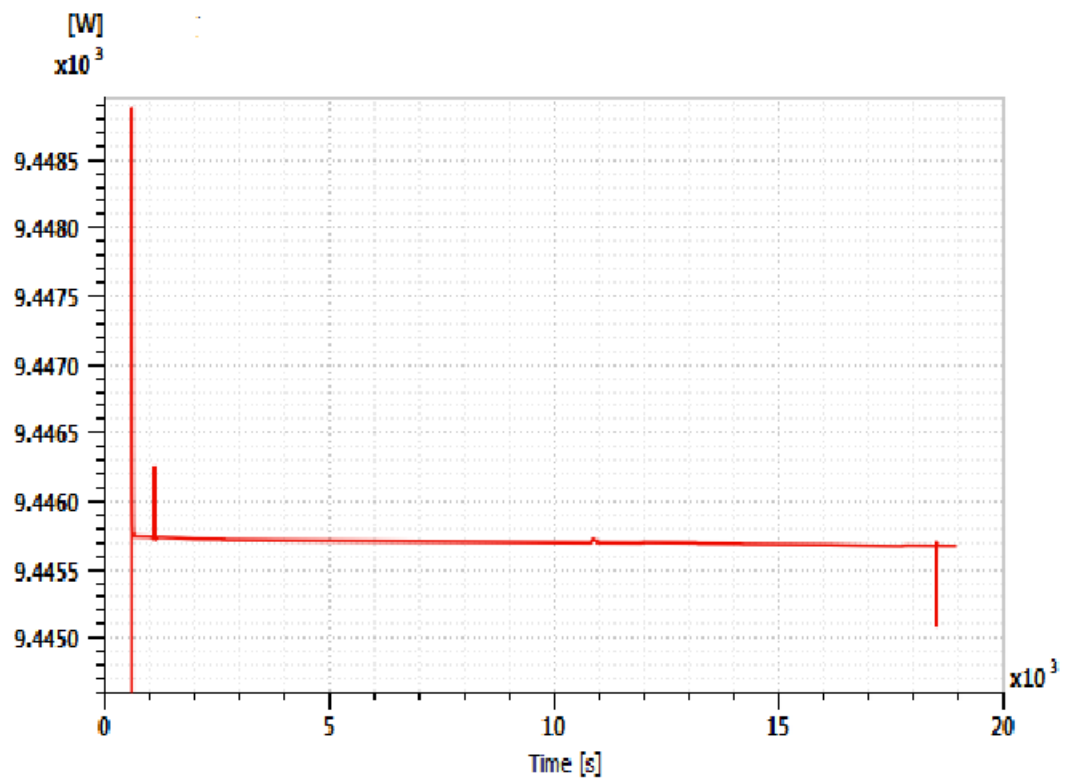


Figure 39. Thermal flow detail in the avionic bay in the cold case for Air Cycle

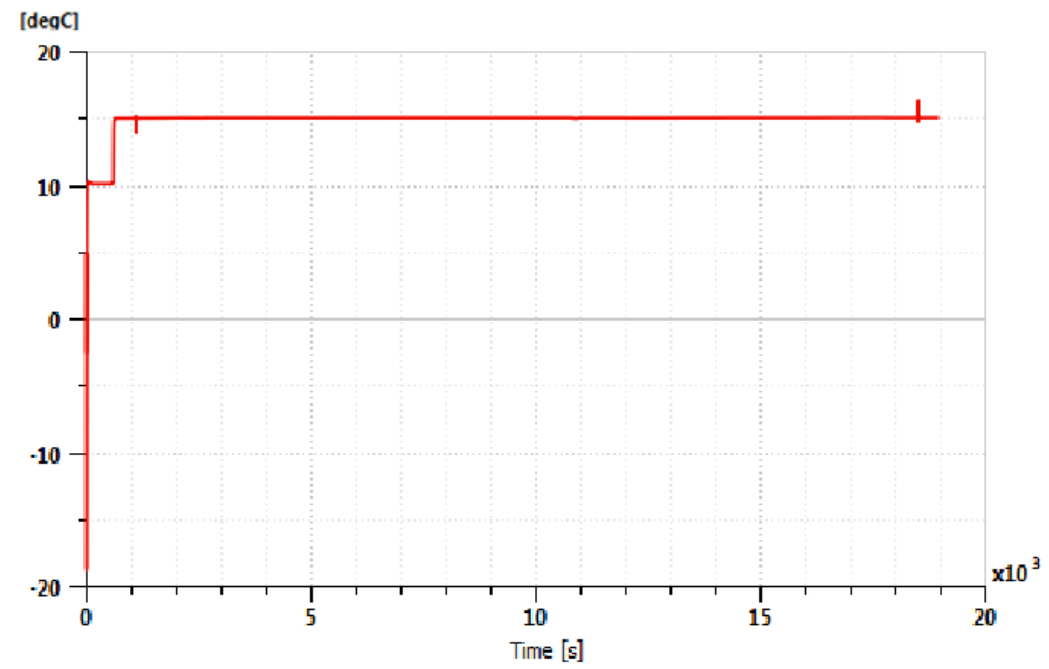


Figure 40. Temperature in the avionic bay for the Air Cycle in the cold case

5. COST ANALYSIS

5.1 Design to Cost

The Design to Cost is an innovative methodology that gives importance to the cost estimation as fundamental part of the design process of a system through a systematic approach that controls the costs of product development and manufacturing.

Recalling the chart related to the Life Cycle Cost, it is possible to see that the costs must be a fundamental part of the project, since, with the product development advancement, it is difficult to modify the costs without return back to previous phase and strongly modify the project.

About the 70% of the project cost will be fixed during the conceptual design phase, while in the development phase the cost will be allocated until the 80% of the total cost. The remaining costs will be fixed during production and operative live.

Instead, the effective expense of the budget in the conceptual design phases will be a little part of the total and grows exponentially in the whole Life Cycle Cost.

Therefore, costs have become an important parameter of design, forming the project triangle that is a triangle with its corner formed from the project scope and schedule, since the quality of the work depends on these three parameters.

These three parameters are competing constraints, because they are dependent by the each other: increasing the scope will increase time and costs, while reducing time will increase costs and reduce the scope and a low budget means increased time and reduced scope.

In particular, the cost depends on several variables such as resources, worker skill and productivity. Therefore, it is important the cost estimation of all the resources needed to achieve the objectives, the budget available and the control of the factors that cause variances in the cost of the system.

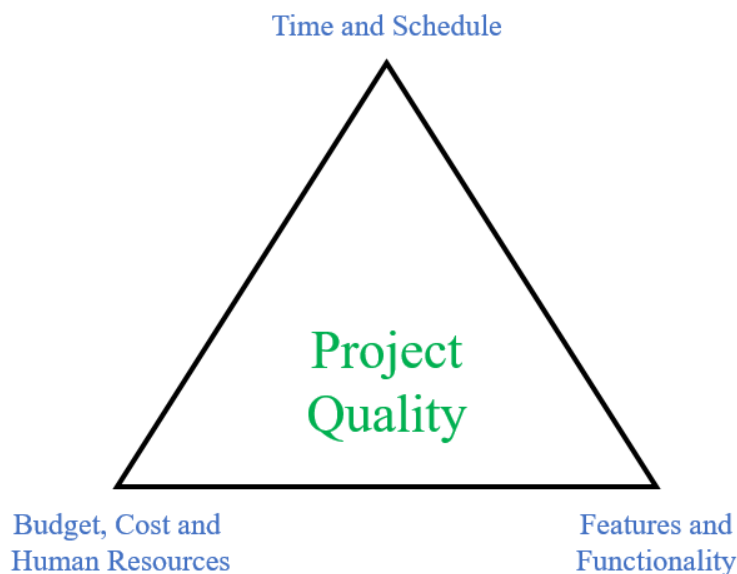


Figure 41. Project triangle

In particular, the cost depends on several variables such as resources, worker skill and productivity. Therefore, it is important the cost estimation of all the resources needed to achieve the objectives, the budget available and the control of the factors that cause variances in the cost of the system.

The results of a well-done cost estimation will be:

- Reduced global life cycle cost of the project in every phase;
- Meeting the customer needs in terms of cost without compromising productivity, safety and quality;
- Reduced time to market, through target cost sharing that allows extra cost anticipation;
- Identifying and reducing useless costs that cause an inefficient production;
- Optimize cost and time regarding production and marketing;
- Evaluating the best compromise between performance and cost.

Therefore, Design to Cost is a decision maker related to cost risks inherent to the project and focuses on the cost of alternatives to make these decisions. Once a decision has been taken, it is possible to proceed with the project.

This means that during the project life cycle there are different phases and each phase must be reviewed in terms of cost, so that it is possible to evaluate the effective cost and compare it with the estimated cost.

Often, the project cost and schedule decrease as the development goes on, and it is important to track the sustained costs, their changes and the causes of these changes.

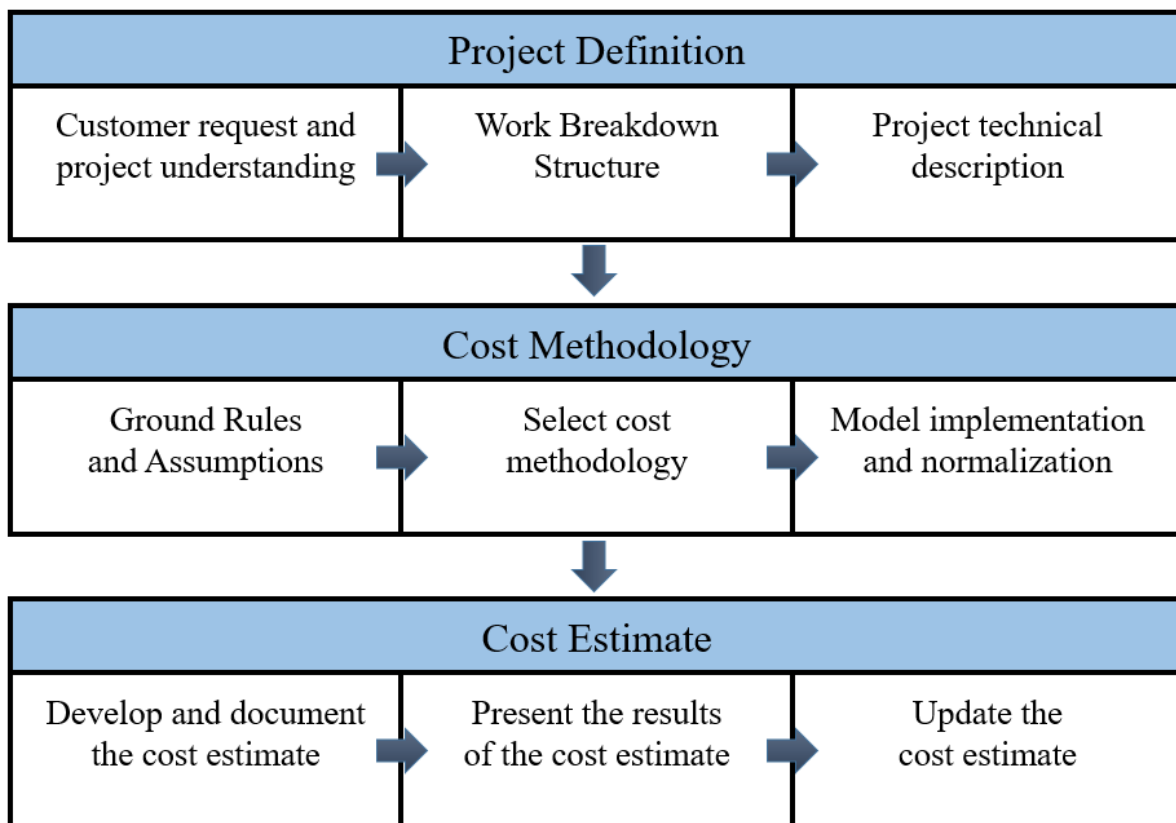


Figure 42. Cost estimation phases

As shown in the Fig. “”, the cost estimating process can be divided in three parts:

- Project Definition;
- Cost Methodology;
- Cost Estimate.

To pass from a phase to another, it is necessary to satisfy some conditions dictated by decision gates that define the maturity of the projects. They represent the most important points of the life cycle of the product, since they assure that new activity does not begin before the accomplishment of the previous ones.

Anyway, the process is not a pure succession of steps, but it has an iterative and nonlinear nature, since the project must be validate in every phase of development, or due to some changes in the baselines or in the assumptions.

5.2 Project Definition

The Project Definition have to:

- identify expectations and requirements and begins to understand the project;
- build a Work Breakdown Structure (WBS) and a technical description;
- identify a technical description of the project development.

These are the bases for cost estimation of the project and are the fundamental steps that may be revisited when new information are available.

Then the first step is to communicate with customer and stakeholder to define enough baselines for the project starting from the given information. This phase need a documentation of what the expectations are, taking into account the purpose of the estimate, the mission needs and goals. It also provide to collect and review all relevant data and the discussion of all the variables with the customer, since it is not always possible to satisfy all the request and a compromise could be needed. Then it is possible to define the workload needed to reach the goals and discuss this factor with the customer.

At this point, the Work Breakdown Structure can be built: it is a first structure of the elements that will be part of the project cost estimation. The WBS divide the project in easily manageable parts organized in an hierarchical structure to facilitate the control of the cost and the schedule.

This structure contains the following information related to the project:

- project planning and scheduling;
- cost estimation and budget formulation;
- project status reporting (schedule, cost, workforce, performance);
- definition of the scope and the specification given by contract;
- documentation.

Then a technical description is defined, providing qualitative and quantitative descriptions of the project, so that the team can understand the project description and estimate the cost. It is necessary to identify those factors that most impact on the cost, review the relevant project data, its characteristic, the different configurations, system risks and so on.

5.3 Cost Methodology

The Cost Methodology is inherent to the approach used for the estimation: this phase provide to the definition of the ground rules and assumptions that most fit the cost model of the system under analysis. In addition, as the analysis goes on, these ground rules and assumptions may be refined.

To define the ground rules and assumptions, there are some points to follow:

- Define the scope of the estimation through a set of schedule ground rules and assumption, specifying what costs are going to be included;
- Gain approval and agreement from the cost estimate reference points;
- Agree with vendors, customers and stakeholders on the ground rules;
- Document the ground rules during the whole process.

These ground rules must provide a definition of the project and of the estimation, letting all the people involved to understand the cost that have been considered. Following this methodology, it is possible to allow a comparison between the present project and the future ones.

Moreover, the ground rules and the assumptions have to refer to both global and element specific.

The first ones are applied to the entire cost estimation, while the second ones are related to each element of the Work Breakdown Structure and goes in the detailed estimation for the specific element, giving information about unit cost, quantity etc.

In addition, to improve the estimation it is useful to have a description of the system characteristics, the mission details, data related to maintenance and logistic support, number of flight per year, operative lifetime, safety level and commercialization approach.

First of all, it is necessary to identify the scope, that define the activities, the hardware elements and the quantities needed by the cost estimate, that could include the costs of designing and fabricating specific components or the costs related to their purchasing, testing, transporting and assembling.

In addition, costs depending on the efficiency and developing time required by contractors and customers are needed, since fulfill these request will influence the cost estimation.

Another factor of influence will be the budget profile due to the dependence of the availability of resources during different period of the program may influence activities, time and costs related.

In fact, the availability of resources and information will include factors like the skill level of the specialists, the labor rates, the number of person involved in every task and the efficiency of the tools involved.

A Make vs Buy Decision could be useful to decide if it will be more convenient a manufactured items, which will require design effort, tooling and product setup, or a purchased item that will require integration and testing in the system contest.

In addition, these ground rules have to take in account the number of units, the spares, the prototypes and their costs. This choice strongly affects those that will be the profit of the system, considering both the production and operative costs. This will also include the availability of existing facilities, their modifications or the assembly of new ones.

During the estimation, others important ground rules are those related to the risk reduction or mitigation, which describe the activities used by estimators to reduce the probability of those effects that could increase the project effort. These risks are estimated during every phase of the project and, obviously, they have their own documentation.

Another factor related to the ground rules will be the currency inflation, since the currency value will change over the years and, so, it is necessary to specify the year in which the estimation is considered.

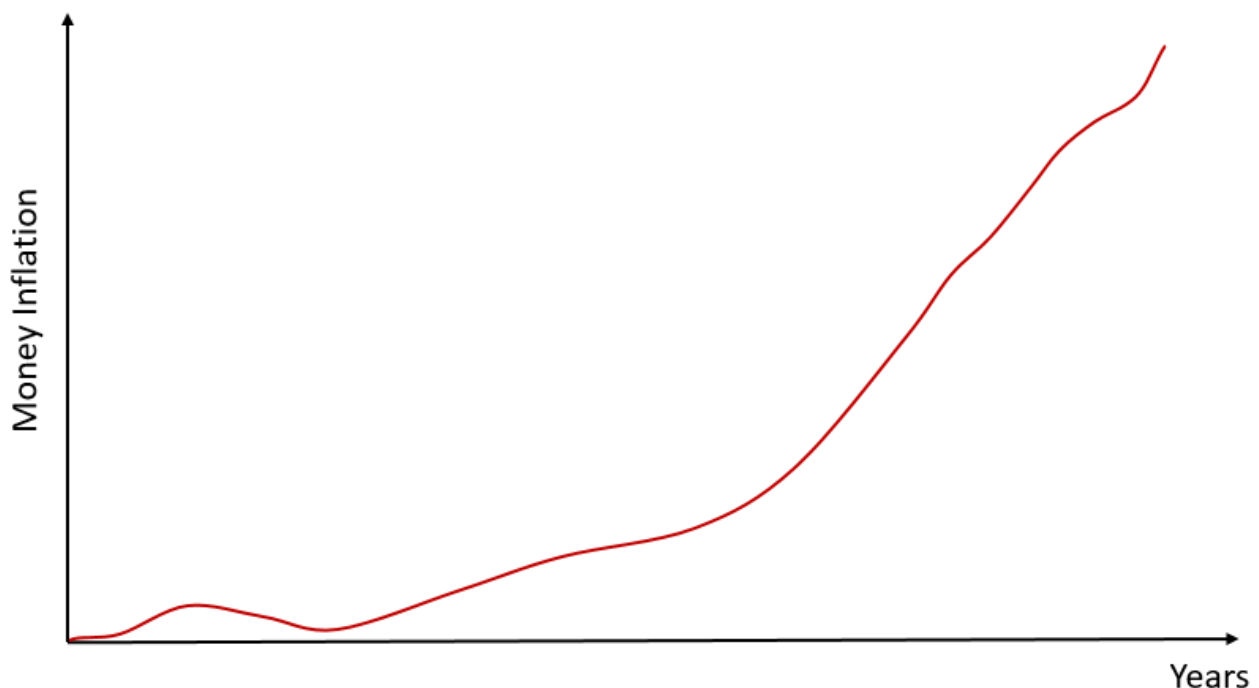


Figure 43. Inflation trend through the years

At this point, once the ground rules and the assumptions have been fixed, it is possible to choose the most appropriate estimating methodology for the cost estimation, according to the choices done till now.

There are basically three type of cost estimating methodology:

- *Analogy Methodology;*
- *Parametric Methodology;*
- *Engineering Build-Up Methodology.*

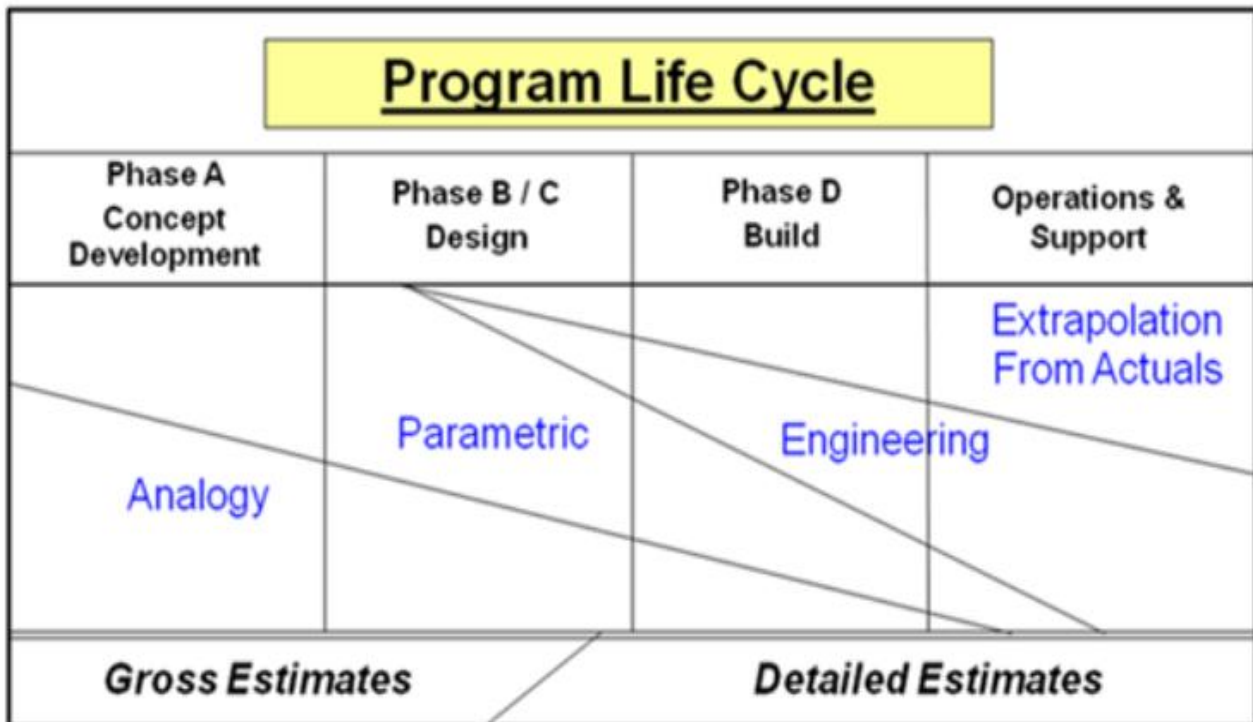


Figure 44. Cost estimation methodologies related to the Program Life Cycle ref.[2]

The *Analogy Methodology*, to estimate the cost of a system, refers to the costs of similar systems using some adjustment due to the differences. The systems must be similar in terms of design and operations.

This methodology requires the effective cost data from the past programs that will be the starting base of the new cost estimation. Then the data are adjusted depending on the difference between the two systems in term of complexity. These adjustments will be subjective and will compromise the validity of the estimation, so the opinion of experts is needed due to obtain a good estimation. It is possible to use an analogy approach when there are enough technical data available to perform an adequate comparison and adjustment or if the system is not well defined yet.

The advantages of this method are:

- Less time needed to obtain a first estimation;
- Availability of historical data;
- Easy to understand;
- Accuracy for minor changes respect the reference system.

Its disadvantages are:

- Can be difficult to identify analogy;
- Based on subjective adjustment;
- Require normalization;
- In some case, there are only few historical data.

The *Parametric Methodology* uses statistical relationship, performance characteristics, personnel skills, design complexity and other variables to perform a cost estimation. This methodology is useful in the case of few known data such weight and dimensions. The Fig. 45 shows the step of this methodology.

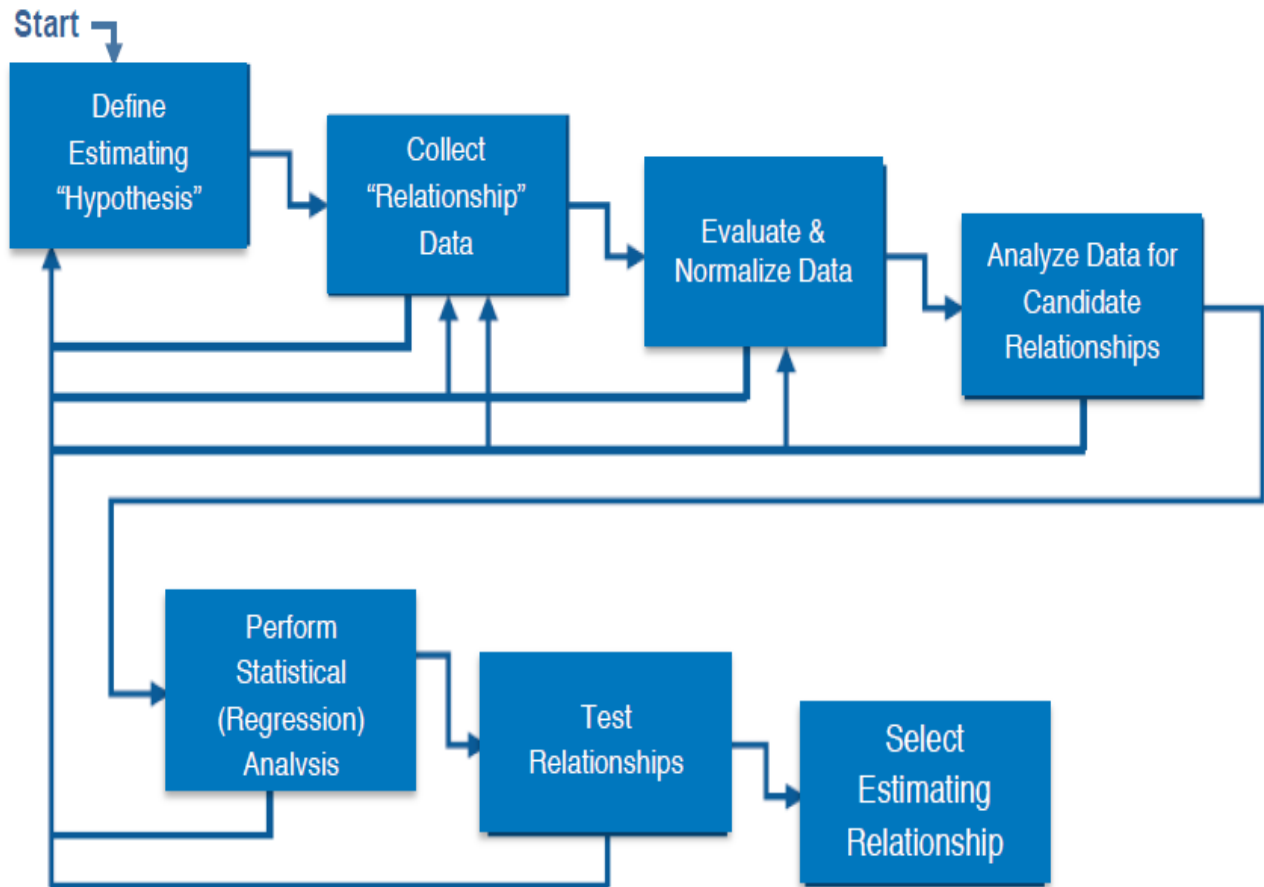


Figure 45. Parametric Methodology phases ref[3]

The process is iterative and has the scope of identify a cost estimating relationship in the form of equations involving dependent variables, which are influenced by changes, and independent variables, which are not influenced by them. In this case, the cost will be the dependent variable, while the independent variables will be the cost drivers identified by the system parameters. Anyway, the cost estimator have to choose what cost drivers to take in account, since each of them strongly influences the cost of the system.

In every project and estimation, the firsts cost drivers come are the requirements, which define a first input for the analysis. However, in most case, requirements may not be enough to obtain ad adequate estimation, so the results have to be adjusted using a series of other cost drivers that may influence the project cost. Cost drivers could be anything that may affect the cost such as methodology, personnel experience, weight and others factors.

Other inputs may be some constraints that have to be considered, such as budget constraints, labor constraints, architecture and process.

The outputs of the estimate may be:

- Manpower loading, which is the number of personnel allocated to the project as function of time;
- Project Duration, which is the time required by the project;
- Effort, which is the effort related to the project completion measured in person-months;
- Refined requirements, since the project evolves continuously;
- Work Breakdown Structure;
- Refined Architecture, since new component may be obtained.

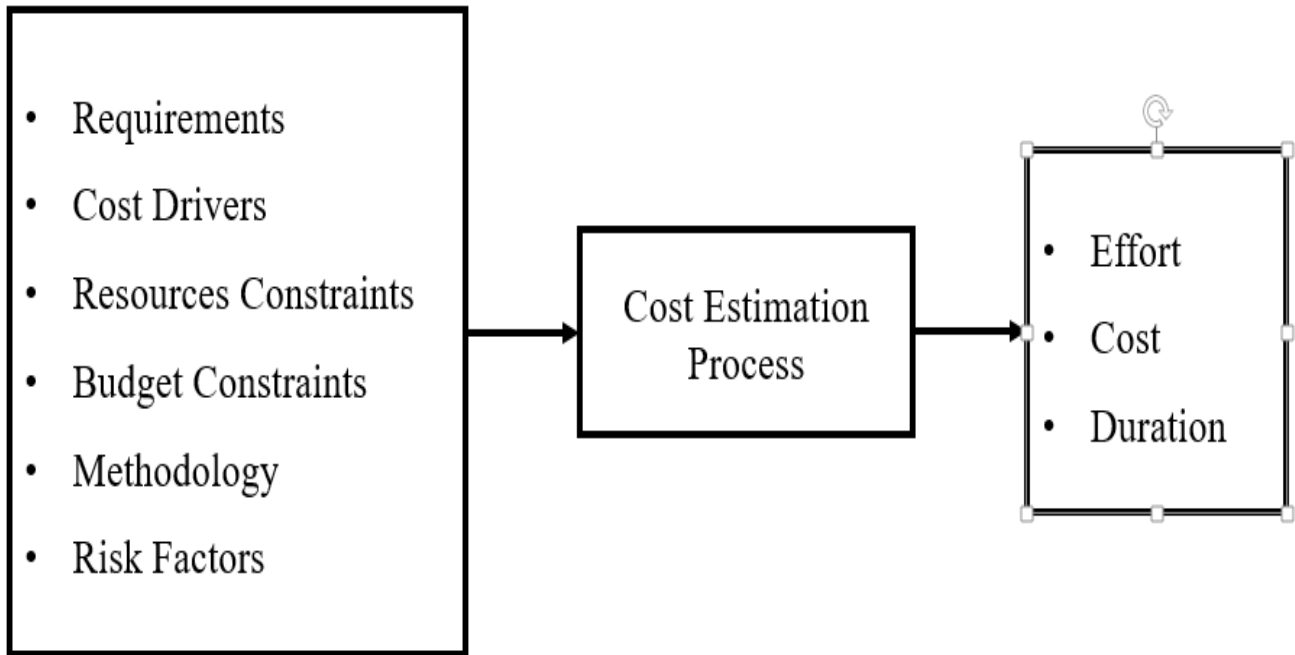


Figure 46. Parametric cost estimation process

The advantages of this methodology are:

- The cost estimating relationships can be used to rapidly evaluate the cost of changes;
- Evaluate the cost from an objective point of view;
- Reliability of the method based on logical correlation and data.

The disadvantages of Parametric Methodology are:

- Definition of the cost estimating relationships requires a lot of time and resources;
- Loss of effectiveness in lack of data;
- Requires a lot of documentation related to the equations, validation and data;
- Difficulty to understand the cost estimating relationships for external users.

The *Engineering Build-Up Methodology*, or bottom-up estimating, provide a cost estimation of the project starting from the cost of the elements of the Work Breakdown Structure, then sums them and adjust the results to considering overheads. This method starts from the lowest level of detail that is easily subject to an estimation of the costs and effort, distinguishing the cost for labor from

the cost for materials and, once they have been evaluated, overheads such as general and administrative expenses, fee and other direct cost are added.

This method requires experienced people that assure the consistency, the reasonability and the completeness of the model, and they have to test and validate it. In addition, it is applicable to mature projects, since a lot of information are required.

The advantages of this method are:

- Intuitiveness;
- Cost given for each element;
- Calculation errors in one element does not compromise the cost estimation of the other components;
- Reusability of the model.

The disadvantages are:

- Elevate costs required to obtain a build-up estimate;
- Not easily changeable;
- Requires different built-up for every scenario;
- Cost drivers are not identified;
- Requires relationships between the elements.

Regardless the choice of the methodology, the cost estimator will perform two essential steps:

- Data collection;
- Data normalization.

Data collection is an extremely costly activity in terms of time and resources, since data needed are not always clear and due to the evolution of the requirements during the project.

In addition, the collected data may need to be normalized before applying them to the cost estimation. The data can also help in prevent or reduce risks resulting in saved time and resources.

Therefore, data collection occurs in order to understand the project and to support the choices done in the following steps. Data are collected by means of:

- Questionnaires;
- Model specifications;
- Interviews;
- Papers, documentation and statistical analysis;
- Project technical and schedule data such as budget, contract, labor rates etc.

Then, collected data may require a normalization that consist in adjustments of the data considering the effects of inflation that reflects the decrease in the purchasing power of the currency over the time. Therefore, in the estimation, the inflation factor represent a multiplier used to account these changes over time and the outlays over a number of years.

Others adjustment can come from the improvements in the process as showed by the learning curves. In fact, the production rate may also affect the data set and, as the production goes on, later produced units will cost less due to improvement of the process and the better understanding of the project.

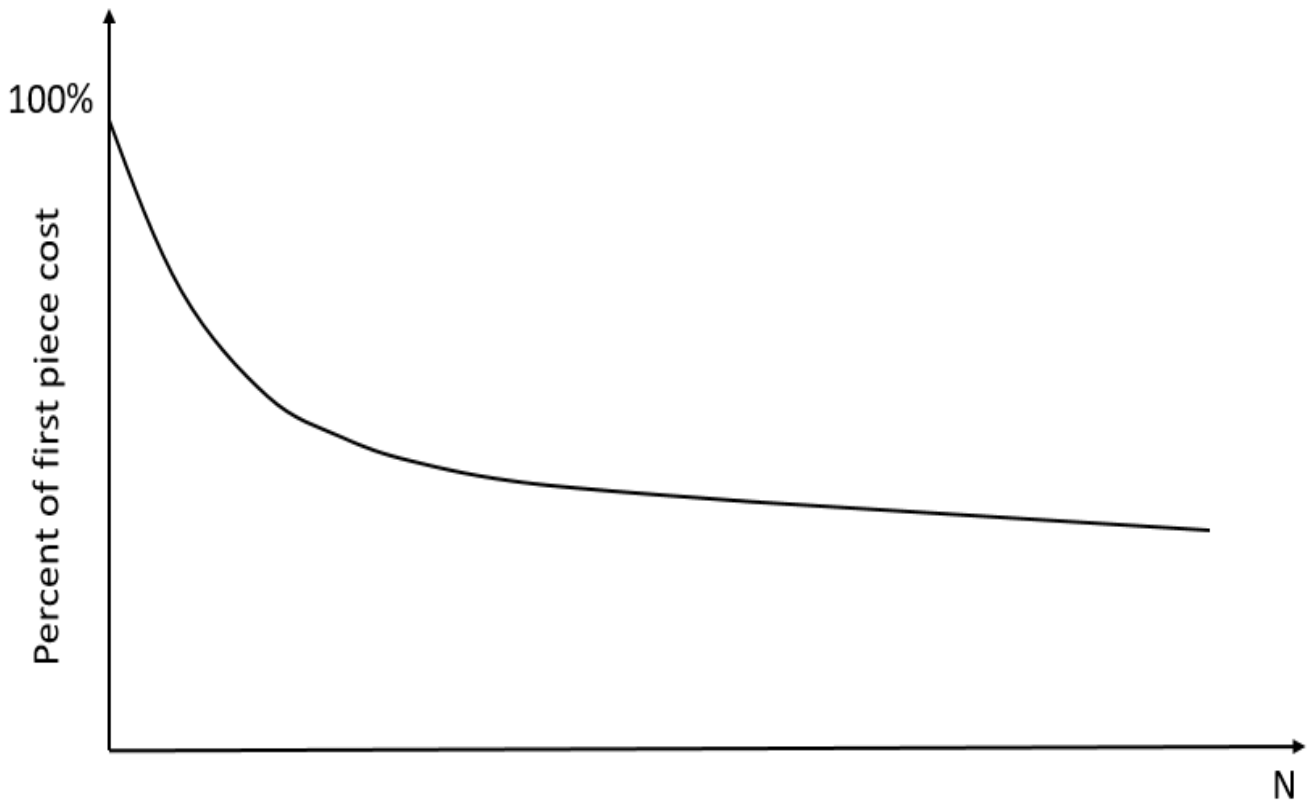


Figure 47. Labor Learning curve

In the end, normalized data should be validated to ensure a consistency of the data collected, in order to avoid anomalies in the results.

5.4 Cost Estimate

The Cost Estimate regards the effective cost estimation, the development of the cost risk assessment, the drawing up of all the documentation and the results.

The main task of this step is to define an initial Life Cost Cycle by means of the following activities:

- Verify the ground rules and the assumptions;
- Use normalized data to populate the model;
- Grant that the estimate includes all the costs;
- Calculate the costs running the model;
- Divide the estimate through the phases;
- Inflation adjustment;

- Update the cost through the previous estimation.

The cost estimation proceed point to point: in fact, with every model run, a point estimate is calculated and it will be the starting point for the calculation of the following point estimate. These points are subjected to uncertainty, so it is possible that they will need some changes, since they are not definitive once calculated.

The next step is to develop and incorporate a cost risk assessment, in order to understand the consistency of the project and evaluate the future overlays. It is an analytical process that identifies and analyzes the critical risks of the project in a defined set of cost and schedule constraints, by means of uncertainty errors. These uncertainties bring to a probabilistic range of cost in which the project must enter.

This cost risk assessment comes from some activities:

- Determination of the cost drivers and risks of the project;
- Developing of probability distributions for the cost drivers;
- Developing of probability distributions for the model uncertainty;
- Running the risk model;
- Identifying the probability of lesser cost respect to the point estimate.

By means of the cost risks analysis, it is possible to quantify the budget needed to obtain an adequate level of confidence.

In addition, the cost drivers have to be submitted to a sensitivity analysis, since each of them influences the cost estimate with minor or major changes. Understand how sensitive the project is to changes in the drivers may help the estimators to take programmed decisions when changes in the project are needed.

During all the processes described until now, data relative to estimations, changes and processing are captured to provide a written documentation that justify the project and the taken decisions.

The final documentation should contain data relative to the estimation of each element, the inputs used, the point estimate for the Life Cycle Cost and a description of the cost-risk analysis. Each argument should be fully explained and in the clearest way possible.

In particular, a complete documentation should contain:

- Models used;
- Methodology applied;
- Explanation of each part of the estimate;
- Reference to data sources;
- Explanation of cost drivers;
- Description of the Life Cycle Cost and the factors that influence it;
- An analysis on the future overlays that have not been included;
- The sensitivity analysis;
- Comparison with previous estimate.

A well-documented estimate takes consistency to a project making it more defensible and credible. In fact, it allows an external user to understand the assumptions of the project and to reuse it for a new estimate or simply to modify it.

Obviously, the level of detail can differ depending on the estimate, but it is possible to define a minimum amount of detail that should be enough to allow another analyst to understand the work done.

At the end of the estimate and its documentation, it is appropriate an external review of the work, in order to check the validity of the work or find issue in it before the presentation.

5.5 Hardware Cost

Generally, a system will consist of hardware components and software components.

Therefore, when performing a cost estimate, the analyst should do an analysis of both of them and then consider the integration costs, in order to obtain the cost of the whole system.

As told previously, the main factors that influence the cost estimate are those that goes under the name of cost drivers. Therefore, it is necessary to identify what these cost drivers are.

For the hardware components, it is possible to define some drivers that are universally valid regardless of the component.

A hardware component is the assembly of the physical part of the system that allows the system to operate.

Then, hardware components cost drivers could be:

- Weight, since increasing weight means more material needed, but it could affect operative costs too, causing an increase in the consumption;
- Volume, intended in terms of encumbrance, because this could take to a need of platform for allocation;
- Type of Equipment, that will define the qualities and the performance level of the system;
- Material can affect the cost, due to the different value of different materials;
- Complexity, more complex component will require more machining in order to obtain the final product;
- Production Technology, since a component can be obtained following different processing, but some of them could be more expensive than the others;
- Operative Scenario, because the operative scenario will define the quality of the components and the relative cost to obtain them;
- Number of Units Produced, since, following the labor learning curve, the more units will be produced, lower the cost of the N^{th} unit will be;
- Team Experience, obviously a more experienced team will use less time to develop and produce a certain component;

- Reliability, influencing the costs relative to maintenance.

Moreover a hardware component can be provided with internal development or purchasing it from external producer.

In the first case, the cost estimate will include the development costs and the component could be tailored for the system, while in the other case the estimate will consider the purchasing cost and could include some integration costs needed to adapt the component to the system.

5.6 Software Cost

Usually software project estimate are more difficult and use to go over budget, over deadline or both. When considering the cost of the software, the factors involved will be different respect to the hardware ones. Also for the software cost estimation, the results will be functions of some cost drivers and depends by the methodology applied.

First, it is necessary to evaluate a characteristic metric dimension for the code that will influence the cost. In particular, the cost of the software may be estimated choosing between two characteristic:

- Source Lines of Code (SLOC), which are the needed line of code to accomplish the function attributed to the system;
- Function Points (FP), which represent the functions and the operations performed by the system to accomplish its tasks;
- Object Points (OP), which are similar to FP, but count the number of reports, screens and modules classifying them as simple, medium and complex.

Each Source Lines of Code may represent a command or a function needed to perform some tasks and each of them will require time to be developed and refined. Some Source Lines of Code may be also constituted of void lines and comments (CLOC), since, generally, they are counted as:

$$SLOC = NCLOL + CLOC$$

In the cost estimate of a software using SLOC, it is also necessary to consider how the written code is, in terms of precision and compactness. This makes the estimate of the SLOC a very difficult task in preliminary phases, since it does not consider the complexity of the instructions and the exact number of SLOC for every task.

Instead, Function Points are based on the system functionalities regardless of the language and so they are easily estimable. In order to calculate the Function Points, it is necessary to describe the system by means of a functional analysis and then identify the different activities performed by the system. Therefore, Function Points are estimated starting from a subjective point of view and, so, different people may estimate different number of Function Points for the same system.

Therefore, it is possible to put in relationship the two unity of measurement. In fact, a factor of conversion, obviously dependent from the programming language, allows the transformation from one to the other.

Programming Language	SLOC/FP
Assembler	320
C	128
C++	55
Pascal	90
Ada	70
Cobol/Fortran	105
Java/VisualBasic	35
HTML-3	15

Table 3. SLOC/FP conversion ratio ref.[8]

The choice of the programming language will be another cost driver that influences the cost estimate, since each language could use a higher or lower number of SLOC to perform a specific task. This will obviously influence the productivity and then the costs.

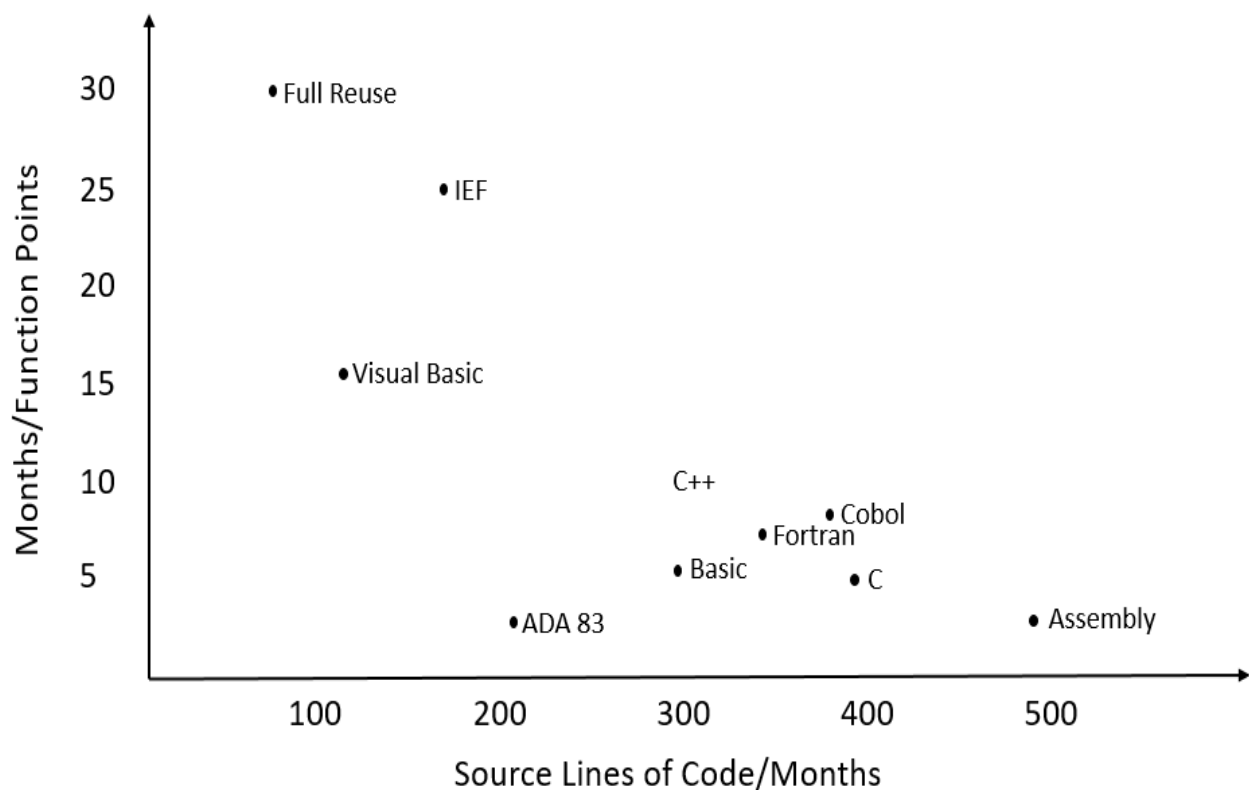


Figure 48. Relation between SLOC and FP in terms of production ref.[9]

In fact, it is possible to fix some parameters like the productivity in terms of Function Point per hour (or SLOC per hour), which is the number of hours needed in order to develop a Function Point (or a SLOC).

Another parameter will be the delivered functionalities related to the productivity rate for those functionalities that can be calculated as Total Cost/Delivered Function Points (or SLOC).

Others cost driver may be:

- the experience of the programming team, since a more experienced team may require less time to develop a certain number of SLOC or Function Points;
- the required calculation power in terms of time response of the software;
- the level of DAL, related with security requirements;
- the number of new SLOC, modified SLOC and reused SLOC;
- function complexity;
- operative applications, since a military software will require higher costs due to the requirements.

5.7 Software Cost Estimate Methodologies

In literature, a lot of software cost estimation exist, each of them based on certain cost drivers. Here are some of them:

- COCOMO (Constructive Cost Model);
- COCOMO II;
- COSYSMO (Constructive System Engineering Cost Model);
- REVIC (Revisited Intermediate COCOMO);
- F-PROM (Effort Prediction Objects Model);
- Object-Oriented Decomposition;
- SLIM (Software Life-cycle Management);
- ESTIMACS;
- Price-to-Win;
- Price-S (Programming Review of Information Costing and Evaluation-Software).

COCOMO is a models used in software cost estimation that was published in 1981 by Barry Bohem. It uses algorithms and parameters in order to provide an estimation of the effort and the schedule of the project.

The main metric used by COCOMO in cost estimation is lines of code counted in thousands, but could also be used function points or object points.

The first model of COCOMO was named COCOMO'81, which has three different models that can be used during the life cycle of the project depending on the maturity of the project:

- Basic Model, which is used for small software at low level of maturity. It does not use the effort adjustment factors, since it will be too approximate;
- Intermediate Model, which is used for more detailed project and uses the effort adjustment factor EAF and has different value for the constants a , b , c and d ;
- Advanced Model, which is used for complete projects to refine the estimate.

The equations used for the COCOMO cost estimation, as previously said, are in terms of effort and schedule duration. Considering a general model, these equations are:

$$PM = a * (KLOC)^b * EAF$$

$$TDEV = c * (PM)^d$$

Where:

- *PM* is the effort in person-month;
- *EAF* is the effort adjustment factor;
- *TDEV* is the schedule time;
- *KLOC* is the number of lines of code counted in thousands;
- *a*, *b*, *c* and *d* are constants based on the mode used.

Then, the total cost estimate for the project may be obtained considering the cost per hour spent, which will depend on the developers. Once this cost is brought to the cost per month, a simple equations let the estimator to obtain the cost of the software:

$$TC = PM * CM$$

Where:

- *TC* is the total cost estimation of the software;
- *PM* is the effort in person-month again;
- *CM* is the cost per month considered.

The four constants *a*, *b*, *c* and *d* depend on the work environment, the size and the constraints and are divided in three modes:

- Organic, which is used for small software in an in-house environment;
- Embedded, which is used when the system is strongly related to hardware, operational procedures and regulations that define tight constraints;
- Semi-detached, which is at an intermediate level between the two previous modes.

Model	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.2	2.5	0.32

Table 4. Values of a and b depending on the model type ref. [4]

The effort adjustment factor *EAF* let the analyst to tailor the estimation depending on the conditions of the development environment.

In the case of Basic Model COCOMO, as previously said, the effort adjustment factor is not considered, so it will be set to a unitary value.

The Intermediate Model COCOMO considers fifteen different cost drivers that can be used to calculate the *EAF*. These costs drivers are divided in four categories:

- Product Attributes;
- Computer Attributes;
- Personnel Attributes;

- Project Attributes.

Moreover, cost drivers have a value depending on a scale obtained from statistical data that rates them from a level that could go from Very Low to Extra High. Once all the cost drivers have been assigned, it is possible to multiply them all together to obtain the EAF.

A table of the cost drivers and their rating is reported below.

Category	Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes	RELY Required Software Reliability	0.75	0.88	1.00	1.15	1.40	-
	DATA Database Size	-	0.94	1.00	1.08	1.16	-
	CPLX Product Complexity	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes	TIME Execution Time Constraint	-	-	1.00	1.11	1.30	1.66
	STOR Main Storage Constraint	-	-	1.00	1.06	1.21	1.56
	VIRT Virtual Machine Volatility	-	0.87	1.00	1.15	1.30	-
	TURN Computer Turnaround Time	-	0.87	1.00	1.07	1.15	-
Personnel Attributes	ACAP Analyst Capability	1.46	1.19	1.00	0.96	0.71	-
	AEXP Applications Experience	1.29	1.13	1.00	0.91	0.82	-
	PCAP Programmer Capability	1.42	1.17	1.00	0.86	0.70	-
	VEXP Virtual Machine Experience	1.21	1.10	1.00	0.90	-	-
	LEXP Language Experience	1.14	1.07	1.00	0.95	-	-
Project Attributes	MODP Modern Programming Practices	1.24	1.10	1.00	0.91	0.82	-
	TOOL Use of Software Tools	1.24	1.10	1.00	0.91	0.83	-
	SCED Required Development Schedule	1.23	1.08	1.00	1.04	1.10	-

Table 5. COCOMO cost drivers ref. [5]

The Advanced Model COCOMO uses some characteristics of the Intermediate Model, since it uses the same cost drivers, but, in this case, they have different values and considers the phase of the project reached. In addition, this model requires the division of the software in different modules and the estimate of the effort of each module. The total effort will be the sum of the effort of all the modules.

It can be divided in the following phases:

- Planning and requirements;
- System design;
- Detailed design;
- Module code and test;
- Integration and test;
- Cost Constructive model.

As seen, the COCOMO model is very simple to apply in cost estimation, but due to its low level of detail, it may lack in precision and may lead to estimation failure, especially considering the continuous improvement of the processes and complexity of the actual software.

COCOMO II is an evolution of the COCOMO'81 model that focuses on development of the software cost and the schedule and on evaluating the effects of software technology improvements.

In broad terms, the estimation is obtained in similar ways to the COCOMO'81 model, but there is a change in the number and the type of cost drivers and in the equations.

This model can be divided in other three sub-model:

- Application Composition model, used to estimate effort and schedule for projects that use rapid application development tools and could work better starting from Object Points;
- Early Design model, which is used for projects that require an analysis of different type of architectures and concepts;
- Post-Architecture model, which is used for top level design, when the project is complete and detailed.

The cost drivers will be different from COCOMO'81 in terms of type and value for every sub-model, but the equation for the effort remains:

$$PM = a * (KLOC)^b * EAF$$

The value of a is fixed to 2.94, while b is 0.91.

The cost driver will be in part different and, again, they are grouped in the four categories:

- Product Factors;
- Platform Factors;
- Personnel Factors;
- Project Factors.

Following, a table of the cost drivers in terms of type and category for COCOMO II is shown.

Category	Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Product Factors	RELY Required Software Reliability	0.82	0.92	1.00	1.10	1.26	-
	DATA Database Size	-	0.90	1.00	1.08	1.16	-
	RUSE Developed for Reusability	-	0.95	1.00	1.07	1.15	1.24
	DOCU Documentation match to Life-Cycle Needs	0.81	0.91	1.00	1.11	1.23	-
Platform Factors	TIME Execution Time Constraint	-	-	1.00	1.11	1.29	1.63
	STOR Main Storage Constraint	-	-	1.00	1.05	1.17	1.46
	PVOL Platform Volatility	-	0.87	1.00	1.15	1.30	-
Personnel Factors	ACAP Analyst Capability	1.42	1.19	1.00	0.85	0.71	-
	APEX Applications Experience	1.22	1.10	1.00	0.88	0.81	-
	PCAP Programmer Capability	1.34	1.15	1.00	0.88	0.76	-
	PCON Personnel Continuity	1.29	1.12	1.00	0.90	0.81	-
	LTEX Language Experience	1.20	1.09	1.00	0.91	0.84	-
	PLEX Platform Experience	1.19	1.09	1.00	0.93	0.86	0.80
Project Factors	SITE Multisite Development	1.22	1.09	1.00	0.91	0.82	-
	TOOL Use of Software Tools	1.17	1.09	1.00	0.90	0.78	-
	SCED Required Development Schedule	1.43	1.14	1.00	1.00	1.00	-

Table 6. COCOMO II cost drivers ref. [6]

Instead, the schedule equation will be:

$$TDEV = \left[3.67 * (\overline{PM})^{(0.28+0.2(b-1.01))} \right] * \frac{SCED\%}{100}$$

Where:

- TDEV is the schedule time in months;

- \overline{PM} is the effort estimated in person-month excluding the SCED cost driver;
- b is the usual constant depending on the model;
- $SCED\%$ is the schedule compression/expansion percentage in the SCED cost driver rating.

COSYSMO is a model of cost estimation developed by Ricardo Valerdi. It is based on a database of more than 50 projects and gives results in terms of effort by means of the following equation:

$$PM = A * Size^E * \prod_i EM_i$$

Where:

- PM is the effort in person-hours;
- A is the calibration constant that is equal to 38.55;
- E is the economy/diseconomy scale;
- EM_i are the effort multiplier of the cost drivers.
- Size is the equivalent size obtained by the size drivers.

The following table shows the costs drivers for the model.

Category	Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Personnel Factors	TEAM Team Cohesion	1,5	1,22	1	0,81	0,66	-
	PCAP Personnel Team Capability	1,48	1,22	1	0,81	0,66	-
	PEXP Personnel Experience/Continuity	1,46	1,21	1	0,82	0,67	-
	PROC Process Capability	1,46	1,21	1	0,88	0,77	0,68
Environment Factors	SITE Multisite Coordination	1,33	1,15	1	0,9	0,8	0,72
	TOOL Tool Support	1,34	1,16	1	0,85	0,73	-
Operational Factors	INST Number of Diversity of Installations/Platforms	-	-	1	1,23	1,51	1,86
	MIGR Migration Complexity	-	-	1	1,24	1,54	1,92
Understanding Factors	RQMT Requirement Understanding	1,85	1,36	1	0,77	0,6	-
	ARCH Architecture Understanding	1,62	1,27	1	0,81	0,65	-
Complexity Factors	TRSK Technology Risk	0,7	0,84	1	1,32	1,74	-
	LSCV Level of Service Requirements	0,62	0,79	1	1,32	1,74	-
	RECU Number of Recursive Levels in Design	0,8	0,89	1	1,21	1,46	-
	DOCU Documentation Match to Life-Cycle needs	0,82	0,91	1	1,13	1,28	-

Table 7. COSYSMO cost drivers ref.[7]

The size is obtained by means of size drivers. These are expressed in terms of:

- Number of System Requirements obtained by system specification;
- Number of physical or logical Interfaces between system components and external systems;
- Number of Algorithms derived in order to achieve system requirements;
- Number of Operational Scenarios that the system must satisfy.

These size drivers are rated on a scale divided in Easy, Nominal and Difficult. The equivalent size may be obtained using a weighted sum of all the size drivers, where Easy are counted as half, Nominal are counted for entire and Difficult ones are doubled.

REVIC (Revisited Intermediate COCOMO) is another model based on the Intermediate COCOMO model. Starting from COCOMO, the Air Force has developed it using coefficient obtained from empirical data calibrated on a specific environment. The equation used is:

$$Effort = 4.44 * KLOC^{1.2} * \prod_i F_i$$

In which F_i are the cost drivers for this model. In particular, compared with the classical COCOMO model, it has two additional ratings for cost driver levels that are Extra High (to some of the cost drivers that does not have it in COCOMO model) and Extremely High. In addition, the value of some cost drivers has been recalibrated based on the environment.

The schedule time will be:

$$Calendar\ Months = 6.3 * Effort^{0.32}$$

The *F-PROM* (Effort Prediction Objects Model) is a model of cost estimate developed to overcome the lacks of COCOMO models in early phases of the project. It provides a continue upgrading of the model during the life cycle of the project, based on variability of the project, data processed, objects involved and resources.

This model identifies the principal causes of variability in a software project as:

- Size, which influences the estimate up to 43%;
- Reuse, with an influence up to 10%;
- Internal complexity of the modules , up to 8%;
- Degree of interaction between objects, up to 6%.

The equations will be similar to the COCOMO ones, but they will be recalibrated every time new data will be available.

The *Object-Oriented Decomposition* is a technic for software cost estimation that focuses on the resolution of:

- The need of specific knowledge of the system in the early phases of the project;
- The need to define the size of the schedule based on this knowledge;
- The need of reliable estimate in the early phases with the few amount of data available.

The estimate is based on empirical data to obtain an envelope of the possible cost estimations. The model has different level of decomposition that can be viewed as phases of the project.

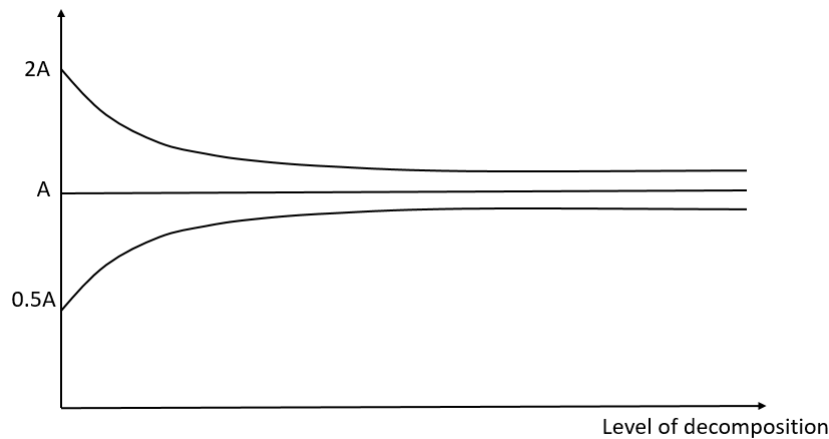


Figure 49. Learning curve for software estimate

The curve in Fig. 49 shows the learning curve for the software estimate of a generic project. The accuracy will increase with the level of decomposition, since functional specifics will be more detailed as the decomposition goes on. B is the rate of exponential decay and it must be defined analyzing the number of completed projects. Obviously, like the other models, it has to be calibrated to the operating environment. This model lacks in available data, because it is a new model.

The *SLIM* model was developed by Putnam in 1974 and it is applied to projects exceeding the 70.000 lines of code. This model considers the effort for software development as a typical Rayleigh curve that represent the manpower as function of time, since the personnel needed will rise smoothly during project development and then drops down in the testing phase. The Rayleigh curves will be different for code development, maintenance, test and validation and management.



Figure 50. Rayleigh curve

The SLIM model provides a calculation of the software project size:

$$Size = CE^{\frac{1}{3}} \left(t^{\frac{4}{3}} \right)$$

Where:

- *Size* is the quantity of functions created in terms of lines of code, function points or object point;
- *C* is the technology factor that considers the used languages, tools, methodologies and standards and it is defined by historical data;
- *E* is the total project effort in person-years;
- *t* is the time from start of detailed design until the operation service.

The Technology factor will be obtained from cost drivers related to:

- process maturity and management practices;
- software engineering practices used;
- level of programming language;
- software environment;
- level of complexity;
- level of skill of the personnel;

Even if the model was developed for software exceeding the 70.000 lines of code, it could be applied to all types and sizes of projects and for all types of metrics.

The effort estimate is derived from the following equation:

$$D = \frac{E}{t^3}$$

Where:

- *D* is a constant named manpower acceleration;
- *E* is the effort in years;
- *t* is the elapsed time to delivery in years.

The manpower accelerations *D* will be 12.3 for new software with many interfaces and interactions with other systems, 15 for standalone systems and 27 for existing systems.

So, the effort can be calculated as:

$$E = \left(\frac{S}{C} \right)^{\frac{9}{7}} D^{\frac{4}{7}}$$

ESTIMACS is a model developed in 1970s by Howard Rubin and it is available as software package. This model focuses on the development phase and it requires Function Points input.

It is divided in five sub-models:

- System development effort estimation, which estimate the development effort in hours;
- Staffing and cost estimation; which uses the effort in hours derived in the previous sub-model and distribute it in the various phases;
- Hardware configuration estimates, which sizes hardware operational resources requirements;
- Risk estimator, which estimates the risk of successfully complete the project;
- Portfolio analyzer, which analyze the whole project.

Price-to-Win estimation is a non-algorithmic technique that focuses on the customer budget rather than system functionalities. This will require that the cost of the project will be fixed in accordance with the customer and the software cost will be restricted by this cost budget.

This allows a cost fixed in accordance with the customer, but this may lead to delay in software delivery.

PRICE-S is a model developed by RCA PRICE Systems that perform cost estimations starting from the size, the complexity and the type of project.

This model allows the calculation of costs and schedules by means of three sub-models:

- Acquisition sub-model, which estimates costs and schedules;
- Sizing sub-model, which estimates the size of the software;
- Life-Cycle Cost sub-model, which estimates the costs for maintenance and support during the early phases.

Unfortunate, this model is not public, but it is a black box model.

Most of the models shown until now require an input related to the size of the project that could be in terms of lines of code, function points or object points.

5.8 Estimation Technique

The *Delphi* technique developed by Rand Corporation allows predictions about events. It could be used as a guide for a group of experts to reach an accordance regarding some issue. Therefore, it may be applied to allow a team of expert to obtain the size of the project in terms of lines of code, function point or object points.

The method is divided in rounds. The first one requires that every expert make individually some assessment related to some issue without the opinion of the others. Then, data are collected, tabulated and distributed to the experts for the second round.

In the second round, experts give another opinion on the same issue, but knowing the opinion of the other experts for the first round. Usually, this leads to narrow the range and to a convergence of the results.

The process will continue until all the parts reach an accordance. The estimate is obtained as an average of weighted estimates:

$$Estimate = \frac{LB_estimate + 4 * ML_estimate + UB_estimate}{6}$$

Where:

- *LB_estimate* is the lower bound of estimate;
- *ML_estimate* is the most likely estimate;
- *UB_estimate* is the upper bound of estimate.

Another method to define the number of lines of code or the number of the function points is the *Function Bang Metric* is a model to define the size of a software based on specification and it uses flow diagrams, data dictionaries, state transition diagrams and entity relationship diagrams.

This model classifies the model in:

- Function-Strong, when the software is related to robotic systems;
- Data-Strong, when the software is related to an information retrieval system;
- Hybrid, when it is a combination of the other two.

To classify a system it is possible to use the ratio RE/FP, in which RE are the number of relationship and FP are the primitive functions. An entity-relationship diagram derives relationship, while primitive functions are calculated from the flow data diagrams.

Function Strong	RE/FP<0.7
Data Strong	RE/FP >1.5
Hybrid	0.7<RE/FP<1.5

Table 8. RE/FP ratio basing on the model type

The size will be calculated as:

$$FBM = \sum_i w_i * CFPI_i$$

With

$$CFPI_i = \frac{(TC_i * \log_2(TC_i))}{2}$$

Where:

- w_i are the weight adjusters;
- TC_i are the number of data tokens of the i^{th} functional primitive in a data flow diagrams.

Therefore, for this model, there are no published empirical data, due to its need for considerable tailoring to certain environments.

6. COST ESTIMATE APPLIED TO THE ECS FOR UAVS

In this chapter, a case of application of cost estimate has been applied by means of the cost estimation software based on a parametric methodology.

In fact, it requires a series of input that will define, by means of a database of equations on which the software is based, an output in terms of costs and effort.

6.1 Scheduling

The first step when starting a new project will be the control of the settings of the Worksheet Sets. Here it is possible to set data related to:

- Unit Cost in terms of cost per hour, which will be related to the costs per hour of the user;
- The costs of the Overtime of the personnel in percentage;
- The costs of Overhead in percentage;
- Time Worked in hours per year;
- Other factors.

By the way, an example of the Worksheet Sets used in the present work is shown in Fig. 51. The Worksheet Sets have been set for every department considered in the development of the system.

	Value	Units	Spread
1 Unit Cost	xx.xx	€/H...	
2 Overtime	0,00%	%	
3 Overhead	0,00%	%	
4 General And Administrative	0,00%	%	
5 Fee Or Profit	0,00%	%	
6 Cost Of Money	0,00%	%	
7 Time Worked	1.824.00	Hours/Year	
8 Availability	0,00	Full Time Equiv...	
9 Distribution	*		
10 Escalation Set	Italy Escalation Table (2016)		
11 Multiplier	1,00		

Figure 51. Worksheet Sets

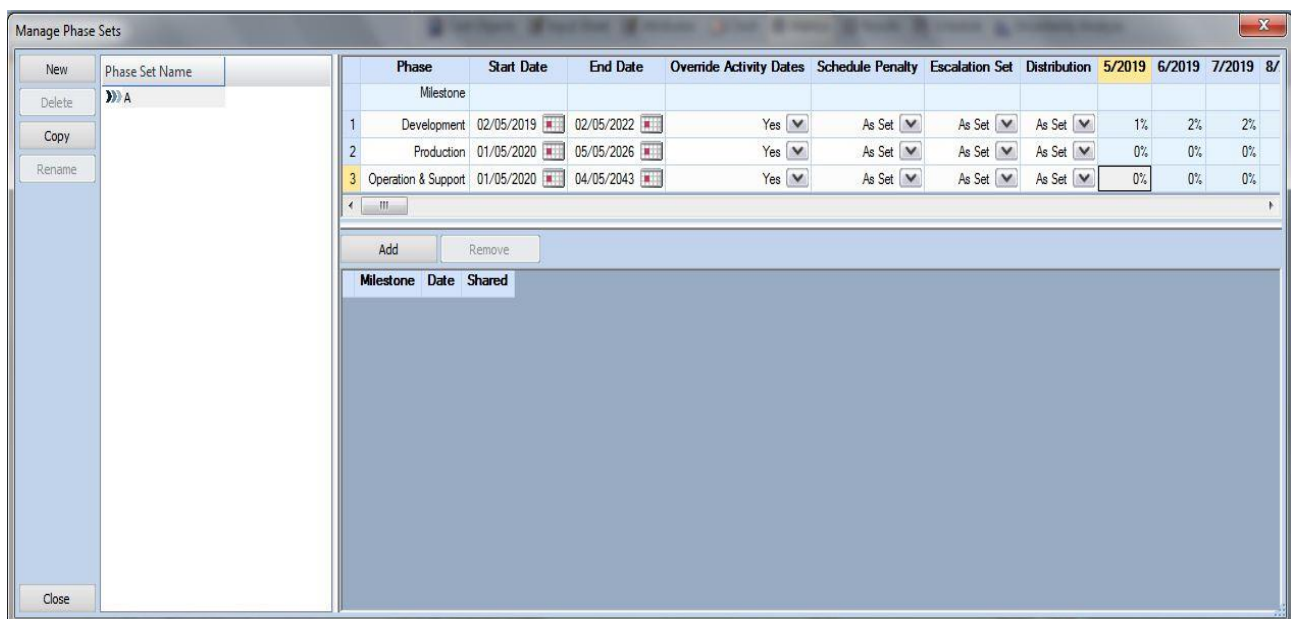
In addition, a schedule of the project need to be defined. The project is divided in the following phases:

- Development;
- Production;
- Operation and Support.

The project starts with the development phase that has been considered of three years.

The production is considered to start after one year from the start of the project and for a duration of six years.

In addition, it has been considered an operative life of twenty-three years, since these are the numbers related to aircraft of the category considered. The Operation and support phase starts with the Production phase, so it starts after one year from the beginning of the project.



Phase	Start Date	End Date	Override Activity Dates	Schedule Penalty	Escalation Set	Distribution	5/2019	6/2019	7/2019	8/
Milestone										
1 Development	02/05/2019	02/05/2022	Yes	As Set	As Set	As Set		1%	2%	2%
2 Production	01/05/2020	05/05/2026	Yes	As Set	As Set	As Set		0%	0%	0%
3 Operation & Support	01/05/2020	04/05/2043	Yes	As Set	As Set	As Set	0%	0%	0%	

Figure 52. Phase Sets

6.2 Product Breakdown Structures

The next step in the cost estimate is the definition of a Product Breakdown Structure (PBS), which will define all the parts of the system and how they are structured.

This exercise has been made for both the architecture analyzed, so two different models have been obtained.

In order to build a Product Breakdown Structure, the software offers a library of objects of different types.

In particular, in this analysis, the following types of objects have been used:

- System, which, in this case, is the higher level of the PBS and represent the system itself;

- Assembly, which corresponds to the integration level of the system or the sub-systems;
- Hardware Component, which implements an hardware part that is developed in-house;
- Hardware COTS, which considers an hardware part purchased as COTS (Commercial Off-The-Shelf);
- Software Component, which is used to describe a software developed in-house.



Figure 53. Parametric Cost Estimation tool library

The two architectures have been modeled following the same logic.

Let us start considering the Vapor Cycle. At the highest level of the Product Breakdown Structure, there is the ECS System object.

Going down on a level in the tree, there are two Assembly objects:

- Aircraft-ECS Integration, which represents the integration of the system with the aircraft;
- ECS integration, which represents all the components that allow the system to perform the functions that it must provide.

The Aircraft-ECS Integration block contains all the components of structural nature related to the aircraft integration, so inside of it there are the following Hardware components:

- Cables, which considers all the wirings related to the system;
- Inlet, which considers the inlets integrated in the aircraft that allow the air flow on the condenser;
- Structure, which consider all the structural components with the only purpose of supporting the system to allocate it in the aircraft.

The ECS Integration block is composed by the components of the system itself and is divided in:

- Control Unit, which represent the dedicated motherboard that controls the functionalities of the system;
- Software, which represent the source code that provides the functionalities of the system;
- Vapor Cycle, represented by an Assembly object, which contains all the component of the Vapor Cycle;
- Back up, another Assembly object used to represent the back-up system used when a failure of the system occurs.

The Vapor Cycle is divided in:

- Compressor, which is one of the fundamental component of this architecture as shown in Chapter 4;
- Condenser, which is another components of the architecture that provides the cooling action of the refrigerant;
- Electric Motor, which is the component that activate the compressor;
- Evaporator, another essential component that provides the heat exchange with avionic bay;
- Expansion Valve, which is another of the components analyzed in Chapter 4;
- Fan, which in the closed loop of the avionic bay provides mass flow to the evaporator, while in the open loop provides mass flow to the condenser during ground operations;
- Filter, which filters the air sent to the avionic bay;
- Integration Material, that considers pipes that link the components;
- Pressure Sensors, which considers, obviously, the pressure sensors;
- Temperature Sensors, which considers the temperature ones.

The back-up system will activate when a failure of the system occurs and it consists simply in opening an inlet to carry external airflow directly in the avionic bay, allowing a recirculation of the air in the avionic bay. In addition, a fan will provide an adequate airflow.

Therefore, the back-up system has been divided in:

- Inlet;
- Fan.

The following figure shows the Product Breakdown structure just described.

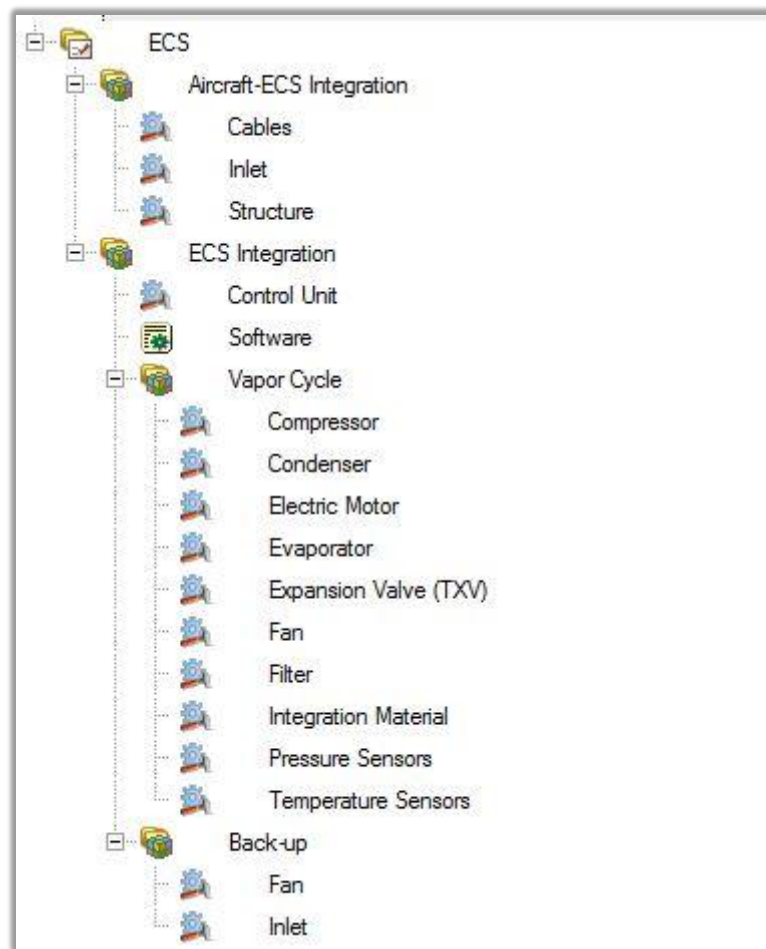


Figure 54. Product Breakdown Structure for the Vapor Cycle

The Air Cycle Product Breakdown Structure is divided in the same way. Again, the highest level is the System object that represent the ECS itself.

Then, in the following level there is the same division in:

- Aircraft-ECS Integration assembly;
- ECS Integration assembly.

This time, the ECS Integration assembly is divided in:

- Control Unit;
- Software;
- Air Cycle;
- Back-up.

The only part of the tree that has changed is the Air Cycle assembly that replace the Vapor Cycle one. The Air Cycle, obviously, changes in terms of components too:

- Bleed Valve, which represent the bleed valve that draws the airflow from the engine;
- By-Pass Valve, which represents the by-pass valves at the precooler and the intercooler;

- Cold Air Unit, which incorporate the three components compressor, shaft and turbine, since, as it will be shown when talking about the inputs, it has been considered as an unique component with a mechanical complexity of an engine, due to the component analogy;
- Differential Pressure Sensors, since this architecture is provided with this type of sensor at the CAU;
- Fan, which are used at the precooler and the intercooler;
- Filter, used upstream of the avionic bay;
- Integration Material, which considers the pipes that link the components of the system;
- Intercooler, that as told in the Chapter 4 is one of the fundamental components for this architecture;
- Non-Return Valve, used to prevent the flow from reversing its direction;
- Precooler, another essential component of the Air Cycle;
- Pressure Sensors;
- Temperature Sensors.

The back-up system is the same as the Vapor Cycle one.

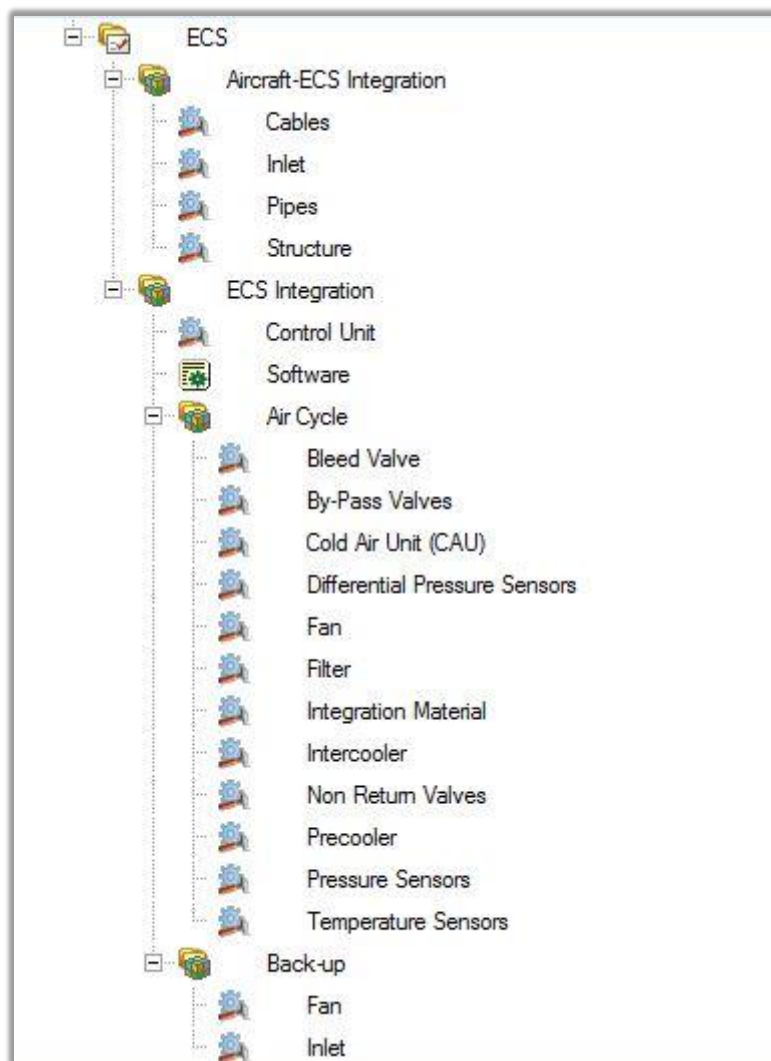


Figure 55. Product Breakdown Structure for the Air Cycle

6.3 Model Inputs for Hardware

Once the Product Breakdown Structure has been populated, each of its elements must be characterized. In fact, each of them has a set of inputs that could be used to refine the estimate.

The System object will require input as:

- Type of equipment;
- Number of Units;
- Number of Prototypes;
- Operating Specification;
- Number of Requirements;
- Complexity of the Requirements;
- Team Capability;
- Others inputs.

The Assembly objects will have similar inputs and will consider the type and the number of interfaces with external parts.

When describing the Hardware Components it is needed a higher level of detail of the component.

1	Start Date	02/05/2019	
2	Quantity Per Next Higher Level	1,00	
3	Additional Units		
4	Number of Additional Production Units	500,00	
5	Number of Additional Prototypes	0,00	
6	Cost Sharing Units		
7	Total Number of Production Units Produced	0	
8	Total Number of Prototypes Produced	0,00	
9	Technical Description		
10	Equipment Type	Propulsion	
11	Operating Specification	1.700	
12	Weight of Structure	3,3600	kg
13	Weight of Electronics	2,6400	kg
14	Volume	9,364	l
15	Manufacturing Complexity for Structure	7,170000	
16	Percent of New Structure	100,00%	%
17	Percent of Design Repeat for Structure	0,00%	%
18	Manufacturing Complexity for Electronics	7,560000	
19	Percent of New Electronics	100,00%	%

Figure 56. Inputs for the Hardware Components

The inputs for Hardware Components will be:

- Number of Units;
- Number of Prototypes;
- Equipment type,
- Operating Specification;
- Weight of Structure;
- Weight of Electronics;
- Volume;
- Manufacturing Complexity for Structure;
- Manufacturing Complexity for Electronics;
- Percent of New Structure;
- Percent of New Electronics;
- Team experience;
- Other factors.

The model will be very sensitive to changes related to the quantity of structure and electronics, since electronics results to be more expensive than a simple structural support element.

In addition, the cost will depend strongly on the percentage of reused structural or electronics and on the technology of production adopted. The manufacturing complexity will depend on the operating specification, on the type of technology, which can be defined by the material and by the shape complexity, and on the type of component that can be described for analogy with the type of components considered in the database.

Manufacturing Complexity for Structure

The Manufacturing Complexity for Structure represents a technology index for the structural portion of the component being described. Manufacturing Complexity is a measure of the component's technology, it's producibility (material machining and assembly tolerances, machining difficulty, surface finish, etc.), and yield. Manufacturing Complexity is a major cost and schedule driver.

The value for Manufacturing Complexity for Structure should be determined either through calibration using historical data from past projects or by taking advantage of one of the many tools in the product designed to help with this critical input. Values can be selected from the PRICE Reference table for Manufacturing Complexity for Structure; by using the Conceptual Complexity generator which uses top level descriptions of the components; or the Detailed Complexity Generator which allows for a detailed description of the many parts that comprise the component.

The Equipment Type drop down input contains typical Manufacturing Complexity values for many equipment types and should be used for guidance in the absence of actual complexity data.

Section Name	Input Field	Units	Description
Operating Specification	Airborne-Commercial		The Operating Specification value indicates the end user's requirements based on the planned operating environment for the hardware component (ground, air, space, sea).
Structural Technology Type	Other Mech		Describes the type of structure being built.
Other Mechanical Assemblies	Engines and Motors		
Structural Technology Detail	Turbo-Jet		

Manufacturing Complexity for Structure: 7,170,000

OK Cancel

Figure 57. Hardware specifications for structure

The Manufacturing Complexity for Electronics will depend on the operating specification, the equipment type (in terms of circuits) and subtype (in terms of signals).

Manufacturing Complexity for Electronics

The Manufacturing Complexity for Electronics represents a technology index for the electronic portion of the component being described. Manufacturing Complexity is a measure of the component's technology, its producibility (component make-up, packaging density, test and reliability requirements, etc.), and yield. Manufacturing Complexity is a major cost and schedule driver.

The value for Manufacturing Complexity for Electronics should be determined either through calibration using historical data from past projects or by taking advantage of one of the many tools in the product designed to help with this critical input. Values can be selected from the PRICE Reference table for Manufacturing Complexity for Electronics (it is recommended that these values only be used when there is no complicated component mixture) or by using the Detailed Complexity Generator which allows for a detailed description of the many parts that comprise the component.

The Equipment Type drop down input contains typical Manufacturing Complexity values for many equipment types and should be used for guidance in the absence of actual complexity data.

☒ Show Descriptions

Section Name	Input Field	Units	Description
Operating Specification	Airborne-Commercial		The Operating Specification value indicates end user's requirements based on the operating environment for the hardware (ground, air, space, sea).
Equipment Type	Small Scale Integrated Circuits		
Equipment Subtype	Digital		

Calculated Manufacturing Complexity for Electronics: 7.560000

OK Cancel

Figure 58. Hardware specifications for electronics

Other inputs are related to the experience of the team, on the facilities involved in the development and on maintenance and support.

By the way, the inputs related to maintenance and operational support have been set to default values, since this inputs require more studies.

6.4 Transformation to a COTS base model

Since here, all the components of the two architectures have been implemented as object of Hardware Component type. This means that the components are developed in-house.

To follow the policy adopted by the most part of the companies, the various components are generally acquired as COTS (Commercial Off-the-Shelf).

In particular, the components related to a structural purpose such as inlets, cable and structure itself are usually developed and produced in-house, while the other elements are purchased as COTS.

For this reason, an iteration between two models has been made for both the configurations. In fact, the first model has been obtained by means of only Hardware Components for the only purpose of obtaining an input cost for the model using COTS Components.

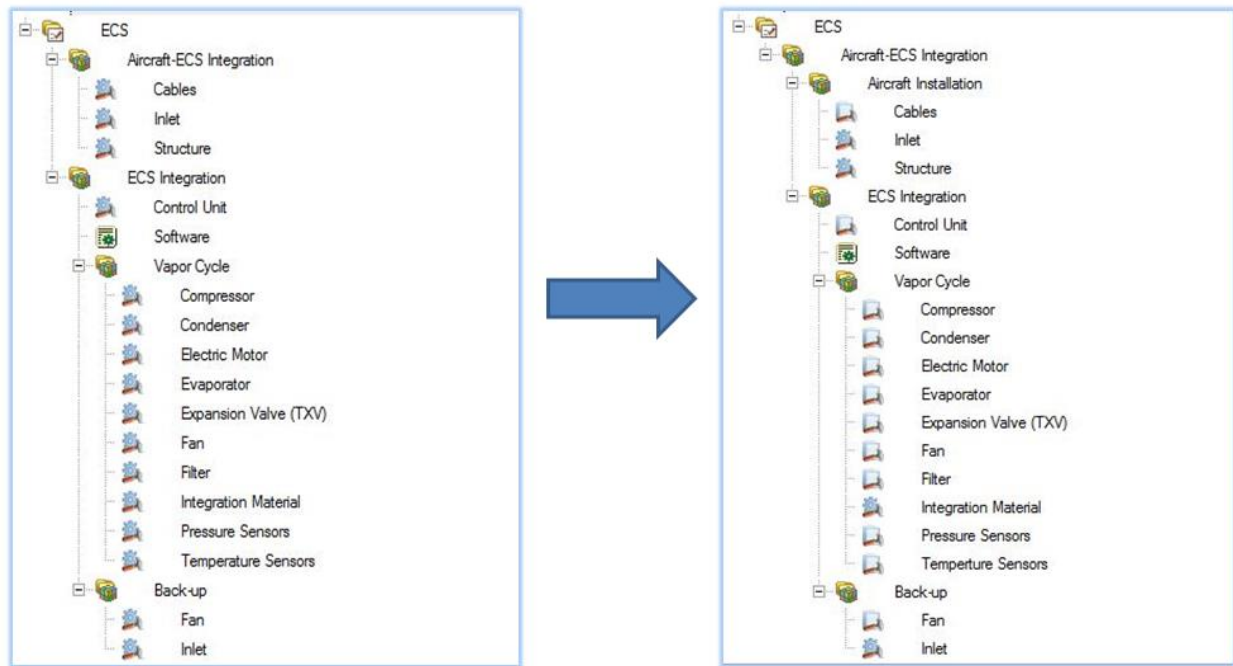


Figure 59. Transition to the COTS components model

In fact, one of the inputs for this type of objects will be the purchasing cost of the single unit and the cost of the prototype.

21	Engineering Complexity	0.300	
22	Electronic Density	0.0000	kg/l
23	Labor Learning Curve	87.00%	%
24	Material Learning Curve	87.00%	%
25	Beginning Production Unit (for Learning Curve)	1	
26	B Factor	0.00%	%
27	Manufacturing Process Index for Labor	1.490	
28	Manufacturing Process Index for Material	1.490	
29	Technology Improvement Control	1.0	
30	Technology Obsolescence Control	0.0	
31	Year of Technology	02/05/2019	
32	External Integration Complexity for Structure	2.00	
33	External Integration Complexity for Electronics	2.00	
34	Hardware Software Integration Factor	50.00	
35	Purchased Hardware Unit Cost	25.300,00	€
36	Prototype Unit Cost	61.624,00	€
37	Scrap Rate Percentage	0.00%	%
38	Prototype Support Adjustment Factor	1.00	
39	Material Index for Development Manufacturing	0.00%	%

Figure 60. COTS components inputs

To obtain valid unit and prototype costs to use as inputs, the model with Hardware Component objects has been modeled considering a mass production of the elements of the Product Breakdown Structure.

In particular, the main input used for this purpose is obviously the number of unit produced. This number has been calibrated in function of the type of component considered.

For example, components like sensors that may be adapted to more projects are considered with a large scale production of about 100.000 units.

Instead, unique components that may be tailored for a certain project are considered with a production of about 500 units.

In fact, the more the units produced, the less expensive will be the production of the lasts units, due to the process improvement and the more knowledge acquired related to the project. This behavior is described by the labor learning curve.

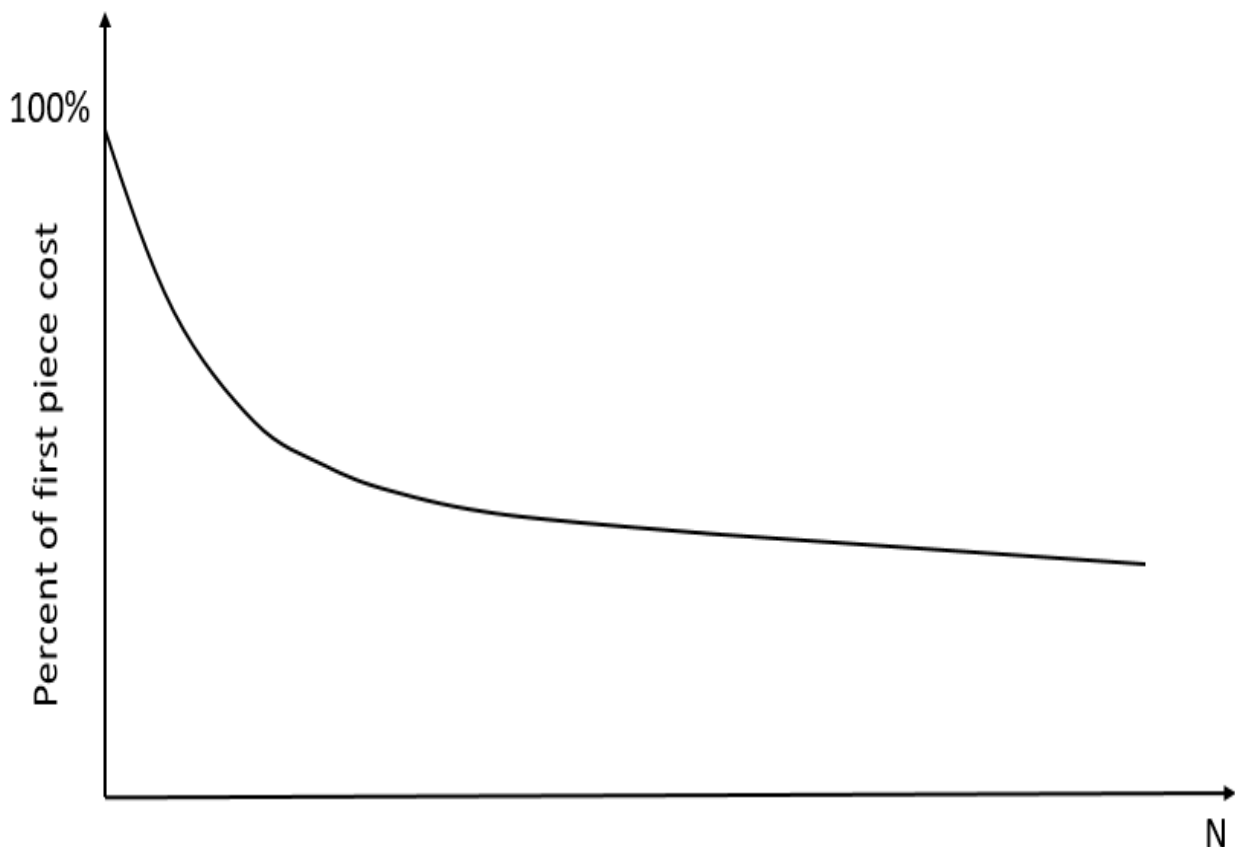


Figure 61. Labor Learning Curve

Then the outputs of the model obtained by means of Hardware Component objects are used as input by increasing it of a 15% of its value to consider the market purchasing cost.

In fact the inputs to define a cost estimation for the software are:

- Application type;
- Functional Complexity, related to the functions the software have to assure;
- Software produced in-house or outsourced or both of them;
- Language adopted;
- Metric adopted;
- Size of the metric adopted;
- Reused Code;
- Adapted Code;
- New Code;
- Repeated Code;
- Team experience;
- Other inputs.

The metric adopted for this analysis are COSMIC Function Points. This type of metric can be obtained starting from the functional analysis and then entered in the tool as inputs.

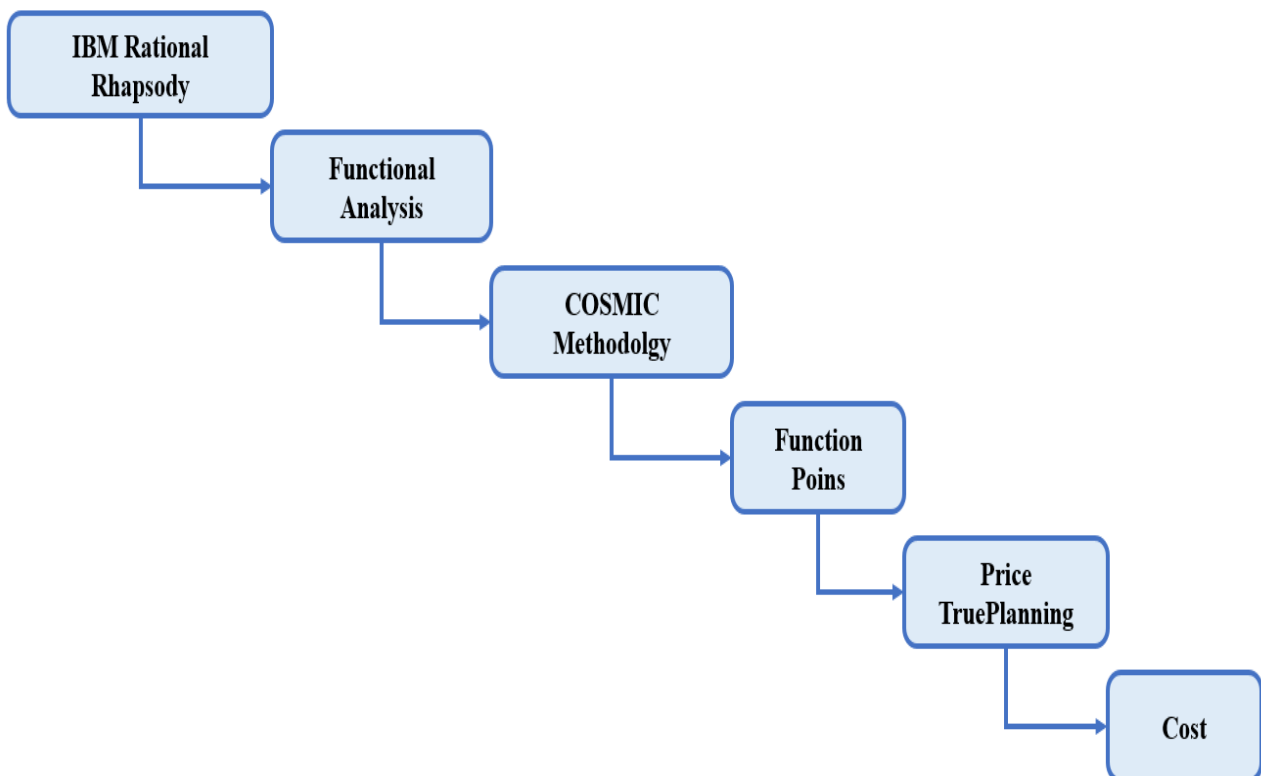


Figure 64. COSMIC methodology process

The COSMIC methodology allows an estimation of the size of the software in terms of Function Points starting from the information exchanges between software and system components, software and actors.

This information have been derived by the White Box Sequence Diagrams counting the number of messages. The messages are divided in four categories:

- *Entry*, exchange of information from an external actor to the controller of the system;
- *Exit*, exchange of information from the controller of the system to an external actor;
- *Write*, exchange of information from the controller of the system to one of the system components;
- *Read*, exchange of information from one of the system components to the controller of the system.



Figure 65. Acquisition of the COSMIC Function Points from the Sequence Diagrams

Therefore, for every Use Case, the Sequence Diagram has been analyzed in order to obtain the Function Points. The Function Points obtained are shown in the relative tables in the data sheet of the software.

Vapor Cycle					
Use Case Name	Entry	Exit	Read	Write	Total
Monitor its health status	0	2	2	1	5
Monitor Status of air filter	1	1	3	1	6
Provide air conditioning	2	3	3	4	12
Provide maintenance	1	1	1	2	5
Shut down	2	1	1	0	4
Start up	1	1	1	2	5
Total	7	9	11	10	37

Table 9. Vapor Cycle COSMIC Function Points

Air Cycle					
Use Case Name	Entry	Exit	Read	Write	Total
Monitor its health status	0	2	3	1	6
Monitor Status of air filter	1	1	3	1	6
Provide air conditioning	2	3	15	4	24
Provide maintenance	1	1	2	2	6
Shut down	2	1	1	0	4
Start up	1	1	1	2	5
Total	7	9	25	10	51

Table 10. Ai Cycle COSMIC Function Points

6.6 Results

The following tables show the results related to the various costs and efforts of the system. The values have been approximated and the cost related to Operation and Support are not reported, due to the exclusion of them from the study. The tool allows a cost analysis for every level of the Product Breakdown Structure, but only the costs for the higher level represented by the ECS System Component are reported.

ECS (Vapor Cycle)	Value	Units
Total Cost	5.800.000	€
Total Labor Hours	35.000	Hours
Development Summary		
Prototype Quantity	1	Units
Development Cost	1.675.000	€
Development Labor Hours	25.000	Hours
Development Duration	36	Months
Production Summary		
Production Quantities	60	Units
Unit Production Cost	59.000	€
Amortized Unit Production Cost	60.000	€
Production Cost	4.100.000	€
Production Labor Hours	9.000	Hours
Production Duration	72	Months
First Piece Cost	76.000	€
First Piece Cost Material	56.000	€
First Piece Cost Labor	20.000	€
Nth Unit Cost	54.000	€
Nth Unit Cost Material	46.000	€
Nth Unit Cost Labor	7.800	€
Project and Productivity Details		
Total Cost for Hardware	4.700.000	€
Total Labor for Hardware	19.000	Hours
Total Cost for Software	670.000	€
Total Labor for Software	11.000	Hours

Table 11. Vapor Cycle results obtained in the Parametric Cost Estimation tool

ECS (Air Cycle)	Value	Units
Total Cost	8.500.000	€
Total Labor Hours	52.000	Hours
Development Summary		
Prototype Quantity	1	Units
Development Cost	2.500.000	€
Development Labor Hours	39.000	Hours
Development Duration	36	Months
Production Summary		
Production Quantities	60	Units
Unit Production Cost	84.000	€
Amortized Unit Production Cost	85.000	€
Production Cost	5.950.000	€
Production Labor Hours	12.000	Hours
Production Duration	72	Months
First Piece Cost	106.000	€
First Piece Cost Material	81.000	€
First Piece Cost Labor	25.000	€
Nth Unit Cost	78.000	€
Nth Unit Cost Material	68.000	€
Nth Unit Cost Labor	9.800	€
Project and Productivity Details		
Total Cost for Hardware	6.500.000	€
Total Labor for Hardware	20.500	Hours
Total Cost for Software	880.000	€
Total Labor for Software	15.000	Hours

Table 12. Air Cycle results obtained in the Parametric Cost Estimation tool

As expected, the Air Cycle presents higher costs and effort, due to the higher number of components and complexity. This difference is of about the 45% in terms of total costs.

Obviously, the costs of the software are also higher for the Air Cycle of about a 50%.

6.7 Other methods applications

In this section, an application of some of the other methodologies previously exposed is presented for the two architecture, in order to validate the estimate obtained by means of the tool or at least to obtain another reference estimate.

For these estimations, it is necessary to obtain an input in terms of software size for the two architectures.

By means of the Delphi method and experts opinions, a value of the Source Lines of Code has been fixed.

In particular, two experts of Leonardo Aircraft Division have provided an individual estimation for the two architectures. Then the estimations have been provided to the other expert and their opinion

has been calibrated. In the end, the Delphi equation has been applied and the estimate of the Source Lines of Code has been obtained for the software cost estimate models.

Estimate	Vapor Cycle	Air Cycle
Expert 1 Estimation	6.000	10.000
Expert 2 Estimation	10.000	12.000
Parametric Cost Estimation tool	8.800	12.200
Delphi Estimation	8.500	11.700

Table 13. SLOC estimations

These lines of code have also been used as input for another Parametric Cost Estimation tool model replacing the COSMIC Function Points, in order to obtain another result.

This model gave as result of 660.000 € for the Vapor Cycle and of 890.000 € for the Air Cycle, very similar to the previous ones.

At this point, it is possible to proceed with the analysis.

The first model that has been analyzed is COCOMO, for which it is necessary to obtain the inputs for the estimate:

- Source Lines of Code;
- EAF;
- Type of model.

For both the architecture has been fixed a type of model Embedded, from which the values for the coefficients has been fixed as shown in the table below.

Model	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.2	2.5	0.32

Table 14. Values of a and b depending on the model type for the ECS

To calculate the Effort Adjustment Factor, it is necessary to fix the costs drivers for the two architectures and then multiply them to obtain it as:

$$EAF = \prod Cost\ Driver$$

In order to fix costs drivers, it is necessary to define how the ratings are assigned. For this purpose, a description of the ratings related to each cost drivers is shown below.

Required Reliability levels are:

- **Very Low:** A software failure simply causes an inconvenience incumbent on the developers to fix the fault;
- **Low:** A software failure has a low level effect, easily-recoverable loss to users;

- **Nominal:** A software failure causes a moderate loss to users, but a situation for which one can recover without extreme penalty;
- **High:** A software failure can lead to a major financial loss or a massive human inconvenience;
- **Very High:** A software failure can cause the loss of human life;
- **Very High:** No rating - defaults to Very High.

This cost driver has been fixed to high both architectures, since a system failure will compromise the mission.

Database Size is related to the database size to be developed, where size refers to the amount of data to be assembled and stored in non-main storage. Its levels are:

- **Very Low:** No rating - defaults to Low;
- **Low:** $D/P < 10$;
- **Nominal:** $10 \leq D/P \leq 100$;
- **High:** $100 \leq D/P \leq 1000$;
- **Very High:** $D/P > 1000$;
- **Extra High:** No rating - defaults to Very High.

Where D/P is the (Database size in bytes or characters) / (Program size in SLOC).

The database size as been fixed Nominal for the Vapor Cycle and High for Air Cycle, since the second one requires a greater database due to the control dynamics of the valves.

The *Product complexity* depends on the architecture of the system. This cost driver has been considered nominal for Vapor Cycle and high for Air Cycle

Execution Time Constant represents the degree of the execution time constraint imposed upon a software project. The ratings are expressed in terms of available execution time expected to be used and they are:

- **Very Low:** No rating - defaults to Nominal;
- **Low:** No rating - defaults to Nominal;
- **Nominal:** $\leq 50\%$ use of available execution time;
- **High:** 70% use of available execution time;
- **Very High:** 85% use of available execution time;
- **Extra High:** 95% use of available execution time.

This driver has been fixed to Extra High, since the system must operate in real time.

Main Constrains is the percentage of main storage expected to be used by the software and any subsystems that consumes main storage resources. Main storage is related to direct random access storage such as disks, tapes, or optical drives. The ratings are:

- **Very Low:** No rating - defaults to Nominal;
- **Low:** No rating - defaults to Nominal;
- **Nominal:** $\leq 50\%$ use of available storage;
- **High:** 70% use of available storage;
- **Very High:** 85% use of available storage;

- **Extra High:** 95% use of available storage.

This driver has been fixed to high due to the high number of sensors in the system.

Virtual Machine Volatility is the level of volatility of the virtual machine underlying the developed software. The virtual machine is the complex of hardware and software the product will call upon to achieve its tasks. The ratings are defined in terms of the relative frequency of major and minor changes as:

- **Major change:** significantly affects roughly 10% of routines under development;
- **Minor change:** significantly affects roughly 1% of routines under development;
- **Very Low:** No rating - defaults to Low;
- **Low:** Major change every 12 months;
- **Nominal:** Major change every 6 months; Minor: 2 weeks;
- **High:** Major: 2 months; Minor: 1 week;
- **Very High:** Major: 2 weeks; Minor: 2 days;
- **Extra High:** No rating - defaults to Very High.

This driver has been fixed to high since the system call upon various actors during its operations.

Computer Turnaround Time represents the level of computer response time experienced by the project team that developed the software product. The response time is the average time from when the developer submits a task to be run until the results are back and available to the user. The ratings are:

- **Very Low:** No rating - defaults to Low;
- **Low:** Interactive;
- **Nominal:** Average turnaround time < 4 hours;
- **High:** 4 - 12 hours;
- **Very High:** > 12 hours;
- **Extra High:** No rating - defaults to Very High.

The system must communicate in an interactive way with the operator, so this cost driver has been fixed to be low.

Analyst Capability is related to the fact that the analysts are involved in the development and validation of system requirements and preliminary design specifications. They consult on detailed design and code activities. They participate in the integration and test phases. The ratings for analyst capability are expressed in terms of percentiles with respect to the overall population of software analysts. The major attributes to be considered are skill, efficiency, thoroughness, teamwork and the ability to communicate. This evaluation should not include experience (that is considered in other factors) and should be based on the ability of the analysts as a team rather than individuals.

- **Very Low:** 15th percentile;
- **Low:** 35th percentile;
- **Nominal:** 55th percentile;
- **High:** 75th percentile;

- **Very High:** 90th percentile;
- **Extra High:** No rating - defaults to Very High.

The analysts have been considered experts, so this driver is very high.

Application Capability is the level of equivalent applications experience of the project team that develops the software. Its levels are:

- **Very Low:** ≤ 4 month experience;
- **Low:** 1 year of experience;
- **Nominal:** 3 years of experience;
- **High:** 6 years of experience;
- **Very High:** 12 years of experience;
- **Extra High:** No rating - defaults to Very High.

The team has been considered expert, so this driver is very high.

Programmer Capability is the capability of the programmers who work on the software. The ratings are expressed in terms of percentiles with respect to the overall population of programmers. The major factors which should be considered in the rating are ability, efficiency, thoroughness, and the ability to communicate and cooperate. The evaluation should not consider the level of experience of the programmers (it is considered by other factors) and it should be based on the ability of the programmers as a team rather than as individuals. The ratings are:

- **Very Low:** 15th percentile
- **Low:** 35th percentile
- **Nominal:** 55th percentile
- **High:** 75th percentile
- **Very High:** 90th percentile
- **Extra High:** No rating - defaults to Very High.

The programmers have been considered to be expert, so this driver is very high.

Virtual Machine Experience is the experience of the project team with the complex of hardware and software that the software itself calls upon to accomplish its tasks, e.g. computer, operating system, or database management system (the programming language is not considered as part of the virtual machine). Its ratings are:

- **Very Low:** ≤ 1 month experience;
- **Low:** 4 months of experience;
- **Nominal:** 1 year of experience;
- **High:** 3 years of experience;
- **Very High:** No rating - defaults to High;
- **Extra High:** No rating - defaults to High.

The team has been considered expert with the system, so this driver is high.

Programming Language Experience represents the level of programming language experience of the project team that develops the software. The ratings are defined in terms of the project team's equivalent duration of experience with the programming language to be used. The ratings are:

- **Very Low:** ≤ 1 month experience;
- **Low:** 4 months of experience;
- **Nominal:** 1 year of experience;
- **High:** 3 years of experience;
- **Very High:** No rating - defaults to High;
- **Extra High:** No rating - defaults to High.

The team has been considered expert with the language, so this driver is high.

Use of Software Tools represents the degree to which software tools are used in developing the software product. The ratings are classified as:

- **Very Low:** Basic tools, e.g. assembler, linker, monitor, debug aids;
- **Low:** Beginning use of more productive tools, e.g. High-Order Language compiler, macro assembler, source editor, basic library aids, database aids;
- **Nominal:** Some use tools such as real-time operating systems, database management system, interactive debuggers, interactive source editor;
- **High:** General use of tools such as virtual operating systems, database design aids, program design language, performance measurement and analysis aids, and program flow and test analyzer;
- **Very High:** General user of advanced tools such as full programming support library with configuration management aids, integrated documentation system, project control system, extended design tools, automated verification system;
- **Extra High:** No rating - defaults to Very High.

It has been considered that the code writing is supported by few tools, so this cost driver has been considered low

Schedule Constraints represents the level of constraint imposed on the project team that develops a software. Ratings are defined in terms of the percentage of schedule stretch-out or acceleration with respect to a nominal schedule for a project requiring a given amount of effort. The levels of rating are:

- **Very Low:** 75% of nominal;
- **Low:** 85% of nominal;
- **Nominal:** 100%;
- **High:** 130% of nominal;
- **Very High:** 160% of nominal;
- **Extra High:** No rating - defaults to Very High.

The constraints imposed have been fixed to a low level, since the project duration considered before operating phase is long enough for the code needed.

For the two architectures, the cost drivers chosen are almost similar. In fact, the ones related to Computer Attributes, Personnel Attributes and Project Attributes are fixed, since they do not depend on the architecture of the system.

The tables related to the cost drivers fixed for the two architectures are shown following.

Vapor Cycle							
Category	Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes	RELY Required Software Reliability	0,75	0,88	1	1,15	1,4	-
	DATA Database Size	-	0,94	1	1,08	1,16	-
	CPLX Product Complexity	0,7	0,85	1	1,15	1,3	1,65
Computer Attributes	TIME Execution Time Constraint	-	-	1	1,11	1,3	1,66
	STOR Main Storage Constraint	-	-	1	1,06	1,21	1,56
	VIRT Virtual Machine Volatility	-	0,87	1	1,15	1,3	-
	TURN Computer Turnaround Time	-	0,87	1	1,07	1,15	-
Personnel Attributes	ACAP Analyst Capability	1,46	1,19	1	0,96	0,71	-
	AEXP Applications Experience	1,29	1,13	1	0,91	0,82	-
	PCAP Programmer Capability	1,42	1,17	1	0,86	0,7	-
	VEXP Virtual Machine Experience	1,21	1,1	1	0,9	-	-
	LEXP Language Experience	1,14	1,07	1	0,95	-	-
Project Attributes	MODP Modern Programming Practices	1,24	1,1	1	0,91	0,82	-
	TOOL Use of Software Tools	1,24	1,1	1	0,91	0,83	-
	SCED Required Development Schedule	1,23	1,08	1	1,04	1,1	-

Table 15. COCOMO cost drivers for Vapor Cycle

Air Cycle							
Category	Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes	RELY Required Software Reliability	0,75	0,88	1	1,15	1,4	-
	DATA Database Size	-	0,94	1	1,08	1,16	-
	CPLX Product Complexity	0,7	0,85	1	1,15	1,3	1,65
Computer Attributes	TIME Execution Time Constraint	-	-	1	1,11	1,3	1,66
	STOR Main Storage Constraint	-	-	1	1,06	1,21	1,56
	VIRT Virtual Machine Volatility	-	0,87	1	1,15	1,3	-
	TURN Computer Turnaround Time	-	0,87	1	1,07	1,15	-
Personnel Attributes	ACAP Analyst Capability	1,46	1,19	1	0,96	0,71	-
	AEXP Applications Experience	1,29	1,13	1	0,91	0,82	-
	PCAP Programmer Capability	1,42	1,17	1	0,86	0,7	-
	VEXP Virtual Machine Experience	1,21	1,1	1	0,9	-	-
	LEXP Language Experience	1,14	1,07	1	0,95	-	-
Project Attributes	MODP Modern Programming Practices	1,24	1,1	1	0,91	0,82	-
	TOOL Use of Software Tools	1,24	1,1	1	0,91	0,83	-
	SCED Required Development Schedule	1,23	1,08	1	1,04	1,1	-

Table 16. COCOMO cost drivers for Air Cycle

For the two architectures, the cost drivers chosen are almost similar. In fact, the ones related to Computer Attributes, Personnel Attributes and Project Attributes are fixed, since they do not depend on the architecture of the system.

The only different cost drivers that differs are those related to the Product Attributes, since these will be related to the architecture complexity and components. Obviously, the Air Cycle is the one with higher ratings related to these cost drivers.

By means of the COCOMO methodology, the following results have been obtained.

	Vapor Cycle	Air Cycle
EAF	0.80	1.04
Effort [person-months]	38	69
Schedule Duration [months]	8	10
Software Cost [€]	530.000	1.000.000

Table 17. COCOMO results

The second model adopted is COCOMO II. As told in Chapter 5, this is an evolution of the previous method, based on the same equation for the effort and a different equation for schedule.

Vapor Cycle							
Category	Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Product Factors	RELY Required Software Reliability	0,82	0,92	1	1,1	1,26	-
	DATA Database Size	-	0,9	1	1,08	1,16	-
	RUSE Developed for Reusability	-	0,95	1	1,07	1,15	1,24
	DOCU Documentation match to Life-Cycle Needs	0,81	0,91	1	1,11	1,23	-
Platform Factors	TIME Execution Time Constraint	-	-	1	1,11	1,29	1,63
	STOR Main Storage Constraint	-	-	1	1,05	1,17	1,46
	PVOL Platform Volatility	-	0,87	1	1,15	1,3	-
Personnel Factors	ACAP Analyst Capability	1,42	1,19	1	0,85	0,71	-
	APEX Applications Experience	1,22	1,1	1	0,88	0,81	-
	PCAP Programmer Capability	1,34	1,15	1	0,88	0,76	-
	PLEX Platform Experience	1,19	1,09	1	0,93	0,86	0,8
	PCON Personnel Continuity	1,29	1,12	1	0,9	0,81	-
	LTEX Language Experience	1,2	1,09	1	0,91	0,84	-
Project Factors	SITE Multisite Development	1,22	1,09	1	0,91	0,82	-
	TOOL Use of Software Tools	1,17	1,09	1	0,9	0,78	-
	SCED Required Development Schedule	1,43	1,14	1	1	1	-

Table 18. COCOMO II cost drivers for Vapor Cycle

Air Cycle							
Category	Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Product Factors	RELY Required Software Reliability	0,82	0,92	1	1,1	1,26	-
	DATA Database Size	-	0,9	1	1,08	1,16	-
	RUSE Developed for Reusability	-	0,95	1	1,07	1,15	1,24
	DOCU Documentation match to Life-Cycle Needs	0,81	0,91	1	1,11	1,23	-
Platform Factors	TIME Execution Time Constraint	-	-	1	1,11	1,29	1,63
	STOR Main Storage Constraint	-	-	1	1,05	1,17	1,46
	PVOL Platform Volatility	-	0,87	1	1,15	1,3	-
Personnel Factors	ACAP Analyst Capability	1,42	1,19	1	0,85	0,71	-
	APEX Applications Experience	1,22	1,1	1	0,88	0,81	-
	PCAP Programmer Capability	1,34	1,15	1	0,88	0,76	-
	PLEX Platform Experience	1,19	1,09	1	0,93	0,86	0,8
	PCON Personnel Continuity	1,29	1,12	1	0,9	0,81	-
	LTEX Language Experience	1,2	1,09	1	0,91	0,84	-
Project Factors	SITE Multisite Development	1,22	1,09	1	0,91	0,82	-
	TOOL Use of Software Tools	1,17	1,09	1	0,9	0,78	-
	SCED Required Development Schedule	1,43	1,14	1	1	1	-

Table 19. COCOMO II cost drivers for Air Cycle

The two previous tables show the cost drivers used for the present valuation.

Most of cost drivers type is equivalent to the COCOMO'81 ones, but with different values due to the data upgrade. By the way, these cost drivers have been fixed to the same ratings of the previous estimation.

Instead, other cost drivers are new or replace others.

The different cost drivers are:

- *RUSE*, fixed to extra high, since the software main functionalities depend on the architectures that are the most used;
- *DOCU*, fixed to very high, since the model will be documented in each phase;
- *PVOL*, fixed to high, since it has been considered that some changes may be applied during project duration;
- *PLEX*, fixed to high, since it has been considered that the team is familiar with the two architectures;
- *PCON*, fixed to high, since some of the personnel may change during the project duration;
- *SITE*, fixed to low, since it has been considered only one site of development.

The results for this model are reported below.

	Vapor Cycle	Air Cycle
EAF	1.71	1.85
Effort [person-months]	35	50
Schedule Duration [months]	5	5
Software Cost [€]	500.000	720.000

Table 20. COCOMO II results

Another result has been obtained by means of the application of the COSYSMO methodology.

This methodology may be applied for both hardware and software, but, for this analysis, the only cost related to software has been considered.

The effort can be obtained as:

$$Effort = A * Size^E * \prod_i EM_i$$

This time, the effort has been calculated in terms of person hour instead of person month.

In addition, this time the size is not provided in terms of source lines of code, but by means of the size drivers used to estimate the equivalent size.

For this model the cost drivers used to calculate the Effort Adjustment Factor are different again. In fact, they are more focused on the requirements and the architecture of the system.

The different cost drivers for COSYSMO are:

- *TEAM* considered nominal as compromise;
- *PCAP* has been fixed to high, considering a capable team;
- *PEXP* has been fixed to high, since the program will endure for many years;
- *PROC* has been fixed to nominal, since the process fixed in the early phases could be changed in later phases;
- *INST* has been fixed to high, since the software may be adapted for more platforms;
- *MIGR* has been fixed to high, since the complexity may change due to requirement refining;
- *RQMT* has been fixed to nominal, since requirement may change as the process gains maturity;
- *ARCH* has been fixed to nominal, since in early phases of the project more components may be identified;
- *TRSK* has been fixed to nominal, since the two architectures are not experimental;
- *LSCV* has been fixed to high, since the requirements given by the customer have to be satisfied, but some compromises may be needed;
- *RECU* fixed to high, since some of the function and components of the system are repeated.

Again, the cost drivers used for these models are reported in tables.

Vapor Cycle							
Category	Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Personnel Factors	TEAM Team Cohesion	1,5	1,22	1	0,81	0,66	-
	PCAP Personnel Team Capability	1,48	1,22	1	0,81	0,66	-
	PEXP Personnel Experience/Continuity	1,46	1,21	1	0,82	0,67	-
	PROC Process Capability	1,46	1,21	1	0,88	0,77	0,68
Environment Factors	SITE Multisite Coordination	1,33	1,15	1	0,9	0,8	0,72
	TOOL Tool Support	1,34	1,16	1	0,85	0,73	-
Operational Factors	INST Number of Diversity of Installations/Platforms	-	-	1	1,23	1,51	1,86
	MIGR Migration Complexity	-	-	1	1,24	1,54	1,92
Understanding Factors	RQMT Requirement Understanding	1,85	1,36	1	0,77	0,6	-
	ARCH Architecture Understanding	1,62	1,27	1	0,81	0,65	-
Complexity Factors	TRSK Technology Risk	0,7	0,84	1	1,32	1,74	-
	LSCV Level of Service Requirements	0,62	0,79	1	1,32	1,74	-
	RECU Number of Recursive Levels in Design	0,8	0,89	1	1,21	1,46	-
	DOCU Documentation Match to Life-Cycle needs	0,82	0,91	1	1,13	1,28	-

Table 21. COSYSMO cost drivers for Vapor Cycle

Air Cycle							
Category	Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Personnel Factors	TEAM Team Cohesion	1,5	1,22	1	0,81	0,66	-
	PCAP Personnel Team Capability	1,48	1,22	1	0,81	0,66	-
	PEXP Personnel Experience/Continuity	1,46	1,21	1	0,82	0,67	-
	PROC Process Capability	1,46	1,21	1	0,88	0,77	0,68
Environment Factors	SITE Multisite Coordination	1,33	1,15	1	0,9	0,8	0,72
	TOOL Tool Support	1,34	1,16	1	0,85	0,73	-
Operational Factors	INST Number of Diversity of Installations/Platforms	-	-	1	1,23	1,51	1,86
	MIGR Migration Complexity	-	-	1	1,24	1,54	1,92
Understanding Factors	RQMT Requirement Understanding	1,85	1,36	1	0,77	0,6	-
	ARCH Architecture Understanding	1,62	1,27	1	0,81	0,65	-
Complexity Factors	TRSK Technology Risk	0,7	0,84	1	1,32	1,74	-
	LSCV Level of Service Requirements	0,62	0,79	1	1,32	1,74	-
	RECU Number of Recursive Levels in Design	0,8	0,89	1	1,21	1,46	-
	DOCU Documentation Match to Life-Cycle needs	0,82	0,91	1	1,13	1,28	-

Table 22. COSYSMO cost drivers for Air Cycle

The next step is to define the size drivers for the two architectures. They are defined in terms of requirements, interfaces, algorithms and operation scenarios.

For the two architecture, the requirements analyzed during Functional Analysis have been considered and classified.

The interfaces have been calculated considering the various component of the system and external systems that communicate with the software. In particular, interfaces related to control dynamics (e.g. PID for the valves in Air Cycle and inlet area control) have been considered difficult. The ones related to data acquisition (e.g. sensors) have been considered easy. The ones related to components operation with dynamic components that do not present a complex control (e.g. fan, CAU, compressor) have been considered nominal.

Algorithms take in account all the measurement and the controls required by the two architectures. The ones related to valve and inlet controls have been considered difficult. The ones related to sensors measurements and comparisons have been considered easy.

The operational scenarios have been considered the same ones for the two architectures: one nominal and two difficult related to the cold case and the hot case.

The tables relative to the size drivers are shown below.

Vapor Cycle				
Category	Easy	Nominal	Difficult	Equivalent
Requirements	6	10	3	19
Interfaces	32	5	2	25
Algorithms	2	0	6	13
Operational Scenarios	0	1	2	5
Total	40	16	13	62

Table 23. COSYSMO size drivers for Vapor Cycle

Air Cycle				
Category	Easy	Nominal	Difficult	Equivalent
Requirements	6	10	3	19
Interfaces	32	11	5	37
Algorithms	2	0	11	23
Operational Scenarios	0	1	2	5
Total	40	22	21	84

Table 24. COSYSMO size drivers for Air Cycle

The table below reports the results for this analysis.

	Vapor Cycle	Air Cycle
EAF	3.19	3.19
Effort [person-hours]	9.800	13.500
Software Cost [€]	570.000	790.000

Table 25. COSYSMO results

At this point it is possible to compare the results obtained.

Methodology	Vapor Cycle Software Cost [€]	Air Cycle Software Cost [€]
Parametric Cost Estimation tool COSMIC Function Points	670.000	880.000
Parametric Cost Estimation tool Source Lines of Code	660.000	890.000
COCOMO	530.000	1.000.000
COCOMO II	500.000	720.000
COSYSMO	570.000	790.000

Table 26. Results comparison

The Air Cycle architecture obviously presents the higher cost for all the methodologies applied, with an average of about 840.000 €.

For the Vapor Cycle, the results give an average cost of about 590.000 €.

In addition, it is possible to notice that the two Parametric Cost Estimation tool estimate are very similar, due to the same inputs set, since they differ only for the metrics adopted.

In addition, the COCOMO model presents the greater gap between the two architecture and this may be due to its weakness in calibration, which seems to be resolved by the refined COCOMO II.

7. CONCLUSIONS

It has been shown as the tools and the methodologies of the Systems Engineering allow obtaining a preliminary cost estimation of the project.

In fact, Systems Engineering approach allows understanding various aspects of the project. By understanding the project, it is possible to obtain information used for the cost estimation.

The Functional Analysis has been used to identify the functionality of the systems and to define some of the components required to accomplish these functions. In addition, as shown, this analysis allowed obtaining a first estimate of the size of the software associated to the system. In fact, the COSMIC Function Points obtained from the Sequence Diagrams led to a correspondent number of Lines of Code similar to the one provided by the experts.

The Performance Analysis allowed a study of the two architectures in order to define the number and type of component required by the system. This analysis has been essential to obtain a Product Breakdown Structure of the system and to define the size of the components.

All the results obtained by these analysis are the inputs for the cost estimates presented in this work.

In addition, software cost estimation has been calibrated by means of Delphi methodology. This method led to an estimate of the Source Lines of Code has been obtained starting from experts opinions.

In fact, different methodologies of cost estimation led to similar results returning a first calibration of the results obtained. The differences in the results come from the low level of detail of this phase of the project.

In fact, as the project gains maturity, the cost estimates will be uploaded and refined with the new information and details obtained. In addition, if the project reach a sufficient level of maturity, it will be possible to apply other methodologies and approaches tailored for more detailed project.

In addition, starting from this thesis work and on a thesis work on Safety Assessment, it will be possible to perform other studies relative to the operative costs that will affect maintainance and support costs (e.g. Mean Time Between Failure, Maintainance-Man Hours, Failure Rate parameters).

Another interesting aspect related to the operative costs could also be the costs estimate related to the integration of the ECS system (e.g. fuel consumption) in a whole product.

Bibliography

- *Introduction to UAV Systems* - Paul Gerin Fahlstrom, Thomas James Gleason
- *Unmanned aircraft Systems - UAVs Design, Development and Deployment*- Reg Austin
- *NASA Cost Estimating Handbook* - National Aeronautics and Space Administration ref.[2]
[3]
- *Parametric Estimating Handbook* – International Society of Parametric Analysts
- *Appendix of Parametric Handbook* – International Society of Parametric Analysts
- *Cost Estimating Guide* – U.S Department of Energy
- *Applicazione del Design to Cost – Caso d'uso Leonardo – Leonardo Divisione Velivoli* - Luca Boggero
- *Software Cost Estimation* – Samuel Lee, Lance Titchkosky, Seth Bowen ref.[4][5]
- *Review of Various Software Cost Estimation Techniques* – Shivangi Shekhar, Umesh Kumar
- *Cost Models for Future Software Life Cycle Processes* – Barry Boehm, Bradford Clark, Ellis Horowitz, Chris Westland
- *Cost Estimation Methods For Software Engineering* – Andre Ladeira
- *A Review of Software Cost Estimation: Tools, Methods, and Techniques* – Muhammad Tosam Bingamawa, Massila Kamalrudin
- *Methods and Models For Estimating a Software Project: An Analytical Approach* – James Themis
- *Applying COCOMO II – A case study* – Darko Milicic ref.[6]
- *Softsar Systems: COCOMO and COSYSMO Estimation Tools* ref.[7]
- *Project Management dei Progetti Software* ref.[8][9]
- *COSYSMO: A System Engineering Cost Model* – Ricardo Valerdi, Barry W. Boehm

Sitography

- https://en.m.wikipedia.org/wiki/Systems_engineering
- https://sites.google.com/site/systemengineeringitaly/home/system_and_systemengineering/wat-system
- https://en.m.wikipedia.org/wiki/Environmental_control_system
- https://en.m.wikipedia.org/wiki/Air_cycle_machine
- <https://aeronauticallecture.blogspot.com/2014/01/vapour-cycle-cooling-system-in-aircraft.html?m=1>
- https://en.m.wikipedia.org/wiki/Vapor-compression_refrigeration

- <https://www.aircraftsystemstech.com/p/vapor-cycle-air-conditioning-system.html?m=1>
- https://en.m.wikipedia.org/wiki/Model-based_systems_engineering
- <https://www.scaledagileframework.com/model-base-systems-engineering>
- <https://re-magazine.ireb.org/articles/modeling-requirements-with-sysml>
- https://en.wikipedia.org/wiki/Project_management_triangle#Scope
- <https://en.wikipedia.org/wiki/Design-to-cost>
- <https://www.computing.dcu.ie/~renaat/ca421/report.html>
- <https://www.geeksforgeeks.org/software-engineering-cocomo-model/>
- http://sunset.usc.edu/research/COCOMOII/cocomo81_pgm/help.html
- <https://www.airforce-technology.com/projects/predator-uav/> ref.[1]