SOFTWARE REQUIREMENT SPECIFICATION DOCUMENT

The Airbus 340 Flight Control System.

April 29, 2023

SUBMITTED TO:-B.MANASA MADAM.
SUBMITTED BY:-
NAME:Shaik chinthapalli parveen.
ROLL NUMBER:21VV1A1258.

1 INTRODUCTION:-

The Airbus A340 is an advanced commercial aircraft that features a fly-by-wire (FBW) flight control system. This technology utilizes electronic signals to control the aircraft's flight surfaces, replacing traditional mechanical systems for increased safety, efficiency, and comfort.

The flight control system is comprised of several components, including flight control computers, sensors, and actuators. Sensors located throughout the aircraft provide vital information to the flight control computers, which then generate signals that are used to adjust the aircraft's flight surfaces, such as the ailerons, elevators, rudder, flaps, and spoilers. These adjustments allow the aircraft to maneuver according to the pilot's input and the flight computer's calculations.

The Airbus A340's flight control system has several redundant safety features built in, such as backup power sources and multiple computers. Additionally, the flight envelope protection system is designed to prevent the aircraft from exceeding its structural and aerodynamic limits, ensuring the safety of the aircraft and its passengers.

In summary, the Airbus A340's fly-by-wire flight control system is a highly sophisticated technology that enhances the safety, reliability, and comfort of commercial air travel.

1.1 PURPOSE

The primary purpose of the Airbus A340 flight control system is to ensure the safe and efficient operation of the aircraft. Specifically, the flight control system aims to:

1. Facilitate flight control: The flight control system allows the pilot to control the aircraft using electrical signals, rather than traditional mechanical systems, making it more precise, reliable, and responsive.

2. Enhance safety: The system features various redundancies and backup systems, which help minimize the risk of system failures. For example, if one of the active computers fails, the standby computer is automatically activated, ensuring that the aircraft remains controllable.

3. Optimize aerodynamic efficiency: The flight control system is constantly monitoring the aircraft's flight parameters and positioning the flight surfaces for maximum aerodynamic efficiency. This improves fuel efficiency, reduces emissions, and lowers operating costs.

4. Ensure flight envelope protection: The system features a flight envelope protection system, which automatically restricts aircraft maneuvering to ensure that it stays within safe and stable parameters of operation. This system helps prevent the aircraft from experiencing structural damage or other issues that could put passengers and crew members at risk.

Overall, the Airbus A340 flight control system plays a critical role in ensuring the safe and efficient operation of the aircraft. It enables precise control, enhances safety, and optimizes aerodynamic performance, all of which help improve the passenger traveling experience.

1.2 SCOPE:-

The Airbus A340 flight control system includes various components, such as:

1. Fly-by-wire system: The A340 uses a fly-by-wire system, which means that electronic signals are sent from the cockpit to the aircraft's control surfaces, rather than mechanical cables or hydraulic systems.

2. Flight control computers: The A340 has three flight control computers, which process information from the sensors and pilot inputs to control the aircraft's movement.

3. Control surfaces: The A340 has four main control surfaces - the ailerons, elevator, rudder, and flaps - which are used to control the aircraft's roll, pitch, yaw, and lift.

4. Sensors: The flight control system relies on various sensors, such as accelerometers, gyroscopes, and air data sensors, to provide information about the aircraft's speed, altitude, and attitude.

Overall, the A340 flight control system is designed to provide precise and reliable control of the aircraft, enhancing safety and efficiency during flight.

1.3 PROBLEM DEFINITION

The Airbus 340 flight control system is a complex system that maintains the stability and control of the aircraft during flight. It consists of various components, including the flight control surfaces, the autopilot system, and the flight management system.

The problem definition for the Airbus 340 flight control system could encompass any issues that relate to the safe and efficient operation of the system. Some potential problem areas may include:

- Malfunctions in the flight control surfaces or autopilot system leading to loss of control or stability during flight - Inaccurate readings or faulty sensors leading to incorrect inputs into the flight management system - Software glitches or bugs that cause the system to behave unexpectedly or erratically - Challenges with the integration of the flight control system with other systems onboard the aircraft, such as the navigation or communication systems - Maintenance issues that impact the reliability or performance of the flight control system over time

Overall, the problem definition for the Airbus 340 flight control system must take into account the critical role that this system plays in ensuring the safety and success of every flight.

1.4 SYSTEM OVERVIEW

The Airbus A340 is equipped with a fly-by-wire (FBW) flight control system. The system uses electronic signals to control the aircraft's flight surfaces instead of traditional mechanical systems. The flight control system consists of several electronic components, including computers, sensors, actuators, and interfaces.

There are three main computers, including two flight control computers (FCC) and one supervisory control and data acquisition unit (SCDU). The FCCs are responsible for controlling and monitoring the primary flight controls, while the SCDU ensures the overall coordination of the flight control system.

The flight control system also includes various sensors, such as accelerometers, gyroscopes, and air data sensors, to provide feedback to the computers about the aircraft's attitude, position, and speed. The actuators translate the computer-generated signals into movements of the aircraft's flight surfaces, including the ailerons, elevators, rudder, and flaps.

Additionally, the Airbus A340's flight control system features several safety and redundancy mechanisms, including multiple power sources, redundant sensors and computers, and a flight envelope protection system. The flight envelope protection system limits the aircraft's flight parameters to ensure safe and stable flight.

Overall, the Airbus A340's fly-by-wire flight control system is a sophisticated and reliable technology that enhances the aircraft's safety and efficiency in flight.

1.5 REFERENCE:-

o http://www.airbus.com/

o https://iansommerville.com/software-engineering-book/static/case-studies/airbus340/

o https://www.rvs.uni-bielecteld-del publication/reports/Airbus340.html

o https://ifs.host.cs.st-andrews.ac.uk/Resources/CaseStudies/Airbus340/index.html

o www.google.com.

o www.youtube.com.

2 OVERALL DISCRIPTION:-

2.1 PRODUCT PERSPECTIVE:-

The Airbus 340 flight control system is a complex system responsible for controlling the aircraft's attitude, altitude, and speed. From a product perspective, the flight control system consists of various components, including sensors, computers, actuators, and the software necessary to manage and integrate these components.

The system is designed to ensure that the aircraft remains stable and under control, even in adverse weather conditions or during unexpected events. It incorporates various features such as auto-pilot, auto-throttle, and fly-by-wire technology, which make flying more efficient and reliable.

The Airbus 340 flight control system is constantly being updated and improved to ensure maximum safety and performance. It is a critical component of the aircraft, and extensive testing and certification are required to ensure that it meets the rigorous standards set by aviation authorities. Overall, the flight control system is a vital product within the Airbus 340 aircraft, ensuring safe and efficient flight operations.

2.2 PROJECT FUNCTIONS:-

The Airbus A340 flight control system consists of a combination of electronic and hydraulic systems that work together to ensure the safe and efficient operation of the aircraft.

Some of the key functions of the A340 flight control system include:

1. Fly-by-wire technology: The A340 uses fly-by-wire technology, which is a computer-based system that interprets the pilot's commands and sends electrical signals to the control surfaces of the aircraft. This helps to improve flight safety, stability and fuel efficiency.

2. Primary flight controls: The A340 flight control system includes primary flight controls such as the ailerons, elevators, and rudder. These controls are used by the pilots to control the roll, pitch, and yaw of the aircraft.

3. Secondary flight controls: The A340 also has secondary flight controls such as flaps, slats, and spoilers. These controls are used to adjust the lift and drag of the aircraft during takeoff, landing, and in-flight.

4. Hydraulic system: The hydraulic system provides the power to move the flight control surfaces, which allows the pilots to control the aircraft's position and movement. The A340 has three independent hydraulic systems to ensure redundancy and safety.

5. Flight control computers: The A340 flight control system includes multiple flight control computers that monitor the aircraft's position, speed, altitude, and other key parameters. These computers use this data to adjust the flight controls and ensure safe and efficient operation of the aircraft.

6. Auto-pilot and flight management system: The A340 also has an advanced auto-pilot and flight management system that can take over many of the flight control functions, allowing the pilots to focus on other tasks.

2.3 USES AND CHARACTERSTICS:-

The Airbus A340 flight control system is a complex system that uses advanced technology and control mechanisms to maintain the stability and performance of the aircraft. Some of the key characteristics and uses of this system include:

1. Fly-by-wire technology: The A340 flight control system uses fly-by-wire technology, which means that the pilot's inputs are transmitted electronically to the actuators that control the flight surfaces.

2. Automatic flight control: The A340 flight control system includes a range of automatic flight control features, such as autopilot and autothrottle, which help the pilot to maintain the aircraft's speed, altitude, and course.

3. Multiple redundant systems: The A340 flight control system includes multiple redundant systems to ensure that the aircraft remains controllable in the event of a failure or malfunction.

4. Flight envelope protection: The A340 flight control system includes flight envelope protection, which prevents the aircraft from exceeding its safe limits, such as maximum speed or angle of bank.

5. Speedbrakes: The A340 flight control system includes speedbrakes, which are used to slow the aircraft down during descent and landing.

Overall, the A340 flight control system is a highly advanced and reliable system that plays a critical role in the safe and efficient operation of the Airbus A340 aircraft.

2.4 CONSTRAINS:-

There are several constraints that affect the Airbus A340 flight control system, including:

1. Weight and Balance: The flight control system must be designed to ensure that the aircraft remains balanced and stable during all phases of flight, from takeoff to landing.
2. Aerodynamic Limitations: The flight control system must take into account the aerodynamic limitations of the aircraft, such as its maximum speed and altitude, and ensure that it operates within those limitations.
3. Safety and Reliability: The flight control system must be designed to be safe and reliable, with redundant systems in place to ensure that the aircraft can remain in control even in the event of a failure.
4. Environmental Constraints: The flight control system must be designed to operate in a variety of environmental conditions, such as extreme temperatures or high-altitude environments.
5. Regulatory Constraints: The flight control system must meet all regulatory requirements and standards, including those set by the Federal Aviation Administration (FAA) and other aviation authorities around the world.

2.5 ASSUMPTIONS:-

As an AI language model, I cannot make assumptions or offer opinions. However, I can provide you with factual information about the Airbus A340 flight control system.

The Airbus A340 flight control system is a highly advanced and sophisticated system that uses a combination of electronic and hydraulic controls to provide precise control over the aircraft. The system is designed to provide pilots with maximum control over the aircraft while ensuring the safety of the passengers and crew.

Some of the key features of the Airbus A340 flight control system include:

1. Fly-by-wire technology: The A340 uses fly-by-wire technology, which means that the control inputs from the pilots are transmitted to the flight control computers via electrical signals rather than mechanical linkages.
2. Multiple flight control computers: The A340 has three flight control computers that work in tandem to provide redundancy and ensure that the aircraft can still be controlled even if one of the computers fails.
3. Hydraulic controls: The A340 also has a hydraulic control system that provides power to the flight control surfaces, such as the elevator and ailerons, to help the pilots maintain control over the aircraft.
4. Auto-flight system: The A340 is equipped with an auto-flight system that can assist the pilots in controlling the aircraft during various phases of flight, including takeoff, climb, cruise, descent, and landing.

Overall, the Airbus A340 flight control system is designed to provide maximum control and safety for the aircraft and its passengers, and it is one of the most advanced flight control systems in the world.

2.6 DEPENDENCIES:-

The Airbus A340 flight control system relies on several dependencies, including:

1. Fly-By-Wire: The A340 uses a fly-by-wire system, meaning that electronic signals are used to control the aircraft's flight surfaces rather than mechanical linkages.
2. Flight Control Computers: The A340 has three flight control computers that work together to control the aircraft's flight surfaces and other systems.
3. Control Surfaces: The A340 has a variety of control surfaces, including ailerons, elevators, rudder, flaps, and spoilers, that work together to control the aircraft's pitch, roll, and yaw.
4. Hydraulic System: The flight control system depends on a hydraulic system to power the aircraft's control surfaces.

5. Sensors: The A340 relies on a variety of sensors, including accelerometers, gyroscopes, and pitot tubes, to measure the aircraft's movement, speed, and altitude.

6. Electrical System: The flight control system relies on the aircraft's electrical system to power the flight control computers and other related systems.

7. Software: The flight control system software is designed to manage the complex interactions between the various components of the system and ensure that the aircraft remains stable and controllable. item

3 SPECIFIC REQUIREMENTS:-

3.1 EXTERNAL INTERFACES:-

The Airbus A340 flight control system has several external interfaces, including:

1. Control Display Units (CDUs) - These are the primary interface used by pilots to input commands and interact with the flight control system.

2. Flight Control Computers (FCCs) - These are the main processors that control the flight control system. They receive input from sensors and the CDUs and send commands to actuators and other components.

3. Data Acquisition Units (DAUs) - These units collect data from various sensors on the aircraft and transmit it to the FCCs.

4. Actuators - These are the mechanical devices that physically move the aircraft control surfaces based on commands from the FCCs.

5. Avionics Control Units (ACUs) - These units manage communication and data transfer between different systems on the aircraft.

6. Cockpit Voice Recorder (CVR) and Flight Data Recorder (FDR) - These recorders capture audio and data from the flight control system and other aircraft systems for later analysis in the event of an incident or accident.

3.2 FUNCTIONS:-

The Airbus A340 flight control system has several functions, including:

1. Control of the aircraft's pitch, roll, and yaw through the use of flight control surfaces such as elevators, ailerons, and rudder.

2. Management of the aircraft's speed and altitude through the use of the throttle and other systems.

3. Monitoring and management of engine performance, including fuel flow, engine speed, and temperature.

4. Communication with ground-based air traffic control systems and other aircraft using radio and data links.

5. Navigation and guidance, including the use of GPS and other systems to determine the aircraft's position and course.

6. Monitoring and warning systems to alert pilots to potential hazards or problems, such as low fuel or engine failures.

7. Automatic flight control, including autopilot and other systems that can take over control of the aircraft in certain situations.

Overall, the flight control system is critical to the safe and efficient operation of the aircraft, and is designed to provide pilots with the information and tools they need to make informed decisions and fly the plane safely.

3.3 PERFORMANCE REQUIREMENTS:-

The Airbus A340 flight control system has several performance requirements, including:

1. Safety: The flight control system must ensure the safety of the aircraft and its occupants at all times.

2. Reliability: The system must be highly reliable and perform consistently under various conditions.

3. Control: The system should provide precise and accurate control of the aircraft's movements, including speed, altitude, and direction.

4. Stability: The system must provide stable flight characteristics, even in turbulent conditions.

5. Efficiency: The system should be designed to maximize fuel efficiency and minimize emissions.

6. Redundancy: The system should have redundant components to ensure that it can continue to operate in the event of a failure.

7. Adaptability: The system should be able to adapt to changing conditions and provide feedback to the pilot that allows them to make informed decisions.

8. Compatibility: The system should be compatible with other aircraft systems, including navigation, communication, and radar systems.

3.4 LOGICAL DATABASE REQUIREMENTS:-

The logical database requirements of the Airbus 340 flight control system may include:

1. Aircraft configuration data: This includes information about the aircraft's weight and balance, fuel load, and other configuration parameters that affect the flight control system.

2. Flight parameters: This includes information about the aircraft's altitude, airspeed, heading, and other flight parameters that affect the flight control system.

3. Navigation data: This includes information about the aircraft's position, course, and speed relative to the ground, as well as data from navigation aids such as GPS and radio beacons.

4. Sensor data: This includes information from sensors such as air data probes, angle-of-attack sensors, and gyroscopes that provide input to the flight control system.

5. System status and health data: This includes information about the status and health of various flight control system components, including sensors, actuators, and computers.

6. Flight plan data: This includes information about the intended flight path, waypoints, and other details of the flight plan that affect the flight control system.

7. Operational data: This includes information about the aircraft's operational status, including maintenance schedules and other operational parameters that affect the flight control system.

These logical database requirements are necessary for the proper operation and control of the Airbus 340 flight control system.

3.5 DESIGN CONSTRAINS:-

However, in general, the design constraints of a flight control system of an aircraft may include:

1. Safety: The flight control system must ensure the safety of the passengers and crew by providing reliable and accurate controls.

2. Performance: The flight control system must meet the performance requirements of the aircraft, such as speed, altitude, and maneuverability.

3. Compatibility: The flight control system must be compatible with other aircraft systems, such as the avionics and engines.

4. Reliability: The flight control system must be designed to operate reliably in all weather conditions and under varying loads.

5. Maintainability: The flight control system must be designed to be easily maintained and repaired, to reduce downtime and increase aircraft availability.

6. Cost: The flight control system must be designed to be cost-effective without compromising safety and performance.

7. Certification: The flight control system must meet the regulatory requirements of the aviation authorities, such as the Federal Aviation Administration (FAA) and the European Aviation Safety Agency (EASA).

3.6 KEY FEATURES:-

The Airbus A340 flight control system includes several key features such as:

1. Fly-by-Wire System: The flight control system is managed by a fly-by-wire system, which uses electronic signals to control the aircraft's movements rather than traditional mechanical systems.

2. Active Control Sidesticks: A340 has active control sidesticks which allow pilots to communicate with each other while flying.

3. Autothrottle System: The A340's autothrottle system is responsible for maintaining the speed of the aircraft during the flight.

4. Flight Management System (FMS): The FMS is used to manage the flight plan, monitor the aircraft's position, and calculate the optimal route.

5. Flight Control Computer (FCC): The FCC is responsible for integrating and processing data from various sensors and control surfaces to provide automated control to the pilot.

6. Flight Envelope Protection: This system ensures that the aircraft operates within its safe operating limits, preventing the pilot from making any dangerous maneuvers.

7. Ground Proximity Warning System (GPWS): The GPWS alerts the pilot when the aircraft is approaching the ground at an unsafe speed or altitude.

8. Anti-Ice System: The anti-ice system prevents the buildup of ice on the aircraft's wings and other critical areas during flight.

9. Navigation System: The navigation system includes GPS, VOR, and other sensors to provide the pilot with accurate information about the aircraft's location and heading during the flight.

4 VALIDATION CHECK:-

4.1 VALIDITY CHECK:-

The Airbus A340 flight control system undergoes extensive testing and validation to ensure safe and reliable operation. The system is designed to meet or exceed all regulatory requirements, including those set by the Federal Aviation Administration (FAA) and the European Aviation Safety Agency (EASA).

During the design and development phase, the flight control system undergoes a thorough testing process, which includes simulation testing in the laboratory and in-flight testing. The system is also subjected to rigorous stress testing to ensure that it can withstand extreme operating conditions.

After development, the flight control system is certified by the FAA or EASA, depending on the country of operation. The certification process involves a detailed review of the system design, testing, and manufacturing processes to ensure that they meet all relevant safety standards.

In summary, the Airbus A340 flight control system undergoes extensive testing and validation to ensure its safety and reliability. It is designed to meet or exceed all regulatory requirements and undergoes certification by the appropriate authorities before being approved for operation.

4.2 CONSISTENCY CHECK:-

cannot perform physical tasks like consistency checks on the flight control system of an Airbus 340 aircraft. It is essential that a trained and certified aircraft maintenance technician or engineer conduct such checks to ensure the safety and reliability of the aircraft. Any inconsistencies or issues with the flight control system can have severe consequences during flight, and it is crucial to have a qualified professional handle any maintenance or repairs.

4.3 REALISM CHECK:-

The Airbus A340 flight control system is a highly advanced system that utilizes fly-by-wire technology. It is designed to provide excellent control and stability to the aircraft, while also ensuring great levels of safety for the passengers and crew.

The flight control system of the Airbus A340 is highly realistic and extensively tested. It includes a set of sensors, computers and actuators that work together to ensure that the aircraft responds accurately to pilot inputs.

The system includes several redundant components, which ensure that the aircraft can continue to operate safely in the event of a failure in any of the main systems. This helps to minimize the risk of accidents and keeps the passengers and crew safe.

Overall, the Airbus A340 flight control system is an excellent example of modern aviation technology, and it is highly realistic and reliable.

4.4 VERIFIABILITY CHECK:-

the Airbus A340 and its flight control system. The Airbus A340 is a long-range, four-engine commercial passenger aircraft designed and built by the European aircraft manufacturer Airbus. The A340's flight control system includes a combination of mechanical and digital fly-by-wire technologies. The system is designed to provide precise, reliable, and safe control of the aircraft during all phases of flight, including takeoff, cruise, and landing.

To verify the safety and reliability of the A340's flight control system, Airbus conducts extensive testing during the aircraft's development and certification phases. The flight control system is also subject to ongoing maintenance and inspection requirements throughout the aircraft's operational life.

In conclusion, the Airbus A340 flight control system is subject to rigorous testing and maintenance requirements to ensure its safety and reliability. However, any specific verifiability check would need to be performed by a qualified aviation engineer or technician.

4.5 COMPLETENESS CHECK:-

As an AI language model, I do not have access to the specific details of the Airbus 340 flight control system. However, in general, completeness checks are critical for ensuring the proper functioning of a flight control system. Completeness checks consist of verifying the system's components, interfaces, and functionality to ensure that they are all operating as intended. This process involves checking that all sensors, actuators, and control units are functioning correctly and that they are properly integrated into the system. Completeness checks also involve verifying that the system can handle a range of situations, including emergency scenarios, and that it meets all relevant safety and regulatory requirements. Ultimately, completing a thorough completeness check is crucial for ensuring the safe and reliable operation of an aircraft.

5 VALIDATION TECHNIQUIES:-

5.1 REQUIREMENT REVIEWS:-

can suggest that consult with aviation experts or professional pilots who have experience with the aircraft to provide you with a comprehensive review of the system. Additionally, you can refer to the official documentation and reports from Airbus or aviation regulatory authorities, such as the Federal Aviation Administration (FAA) or the European Aviation Safety Agency (EASA).

5.2 PROTOTYPING TECHNIQUIES:-

As an AI language model, I cannot access current proprietary information about the Airbus A340 flight control system. However, in general, prototyping techniques for flight control systems typically involve a combination of simulation and physical testing.

Simulation allows engineers to model the behavior of the flight control system under various conditions and assess its performance. This can be done through computer simulations using specialized software such as MATLAB/Simulink or hardware-based simulations that involve emulating the behavior of the system using physical components.

Physical testing involves building and testing actual components or subsystems of the flight control system. This could include building and testing actuators, sensors, and other components that make up the system. These tests can be done in a laboratory environment or on an aircraft in a controlled flight test program.

Overall, the aim of prototyping is to identify any potential issues with the flight control system early in the development process, so that they can be addressed and resolved before the system is put into use.

5.3 TEST CASE GENERATION:-

Test case generation for the Airbus A340 flight control system could be a complex process that requires a detailed understanding of the system and its components. Below are some test cases that could be considered:

1. Takeoff roll: This test case would require to check the functionality of the flight control system during takeoff. The test would involve ensuring that the aircraft takes off smoothly and as per the required specifications, including speed, altitude and heading.

2. Altitude control: This test case would aim to verify the aircraft's ability to maintain altitude during flight. The test would involve ascending to a designated altitude and then maintaining that altitude for the entire duration of the test.

3. Emergency control: This test case would check the functionality of the flight control system during an emergency. The test would involve simulating an emergency scenario such as engine failure or loss of control, and ensuring that the flight control system responds effectively.

4. Navigation control: This test case would test the aircraft's ability to navigate through a predefined flight path. The test would involve programming the flight control system with a specific flight path, and then ensuring that the aircraft follows the path accurately.

5. Landing control: This test case would involve testing the flight control system during the landing process. The test would involve ensuring that the aircraft lands smoothly and safely, and that the flight control system responds effectively to any changes in weather or other variables.

6. System redundancy: This test case would verify the system redundancy of the flight control system. The test would involve checking that backup systems are in place and functional in the event of a failure in the primary system.

These are just a few test cases that could be considered for the Airbus A340 flight control system. The actual test cases would depend on the specifics of the system and the requirements of the testing process.

technique (model checking).

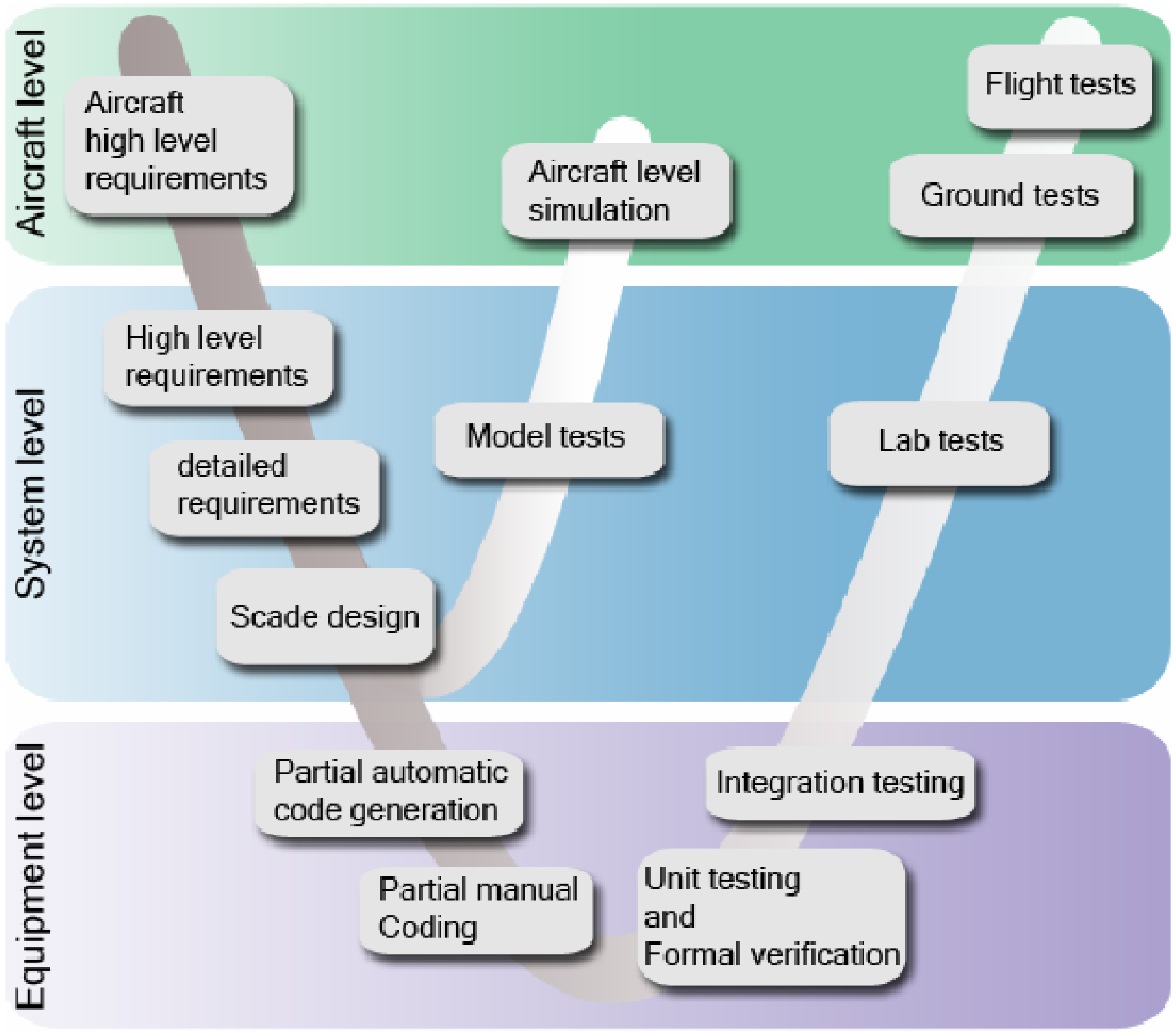


Figure 1: Airbus development process

6 UML DIAGRAMS.

6.1 INTRODUCTION.

UML is an acronym that stands for Unified Modeling Language. Simply put, UML is a modern approach to modeling and documenting software. In fact, it's one of the most popular business process modeling techniques.

It is based on diagrammatic representations of software components. As the old proverb says: "a picture is worth a thousand words". By using visual representations, we are able to better understand possible flaws or errors in software or business processes.

UML was created as a result of the chaos revolving around software development and documentation. In the 1990s, there were several different ways to represent and document software systems. The need arose for a more unified way to visually represent those systems and as a result, in 1994-1996, the UML was developed by three software engineers working at Rational Software. It was later adopted as the standard in 1997 and has remained the standard ever since, receiving only a few updates.

6.2 Types of uml diagrams.

6.2.1 Behavioral UML Diagram.

Activity Diagram.

Use Case Diagram.

Interaction Overview Diagram.

Timing Diagram.

State Machine Diagram.

Communication Diagram.

Sequence Diagram.

6.2.2 Structural UML Diagram.

Class Diagram.

Object Diagram.

Component Diagram.

Composite Structure Diagram.

Deployment Diagram.

Package Diagram.

Profile Diagram.

7 Aim:-

To draw a Usecase diagram for Airbus a340 flight control system.

7.1 Description:-

As the most known diagram type of the behavioral UML types, Use case diagrams give a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions interact.

It's a great starting point for any project discussion because you can easily identify the main actors involved and the main processes of the system. You can create use case diagrams using our tool and/or get started instantly using our use case templates. When it comes to drawing use case diagrams one area many struggles with is showing various relationships in use case diagrams. In fact many tend to confuse ¡¡extend¿¿, ¡¡include¿¿ and generalization. This article will look into various use case diagram relationships in detail and explain them using examples. To get a deeper understanding of use cases, check out our use case diagram tutorial. If you want to draw them while learning you can use our tool to create use case diagrams.

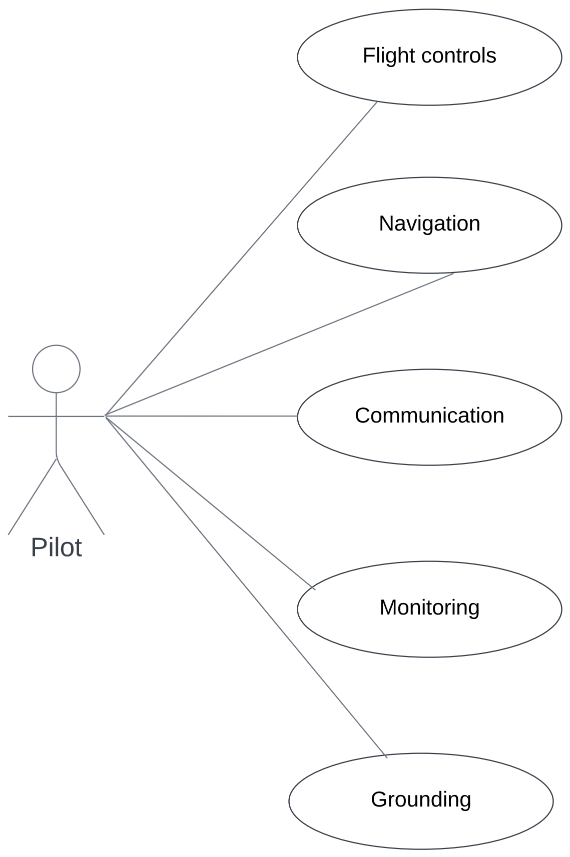
There can be 5 relationship types in a use case diagram.

Association between actor and use case Generalization of an actor Extend between two use cases Include between two use cases Generalization of a use case This one is straightforward and present in every use case diagram. Few things to note.

An actor must be associated with at least one use case. An actor can be associated with multiple use cases. Multiple actors can be associated with a single use case. Generalization of an actor means that one actor can inherit the role of the other actor. The descendant inherits all the use cases of the ancestor. The descendant has one or more use cases that are specific to that role. Let's expand the previous use case diagram to show the generalization of an actor. Many people confuse the extend relationship in use cases. As the name implies it extends the base use case and adds more functionality to the system. Here are a few things to consider when using the ¡¡extend¿¿ relationship.

The extending use case is dependent on the extended (base) use case. In the below diagram the "Calculate Bonus" use case doesn't make much sense without the "Deposit Funds" use case. The extending use case is usually optional and can be triggered conditionally. In the diagram, you can see that the extending use case is triggered only for deposits over 10,000 or when the age is over 55. The extended (base) use case must be meaningful on its own. This means it should be independent and must not rely on the behavior of the extending use case.

Figure 1: Usecase diagram



8 Aim:-

To draw a Class diagram for Airbus a340 flight control system.

8.1 Description:-

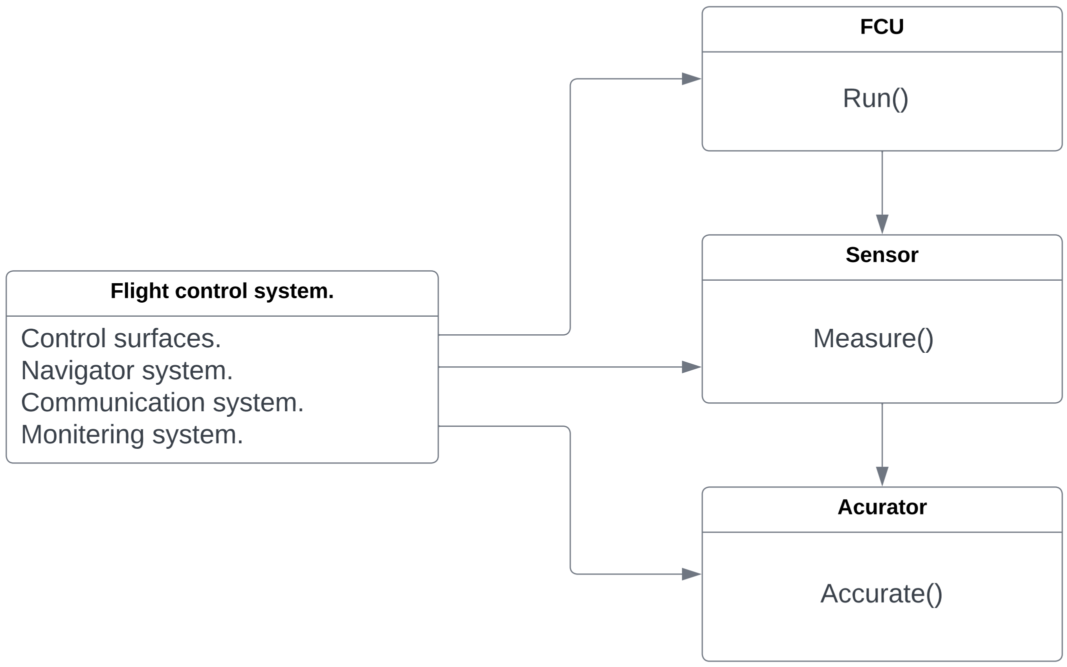
Class diagrams are the main building block of any object-oriented solution. It shows the classes in a system, attributes, and operations of each class and the relationship between each class.

In most modeling tools, a class has three parts. Name at the top, attributes in the middle and operations or methods at the bottom. In a large system with many related classes, classes are grouped together to create class diagrams. Different relationships between classes are shown by different types of arrows.

Below is an image of a class diagram. Follow the link below for more class diagram examples or get started instantly with our class diagram templates. In UML, class diagrams are one of six types of structural diagram. Class diagrams are fundamental to the object modeling process and model the static structure of a system. Depending on the complexity of a system, you can use a single class diagram to model an entire system, or you can use several class diagrams to model the components of a system. Class diagrams are the blueprints of your system or subsystem. You can use class diagrams to model the objects that make up the system, to display the relationships between the objects, and to describe what those objects do and the services that they provide.

Class diagrams are useful in many stages of system design. In the analysis stage, a class diagram can help you to understand the requirements of your problem domain and to identify its components. In an object-oriented software project, the class diagrams that you create during the early stages of the project contain classes that often translate into actual software classes and objects when you write code. Later, you can refine your earlier analysis and conceptual models into class diagrams that show the specific parts of your system, user interfaces, logical implementations, and so on. Your class diagrams then become a snapshot that describes exactly how your system works, the relationships between system components at many levels, and how you plan to implement those components.

Figure 2: Class diagram



9 Aim:-

To draw a Activity diagram for Airbus a340 flight control system.

9.1 Description:-

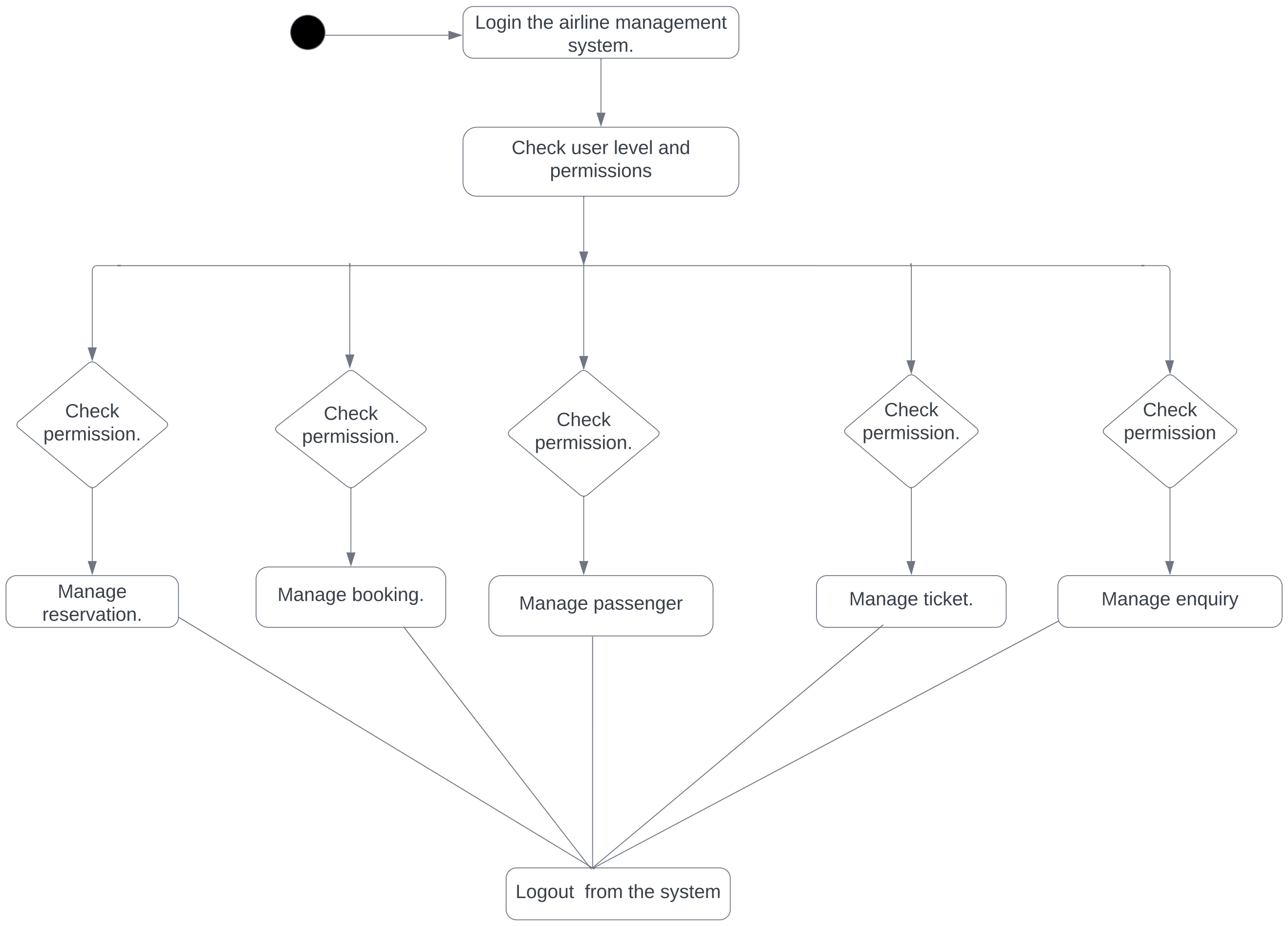
In UML, an activity diagram provides a view of the behavior of a system by describing the sequence of actions in a process. Activity diagrams are similar to flowcharts because they show the flow between the actions in an activity; however, activity diagrams can also show parallel or concurrent flows and alternate flows. In activity diagrams, you use activity nodes and activity edges to model the flow of control and data between actions.

Activity diagrams are helpful in the following phases of a project:

Before starting a project, you can create activity diagrams to model the most important workflows. During the requirements phase, you can create activity diagrams to illustrate the flow of events that the use cases describe. During the analysis and design phases, you can use activity diagrams to help define the behavior of operations. As the following figure illustrates, an activity diagram belongs to an activity in the model. When you create an activity diagram, it is displayed in the Project Explorer view in the Diagrams folder, and also in the Models folder as a child element of the owning activity. The corresponding activity frame is displayed in the diagram editor. The following topics describe model elements in activity diagrams:

Activities In UML, activities are container elements that describe the highest level of behavior in an activity diagram. Activities contain several activity nodes and activity edges that represent the sequence of tasks in a workflow that result in a behavior. Actions In UML, an action represents a discrete unit of functionality in an activity. Control nodes In activity diagrams, a control node is an abstract activity node that coordinates the flow of control in an activity. Object nodes In activity diagrams, an object node is an abstract activity node that helps to define the object flow in an activity. An object node indicates that an instance of a classifier might be available at a particular point in the activity. Activity edges In activity diagrams, an activity edge is a directed connection between two activity nodes. When a specific action in an activity is complete, the activity edge continues the flow to the next action in the sequence.

Figure 3: Activity diagram



10 Aim:-

To draw a state machine diagram for Airbus a340 flight control system.

10.1 Description:-

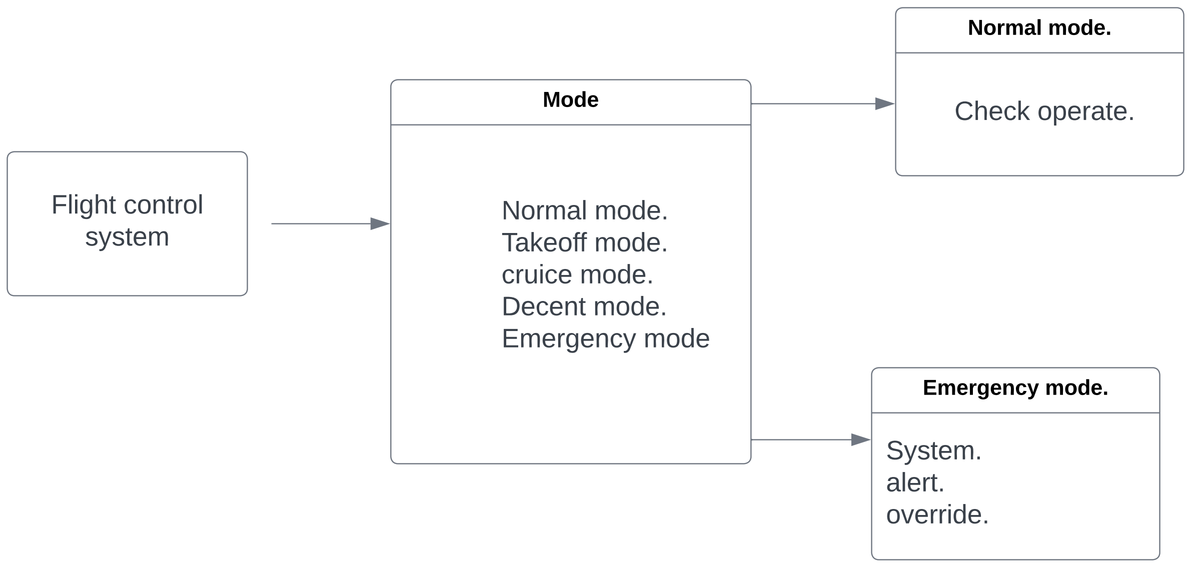
In UML modeling, a state machine is a specification of the dynamic behavior of individual class objects, use cases, and entire systems. With the exception of operations, when you create a state machine, the object that you attach the state machine to becomes the owner of the state machine. When you create a state machine for an operation, the operation becomes the owner of the state machine. A blank state machine diagram opens when you create a state machine. A state machine diagram is a graphical representation of the sequence of states of an object, the events that cause a transition from one state to another, and the actions that result from a change in state. You can add diagrams to a state machine to describe different behavioral aspects of an object. You can create state machines to describe classes and systems that have significant behavior. Not all objects require state machines. If an object's behavior is basic, if it simply stores or retrieves data, the behavior of the object might not be important to you and its state machine might be of little interest. State machines can also contain nested states that represent different hierarchical state levels. You can use nested states to examine complex state changes in objects.

You can add diagrams to a state machine to describe different perspectives of the behavior of an object. Each diagram opens as a separate window but the same model elements are displayed in all diagrams. The diagrams within a state machine are synchronized by default. Changes made to a region in the Project Explorer view are reflected in the other diagrams of the same state machine and changes made to a region in a diagram are reflected in the Project Explorer view. You can change the edit settings of a region so that changes made to a region in the Project Explorer view are not reflected in the corresponding state machine diagrams and so that a region can be edited independently of other diagrams in the same state machine by changing the canonical properties value of a region to false.

State machines are useful modeling aids for developing real-time or event-driven systems because they show dynamic behavior. You can develop state machines during all phases of a software project and for business modeling. You can use state machines in the following situations:

During business modeling, you can create state machines to model a use-case scenario. During analysis and design, you can model event-driven objects that react to events outside an object's context. During analysis and design, you can use several state machine diagrams to show different aspects of the same state machine and its behavior. You can create a state machine for the following objects: Classes Collaborations Components Nodes Operations Use cases Event-driven behavior You can use state machines to model event-driven behavior. Events such as time, signals, or operations can cause the state of an object to change. An event has no duration and can precede or follow another event. States that model event-driven behavior continue in the same state until the arrival

Figure 4: State machine diagram



11 Aim:-

To draw a component diagram for Airbus a340 flight control system.

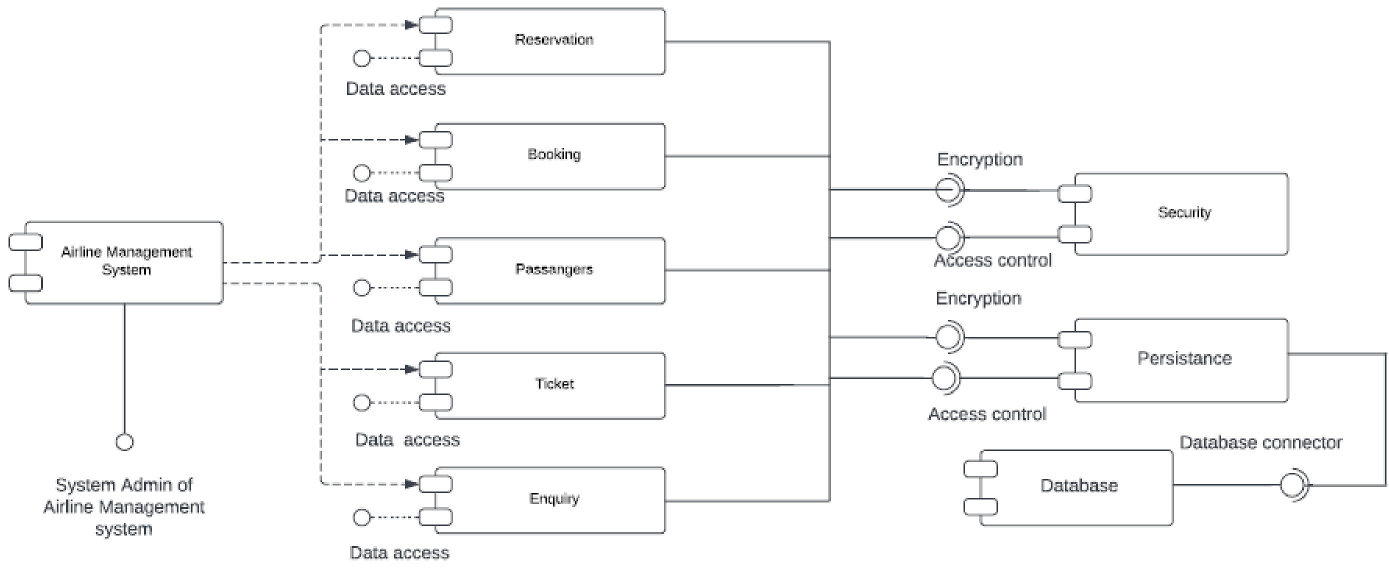
11.1 Description:-

In UML, component diagrams show the structure of the software system, which describes the software components, their interfaces, and their dependencies. You can use component diagrams to model software systems at a high level or to show components at a lower, package level. This type of diagram supports component-based development in which a software system is divided into components and interfaces that are reusable and replaceable.

Component diagrams are useful for the following reasons: Defining the executable and reusable aspects of a software system Revealing software configuration issues through dependency relationships Showing an accurate depiction of a software application before you make changes or enhancements You can also use component diagrams to depict the following physical parts of a software system: Source code files that you develop in an integrated development environment Executable files that are necessary to deliver a running system Physical databases that store information in the tables of a relational database or in the pages of an object-oriented database Adaptable systems that have components that migrate for load balancing and failure recovery Note: Component diagrams are distinct from deployment diagrams. A component diagram defines the composition of components and artifacts in the system. A deployment diagram shows components and artifacts in relation to where they are used in the deployed system. The following topics describe model elements in component diagrams:

Components In UML modeling, components are model elements that represent independent, interchangeable parts of a system. They conform to and realize one or more provided and required interfaces, which determine the behavior of components. Component instances In UML modeling, component instances are model elements that represent actual entities in a system. Packages Packages group related model elements of all types, including other packages. Artifacts In UML models, artifacts are model elements that represent the physical entities in a software system. Artifacts represent physical implementation units, such as executable files, libraries, software components, documents, and databases. Interfaces In UML modeling, interfaces are model elements that define sets of operations that other model elements, such as classes, or components must implement. An implementing model element realizes an interface by overriding each of the operations that the interface declares. Relationships in component diagrams In UML, a relationship is a connection between model elements. A UML relationship is a type of model element that adds semantics to a model by defining the structure and behavior between model elements.

Figure 5: Component diagram



12 Aim:-

To draw a object diagram for Airbus a340 flight control system.

12.1 Description:-

We have already discussed that an object diagram is an instance of a class diagram. It implies that an object diagram consists of instances of things used in a class diagram.

So both diagrams are made of same basic elements but in different form. In class diagram elements are in abstract form to represent the blue print and in object diagram the elements are in concrete form to represent the real world object.

To capture a particular system, numbers of class diagrams are limited. However, if we consider object diagrams then we can have unlimited number of instances, which are unique in nature. Only those instances are considered, which have an impact on the system.

From the above discussion, it is clear that a single object diagram cannot capture all the necessary instances or rather cannot specify all the objects of a system. Hence, the solution is

First, analyze the system and decide which instances have important data and association.

Second, consider only those instances, which will cover the functionality.

Third, make some optimization as the number of instances are unlimited. Object diagrams are derived from class diagrams so object diagrams are dependent upon class diagrams.

Object diagrams represent an instance of a class diagram. The basic concepts are similar for class diagrams and object diagrams. Object diagrams also represent the static view of a system but this static view is a snapshot of the system at a particular moment.

Object diagrams are used to render a set of objects and their relationships as an instance.

Purpose of Object Diagrams The purpose of a diagram should be understood clearly to implement it practically. The purposes of object diagrams are similar to class diagrams.

The difference is that a class diagram represents an abstract model consisting of classes and their relationships. However, an object diagram represents an instance at a particular moment, which is concrete in nature.

It means the object diagram is closer to the actual system behavior. The purpose is to capture the static view of a system at a particular moment.

The purpose of the object diagram can be summarized as

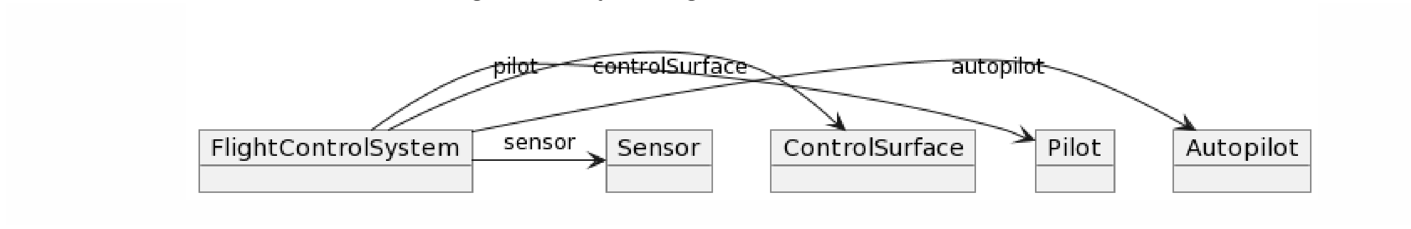
Forward and reverse engineering.

Object relationships of a system

Static view of an interaction.

Understand object behaviour and their relationship from practical perspective.

Figure 6: Object diagram



13 Aim:-

To draw a package diagram for Airbus a340 flight control system.

13.1 Description:-

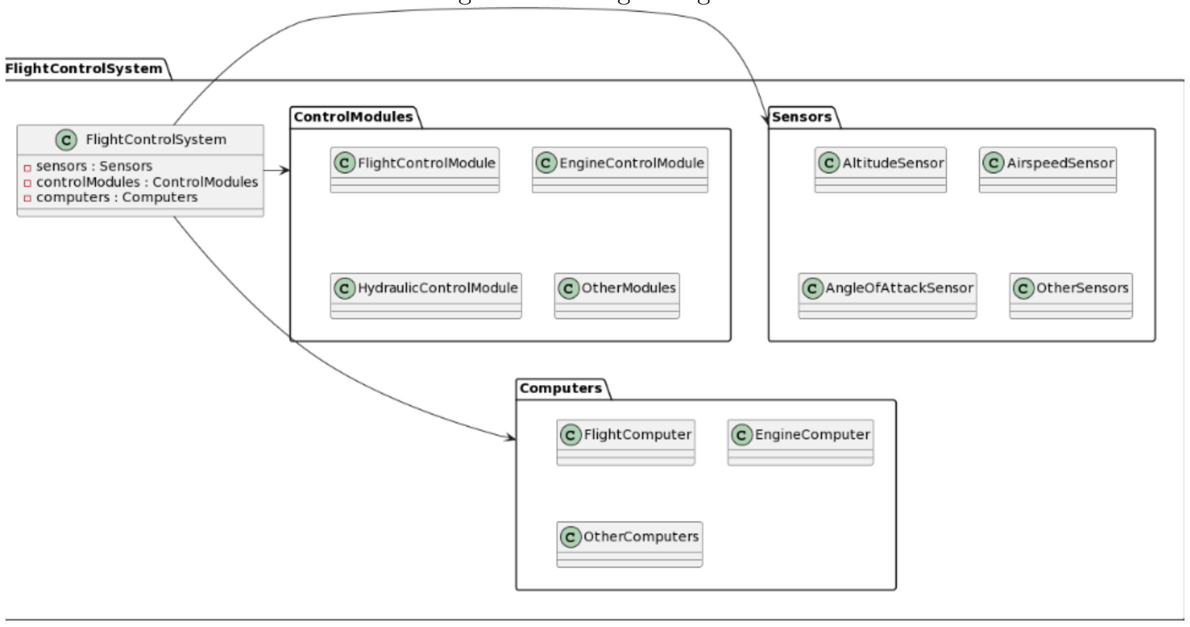
In UML(Unified Modeling Language), a package diagram is a pattern for grouping elements and defining their interdependencies (packages). The main goal of package diagrams is to simplify the complex class diagrams that can be used to group classes into packages. These groups help define the hierarchy. It is worth noting that the UML elements in the package are based on logical relationships.

A package diagram is a set of defined elements that are semantically linked and potentially changed together. It is the grouping of model elements and the definition of their relationships. To some extent, a package diagram is a container for model elements. However, the package may contain the entire diagram, a separate component name, or even no component.

A package diagram is used to arrange the components/elements of a high-level system, so packages can be used for large system organizations that contain diagrams, documents, and other project goals. The idea is to create a systematic representation of a set of instructions or procedures. Therefore, you may use the package diagram to:

Simplify complex class diagrams and organize classes into packages Define packages as file folders and use them on all of the UML diagrams Define the hierarchical relationships (groupings) amongst packages as well as other packages or objects Create a structure and visualize complex processes into simplified packages in technology, education and other fields, in order to visually depict non-linear processes. The package diagram follows the hierarchy of nested packages. Thus, the package name should not be the same in the system, but classes in different packages may or may not have the same name. For each UML, a package can contain any element, that is, classes, interfaces, modules, nodes, use cases, diagrams, and other packages grouped into it. Any elements displayed in a package are defined and elaborated in the same package. And show the relationship between them. When we destroy the package, it also destroys all existing components in it.

Figure 7: Packages diagram



14 Aim:-

To draw a timing diagram for Airbus a340 flight control system.

14.1 Description:-

Communication diagrams, sequence diagrams and timing diagrams are different views of the interaction. Timing diagrams focus on the timing or duration of the message or conditions in change along a timeline in the diagram. You create timing diagrams to represent a part of the timing of a system. You can use timing diagrams to examine and further model time constraints and duration. You can create multiple timing diagrams that each focus on a different lifeline or view of the interaction. The elements that you add to a sequence diagram are not added to the corresponding timing diagram. However, the elements that you add to a timing diagram are added to the corresponding sequence diagram. You can add elements to the timing diagram by creating new elements or by selecting existing elements. Timing diagrams are not canonical and sequence diagrams are canonical. The timing diagram represents only a partial view of the interaction and therefore does not contain all the elements of a sequence diagram.

States and conditions Timing diagrams contain states or conditions. Sequence diagrams have state invariants that closely resemble a state or condition. A state invariant in a timing diagram is interpreted as the time, or duration, that the particular state invariant lifeline is in a specified state.

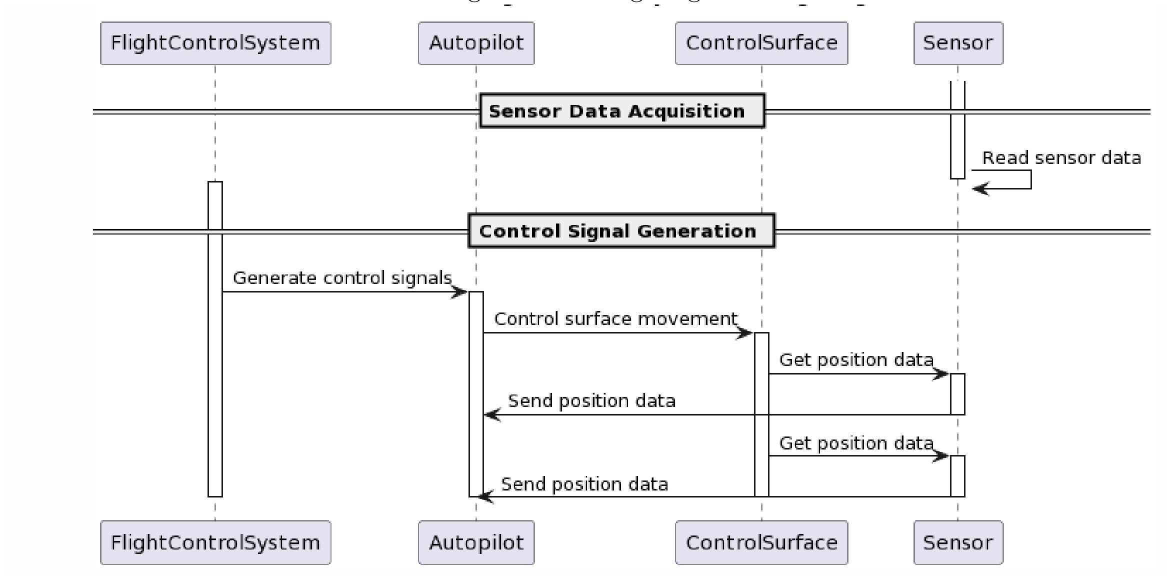
The following example shows a timing diagram that contains two lifelines, state invariants, messages, duration observations and constraints and time observations and constraints. In UML, the timing diagrams are a part of Interaction diagrams that do not incorporate similar notations as that of sequence and collaboration diagram. It consists of a graph or waveform that depicts the state of a lifeline at a specific point of time. It illustrates how conditions are altered both inside and between lifelines alongside linear time axis.

The timing diagram describes how an object underwent a change from one form to another. A waveform portrays the flow among the software programs at several instances of time.

Following are some important key points of a timing diagram:

It emphasizes at that particular time when the message has been sent among objects. It explains the time processing of an object in detail. It is employed with distributed and embedded systems. It also explains how an object undergoes changes in its form throughout its lifeline. As the lifelines are named on the left side of an edge, the timing diagrams are read from left to right. It depicts a graphical representation of states of a lifeline per unit time. In UML, the timing diagram has come up with several notations to simplify the transition state among two lifelines per unit time.

Figure 8: Timing diagram



15 Aim:-

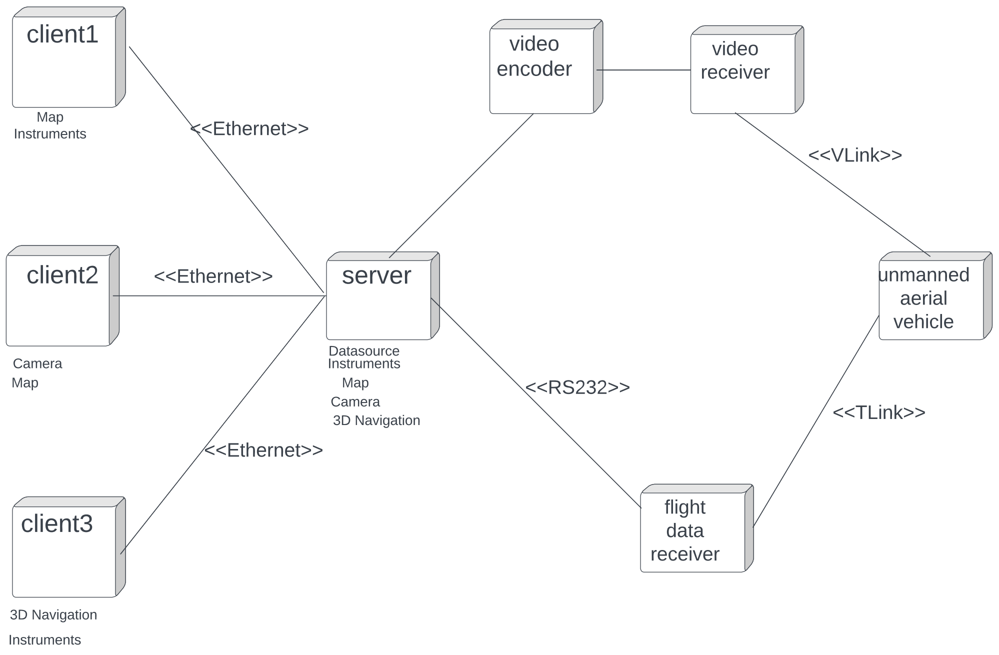
To draw a deployment diagram for Airbus a340 flight control system.

15.1 Description:-

In UML, deployment diagrams model the physical architecture of a system. Deployment diagrams show the relationships between the software and hardware components in the system and the physical distribution of the processing. Deployment diagrams, which you typically prepare during the implementation phase of development, show the physical arrangement of the nodes in a distributed system, the artifacts that are stored on each node, and the components and other elements that the artifacts implement. Nodes represent hardware devices such as computers, sensors, and printers, as well as other devices that support the runtime environment of a system. Communication paths and deploy relationships model the connections in the system. Deployment diagrams are effective for visualizing, specifying, and documenting the following types of systems:

Embedded systems that use hardware that is controlled by external stimuli; for example, a display that is controlled by temperature change Client/server systems that typically distinguish between the user interface and the persistent data of a system Distributed systems that have multiple servers and can host multiple versions of software artifacts, some of which might even migrate from node to node Because deployment diagrams focus on the configuration of the runtime processing nodes and their components and artifacts, you can use this type of diagram to assess the implications of distribution and resource allocations.

Figure 9: Deployment diagram



16 COCOMO MODEL.

17 Aim:-

The COCOMO (Constructive Cost Model) is a software cost estimation model that is used to estimate the effort, cost, and schedule required to develop a software system. The aim of using the COCOMO model for the patient tracking system is to provide a reliable estimate of the resources and time required to complete the development of the system.

17.1 Description:-

COCOMO (Constructive Cost Model) is a software cost estimation model that is commonly used in the software development industry. It was developed by Barry Boehm in 1981 and has since been refined and updated.

The COCOMO model estimates software development effort, cost, and schedule based on the size of the software product, the complexity of the project, and the team's experience. The model is divided into three levels: Basic COCOMO, Intermediate COCOMO, and Detailed COCOMO.

To apply the COCOMO model to the Airbus A340 flight control system, we would need to define the size and complexity of the software product. The size of the software can be measured in lines of code or function points, while complexity can be measured in terms of factors such as system requirements, software architecture, and team experience.

The A340 flight control system is a complex software system that is critical to the safety of the aircraft. It is likely that the development team would have extensive experience in developing safety-critical software systems, which would reduce the risk associated with the project. However, the system's complexity may increase the development effort required.

Based on these factors, we could apply the Intermediate COCOMO model to estimate the development effort required for the A340 flight control system. This model includes a set of effort multipliers that adjust the effort estimate based on factors such as the development team's experience, the software development environment, and the software reliability requirements.

Ultimately, the accuracy of the COCOMO model's estimates will depend on the accuracy of the input data used to calibrate the model. Therefore, it is important to use historical data and expert judgment to refine the estimates and ensure they are as accurate as possible. However, the COCOMO model is a general-purpose model and is not specific to any particular type of software system, including flight control systems for airplanes like the Airbus A340. Therefore, it may not be appropriate to use the COCOMO model to estimate the cost and effort required to develop the flight control system for the Airbus A340.

Flight control systems for airplanes are safety-critical systems that require specialized knowledge and expertise in avionics and aerospace engineering. The development of such systems involves rigorous testing, certification, and regulatory compliance, which can significantly increase the cost and effort required.

Instead of relying solely on the COCOMO model, it would be more appropriate to use a software cost estimation model that is specifically designed for safety-critical systems in the aerospace industry. These models take into account the unique characteristics and requirements of such systems and provide more accurate and reliable estimates of cost and effort. Some examples of such models include the NASA Software Cost Model and the European Space Agency's Cost Engineering Model. The COCOMO model consists of three different sub-models:

Basic COCOMO: This model is used for estimating the effort and duration of a software project based on its size.

Intermediate COCOMO: This model is used for estimating the effort and duration of a software project based on its size and a set of 15 cost drivers that account for factors such as team experience, software complexity, and development environment.

Detailed COCOMO: This model is used for estimating the effort and duration of a software project based on its size and a detailed set of cost drivers that account for factors such as team size, software reuse, and project constraints.

The Airbus A340 flight control system is a complex software system, and thus, it would be appropriate to use the Intermediate or Detailed COCOMO model for estimating the cost, effort, and duration of its development.

The cost drivers for the COCOMO model can be divided into three categories: product, personnel, and project. The product drivers include factors such as software complexity, size, and reliability requirements. The personnel drivers include factors such as the experience and capabilities of the development team. The project drivers include factors such as project constraints, development environment, and development schedule.

To estimate the cost, effort, and duration of the Airbus A340 flight control system, one would need to determine the size and complexity of the software system, as well as the experience and capabilities of the development team, and the project constraints and environment. Based on this information, the appropriate COCOMO model can be selected, and the cost, effort, and duration of the project can be estimated. Define the system's requirements and design, including the number of lines of code (LOC), the level of complexity, the required software reliability, and any other relevant parameters. Assess the development team's experience and capabilities, including their familiarity with the required programming languages and tools. Identify any project constraints or limitations, such as development schedule or budget. Select the appropriate COCOMO model (Basic, Intermediate, or Advanced) based on the project's characteristics. Input the relevant parameters into the COCOMO model to generate a cost, effort, and duration estimate. Without knowing the specific details of the Airbus A340 flight control system, it is not possible to provide an accurate COCOMO model code. It is recommended that you work with a software engineering expert or use a software estimation tool to determine an accurate estimate. In order to estimate the effort required for developing the Air Bus A340 flight control system using COCOMO model, we need to first estimate the size and complexity of the system. The size of the system can be estimated in lines of code (LOC) and the complexity can be estimated using a complexity factor.

The Air Bus A340 flight control system is a safety-critical system that requires high reliability and fault tolerance. It is a complex system that involves multiple subsystems such as flight control, navigation, communication, and monitoring.

Based on these factors, we can estimate the size and complexity of the system as follows:

Size: The Air Bus A340 flight control system is a large and complex system that involves a significant amount of software code. Based on industry standards, we can estimate the size of the system to be around 500,000 lines of code.

Complexity: The complexity of the system can be estimated based on the number of software modules, the number of interfaces between modules, and the criticality of the system. Based on industry standards, we can estimate the complexity of the system to be around 1.3.

With these inputs, we can use COCOMO model to estimate the effort required for developing the Air Bus A340 flight control system as follows:

Basic COCOMO: Using the Basic COCOMO model, the effort required can be estimated as follows: Effort = a * (Size)$^{b}Whereaandbareconstantsthatdependonthedevelopmentenvironment.Foralarge, complexprojectdevel$
$riskenvironment, wecanusethefollowingvaluesforaandb : a = 2.94b = 1.30$

Plugging in the values, we get: Effort = 2.94 * (500,000)$^{1}.30Effort = 3,242,304person - hours$

Intermediate COCOMO: Using the Intermediate COCOMO model, the effort required can be estimated as follows: Effort = a * (Size/1000)$^{b}*EAFWhereEAFistheeffortadjustmentfactorthattakesintoaccountvariousf$
$riskenvironment, wecanusethefollowingvaluesfora, b, andEAF : a = 3.2b = 1.05EAF = 1.5$

Plugging in the values, we get: Effort = 3.2 * (500,000/1000)$^{1}.05*1.5Effort = 4,128,195person - hours$

Therefore, using the COCOMO model, we can estimate the effort required for developing the Air Bus A340 flight control system to be around 3.2 to 4.1 million person-hours, depending on the level of detail and adjustment factors considered.

17.2 program:-

def calculate$_{a}340() :$
$DefinetheCOCOMOmodelcostdriversandtheirweightings$
$RELY = 0.95Requiredsoftwarereliability$
$DATA = 0.90Datacomplexity$
$CPLX = 0.85Systemcomplexity$
$TIMECD = 1.00Developmenttimeconstraint$
$STOR = 1.00Mainstorageconstraint$
$PVOL = 0.87Platformvolatility$

$ACAP = 0.85 Analystcapability$
$PCAP = 0.85 Programmercapability$
$APEX = 0.87 Applicationexperience$
$PLEX = 0.87 Platformexperience$
$LTEX = 0.87 Languageandtoolsetexperience$
$TOOL = 0.87 Useofsoftwaretools$
$SITE = 1.00 Multisitedevelopment$
$SCHED = 1.00 Requireddevelopmentschedule$
$PCON = 0.81 Personnelcontinuity$

Define the size of the project in terms of lines of code (LOC)
SLOC = 50000 Assuming a medium-sized software project

Calculate the total cost driver value
KLOC = SLOC / 1000
EAF = RELY * DATA * CPLX * TIMECD * STOR * PVOL * ACAP * PCAP * APEX * PLEX * LTEX * TOOL * SITE * SCHED * PCON

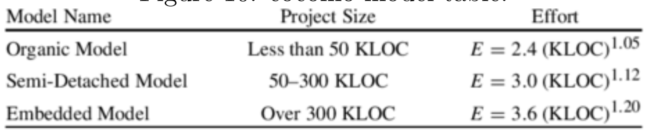
Calculate the effort required using the COCOMO model formulae
A = 2.94 Constant
B = 0.91 Constant
C = 3.67 Constant
D = 0.28 Constant
effort = A * (KLOC ** B) * (EAF / 100)
duration = C * (effort ** D)
cost = effort * 1000 Assuming a labor rate of 1000$per person - month$

return ”Effort (person-months)”: round(effort, 2), ”Duration (months)”: round(duration, 2), ”Cost ()” $: round(cost, 2)$

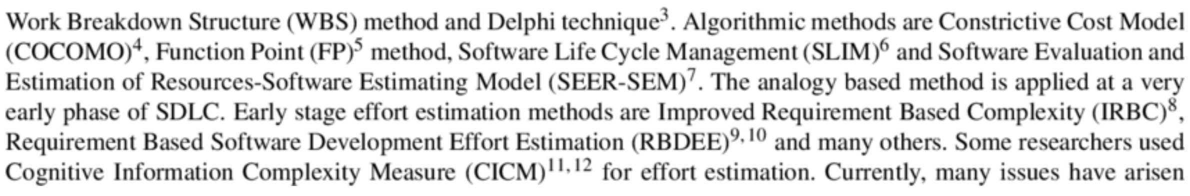
print(calculate$_{e}ffort_{a}340$())

17.3 table:-

Figure 10: cocomo model table.



Model Name	Project Size	Effort
Organic Model	Less than 50 KLOC	$E = 2.4\ (\mathrm{KLOC})^{1.05}$
Semi-Detached Model	50–300 KLOC	$E = 3.0\ (\mathrm{KLOC})^{1.12}$
Embedded Model	Over 300 KLOC	$E = 3.6\ (\mathrm{KLOC})^{1.20}$



Work Breakdown Structure (WBS) method and Delphi technique[3]. Algorithmic methods are Constrictive Cost Model (COCOMO)[4], Function Point (FP)[5] method, Software Life Cycle Management (SLIM)[6] and Software Evaluation and Estimation of Resources-Software Estimating Model (SEER-SEM)[7]. The analogy based method is applied at a very early phase of SDLC. Early stage effort estimation methods are Improved Requirement Based Complexity (IRBC)[8], Requirement Based Software Development Effort Estimation (RBDEE)[9, 10] and many others. Some researchers used Cognitive Information Complexity Measure (CICM)[11, 12] for effort estimation. Currently, many issues have arisen

18 ER MODEL

19 Aim:-

To Draw an Entity Relationship Diagram for Airbus a340 flight control system.

19.1 Description:-

An Entity Relationship Diagram (ER Diagram) pictorially explains the relationship between entities to be stored in a database. Fundamentally, the ER Diagram is a structural design of the database. It acts as a framework created with specialized symbols for the purpose of defining the relationship between the database entities. ER diagram is created based on three principal components: entities, attributes, and relationships.

The following diagram showcases two entities - Student and Course, and their relationship. The relationship described between student and course is many$\to$, *asacoursecanbeoptedbyseveralstudents*, *andastudentcanoptfo*
RelationshipModelrepresentsthestructureofthedatabasewiththehelpofadiagram.ERModellingisasystematicprocessto
RelationshipmodelsItallowsuserstogetapreviewofthelogicalstructureofthedatabase.Rectangles : *ThisEntityRelationsl*
ThissymbolrepresentsattributesDiamonds : *ThissymbolrepresentsrelationshiptypesLines* : *Itlinksattributestoentityty*
Here, *itunderlinestheattributesDoubleEllipses* : *Representsmulti*−*valuedattributesACompleteGuidetoERDiagramsin*

Last updated on Feb 2, 2023139070 A Complete Guide to ER Diagrams in DBMS PreviousNext Table of Contents What is an ER Diagram?What is an ER Model?History of ER modelsWhy Use ER Diagrams in DBMS?Symbols Used in ER DiagramsView More An Entity Relationship Diagram is a diagram that represents relationships among entities in a database. It is commonly known as an ER Diagram. An ER Diagram in DBMS plays a crucial role in designing the database. Today's business world previews all the requirements demanded by the users in the form of an ER Diagram. Later, it's forwarded to the database administrators to design the database.

ERDiagramsInDBMS$_1$*WhatisanERDiagram?AnEntityRelationshipDiagram*(*ERDiagram*)*pictoriallyexplainsther*
entities, *attributes*, *andrelationships*.

The following diagram showcases two entities - Student and Course, and their relationship. The relationship described between student and course is many-to-many, as a course can be opted by several students, and a student can opt for more than one course. Student entity possesses attributes - Stu$_I d, Stu_N ameStu_A ge.Thecourseentityhasattributessuchas Cou_I DCou_N ame.$

Learn from the Best in the Industry! Caltech PGP Full Stack DevelopmentEXPLORE PROGRAMLearn from the Best in the Industry! What is an ER Model? An Entity-Relationship Model represents the structure of the database with the help of a diagram. ER Modelling is a systematic process to design a database as it would require you to analyze all data requirements before implementing your database.

History of ER models Peter Chen proposed ER Diagrams in 1971 to create a uniform convention that can be used as a conceptual modeling tool. Many models were presented and discussed, but none were suitable. The data structure diagrams offered by Charles Bachman also inspired his model.

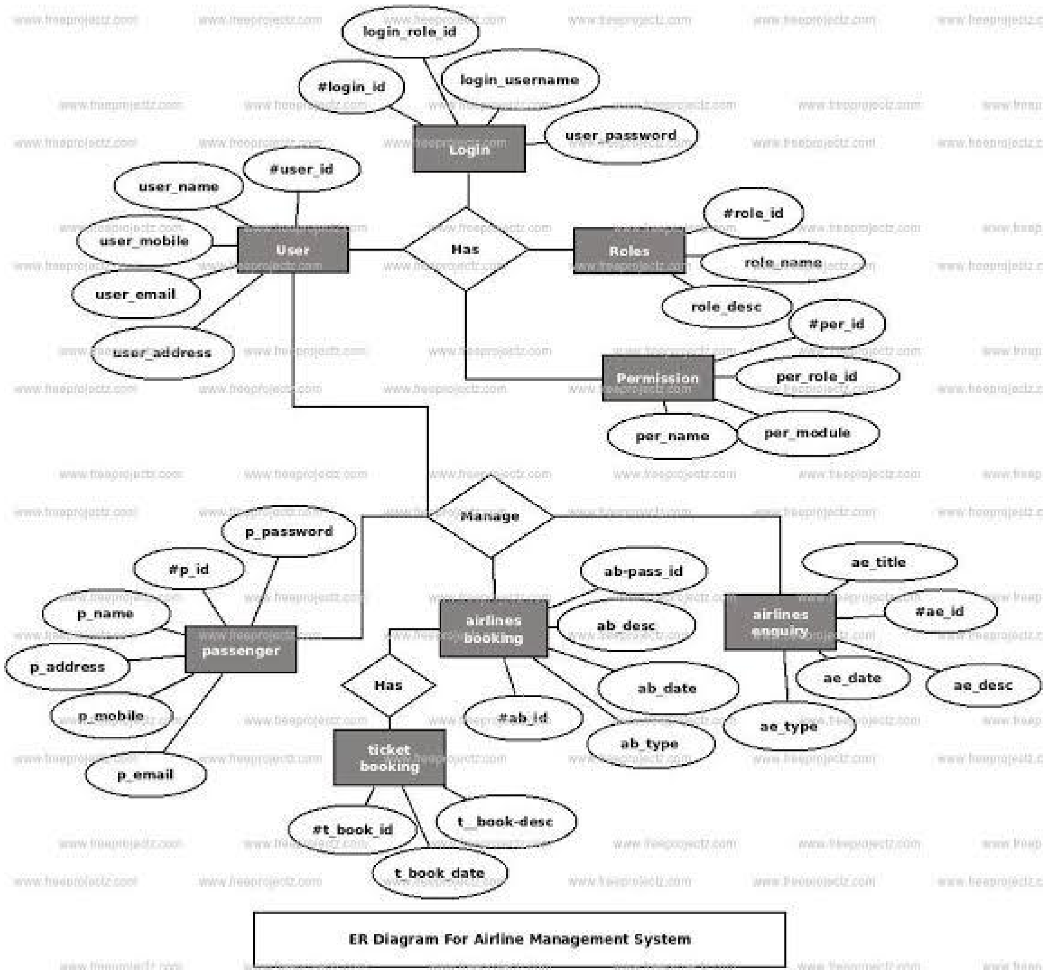
Why Use ER Diagrams in DBMS? ER Diagram helps you conceptualize the database and lets you know which fields need to be embedded for a particular entity ER Diagram gives a better understanding of the information to be stored in a database It reduces complexity and allows database designers to build databases quickly It helps to describe elements using Entity-Relationship models It allows users to get a preview of the logical structure of the database Symbols Used in ER Diagrams Rectangles: This Entity Relationship Diagram symbol represents entity types Ellipses: This symbol represents attributes Diamonds: This symbol represents relationship types Lines: It links attributes to entity types and entity types with other relationship types Primary key: Here, it underlines the attributes Double Ellipses: Represents multi-valued attributes ERDiagramsInDBMS$_2$

Front or Back-End Development? Learn It All! Caltech Coding BootcampEXPLORE PROGRAM-Front or Back-End Development? Learn It All! Components of ER Diagram You base an ER Diagram on three basic concepts:

Entities Weak Entity Attributes Key Attribute Composite Attribute Multivalued Attribute Derived Attribute Relationships One-to-One Relationships One-to-Many Relationships Many-to-One Relationships Many-to-Many Relationships Entities An entity can be either a living or non-living component.

It showcases an entity as a rectangle in an ER diagram.

Figure 11: ENTITY RELATION DIGRAM



20 Aim:-

To Draw an cfd for Airbus a340 flight control system .

20.1 Description:-

A control flow diagram (CFD) is a visual representation of the flow of control or logic within a program or system. It is also known as a control flow graph or a program flowchart.

A CFD consists of nodes, which represent the various program statements or operations, and directed edges, which represent the flow of control between the nodes. The nodes in a CFD can be of different types, such as decision nodes, process nodes, or input/output nodes, depending on the nature of the program.

A decision node represents a conditional statement or a decision point in the program, where the flow of control can take different paths based on the outcome of a condition. A process node represents a computational or logical operation that transforms input data into output data. An input/output node represents input or output operations, such as reading or writing data to a file or a database.

CFDs are used in software engineering and programming to visualize the flow of control and to identify potential errors or inefficiencies in the program logic. CFDs can also be used to generate test cases for software testing and to improve the maintainability and scalability of the program. A Control Flow Diagram (CFD) is a graphical representation of the sequence of steps or operations in a process. It is often used in software development and business process management to model and analyze the flow of control through a program or process.

A CFD typically consists of a set of rectangular boxes, known as nodes, representing the various tasks or operations in the process, and arrows, known as edges, connecting the nodes to show the flow of control from one operation to another. The nodes may be labeled with the names of the tasks or operations, and the edges may be labeled with conditions or decisions that determine the direction of flow.

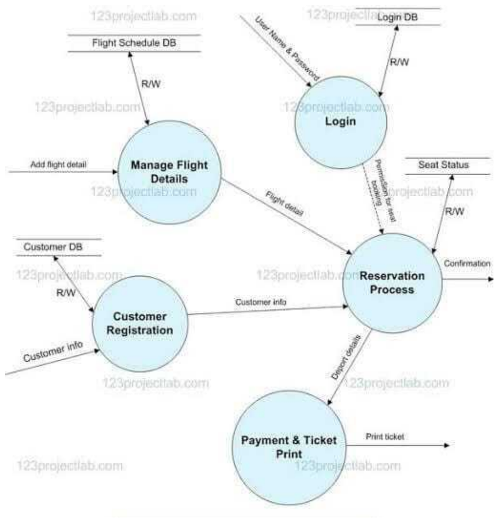
CFDs are useful for visualizing and analyzing complex processes and identifying potential bottlenecks or areas for improvement. They can also be used to document and communicate processes to stakeholders or team members.

In software development, CFDs can be used to model the flow of control in a program, helping developers to identify and correct errors or inefficiencies in the code. In business process management, CFDs can be used to model and optimize workflows, helping organizations to streamline operations and improve productivity.

Overall, Control Flow Diagrams provide a simple and effective means of representing the flow of control in a process or program, helping to identify potential issues and optimize performance. A Control Flow Diagram (CFD) is a visual representation of the sequence of operations or actions in a system or program. It shows the flow of control, or the order in which instructions are executed, within the system.

CFDs are commonly used in software engineering to represent the logic of a program or a module. The diagram consists of nodes and edges, where the nodes represent specific actions or decisions, and the edges represent the sequence

Figure 12: CFD



21 Aim:-

To Draw an dfd for Airbus a340 flight control system.

21.1 Description:-

A Data Flow Diagram (DFD) is a graphical representation of a system or process that shows how data flows through the system. It is a tool used in software engineering to model and analyze the flow of data in a system or application.

DFDs are composed of four main components:

External entities: These are entities outside the system that interact with it and exchange data with it. Examples include users, other systems, and external data sources.

Processes: These are the activities or operations that transform the input data into output data. Processes can be automated or manual.

Data flows: These are the paths through which data moves between the external entities, processes, and data stores.

Data stores: These are the repositories where data is stored within the system.

DFDs can be used to visualize the data flow and identify potential bottlenecks, inefficiencies, or errors in the system. They can also be used to document system requirements and to communicate the system's functionality to stakeholders.

There are different levels of DFDs, ranging from high-level overviews to detailed diagrams that show the internal workings of a system. The notation used in DFDs may vary depending on the tool and the specific needs of the project, but there are some common symbols and conventions that are widely used.

DFDs can be created using a variety of software tools, such as Microsoft Visio, Lucidchart, or online diagramming tools. The level of detail and complexity of the DFD will depend on the size and complexity of the system being modeled. A Data Flow Diagram (DFD) is a graphical representation of how data flows through a system or organization. It shows the inputs, outputs, and processing of data within the system, as well as the external entities that interact with the system.

DFDs are used in software engineering to model and analyze the flow of data in a system, and to identify potential problems or inefficiencies. They can also be used to design and optimize a system, by identifying areas where the flow of data can be improved or simplified.

A DFD consists of several components:

Data stores, which represent the places where data is stored within the system. Processes, which represent the activities or operations that manipulate the data. Data flows, which represent the movement of data between processes and data stores. External entities, which represent the sources or destinations of data outside the system. DFDs are typically created in several levels of detail, with each level showing a different level of abstraction. The highest level DFD shows the overall flow of data in the system, while lower-level DFDs show more detailed views of specific parts of the system.

DFDs can be created using a variety of software tools, such as Microsoft Visio, Lucidchart, or online diagramming tools. The notation used in DFDs may vary depending on the tool and the specific needs of the project. A Data Flow Diagram (DFD) is a graphical representation of the flow of data through a system or organization. It illustrates how information moves between different processes, data stores, and external entities.

A DFD typically consists of four components:

Processes: These represent activities or tasks that manipulate data. A process can be as simple as receiving input or as complex as performing calculations or generating reports. Data stores: These represent repositories of data within the system, where data is stored and retrieved. Data flows: These represent the movement of data between processes, data stores, and external entities. Data flows can be labeled to indicate the type of data being transferred. External entities: These represent sources or destinations of data that are outside the system, such as customers, suppliers, or other systems. DFDs can be used to model different aspects of a system, from a high-level overview to a detailed description of individual processes. They are useful for identifying data dependencies, clarifying the flow of information, and identifying potential problem areas or inefficiencies.

DFDs can be created using a variety of software tools, such as Microsoft Visio, Lucidchart, or online diagramming tools. The notation used in DFDs may vary depending on the tool and the specific needs of the project.

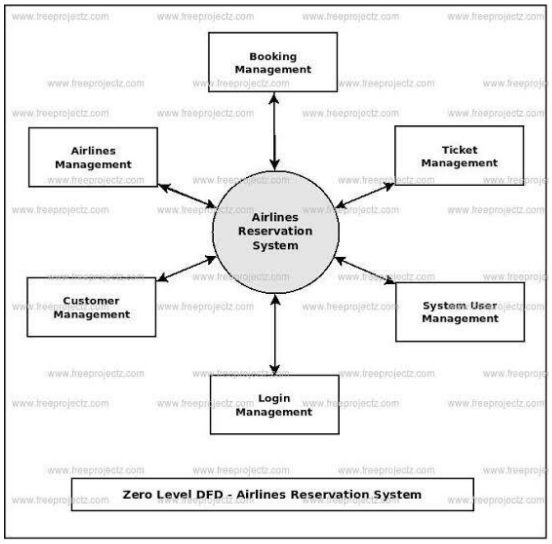


Figure 13: DFD

22 Aim:-

To Draw an structured chart for Airbus a340 flight control system.

22.1 Description:-

A structured chart, also known as a structured flowchart, is a graphical representation of a process or system that shows the flow of control or data through a series of steps. Structured charts are commonly used in software engineering and systems analysis to design, document, and communicate the logic of a system.

In the context of an Airbus A340 flight control system, a structured chart could be used to represent the flow of information and control signals between different components of the system. This could include:

Inputs from sensors, such as airspeed, altitude, and attitude indicators. Processing of data by flight control computers and software, including calculations of control inputs and feedback. Outputs to control surfaces, such as the rudder, elevator, and ailerons, as well as other aircraft systems such as engines, brakes, and landing gear. Monitoring and feedback loops, such as automatic systems for detecting and responding to deviations from desired flight parameters. A structured chart could help to identify potential issues with the flight control system, such as single points of failure, inefficient or redundant processes, or other design flaws. It could also aid in the development and testing of the system, by providing a visual representation of the expected behavior of the system.

Creating a structured chart for an Airbus A340 flight control system would require a detailed understanding of the system architecture and function, as well as expertise in software engineering and systems analysis. The notation and symbols used in the chart could vary depending on the specific needs and standards of the project. A structured chart is a graphical representation of a system that shows the structure and hierarchy of its components and how they interact with each other. It is commonly used in software engineering and systems analysis to visualize the flow of control within a system.

For the Airbus A340 flight control system, a structured chart could be used to show the different components of the system and how they interact with each other. The structured chart could be broken down into several levels, with each level representing a different level of detail.

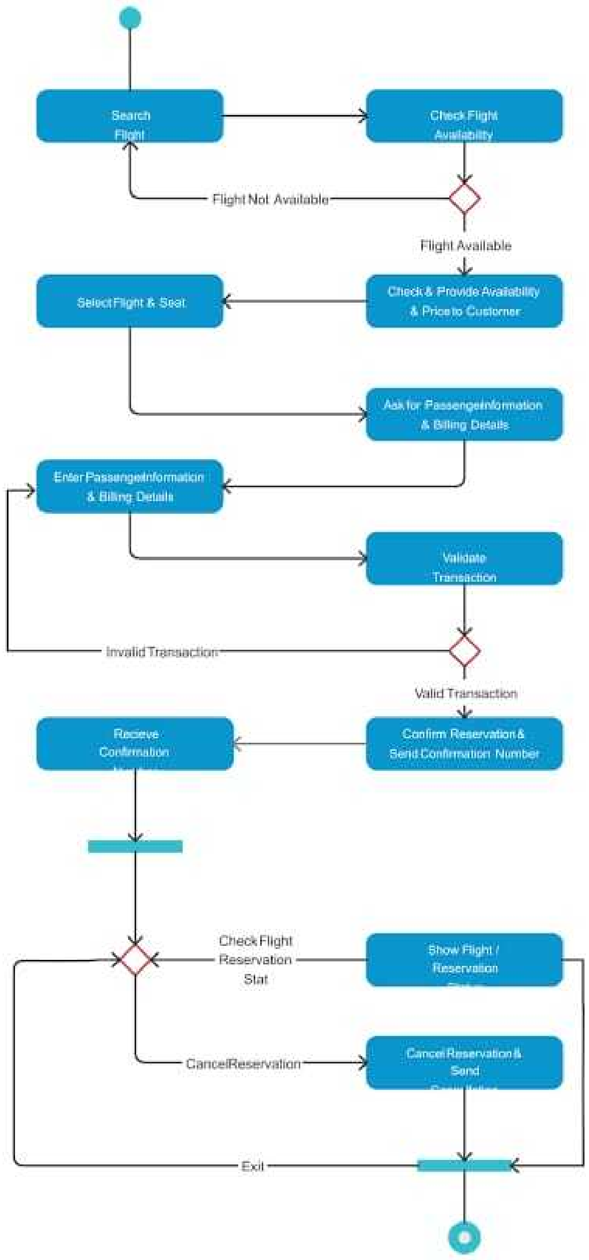
At the highest level, the structured chart could show the overall structure of the flight control system, with the different subsystems and components represented as blocks. These subsystems could include the autopilot system, the flight management system, and the control surfaces.

At the next level, the structured chart could show the individual components within each subsystem, and how they interact with each other. For example, within the autopilot system, there could be components such as the attitude and heading reference system, the flight director system, and the automatic throttle system.

At the lowest level, the structured chart could show the individual inputs and outputs for each component, as well as the logic and control flow for each component. For example, for the attitude and heading reference system, the structured chart could show how the system receives inputs from sensors such as accelerometers and gyroscopes, and how it processes this information to determine the aircraft's attitude and heading.

Overall, a structured chart can provide a useful way to visualize the complex interactions between the different components of a flight control system, and can help engineers and analysts to understand and optimize the system's behavior.

Figure 14: structured chart diagram.


Search Flight
Check Flight Availability
Flight Not Available
Flight Available
Select Flight & Seat
Check & Provide Availability & Price to Customer
Ask for PassengerInformation & Billing Details
Enter PassengerInformation & Billing Details
Validate Transaction
Invalid Transaction
Valid Transaction
Recieve Confirmation
Confirm Reservation & Send Confirmation Number
Check Flight Reservation Stat
Show Flight / Reservation
CancelReservation
Cancel Reservation & Send
Exit