ExamPrepMDSE1

maandag 30 januari 2023 16:14

1.What is a complex system and how it can be developed, tested, and validated?
2. What are the scope of System Engineering and its relationship to traditional engineering?

Complex system is the idea of an integrated system which is not only about "one" variable. It interacts with multiple perspective to achieve what it is intended. Because it is more a little different than a normal system, we would tackle it with three main paradiam,

Communication: tackles with sysml.
Complexity: Model driven engineering. MDSE: concerns with the bigger picture.
Lack of understanding: tackle with creativety

The main difference between system engineering to regular engineering is the holisticness approach. Both involve problem solving in the sense of problem and solution. Yet the view in the system is more about the whole. It includes also technical project manager.

communication : CAN
complexity.
Lack of understanding
passenger centric ⟷ vehicle centric

3. What is Automotive Software Engineering and what is the new software operating model for OEMs?
4. Which are the most important Automotive Standards? Give a short description of the specific standard.

The software is the main inovator of the current industry. The amount of software has increased drastically.

vehicle centric functional domain → passenger centric functional domain.

Automotive systems can be categorized into

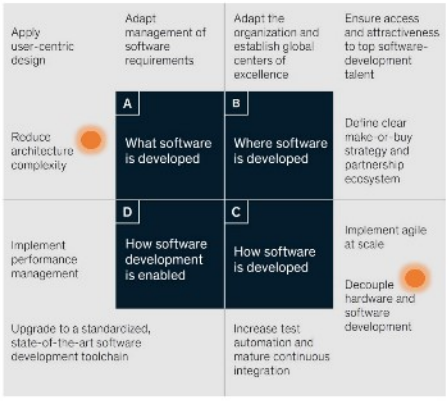
Vehicle-centric functional domains	Passenger-centric functional domains covering
• Powertrain control • Chassis control • Active/passive Safety Systems	• Multimedia/telematics, body/-comfort • Human-Machine Interface

Shared mobility, electrification, connectivety and autonomous vehicles are the most prominant development.
Automotive Software Engineering is the field of engineering that deals with the design, development, and maintenance of software systems for vehicle. The new software operating model for OEMs (Original Equipment Manufacturers) is focused on creating a more agile, efficient, and flexible software development process that enables rapid response to changing market conditions and customer needs.

connectivety, Shared mobility electrification are the most prominent.
agile design: = enable rapid technology

The new fotware operating model for OEM: is the four critical dimentions.

1. What is software development: architecture design requirements.
2. Where is the software developed within the organisation. : location talent partnerships involved.
3. How is software developed: development methadologies- agile at scle changes in development testing processes.
4. How is software fevelopment enables: performance managment: toolchain infrastructure.



SOFL

- Standards
- Safety
- cybersecurity
- Functionality

Automotive standard:

1. ISO 26262 - Automotive Safety: This is an international standard for the functional safety of electrical and/or electronic systems in production vehicles. It provides guidelines for identifying and mitigating potential safety hazards, and includes requirements for the development process, including risk assessment, testing, and validation.
2. ISO/PAS 21448 [1] - Functionality: This standard provides guidelines for the functional safety of advanced driver-assistance systems (ADAS) and automated driving systems (ADS). It covers functional safety requirements for the design, development, production, and maintenance of these systems. void every hazard associated with both the intended functionality and any unintended functionality resulting from vehicle use in the real world
3. ISO 21434 - This standard provides guidelines for the cybersecurity of road vehicles. It covers the protection of vehicles and their systems against unauthorized access, manipulation, or disruption. It also provides guidelines for secure communication, software updates, and cyber resilience.
4. ISO/SAR 21434 - Cybersecurity: This is a standard for the cybersecurity of road vehicles. It provides guidelines for the protection of vehicles and their systems against unauthorized access, manipulation, or disruption. It also provides guidelines for secure communication, software updates, and cyber resilience.

Integrating these standards into an "Integrated Automotive Safety & Security Standard" would involve aligning the various guidelines and requirements from these standards to create a comprehensive and cohesive framework for ensuring the safety and security of automotive systems. This would help ensure that the design, development, and production of vehicles are done with both safety and security in mind, and that the systems and components used meet the highest standards for performance and reliability.

One is is security and safety (21434. 26262) could be seen as requirements before the implementation where as safety and secturity (26262, 21434) are majority to to validate the syste,.
Moreover: we have the Autosar: which is focuses on implementation and realization of automotive systems. It helps to simplify the model sharing among different developments teams and lowers the average initial learning effort.
One of the biggest problems faced by the Automotive Companies was that the software must often be rewritten from scratch when hardware is changed. AUTOSAR was primarily formed to address this challenge.

Mirsa provides a coding standard in c to develop an automotive system.

Problem: Very costly!

functional requirement, non functional requirement,

What is Software Development Process and which are its phases? Describe in short each phase.
6. What is a requirement and why Requirements Engineering is important?
7. How do we get good and right requirements?
8. How many requirements types are in Automotive Software? Give a short description per type.
9. Describe in short the properties of a requirement.

Requirements engineer is important as it forces the engineer to think on the problem in a more holistic and iterative perspective, it would make sure that the specification required is met both in the context of what, and how.

How do we get right requirement.

Start general
slowly break them down and research the components you found, make them clear and atomic

The Software Development Process is a systematic approach to creating software systems. It involves a series of phases that ensure that the software is developed in a controlled, repeatable, and efficient manner. The phases of the software development process typically include:

1. Requirements Gathering: This phase involves identifying and documenting the requirements for the software system. This includes defining what the system should do, and what constraints and limitations it must adhere to.
2. Analysis and Design: In this phase, the requirements are analyzed, and a high-level design of the system is created. This includes defining the architecture, data structures, algorithms, and interfaces.
3. Implementation: During this phase, the code is written and the system is built. This involves translating the design into code and integrating the various components of the system.
4. Testing: In this phase, the system is thoroughly tested to ensure that it meets the requirements and functions as expected. This includes both functional testing (to ensure the system does what it's supposed to do) and non-functional testing (to ensure that the system meets performance, reliability, and other requirements).
5. Deployment: Once the system has been tested and any issues have been resolved, it can be deployed into the production environment. This may involve installation on the target platform, configuring the system, and setting up any required databases or other infrastructure.
6. Maintenance: This phase involves ongoing support and improvement of the system. This includes fixing bugs, adding new features, and making other changes to improve the system's performance and reliability.

Requirement:
Result from one another. It ensure changes of the environment. And more importantly, integrate them into the model in an efficient way.

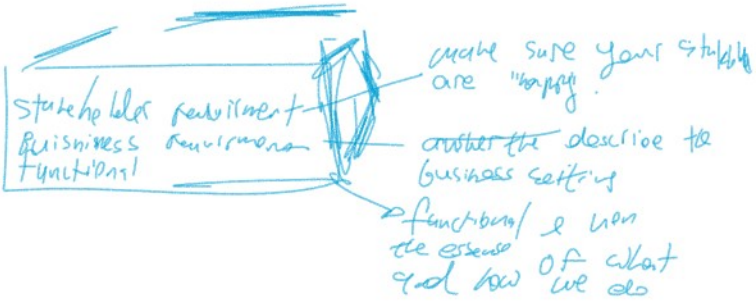
It must be clear and entail the key concept. Requirement engineering, is the dicipline that enables the understanding of the requirement. The key point needs to be specify by the requirement.

Why important:
1. All systems approaches identify a need to understand the requirements properly.
2. Fitness to purpuse
3. The project should be tackle.
THIS RESULT IN A CONCISE DEFINITION OF THE SYSTEM

The benefit:
Increase the possibility to optimal decision.
Every decision can be traced back to the source.
Can be verified easier
Quality assurance.

The requirements should be independent of any specific solution
Requirements are 'constraints' because they will limit, in some way, how that requirement may be realized.

How to capture?
Start somewhere: produce an initial step.
State specific individuals requirement. Specification
Analysis: understanding: the nature of understanding. How and why? Ask yourself always.
Organise: grouping into categories. Embracing more context.
Managments: model them, and must be properly controlled.



se User-Centric System Design - UCSD methods to develop systems, products, or services that conform to User capabilities and limitations, not vice versa, and enable them to accomplish Mission outcomes!

What we try: to understand the model operational needs. Who are the key stakeholder.
We can do it with stories, or enable users to give their input.

Collect
QFD analyse and derive true operational needs.
Use ON to identify the capabilities
Derive specification requirements for the system
Identify the essential requirements

REquirementL system definition that is either needed or wanted.
Functional- What
Non functional How

Requirements often evolve. But the original are also important (the foundation).
Should be clear
Should not be specific.
Who needs this requiements
What needs this requirement.
Should be verified and validated.
Priorotized: what is more important.

Stakeholder requirements
client

Business Requirements:
High-level requirements
Encapsulates, in a concise manner, one single requirement that describes what the business is setting out to do.

Functional requirements:
What the system supposed to do.

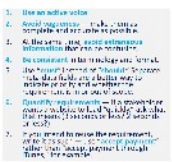
We need to analyze the relation between the stakeholder to non functional requirements.

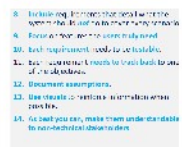
It's a verb: 'do' , 'provide' build

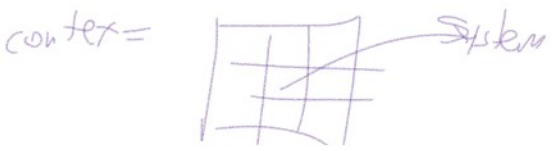
Context modeling:
Identify the boundary of the system
Separate outside world with the inside world of the system
9 boxes.
The highest level of system requirement.

Strategy,
Market,
Mission

How to write:







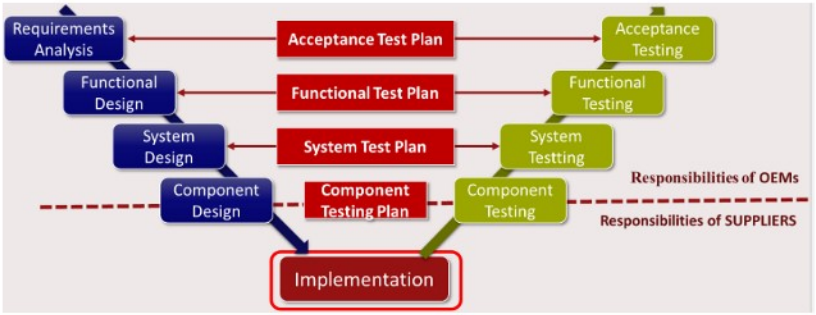
It is important to be able to differentiate between the different types of requirement
Look through a list of generic requirements asking which of the three broad types of requirements classifies each one

Non functional requirements:
Define what the system is supposed to do
A system can still work if NFR are still met. But might not exceed the expected expectation.
How do we do it

It essentially defines the performance attribute of the software system, Non functional is an attribute whereas functional requirement is a verb.

Example: a particular standard must be met, comply with specific legislation, or must be utilized by a software ... (sysml)

1. What are the architectural principles? Describe them in short.
2. Which are the Architectural Views used in Software Automotive? Describe them in short.
3. What are the Automotive Software Architecture Styles? Describe at least three styles and where are they used in the car.
4. How can the Automotive Software Architecture Styles be modeled?



Requirement analysis: broken down into small pieces: What should be implemented
Functional design: High level decision about the allocation of functionality to the logical part of the system.

System Design: The software architects describe the high level design of the software. Allocate them to computational nodes ECU.

Component design: Should be designed in detailed.

Implementation:
The design for each component is implemented in programming languages relevant for the design

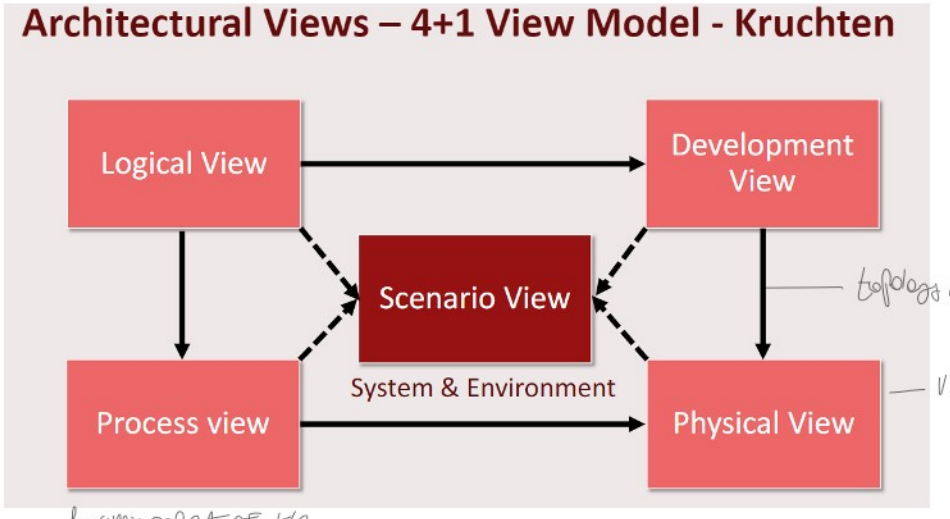
ASA: architecture software automotive:

Software is packed into subsystem based on their logic -> ECU -> design of computer system or platform -> Functions.

An architectural view is a representation of one or more structural aspects of an architecture that illustrates how the architecture addresses one or more concerns held by one or more of its stakeholders

Architecture Principles
S – Single - Responsibility Principle: should have only one responsibility
O – Open - Closed Principle : open for extend without modifying.
L - Liskov Substitution Principle: Derivation
I - Interface Segregation Principle: do not depend on the functionality they use. We can decouple with uneeded responsibilities.
D - Dependency Inversion Principle: high level module should not depend on low level ones.

view



View:

Logical: sysML
Physical/Dev: topology of software components
Functional: Dynammic aspects of the system
Software of the car

dynamic aspect of the System

4+1 is a view model used for "describing the architecture of software-intensive systems, based on the use of multiple, concurrent views. The views are used to describe the system from the viewpoint of different stakeholders, such as end-users, developers, system engineers, and project managers. The four views of the model are logical, development, process and physical view. In addition, selected use cases or scenarios are used to illustrate the architecture serving as the 'plus one' view. Hence, the model contains 4+1 views:

Logical view: The logical view is concerned with the functionality that the system provides to end-users. SysML diagrams are used to represent the logical view, and include block definition diagrams, and state diagrams.
Process view: The process view deals with the dynamic aspects of the system, explains the system processes and how they communicate, and focuses on the run time behavior of the system. The process view addresses concurrency, distribution, integrator, performance, and scalability, etc. UML diagrams to represent process view include the sequence diagram, activity diagram.
Development view: The development view illustrates a system from a programmer's perspective and is concerned with software management. This view is also known as the implementation view. UML Diagrams used to represent the development view include the Package diagram and the Component diagram.
Physical view: The physical view (aka the deployment view) depicts the system from a system engineer's point of view. It is concerned with the topology of software components on the physical layer as well as the physical connections between these components. SysML diagrams used to represent the physical view include the BDD.
Scenarios: The description of an architecture is illustrated using a small set of use cases, or scenarios, which become a fifth view. The scenarios describe sequences of interactions between objects and between processes. They are used to identify architectural elements and to illustrate and validate the architecture design. They also serve as a starting point for tests of an architecture prototype. This view is also known as the use case view.
The 4+1 view model is generic and is not restricted to any notation, tool or design method. Quoting Kruchten,
The "4+1" view model is rather "generic": other notations and tools can be used, other design methods can be used,
especially for the logical and process decompositions, but we have indicated the ones we have used with success.
— Philippe Kruchten, Architectural Blueprints—The "4+1" View Model of Software Architecture

Architecture Styles:

Flexibility its important: all components can be interchangable.

Monolithic: Known as safe mechanisms: static variables, no memory management no dynamic structures. System is a large component. For safety is critical -> components in real time has smaller overhead.

On the other hand, micro kernel is a modular system that has this overhead.

A monolithic kernel is a single large block of code that provides all the essential services for the operating system to function. In contrast, a microkernel has a much smaller codebase and provides only the basic services, delegating other functions to separate processes running in user space. This results in a more modular and scalable system, but can also increase complexity and overhead.

Client server: decouple between client and server provider. Publisher subscriber style for event friven style. It is used in Telemetry
Publisher subscriber : loose coupling. Publisher does not know the subscriber. This is good for distributed system. Disadvantage: lack of control and non synchronous update.

Event driven application: Listeners: react to incoming data. In automotive, driver assistant, sensors and actuators.
Middleware: Cobra: is a broker which mediates the usage of resources between components. This is the autosar standard.

Service oriented: loose coupling between components via internet based protocol.

Describting architectures: SYSML

Testing And integration:

We always compile: to ensure a change is successfully deployed on the target ECU.
Configuration: Configure parameters without modifying the internal structure.

Compilation is the process of transforming source code written in a programming language into machine-readable code. During the compilation process, the source code is transformed into an executable file or a library that can be run on a computer. This process checks the syntax and semantics of the code and generates error messages if any issues are found.
Configuration refers to the process of adjusting the software to meet specific requirements and preferences. Configuration involves the selection and configuration of the software components that are needed to build a specific system. This process is typically performed by setting specific values for parameters and options, choosing specific features and functionalities, and customizing the software to meet specific requirements.

Configuration: tested via multiple scenarios.
Run time variability: we can change the software after we compile.
When two different configuration are used, the software is compiled. (once)

For integration:

Integration - first stage integration- later phase integration
Implementation- desktop in loop, model in loop : we check the logic. HIL: check the actual hardware.

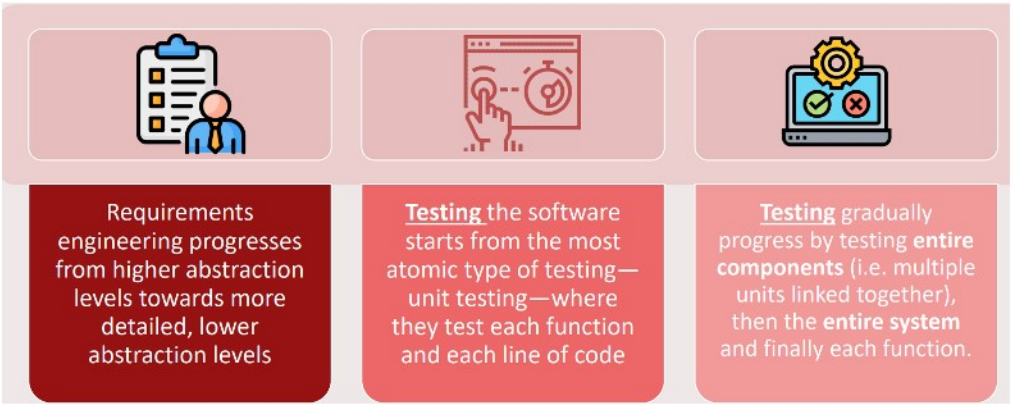
This can be seem
Software implementation/ integration
Software-hardware implementation/ integration
Hardware implementation/integration

Software Integration	Software-Hardware Integration	Hardware Integration
Two (or more) software components are put together and their joint functionality is tested. Integration depend on the what is integrated: 1. *Merging of the source code if the integration is on the source code level* 2. *Linking of two binary code bases together* 3. *Parallel execution to test interoperability.*	**The software is integrated (deployed) to the target hardware platform.** The focus is on the ability of the complete ECU to be executed and the main testing type is **component testing**.	**The focus is on the integration of the ECUs with the electrical system.** The focus is on the interoperability of the nodes and basic functionality, such as communication. The testing related to this type of integration is **system testing**.

Test Driven Development:
Try to break the system. Creating test case and using it to test the pipeline.
But its hard: why: you have infinite test to make but needs only one to fail. Therefore we need to run a lot of tests.

We keep track on the amount of software code we checked.

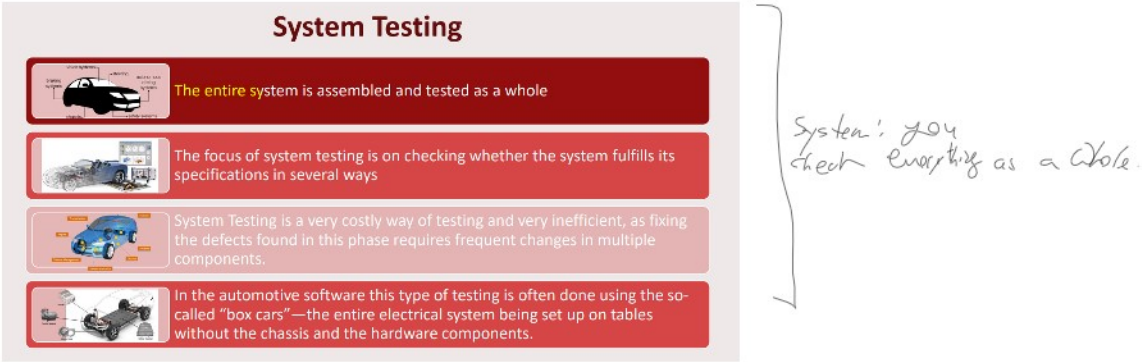
Logic test:
Unit - integration-system



Testing pahses in automotove

Accepteance test, functional test, system test, component test -> only then we implement.
Component test: to verify that the structure and behaviour are implemented corr4cty. The focus is on the interaction components.

We simulate the environment



What we check in testing?
If the system complies to the original requirement and specification.

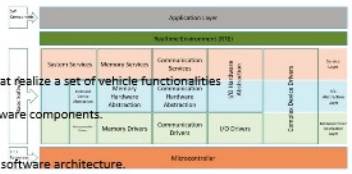
Construction database

Contain all the design information of the electrical system. It is controlled differently in different versions of car. Each of the requirement set has a set of requirements linked to it.

What are the tools? In Rhapsory you would find some.

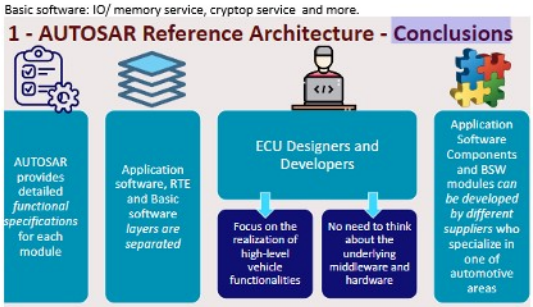
Autosar:
The main objective for it is the break the dependency between the developed software to the used hardware.
Therefore it is proposed to break it down into two components.
We don't want that a small change in the hardware would require a total change of the

Application layer:
Port and software ports.
- Reference architecture:
  - Build upon the ECU hardware.
- Application software: software components that realize a set of vehicle functionalities be exchanging data using interface.
- RTE: Control the communication between software components.
- Basic software



Real time : communication channel: data buffer: the software architecture.

Basic software: IO/ memory service, cryptop service and more.



The middleware:
refers to a software layer that acts as an intermediary between the application layer and the underlying hardware. It provides functionalities such as communication, data management, and functional safety to the applications, abstracting the complexity of the hardware.

Logical and physical architectural views of automotive software systems and their ECUs are mostly done following
the MDA (Model-Driven Architecture).

Physical system architecture and the physical ECU architecture are described by means of architectural models.
Reference architecture:
OEMs, Original Equipment Manufacturers are commonly responsible for the logical and physical design of the system.
A hierarchy of suppliers is responsible for the physical design of specific ECUs, implementation of their application and middleware
software and the necessary hardware.

Once again, autosar meant to connect and interconnect all components and make the system overall more adaptable
Overall we want to standardize the components:

1. ECU reference: make them more resusable.
2. Develop methadology: enable collaboration.
3. Language: enables smooth exchange.
4. ECU middleware. We can divide the software and hardware.

AUTOSAR (Automotive Open System Architecture) is a standard for the development of automotive software systems. It is a collaborative partnership between car manufacturers, suppliers, and tool developers to create a common, open, and standardized architecture for automotive software systems.
The architecture of AUTOSAR is divided into four layers:

1. Application Layer: It contains the high-level functions and services required for the specific application.
2. Runtime Environment Layer: It contains the basic software components necessary for the operation of the application software. This layer includes the operating system and middleware components.
3. Platform Layer: It defines the hardware-specific interfaces and contains the components required for communication with the underlying hardware.
4. Hardware Layer: It represents the physical components and devices of the electronic control unit (ECU).

AUTOSAR aims to provide a standardized and reusable software architecture that can be used across different vehicle platforms, reducing the cost and effort of software development while ensuring compatibility and interoperability between ECUs.

1. A complex system can be defined as a system made up of multiple interrelated components, working together to achieve a common goal. It can be developed using a systematic approach, including requirements gathering, design, implementation, testing, and validation. Good testing and validation practices are essential to ensuring the reliability and safety of a complex system.
2. System engineering is a multidisciplinary field that deals with the design and development of complex systems. It encompasses a wide range of disciplines, including electrical, mechanical, and software engineering, and is concerned with the overall management of a system project. It is related to traditional engineering in that it involves the application of engineering principles to the design and development of a system, but is broader in scope, as it encompasses the entire lifecycle of a system, from concept to retirement.
3. Automotive software engineering is a subfield of software engineering that deals with the design and development of software for use in the automotive industry. The new software operating model for OEMs (original equipment manufacturers) emphasizes collaboration and partnership between OEMs and suppliers, as well as a focus on software development best practices and safety-critical systems.
4. The most important automotive standards include: ISO 26262 (functional safety for road vehicles), MISRA C (guidelines for the use of the C programming language in safety-critical systems), and ASPICE (process assessment for the automotive industry).
5. The software development process typically consists of the following phases: requirements gathering, design, implementation, testing, and maintenance. In each phase, different activities are performed to ensure that the software meets the needs of the stakeholders and is of high quality.
6. A requirement is a statement of a desired feature or behavior of a system. Requirements engineering is important because it helps to ensure that the system is designed and developed to meet the needs of the stakeholders.
7. Good and right requirements can be obtained through effective communication with stakeholders, a thorough understanding of the system and its context, and the use of best practices in requirements engineering.
8. There are several types of requirements in automotive software, including functional requirements, performance requirements, and safety requirements. Each type specifies a different aspect of the system's behavior.
9. Requirements have several properties, including: traceability, uniqueness, completeness, consistency, and testability.
10. Context modeling is a technique for representing the relationships and dependencies between a system and its environment. It can be modeled using a variety of techniques, including UML diagrams and use case models.
11. Architectural principles are high-level guidelines for the design and development of a system's architecture. Examples include: separation of concerns, modularity, and maintainability.
12. The architectural views used in automotive software include: functionality, performance, and physical views. Each view provides a different perspective on the system and its components.
13. Automotive software architecture styles include: layered, event-driven, and component-based. Each style is used in different parts of the car and is chosen based on the specific requirements of the system.
14. Automotive software architecture styles can be modeled using UML diagrams, such as class diagrams and sequence diagrams.
15. Automotive software variant management is the process of managing the different variations of a software product that may be needed for different customers or markets. It can be implemented using a variety of techniques, including branching and merging in version control systems.
16. The integration stages of automotive software development include: integration of individual components, integration of subsystems, and integration of the entire system. Each stage involves testing and verification to ensure that the components are working together correctly.