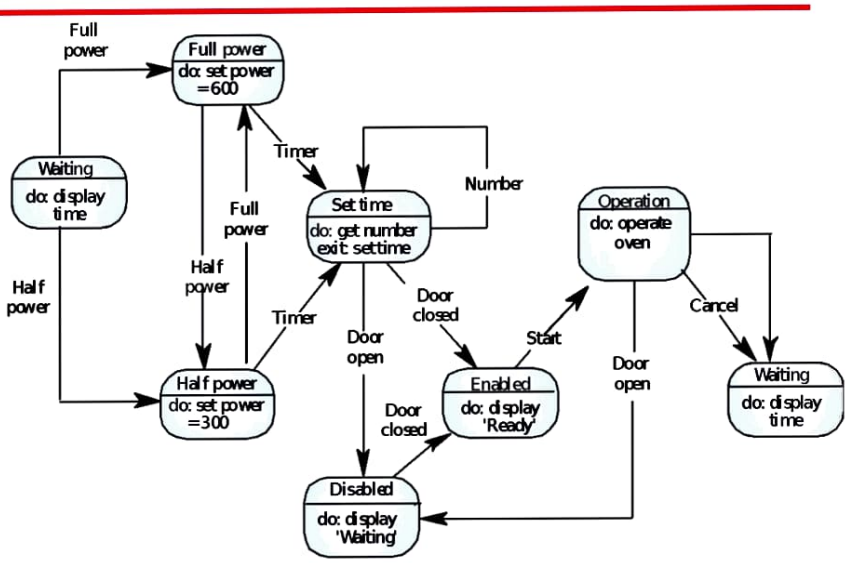
State machine models

- These model the behaviour of the system in response to external and internal events
- They show the system's responses to stimuli so are often used for modelling real-time systems
- State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another
- Statecharts are an integral part of the UML

Microwave oven model



Microwave oven state description

State	Description
Waiting	The oven is waiting for input. The display shows the current time.
Half power	The oven power is set to 300 watts. The display shows 'Half power'.
Full power	The oven power is set to 600 watts. The display shows 'Full power'.
Set time	The cooking time is set to the user's input value. The display shows the cooking time selected and is updated as the time is set.
Disabled	Oven operation is disabled for safety. Interior oven light is on. Display shows 'Not ready'.
Enabled	Oven operation is enabled. Interior oven light is off. Display shows 'Ready to cook'.
Operation	Oven in operation. Interior oven light is on. Display shows the timer countdown. On completion of cooking, the buzzer is sounded for 5 seconds. Oven light is on. Display shows 'Cooking complete' while buzzer is sounding.

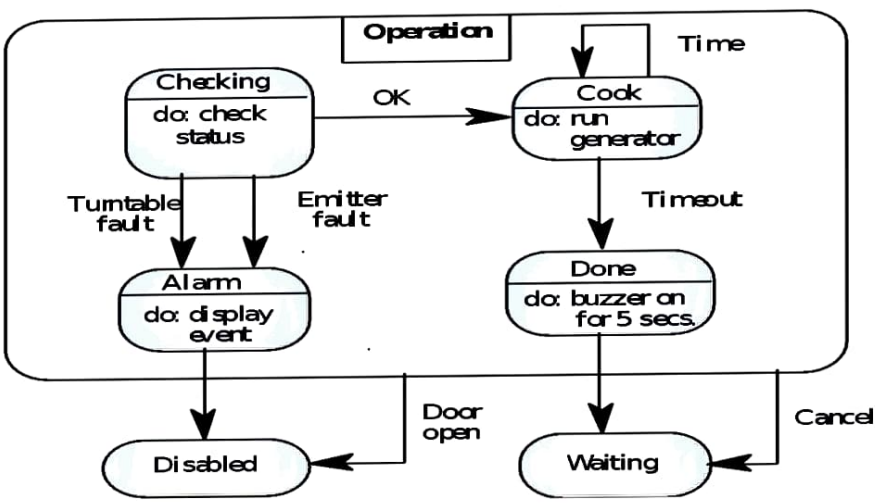
Microwave oven stimuli

Stimulus	Description
Half power	The user has pressed the half power button
Full power	The user has pressed the full power button
Timer	The user has pressed one of the timer buttons
Number	The user has pressed a numeric key
Door open	The oven door switch is not closed
Door closed	The oven door switch is closed
Start	The user has pressed the start button
Cancel	The user has pressed the cancel button

Statecharts

- Allow the decomposition of a model into sub-models (see following slide)
- A brief description of the actions is included following the 'do' in each state
- Can be complemented by tables describing the states and the stimuli

Microwave oven operation



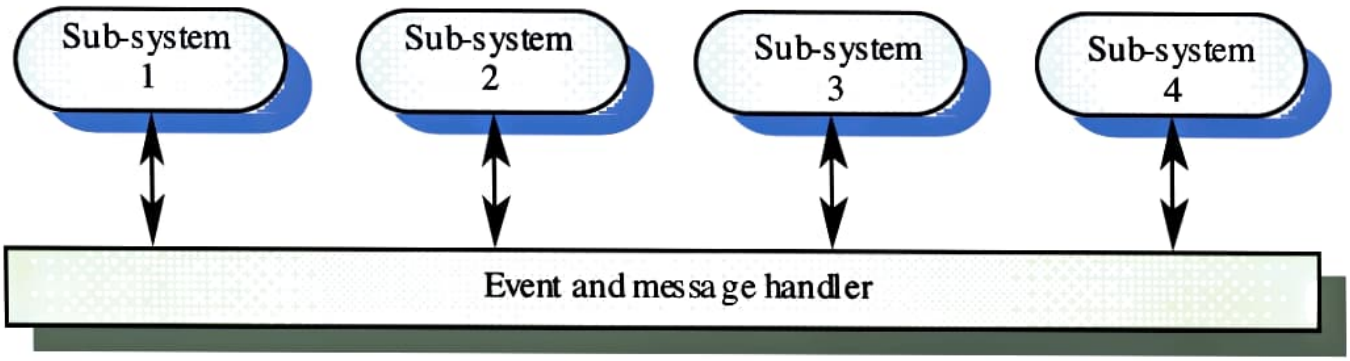
Event-driven systems

- Driven by externally generated events where the timing of the event is outwith the control of the sub-systems which process the event
- Two principal event-driven models
  - Broadcast models. An event is broadcast to all sub-systems. Any sub-system which can handle the event may do so
  - Interrupt-driven models. Used in real-time systems where interrupts are detected by an interrupt handler and passed to some other component for processing
- Other event driven models include spreadsheets and production systems

Broadcast model

- Effective in integrating sub-systems on different computers in a network
- Sub-systems register an interest in specific events. When these occur, control is transferred to the sub-system which can handle the event
- Control policy is not embedded in the event and message handler. Sub-systems decide on events of interest to them
- However, sub-systems don't know if or when an event will be handled

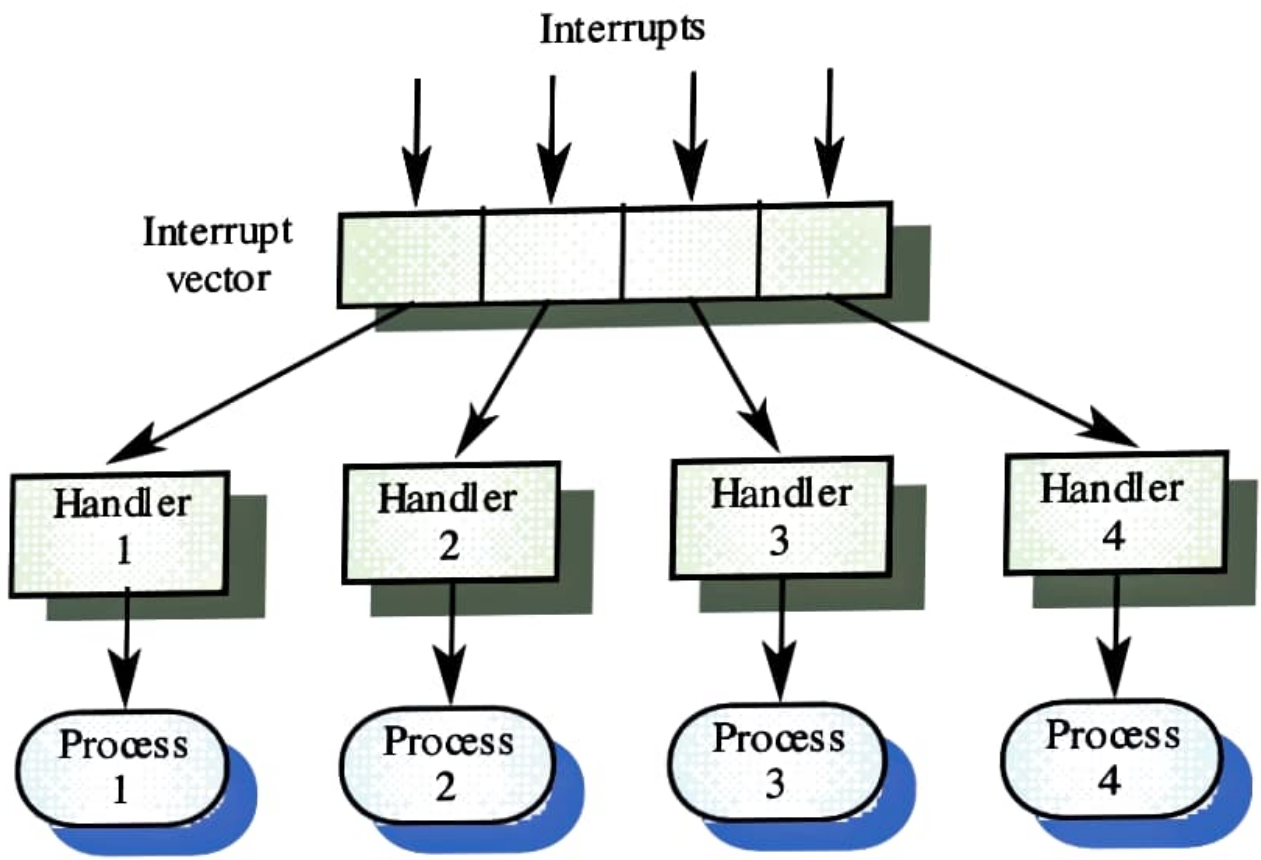
Selective broadcasting



Interrupt-driven systems

- Used in real-time systems where fast response to an event is essential
- There are known interrupt types with a handler defined for each type
- Each type is associated with a memory location and a hardware switch causes transfer to its handler
- Allows fast response but complex to program and difficult to validate

Interrupt-driven control



Modular decomposition

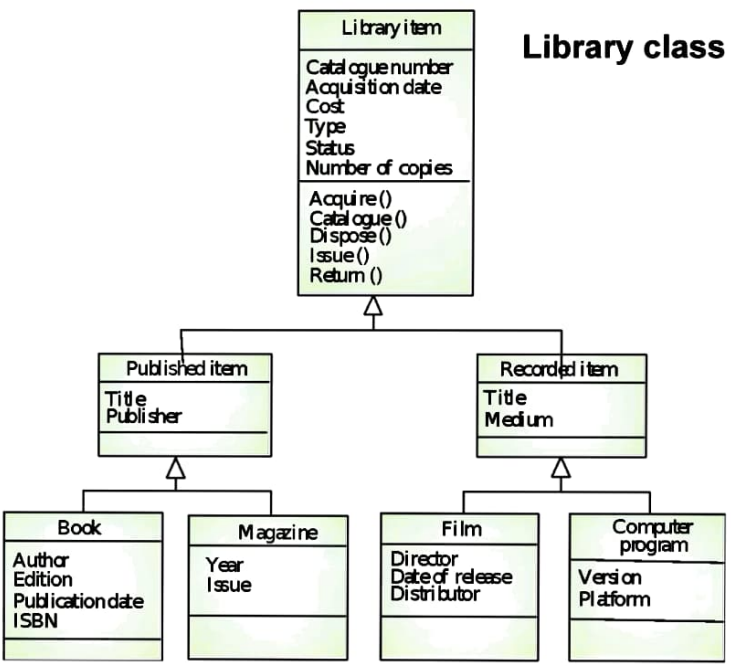
Inheritance models

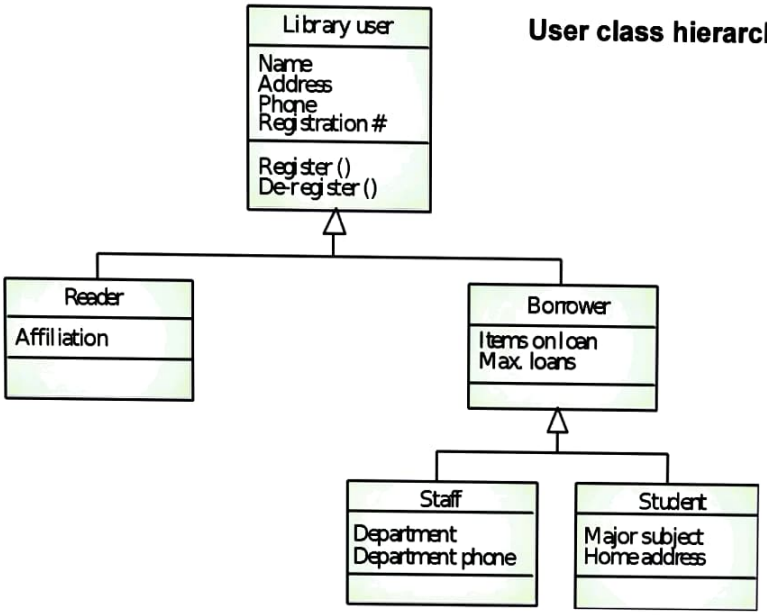
- Organise the domain object classes into a hierarchy
- Classes at the top of the hierarchy reflect the common features of all classes
- Object classes inherit their attributes and services from one or more super-classes. these may then be specialised as necessary
- Class hierarchy design is a difficult process if duplication in different branches is to be avoided

The Unified Modeling Language

- Devised by the developers of widely used object-oriented analysis and design methods
- Has become an effective standard for object-oriented modelling
- Notation
  - Object classes are rectangles with the name at the top, attributes in the middle section and operations in the bottom section
  - Relationships between object classes (known as associations) are shown as lines linking objects
  - Inheritance is referred to as generalisation and is shown 'upwards' rather than 'downwards' in a hierarchy

Library class hierarchy

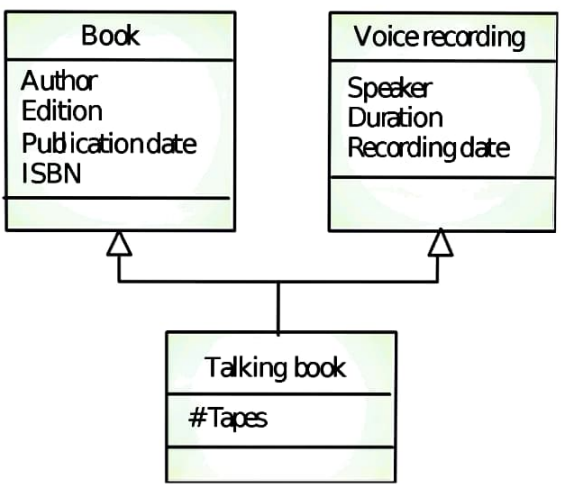




Multiple inheritance

- Rather than inheriting the attributes and services from a single parent class, a system which supports multiple inheritance allows object classes to inherit from several super-classes
- Can lead to semantic conflicts where attributes/services with the same name in different super-classes have different semantics
- Makes class hierarchy reorganisation more complex

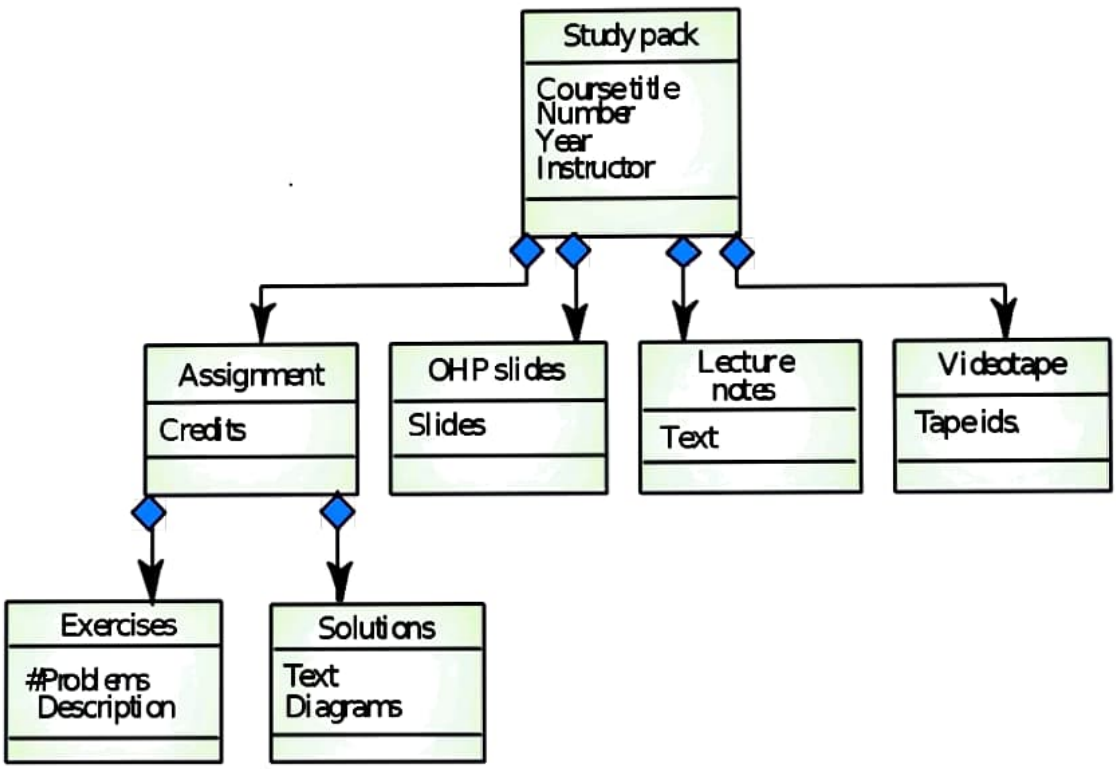
Multiple inheritance



Object aggregation

- Aggregation model shows how classes which are collections are composed of other classes
- Similar to the part-of relationship in semantic data models

Object aggregation

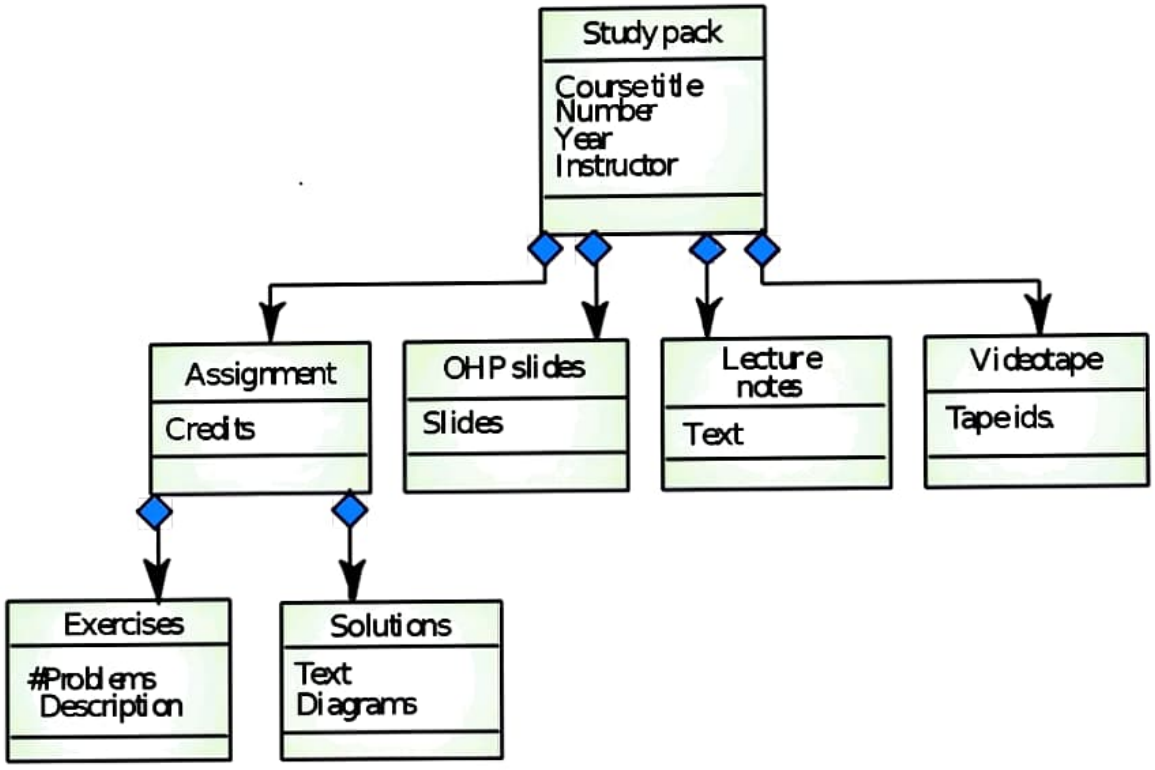


Object aggregation

- Aggregation model shows how classes which are collections are composed of other classes
- Similar to the part-of relationship in semantic data models

3b & 4a same

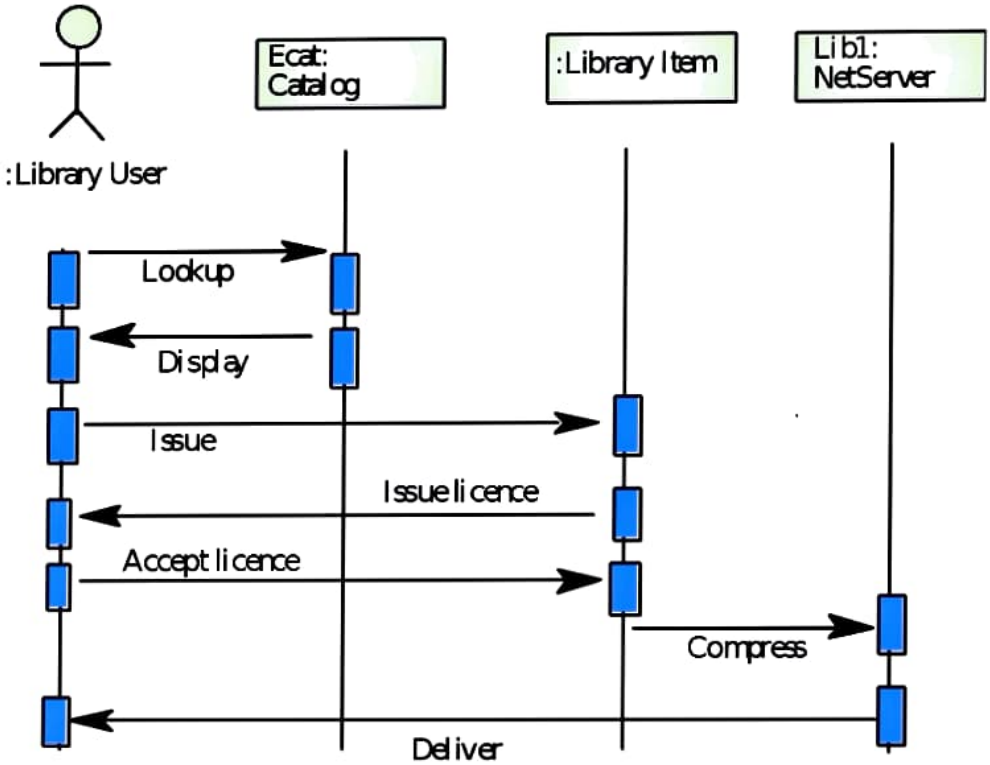
Object aggregation



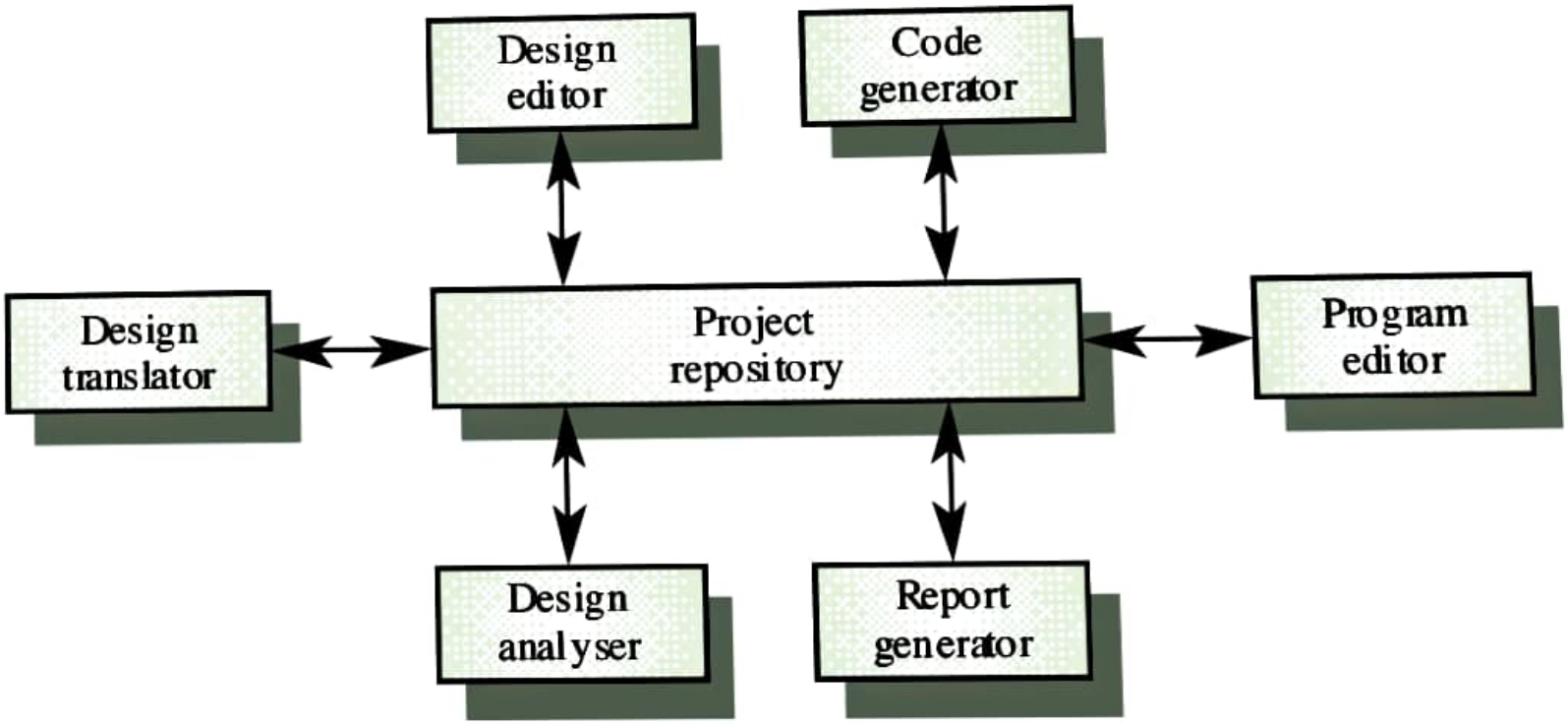
Object behaviour modelling

- A behavioural model shows the interactions between objects to produce some particular system behaviour that is specified as a use-case
- Sequence diagrams (or collaboration diagrams) in the UML are used to model interaction between objects

Issue of electronic items



CASE toolset architecture

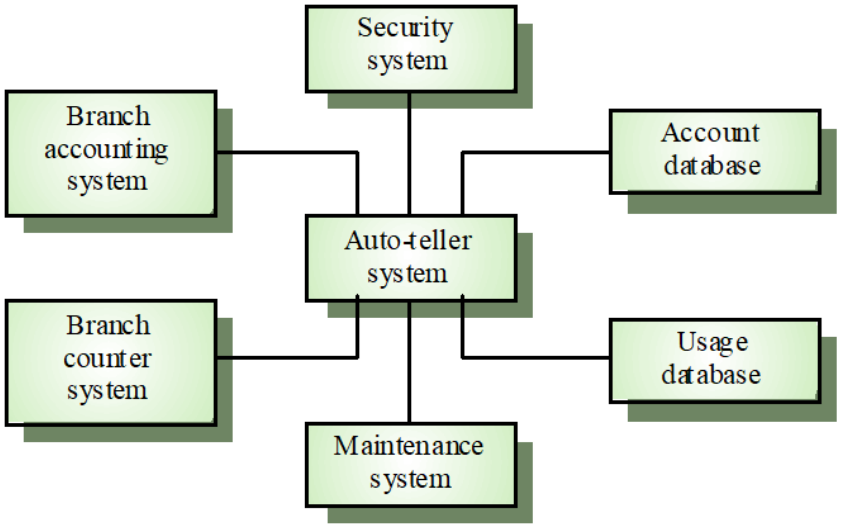


Question 6

Context models

- Context models are used to illustrate the boundaries of a system
- Architectural models show that a system and its relationship with other systems
  - Context models simply show the other systems in the environment, not how the system being developed is used in that environment.

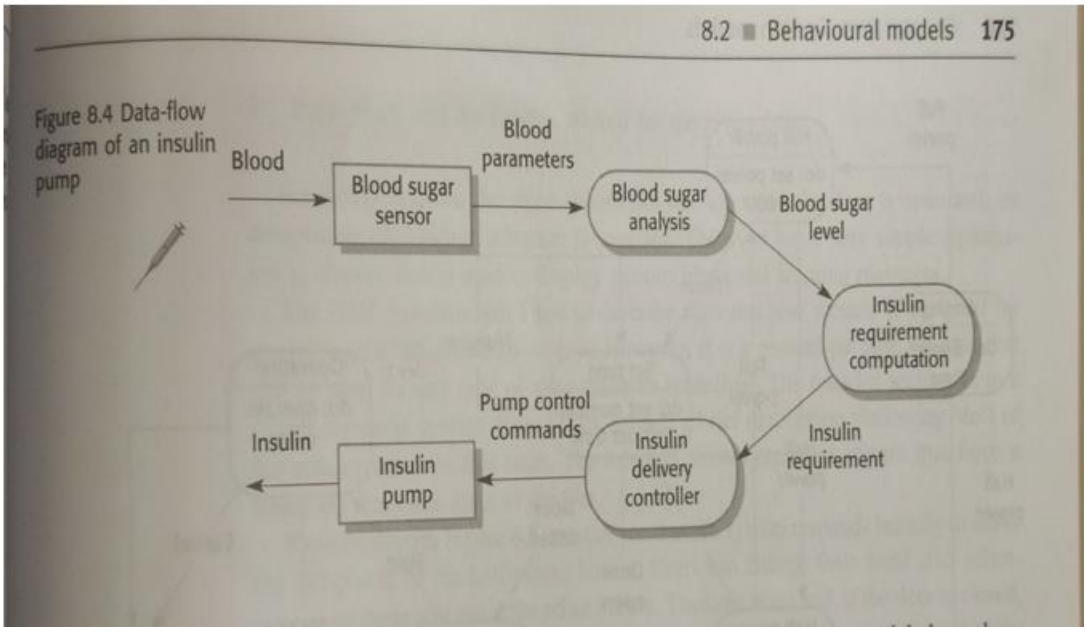
The context of an ATM system



Question 7

Data flow diagrams

- DFDs model the system from a functional perspective
- The data is transformed at each step before moving on to the next stage. Processing step cold be to filter duplicate records in a customer data base.
- Tracking and documenting how the data associated with a process is helpful to develop an overall understanding of the system
- Data flow diagrams may also be used in showing the data exchange between a system and other systems in its environment

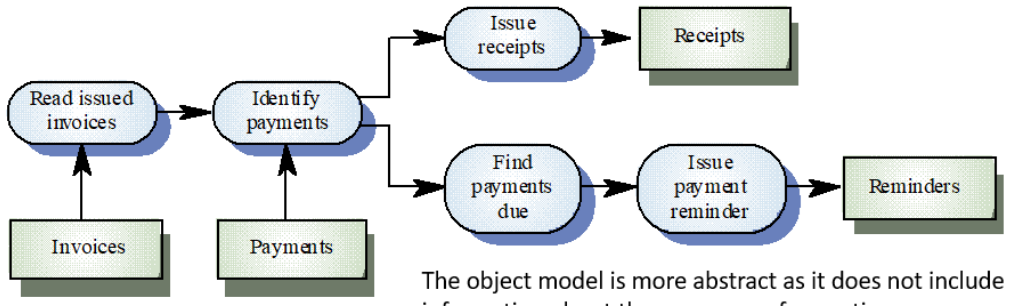


Question 8

Function-oriented pipelining

- In a function-oriented pipeline or data-flow model, functional transformations process their inputs and produce outputs

Invoice processing system



The object model is more abstract as it does not include information about the sequence of operations.

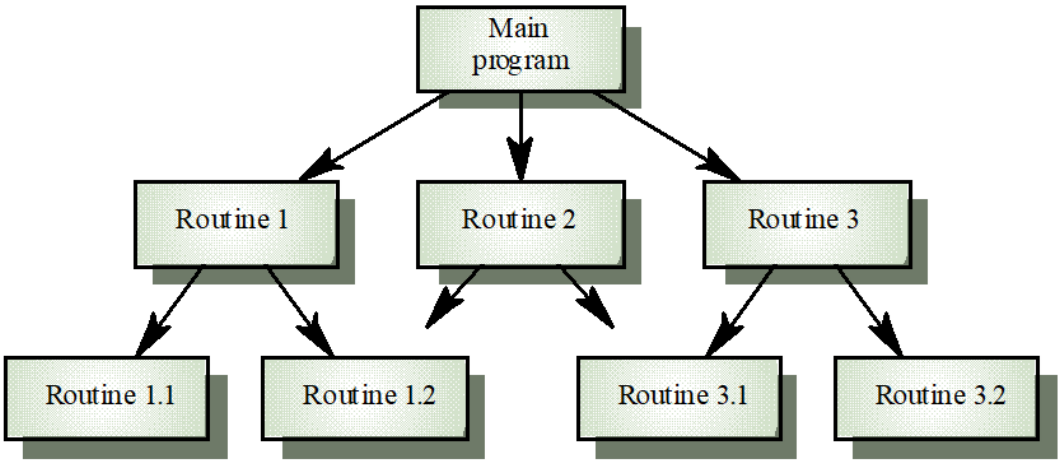
Question 9

Centralised control

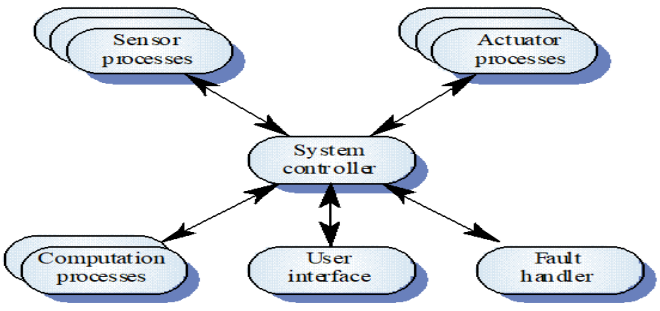
Centralized control models fall into two classes, depending on whether the controlled sub-systems execute sequentially or in parallel.

- Call-return model
  - Top-down subroutine model where control starts at the top of a subroutine hierarchy and moves downwards. Applicable to sequential systems
- Manager model
  - Applicable to concurrent systems. One system component controls the stopping, starting and coordination of other system processes. Can be implemented in sequential systems as a case statement

Call-return model



Real-time system control



The system controller process decides when processes should be started or stopped depending on system state variables. It checks whether other processes have produced information to be processed or to pass information to them for processing. The controller usually loops continuously, polling sensors and other processes for events or state changes.

Question 10

7. What is repository model? Explain its advantages and disadvantages.

6 Mark question | Asked in (TU CSIT) Software Engineering 2075

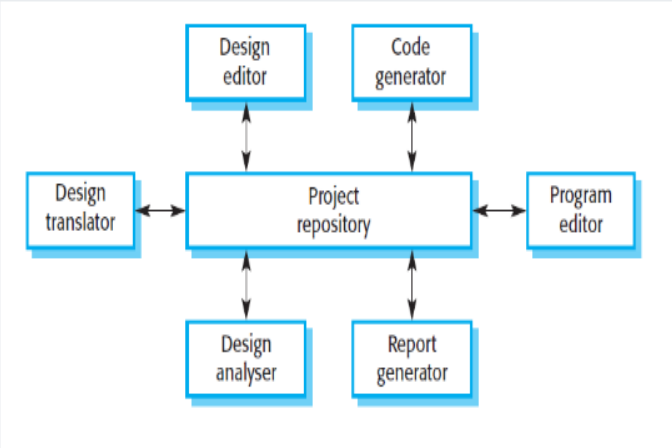
Solution

A repository model is a system that will allow interfacing sub-systems to share the same data. Sub-system must exchange data so that they can work together effectively. This may be done in two ways:

1. All shared data is held in a central database that can be accessed by all subsystems. It is called repository model.

2. Each sub-system maintains its own database. Data is interchanged with other sub-systems by passing messages to them.

Example: CASE Toolset



Advantages:

- It is an efficient way to share large amounts of data. There is no need to transmit data explicitly from one sub-system to another.
- Sub-systems that produce data need not be concerned with how that data is used by other subsystems.
- Activities such as backup, security, access control and recovery from error are centralized.

Disadvantages:

- It is a compromise between the specific needs of each tool. Performance may be adversely affected by this compromise.
- Evolution may be difficult as a large volume of information is generated according to an agreed data model.
- Different sub-systems may have different requirements for security, recovery and backup policies. The repository model forces the same policy on all subsystems.

Question 11

Layered Technology in Software Engineering

Difficulty Level : Easy • Last Updated : 09 Jun, 2022

Read Discuss Practice Video Courses

Software engineering is a fully layered technology, to develop software we need to go from one layer to another. All the layers are connected and each layer demands the fulfillment of the previous layer.

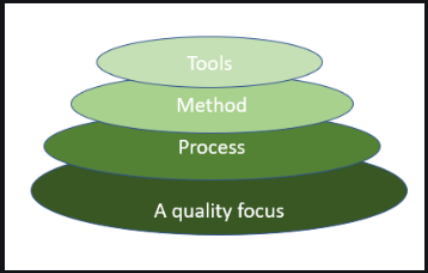
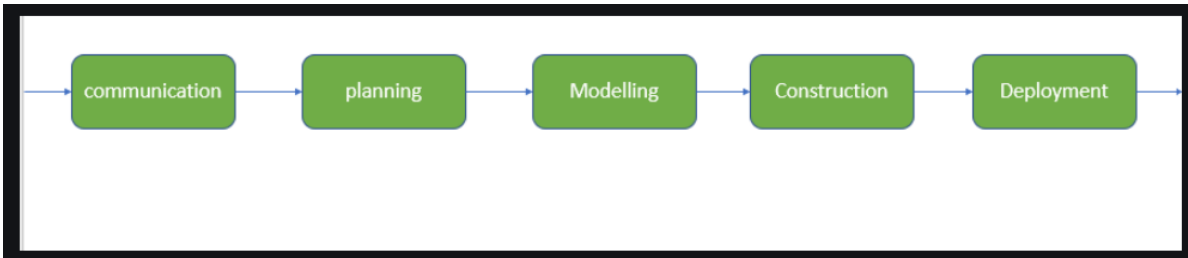


Fig: The diagram shows the layers of software development

Layered technology is divided into four parts:

1. A quality focus: It defines the continuous process improvement principles of software. It provides integrity that means providing security to the software so that data can be accessed by only an authorized person, no outsider can access the data. It also focuses on maintainability and usability.

2. Process: It is the foundation or base layer of software engineering. It is key that binds all the layers together which enables the development of software before the deadline or on time. Process defines a framework that must be established for the effective delivery of software engineering technology. The software process covers all the activities, actions, and tasks required to be carried out for software development.



Process activities are listed below:-

- **Communication:** It is the first and foremost thing for the development of software. Communication is necessary to know the actual demand of the client.
- **Planning:** It basically means drawing a map for reduced the complication of development.
- **Modeling:** In this process, a model is created according to the client for better understanding.
- **Construction:** It includes the coding and testing of the problem.
- **Deployment:-** It includes the delivery of software to the client for evaluation and feedback.

3. Method: During the process of software development the answers to all "how-to-do" questions are given by method. It has the information of all the tasks which includes communication, requirement analysis, design modeling, program construction, testing, and support.

4. Tools: Software engineering tools provide a self-operating system for processes and methods. Tools are integrated which means information created by one tool can be used by another.

Question 12 13

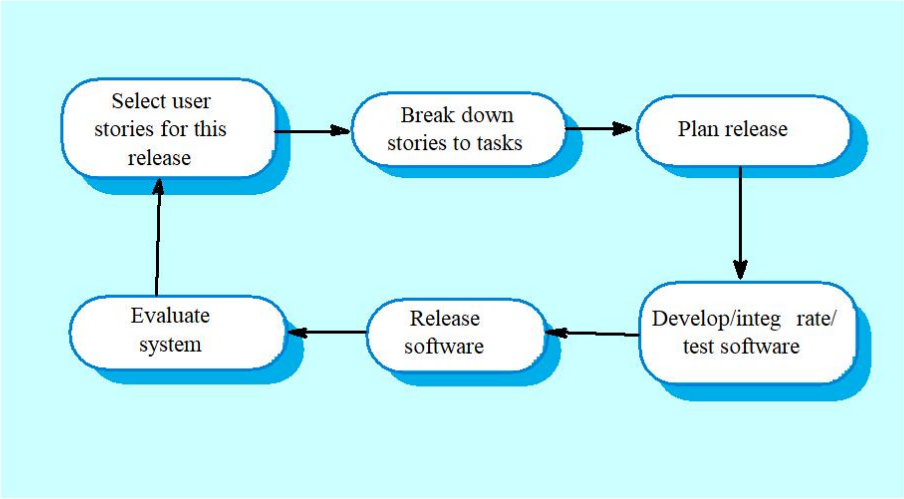
eXtreme Programming

- **Extreme Programming (XP) is an agile software development framework** that aims to produce higher quality software, and higher quality of life for the development team.
- https://www.youtube.com/watch?v=kFM2Vcu-BRo

Extreme programming

- Perhaps the best-known and most widely used agile method.
- Extreme Programming (XP) takes an 'extreme' approach to iterative development.
  - New versions may be built several times per day;
  - Increments are delivered to customers every 2 weeks;
  - All tests must be run for every build and the build is only accepted if tests run successfully.

The XP release cycle



Extreme programming practices 2

Pair Programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective Ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers own all the code. Anyone can change anything.
Continuous Integration	As soon as work on a task is complete it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of over-time are not considered acceptable as the net effect is often to reduce code quality and medium term productivity (a process that avoids long working hours.)
On-site Customer	A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

Question 14

Principles of agile methods

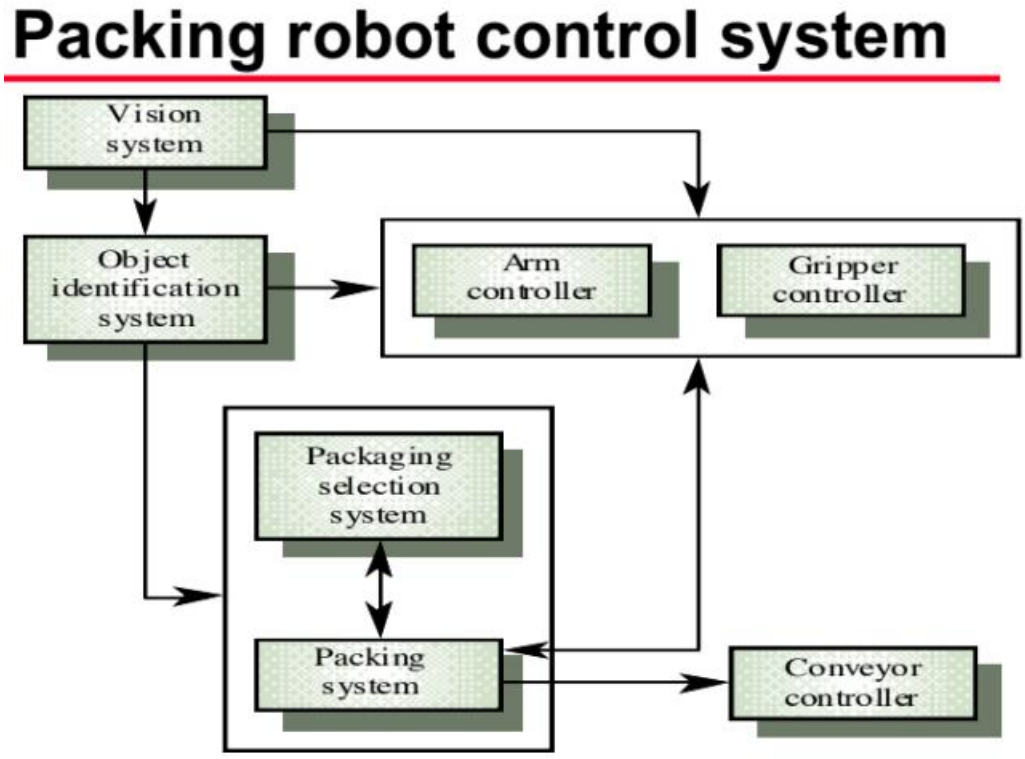
Principles of agile methods

Principle	Description
Customer involvement	The customer should be closely involved throughout the development process. Their role is provide and prioritise new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognised and exploited. The team should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and design the system so that it can acc ommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process used. Wherever possible, actively work to eliminate complexity from the system.

Question 17

ROBOT

For example, Figure 11.1 is an abstract model of the architecture for a packing robot system that shows the sub-systems that have to be developed. This robotic system can pack different kinds of object. It uses a vision sub-system to pick out objects on a conveyor, identify the type of object and select the right kind of packaging. The system then moves objects from the delivery conveyor to be packaged. It places packaged objects on another conveyor. Other examples of architectural designs at this level are shown in Chapter 2 (Figures 2.6 and 2.8).

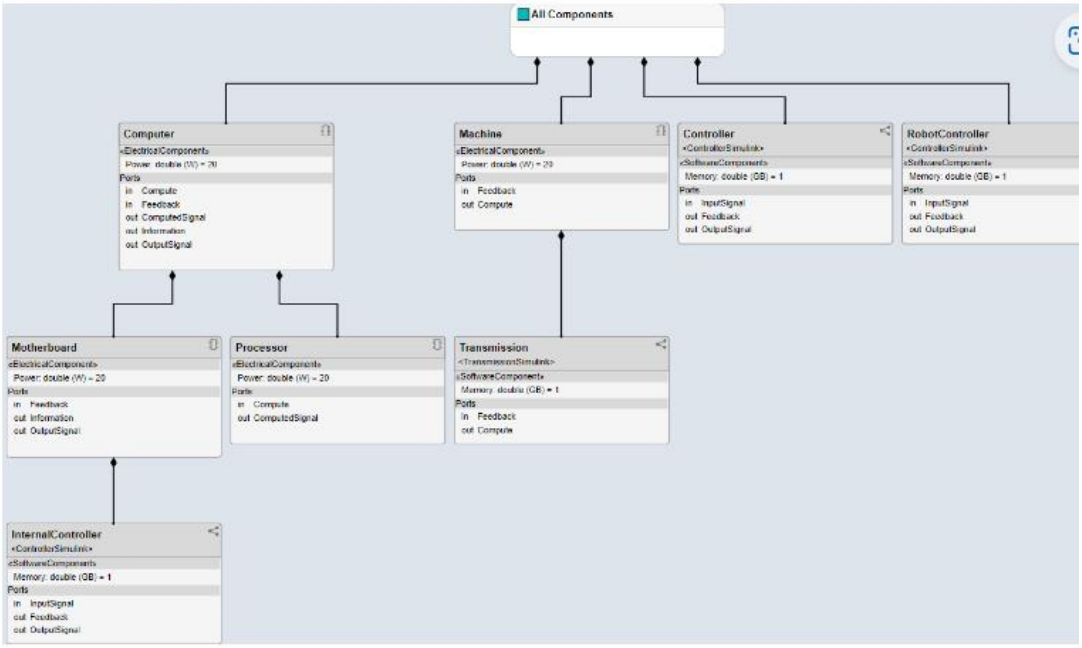


Question 16

PRINCIPLES

Principle or practice	Description
Incremental planning	Requirements are recorded on Story Cards and the Stories to be included in a release are determined by the time available and their relative priority. The developers break these Stories into development 'Tasks'. See Figure 17.6 and Figure 17.7.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.
Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers own all the code. Anyone can change anything.
Continuous integration	As soon as work on a task is complete it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium-term productivity
On-site customer	A representative of the end-user of the system (the Customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

Question 15



The component hierarchy diagram shows a single root, which is the view specification itself. The root corresponds to the name of the view shown in the component diagram. The connections in the component hierarchy diagram originate from the child components and end with a diamond symbol at each parent component.