Chapter 2 Software Development :

Software development -

1. Planning
2. Requirement Analysis
3. Design
4. Implementation
5. Verification & Validation
6. Maintenance & Evolution

Planning -

1. Identify business value
2. Analyze feasibility
3. Develop work plan
4. Staff the project
5. Estimate
6. Identify risk

Requirements analysis -

1. Process of establishing what services are required and the constraints on the system's operation and development
2. **2 major activities:**
    a. **Requirements elicitation (Questions to ask)**
        i. Who are the stakeholders of this project?
        ii. What do the system stakeholders require or expect from the system?
        iii. Interviews, questionnaires, meetings, etc.

b. **Requirements specification**
    i. Defining the requirements in detail
    ii. Use cases are the major part of a requirements specification

Design -

1. Design is how we build the system
2. It is different from Requirement Analysis as analysis is concerned with what is to be built

Design comprises of different aspects:

1. **Architectural design**
    a. Major structure of design (eg. MVC)
2. **Sub system design**
    a. Describe interface/protocol of each sub-system
    b. Structure of each sub-system
    c. Structure can describe how each sub-system sub-divides into package, components
3. **Detailed design**
    a. Description of each class, methods, attributes/data-types
4. **Persistent data design**
    a. Description of database, tables, primary and foreign keys
5. **GUI design**
    a. User interface

Implementation -

1. Construction - Writing code
2. Installation - Deployment

Verification & Validation -

1. Show that system conforms to specifications and meet requirements
2. Testing is the most commonly used V & V activity
3. System testing involves executing test cases that are derived from the specification of real data
    a. **Verification -**
        i. Does it perform exactly as specified in the use cases?
    b. **Validation -**
        i. Bringing in users to demonstrate features and get feedbacks

Maintenance & Evolution -

1. Software does not wear out so there is a need to constantly update it.
2. New requirements like functionalities or UI improvements will emerge at any time in software development lifecycle
3. Changes in business environments (competition, law, new markets, new audience)
4. Change in infrastructure environments (new server/equipments)
5. New technology updates and standards (new os)
6. Bug fixes
7. Performance improvements

Types of maintenance -

1. **Adaptive maintenance:**
    a. Changing the system in response to changes in its environment so it continues to function (Hardware changes/OS changes)
2. **Corrective maintenance:**
    a. Fixing bugs and errors
3. **Perfective maintenance:**
    a. Changing the system's functionality to meet the changing needs (Add Functionalities)

When changes are made to a software, other coordinated changes are often needed at the same time in other parts of the software and that is called **Change propagation** and if the changes are not completed immediately, it will incur a "**debt**"

Technical Debt -

Concept in software development that reflects the implied cost of additional rework caused by choosing an easy solution instead of using a better approach that will take a longer time

Laws of software evolution -

1. **Continuing change -**
    a. Program needs to be continually adapted as new requirements will emerge as environment evolves
2. **Increasing complexity -**
    a. As the program evolves, it will get more complex (unpaid technical debts increase software entropy). Accumulated debt makes it difficult to change.
    b. To reduce or maintain it, refactoring and restructuring it might be needed
3. Self regulation
4. Conservation of organizational stability

5. Conservation of familiarity
6. **Continuing growth -**
    a. Functional content of a program must be continually increased as users will demand more functions to make their quality of life easier to perform tasks
7. **Declining quality -**
    a. Evolving programs will be perceived as declining quality unless maintained and adapted to changing operational environment to improve/maintain reliability and performance
8. Feedback system

Chapter 3 Software Process Models and Ethics :

Software process models -

1. Waterfall model
2. Prototyping model
3. Iterative/incremental model
4. Spiral model
5. Rational Unified Process
6. Agile method

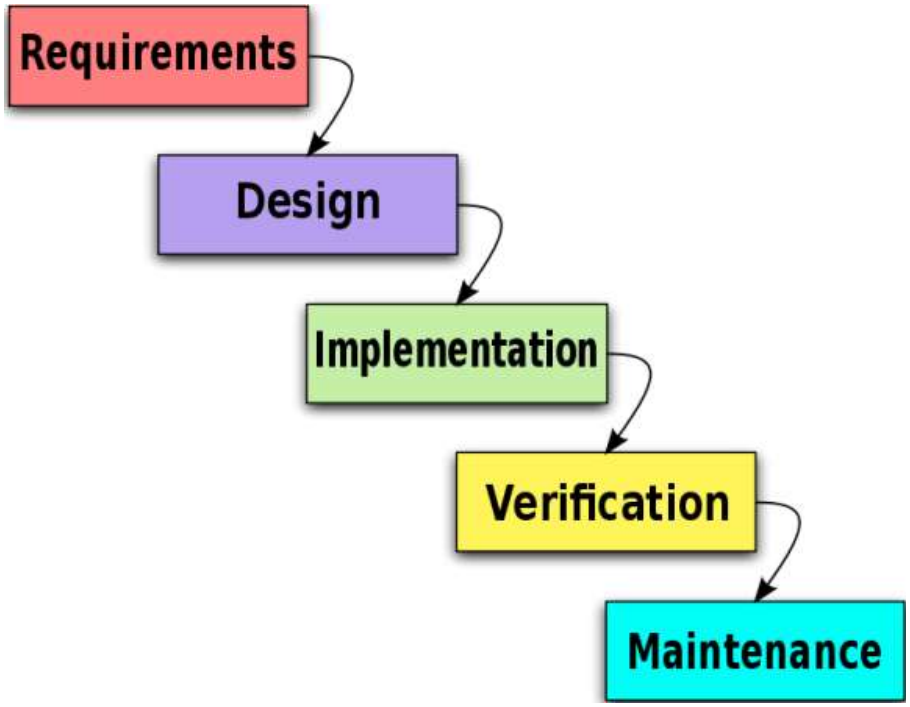
Software Process is a **structured set of activities** required to develop a software system

Many different processes but they all involve

1. Requirements specification - What system should do
2. Design & Implementation - Defining the organization of the system and implementation of the system
3. Verification & Validation - checking if it conforms to the specification and customer needs
4. Maintenance & Evolution - Changing the system in response to changing customer needs

A software process model is an **abstract representation of a process**. It presents a description of a process from some particular perspective.

Waterfall model

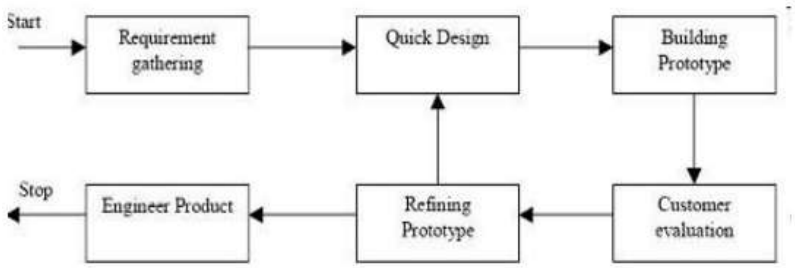


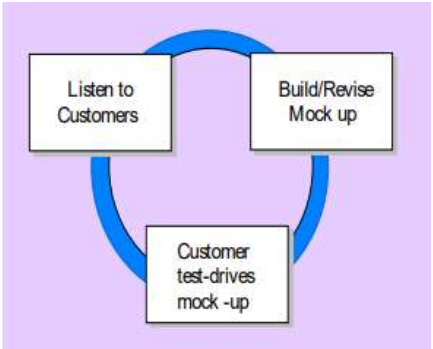
Waterfall model is a software development process which has to be complete and absolutely correct before moving onto the next phase. Progress flows from the top to the bottom like a waterfall.

Limitations of Waterfall Model

1. Difficulty of responding to changing customer requirements
2. Designer will have to fully predict problem area of the system and produce a correct design before implementation is started
3. Model is only appropriate when requirements are well-understood and changes will be fairly limited during the design process

Prototyping model





Gather requirements -> Wireframing -> Prototyping -> Customer evaluation of prototype -> Refine and repeat process until customer & developer is satisfied -> Develop product

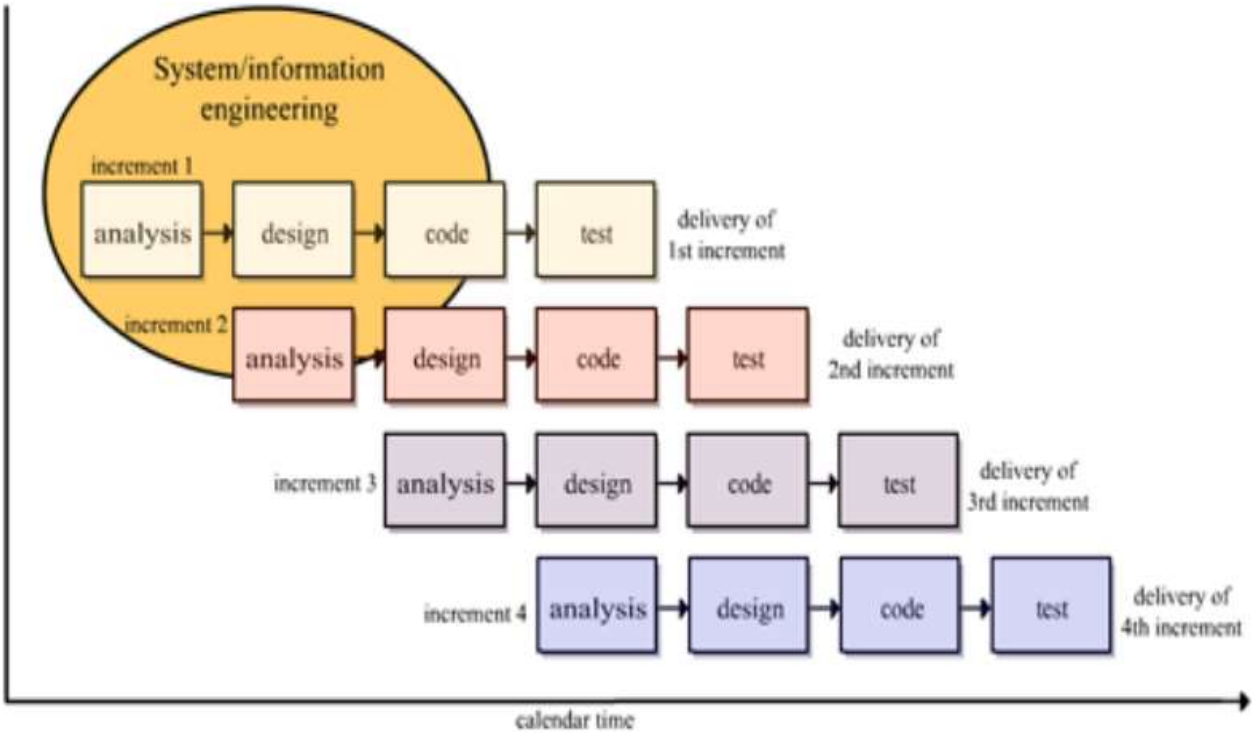
Benefits of prototyping model

1. Users are actively involved in the development
2. Reduced time and costs
    a. Prototype helps users get a better understanding of the system being developed
    b. Prototype improve the quality of requirements and specification provided to developers
    c. Quicker user feedback leading to better solutions
3. Best projects to use prototyping
    a. Human-Computer interface

Limitation of prototyping model

1. User can be confused that the prototype is the final product
2. Developers can become attached to the prototype which can lead to problems like attempting to convert a limited prototype into a final system when it does not have an appropriate underlying architecture.
3. Excessive development time of the prototype.
4. Expense of implementing prototyping might cost too much for startups.

Incremental and Iterative model



Development a system through iterative cycles in small portions. (**Incremental**)
Delivery is **broken down** into increments and each increment delivers parts of the required functionality. User requirements are prioritized and the highest priority requirements are included in early increments. Each iteration, design modifications are made and new functional capabilities are added

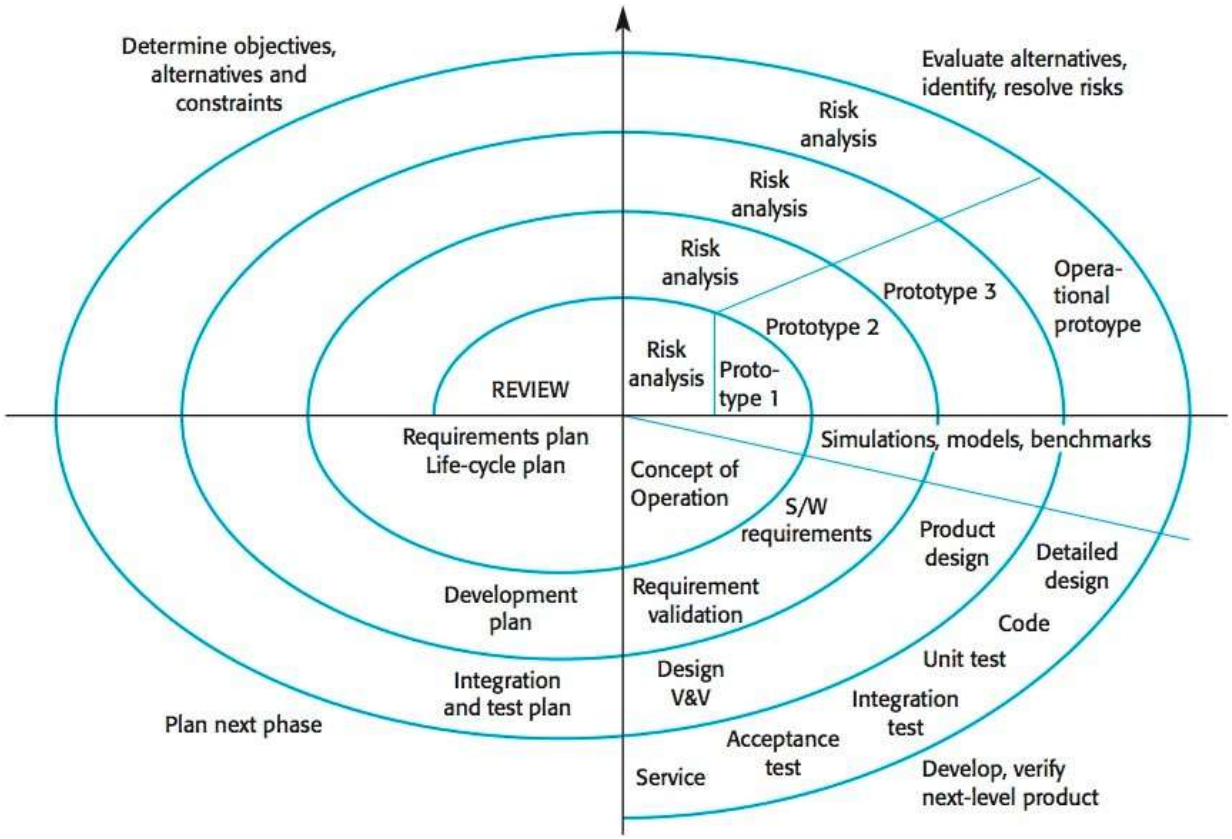
Benefits of Incremental and iterative model

1. Feedback from early stages used in developing latter stages
2. Cost is reduced to accommodate changing customer requirements
3. More rapid delivery and deployment of useful software to customer (customer are able to use and gain value from software early)
4. Early increments act as a prototype to help elicit requirement for later increments
5. Lower risk of overall failure
6. Highest priority services receive the most testing

Cons of Incremental and iterative model

1. Loss of economy of scale (Some cost will be repeated)
2. System structure tends to degrade as new increments are added
    a. Technical Debt
    b. Improving further software change will become increasingly difficult and costly

Spiral Model -



Combination of prototyping with iterative model. Starts from the center and spirals outwards, hence, named Spiral Model

Spiral model is influential in helping people think about iteration in software processes and introduction of risk-driven approach to development. This is mostly used in large projects such as game development as well as military projects.

Agile -

Agile refers to a set of methods and practice :

1. Requirements and solutions evolve through **collaborative effort** of a self organizing cross functional team and their customer and users
2. **Adaptive planning**, **evolutionary development**, **early delivery** and **continuous improvement**
3. **Rapid** and **flexible** to change

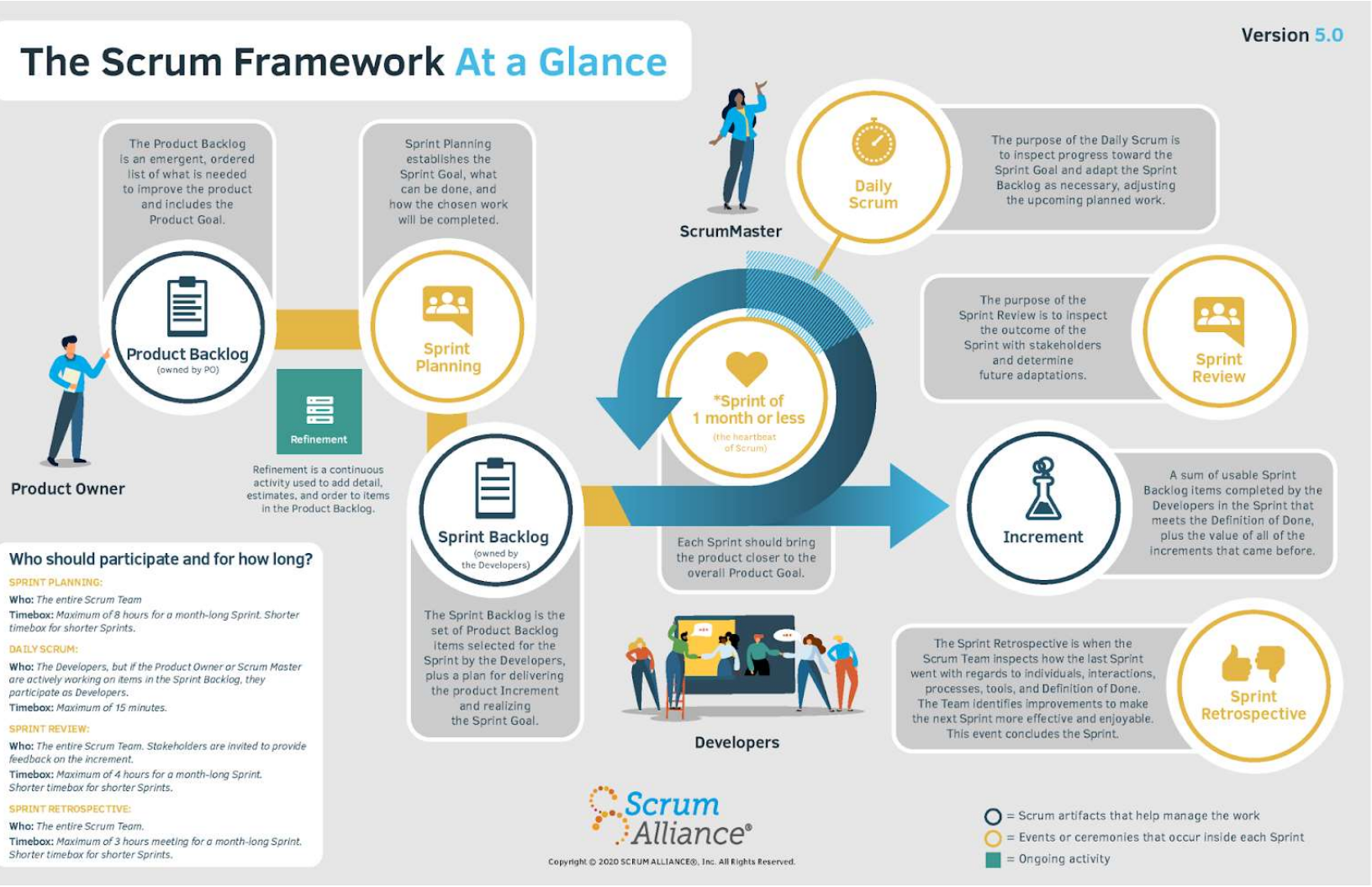
Principles :

1. **Highest priority** is to satisfy customer through **early** and **continuous delivery**
2. Welcome **changing requirements**, even late in development.
3. Deliver working software **frequently**, with a **preference** to a **shorter** timescale from a couple of weeks to months
4. Business people and developers **must work together**
5. Build projects around **motivated individuals**. Giving them the **environment**, **support** and **trust** to get the job done.
6. Most efficient and effective method to convey information in a team is through face to face conversations
7. Working software is the primary measure of progress.
8. Agile processes promote **sustainable development**. Sponsors, developers, and users should be able to maintain constant pace indefinitely.
9. Continuous attention to **technical excellence and good design** enhances agility.
10. **Simplicity –** Efficient coding to get the function
11. Best architecture, requirements, and design emerge from **self-organizing** teams.
12. At regular intervals, the team will **reflect** on how to become more **effective**, then **tune** and **adjust** its behavior accordingly

Agile software development methods -

1. **Scrum**
2. **Extreme Programming**
3. Dynamic System Development Methods
4. **Kanban**
5. Lean Software Development

Scrum -



Sprints

1. Scrum projects make progress in a series of “sprints” (i.e. iterations)
2. Typical duration is 2-4 weeks or a calendar month at most
3. Product is designed, coded, and tested during the sprint

Product backlog

1. A typical scrum backlog comprises of :
    a. Features
    b. Bugs
    c. Technical work (e.g upgrade softwares)
    d. Knowledge acquisition (e.g learning java)
2. **Prioritized** by the product owner
3. Reprioritized at the start of each sprint

User stories

A short and simple description of a feature told from the perspective of a user of the system.

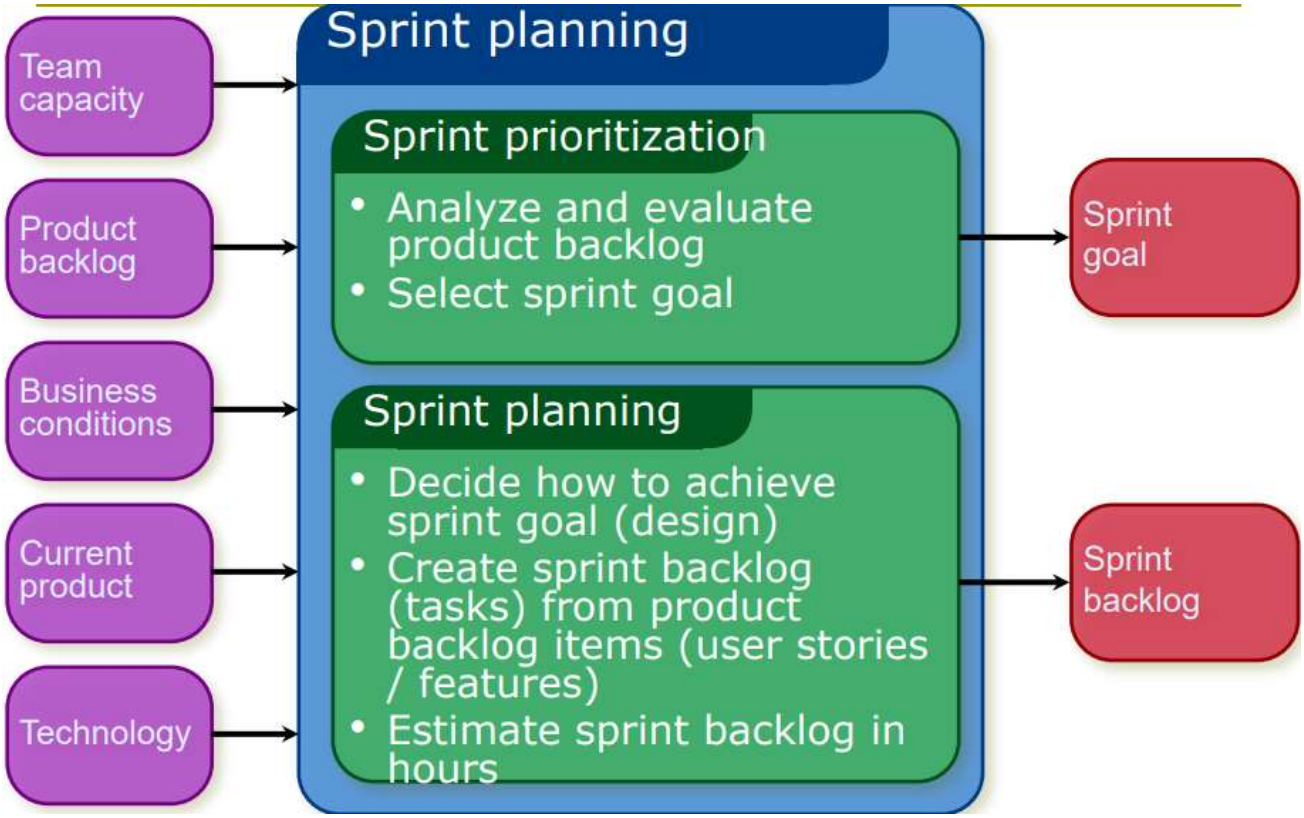
Template -

As a < user >, I want to < goal > so that < reason >.

Example -

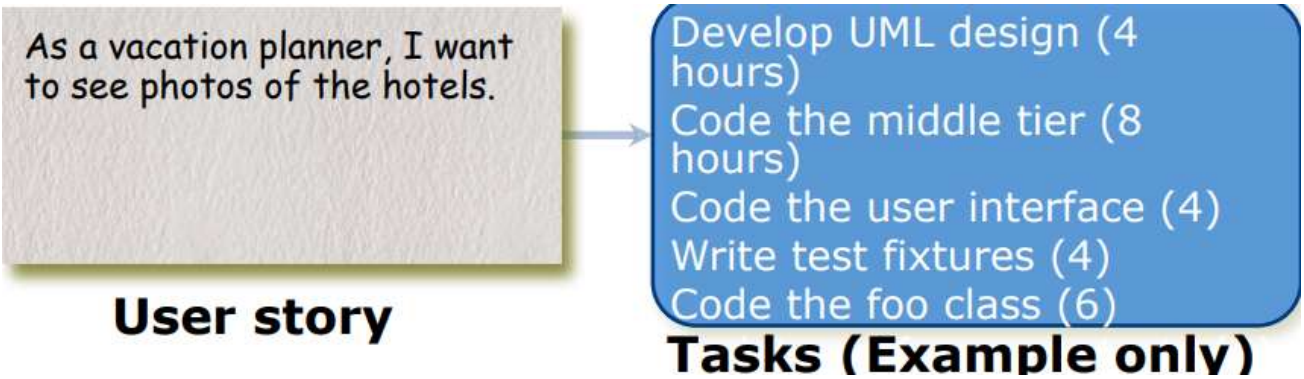
1. As a staff member, I want to view my schedule so that I know when my work shifts.
2. As a customer, I want to view available products so that I can find the stores with the product available.

Sprint planning



Team decides on the **goal** for a particular sprint, and then selects items from the product backlog they commit to complete to achieve the sprint's goal.

Sprint backlog is created with tasks identified and each is estimated (e.g 1-16 hours) collaboratively.



Scrum Team

1. Product owner (Customer)
    a. Define the feature of product
    b. Decide on release date and content
    c. Prioritize features according to market
    d. Adjust features and priority every iteration
    e. Accept or reject work results

2. Scrum Master (Team leader)
    a. Represents management to the project, ensure that the team is fully functional and productive
    b. Responsible for enacting Scrum values and practices

3. The team
    a. Self organizing and cross-functional - Programmers, testers, UX designer etc.

Scrum meetings

1. **Sprint planning meeting -**
    a. Start of sprint planning meeting = Start of sprint
    b. Selecting product backlog items to be delivered during the current sprint
    c. Attended by product owner, scrum master, scrum team, and maybe stakeholders
    d. Focuses on

        i. What is to be built during the sprint?

        Product owner will present an initial goal, then will be broken down into backlog items with different priorities which the product owner will then present the highest priority items to the team. Team members then ask questions to get a clear picture of what is to be created.

ii. How will the team build it?

Some backlog items will be broken down to smaller tasks to complete the backlog item.

e. Team members should remember and accept the fact that situation, conditions and requirements can and will change during the course of the sprint. Therefore, it should not be expected to create a perfect sprint plan. They should put enough effort that they roughly confirm they have selected the right set of the product backlog items

f. Official timebox for meetings is 2 hours per week, which means for a 2 week long sprint it is 4 hours

g. After the how and what plan of the meeting is completed, the team will revisit the sprint goal and decide if any changes are required to be made based on the outcome of the meeting .

2. Daily scrum meeting -

a. To update the team on progress

i. What did you do yesterday?

ii. What will you do today?

iii. Are there any impediments in your way?

3. Sprint review meeting -

a. Entire team will be present including the product owner and the team will explain how much of the sprint goal they were able to accomplish. Team members will demonstrate the stories and tasks they completed and discuss the incomplete ones, and hurdles faced. The review for a month-long sprint typically takes four hours.

b. Stakeholders interact with the product and raise questions they have regarding the product.

c. Product owners discuss the state of the backlog and project delivery dates, incorporating the state of the market, technology and latest feedback. The

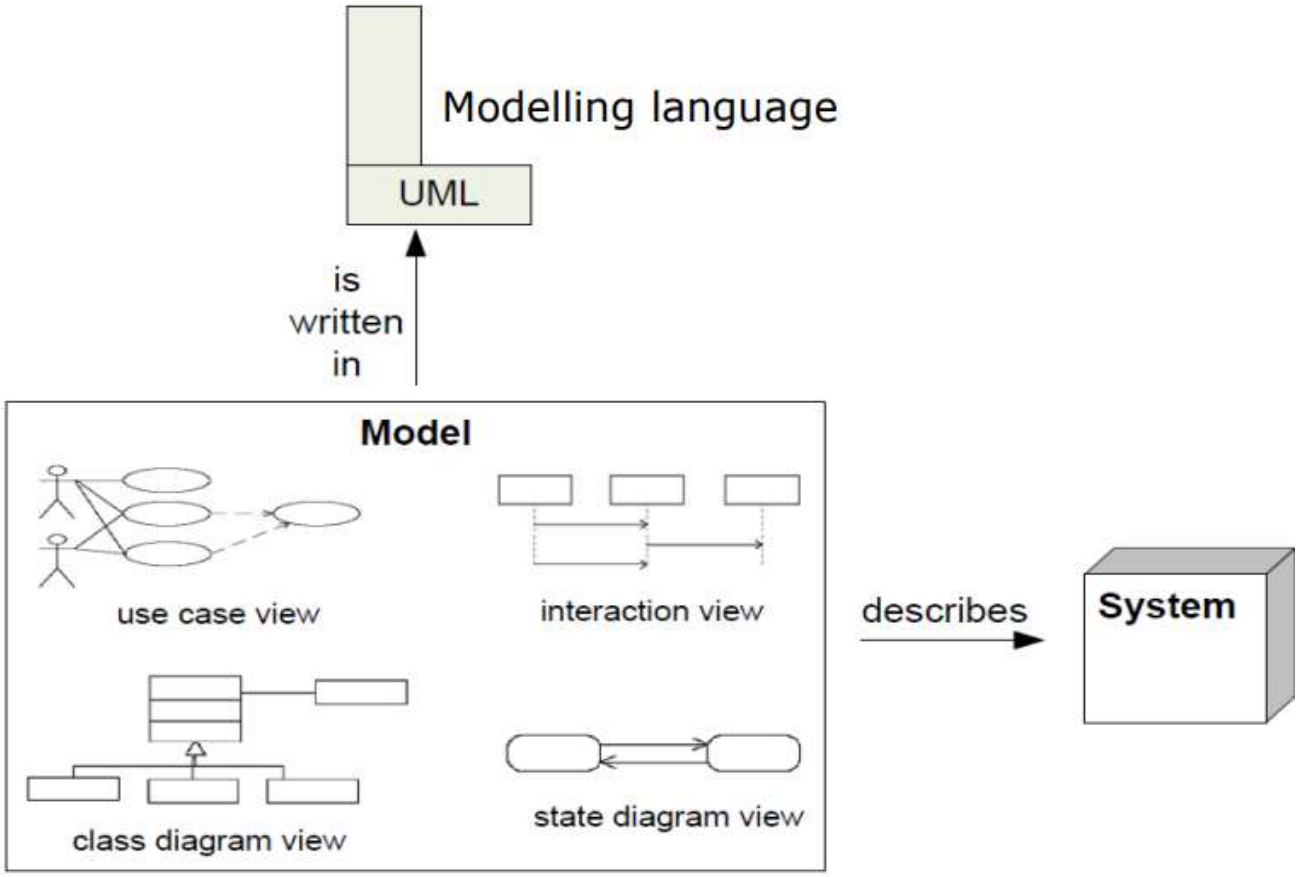
potential capabilities, release timelines, and budget is also reviewed. The sprint review offers valuable for the next sprint planning meeting

4. **Sprint retrospective -**
    a. The team discusses:
        i. What went well in the sprint
        ii. What could be improved
        iii. What will we commit to improve in the next Sprint

Advanced Unified Modeling Language (UML)

Two important component of a software development methodology -

- Process (Chapter 3)
- Modeling Language (UML Diagrams)
    - Describe a system using a models which has clear syntax and semantics suitable for both human and computer interpretation



System is described by a model and is written in a modeling language.

UML-2 diagrams

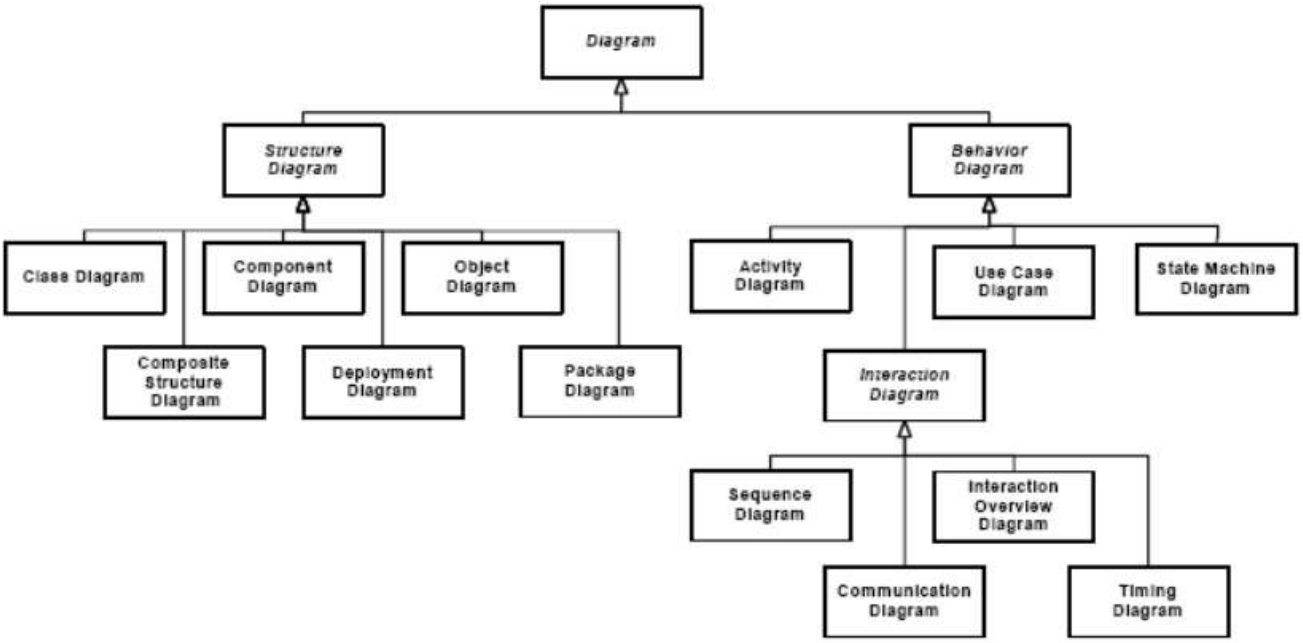
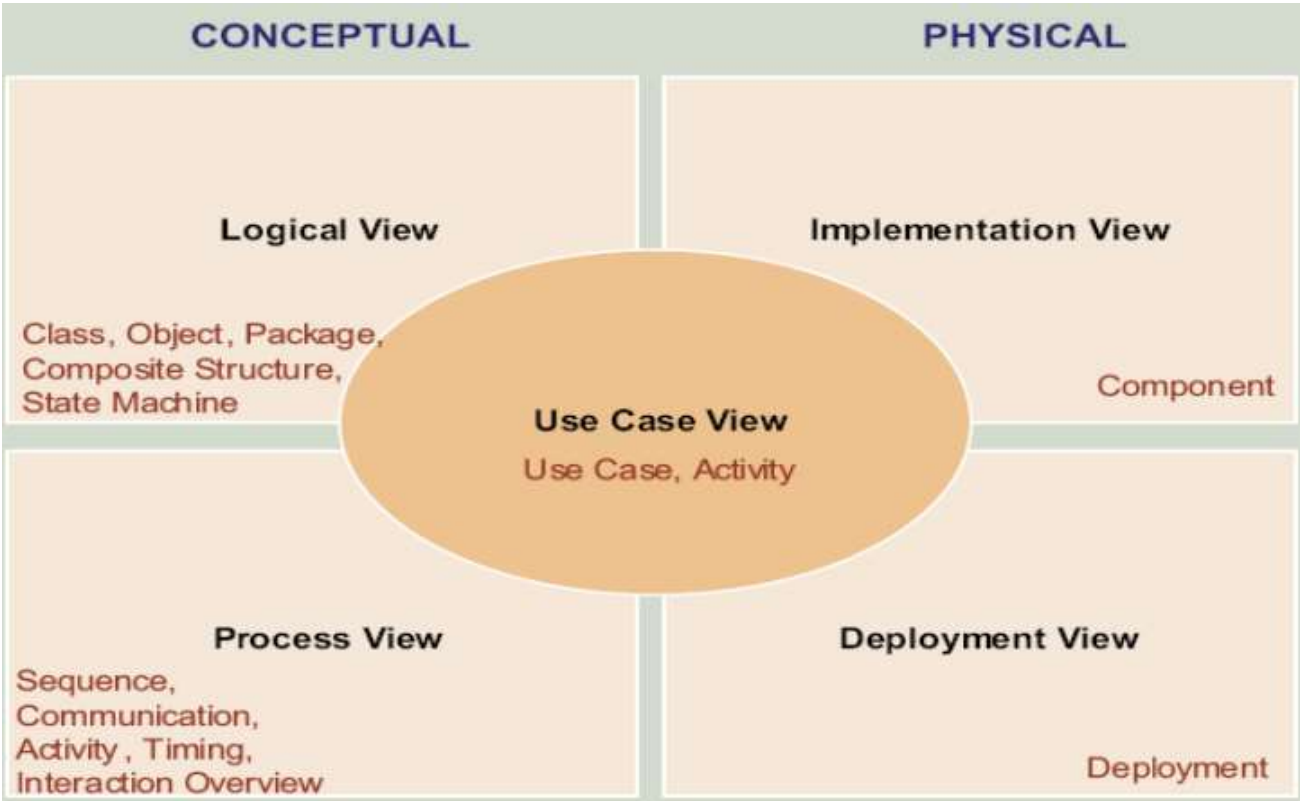


Diagram Views



Conceptual and Physical

The views are used to describe the system from viewpoints of different stakeholder such as end users, developers and project managers

Views and their Diagrams

Use Case	Logical	Process	Implementation	Deployment
Use Case	Class	Sequence	Component	Deployment
Activity	Object	Communication		
	Package	Activity		
	Composite Structure	Interaction Overview		
	State Machine	Timing		

Logical and Process View	Implementation and Deployment View
• Conceptual Level • Designed by analysis • **Logical view** is the **functionality** aspect the system provides to end users • **Process View** is the dynamic aspect of the system processes and how they communicate and focuses on the **runtime** behavior of the system	• Physical Level • Representation of application components as built and deployed

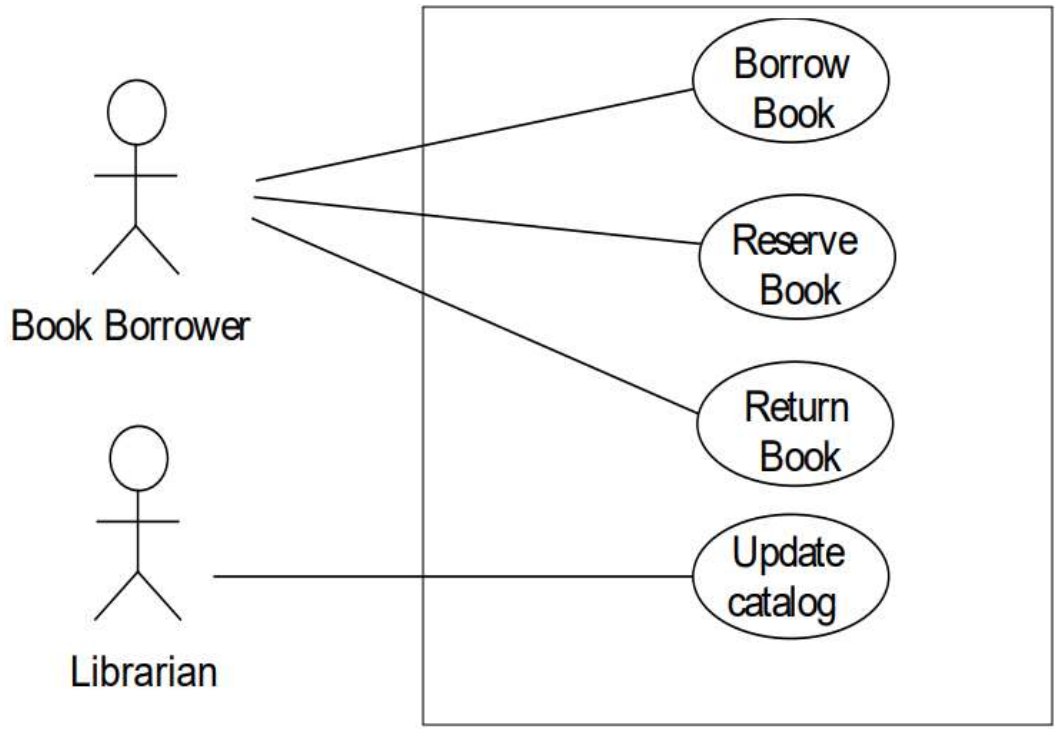
Use case view -

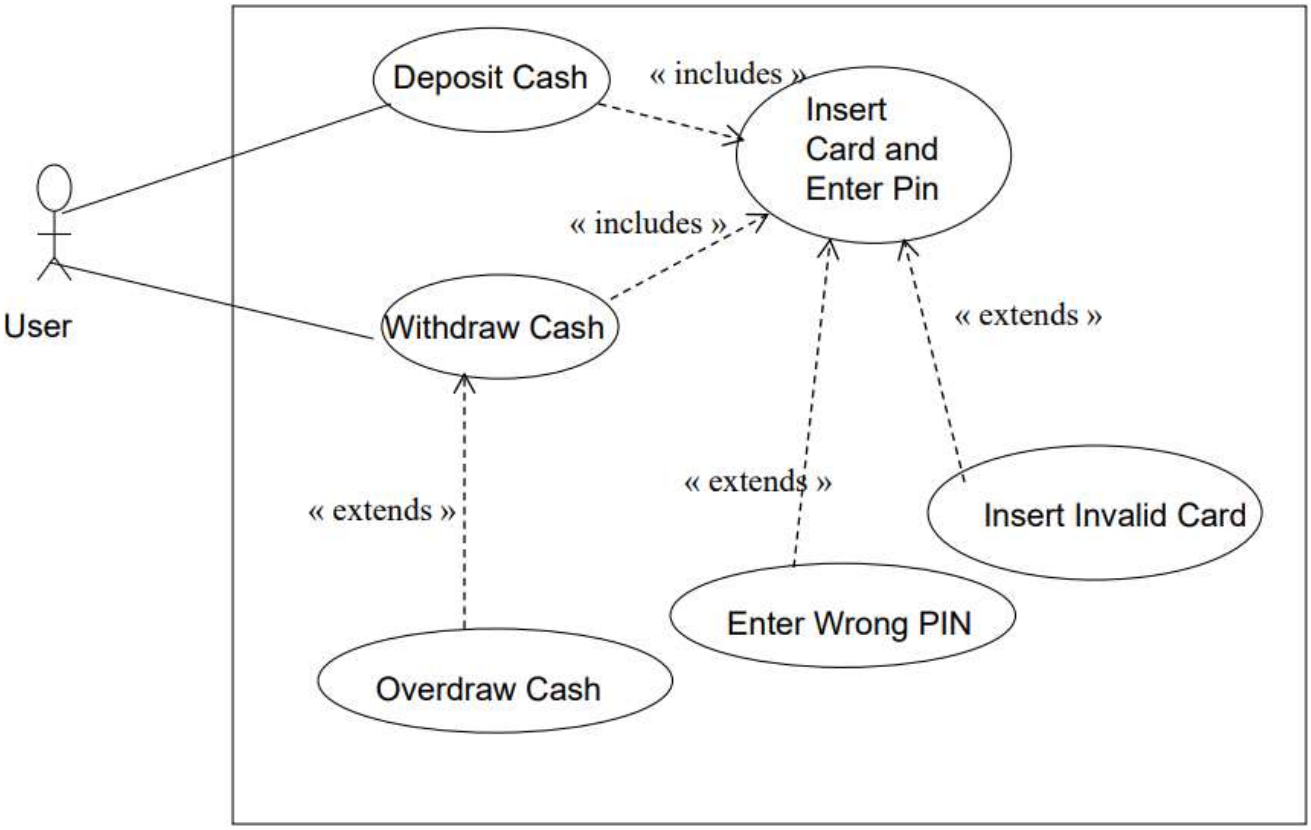
Activity diagram - Shows the flow of event without trigger (event) mechanism

Use case diagram - Shows the **functional requirements** of a system as one or more **actors** (User/External system) interacting with one or more **use cases** (Functionality).

<u>Use case specifies the behavior from user's point of view and not the system's</u>

Use case diagrams





Extends and Includes -

Includes -

A part of the use case that requires the stakeholder to fulfill when performing the interaction.

Extends -

Optional part of the use case which is a complex/alternative course of the interaction that occurs under certain conditions.

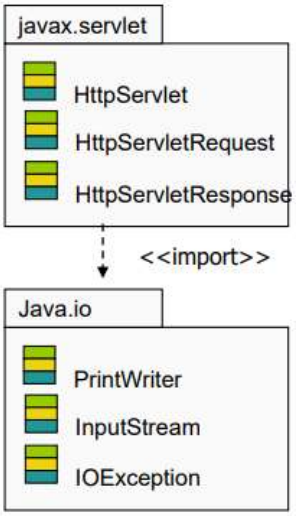
Use case description -

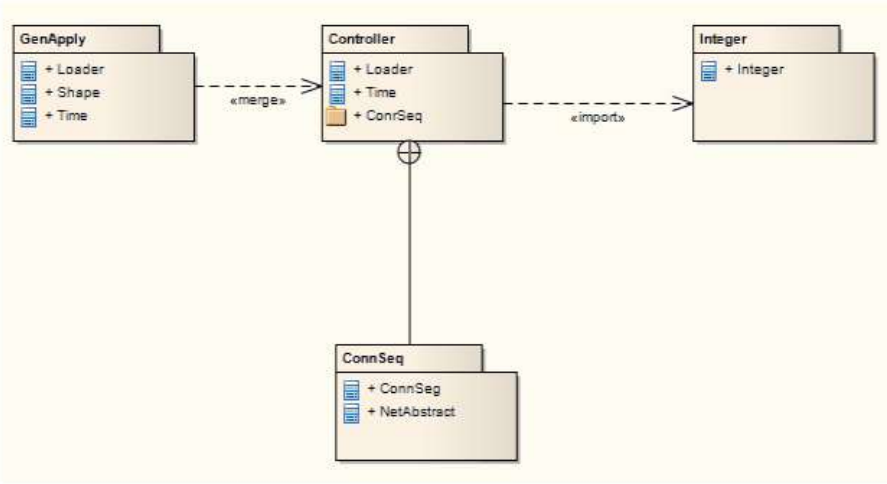
Name: Borrow Book	**ID: 2**
Stakeholders and goals: Student – wants to borrow book from university library	
Description: A borrower wants to borrow a book and must present his/her ID card to the system.	
Actors: Borrower	
Trigger: The borrower in the library brings a book(s) to the librarian's desk	
Normal flow: 1. The borrower swipes his/her card through the card reading machine 2. The system checks the ID of the card in the database 3. The borrower passes the book through the bar-code scanner 4. The system displays the date for returning the book 5. Repeat steps 3-4 for each book to borrow 6. End	
Sub-flows: None	
Alternative/Exceptional flows: 2a Invalid card: The *Invalid Card* use case is performed 2b Unpaid fine: The *Pay Fine* use case is performed 2c Book overdue: 2c1 – The system displays a message displaying the book overdue, the days overdue, and whether a fine must be paid	

Logical View

Package diagram -

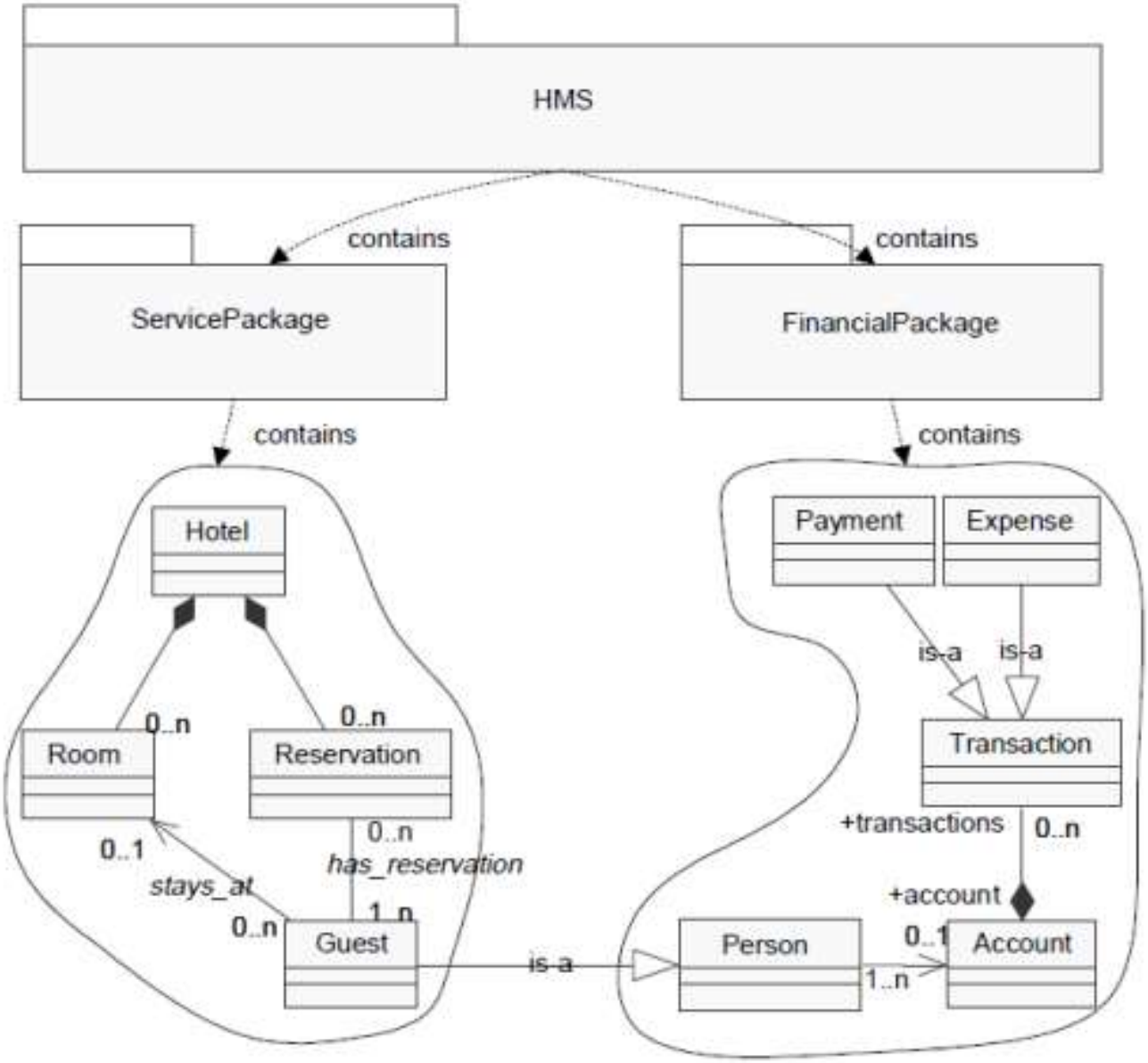
To show group of classes with a namespace





Often used to represent division of work among teams in a larger project and each team will develop code for a specific part of the large project. Each package diagram has many classes - a subset of which will be used from code in other packages.

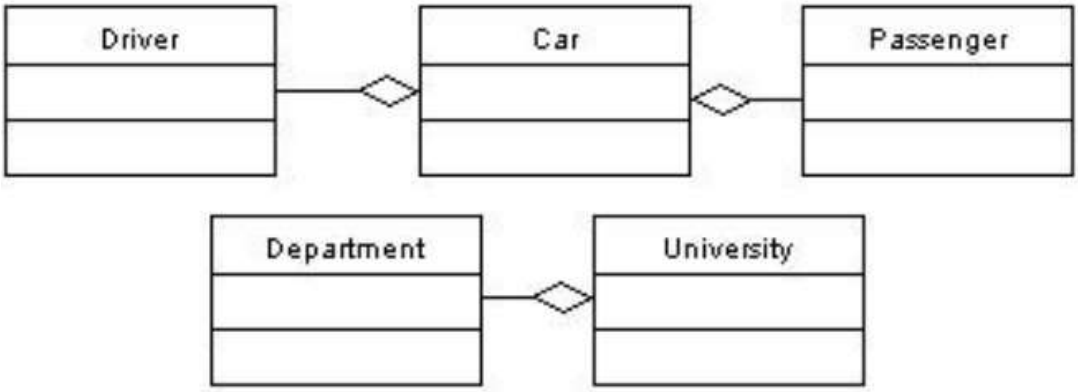
Package to Class Diagrams



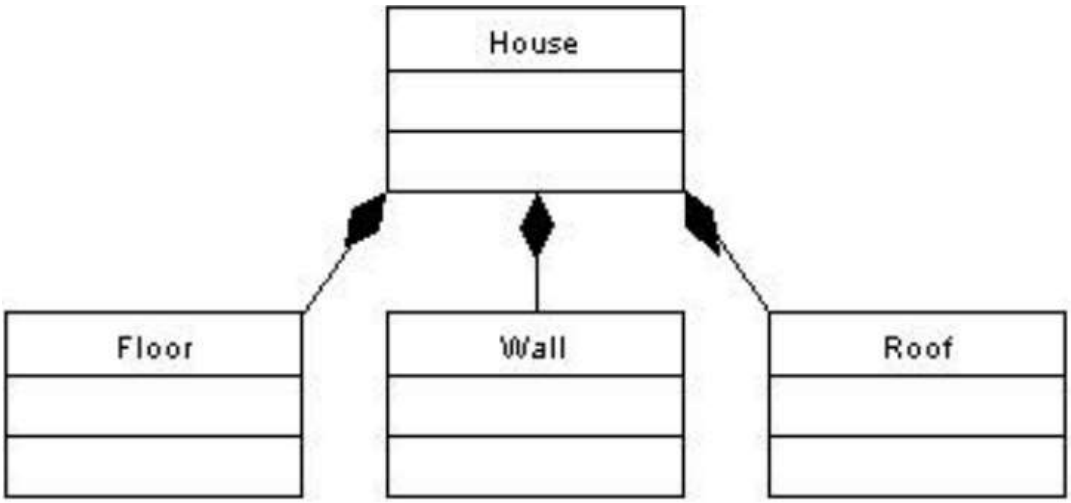
Class diagrams -

Relationships -

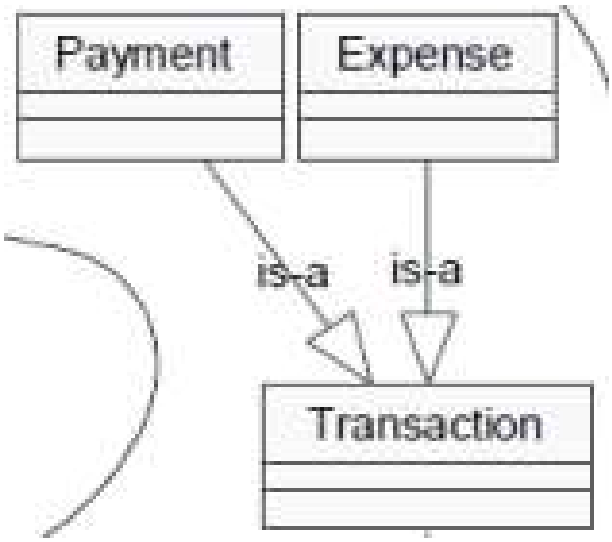
1. **Aggregation -**
    a. A special form of association where it is an 'assembly-component' ("is a part of") relationship.
    b. <u>Objects can exist independently</u>
    c. Eg.



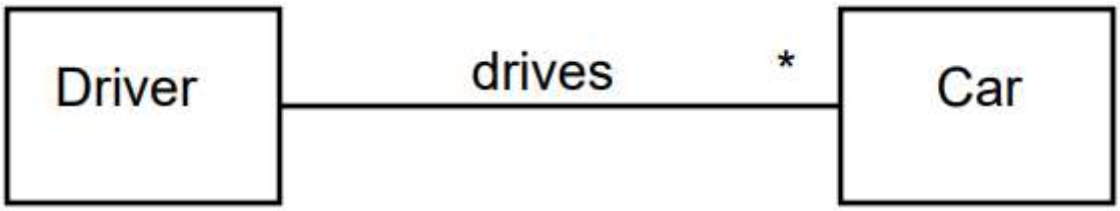
2. **Composition -**
    a. A special form of association where an object "is composed of" another object
    b. <u>Objects cannot exist independently</u>
    c. Eg.



3. **Inheritance -**
    a. A special form of association which consist of a super class and a sub class where subclass will inherit all properties of the superclass
    b. Eg. Transaction is a subclass of Payment and Expense where it inherits all the properties from both the class



4. **Association -**
    a. A generic relationship that is often the type of 'has -a', 'uses', 'communicates with', 'makes request of'
    b. Eg 1. A citizen can having driving/riding license (One to many association as the license can only have one owner but a citizen can have a driving, a boat, a motorcycle, and a plane license)
    c. Eg 2.



Exercise - (Class Diagram)

DDT is a business that rents diving equipment and boats.

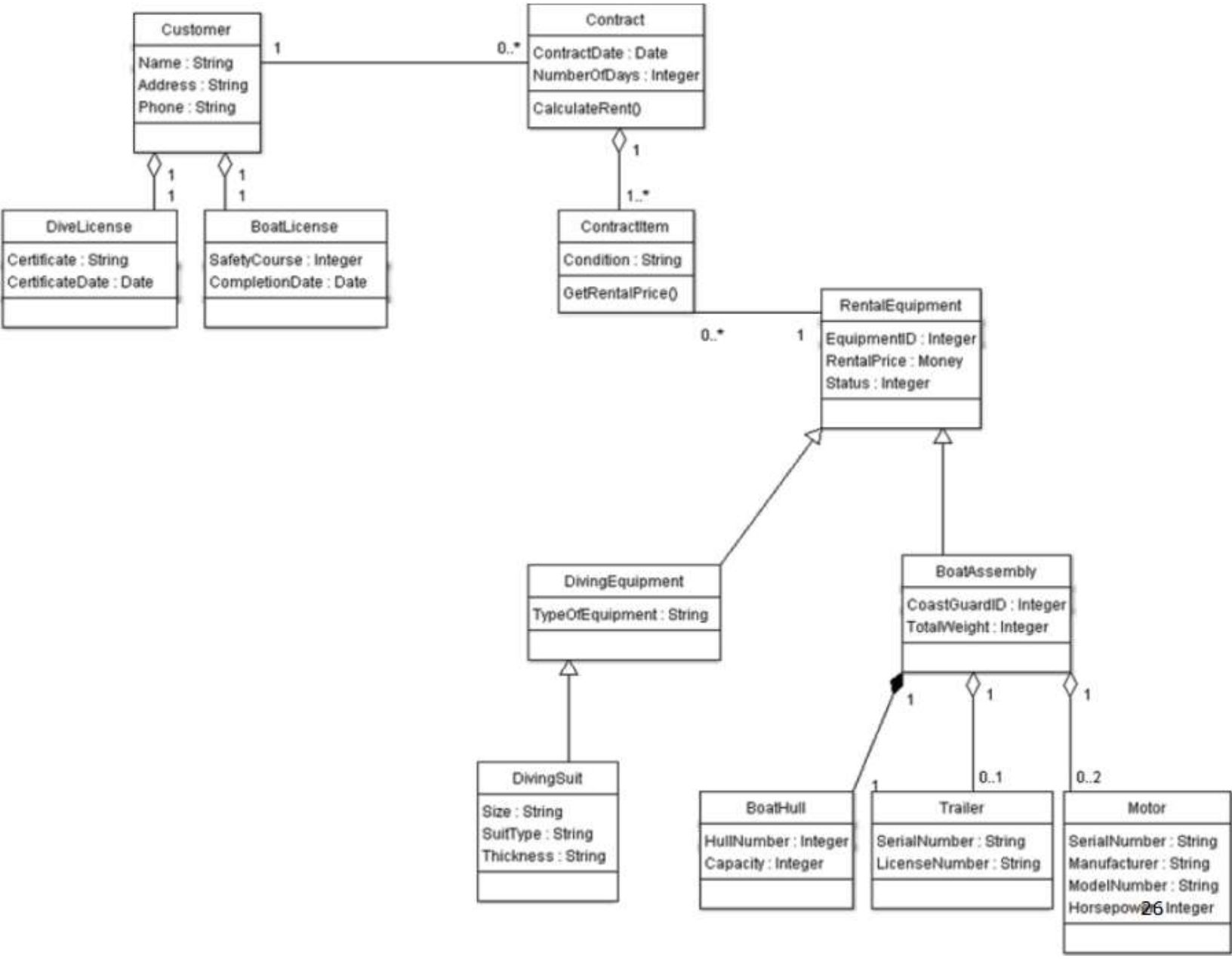
1. Customers can rent either diving equipment or boats from DDT. Each customer has a different license (boat license or dive license).
2. When the customer has made a decision about what to rent, a rental agreement, or contract, is produced and signed.
3. DDT rents all of the usual diving equipment such as tanks, regulators, weight belts, diving suits, and depth gauges. Diving suits come in various sizes, thicknesses, and types (dry and wet).
4. Some boats are rented with a trailer, some without a motor, and some with one or two motors. Trailers and motors always stay with the same boat; they are never rented by themselves or taken away from the boat they belong to (except to be serviced or replaced).
5. A customer can rent one piece of equipment or many pieces of equipment; for example, a diving customer can rent only a regulator or a collection of items for diving. A customer who already rented a boat can rent diving equipment. Thus, A contract contains many items of equipment. (In other words, each piece of rental equipment is rented many times, and over time it is associated with many contracts)

Tip highlight nouns

Answer next page

Answer

This is just one way of representing it.



State Diagram -

Behavioral model that specifies the sequences of states an object goes through during its lifetime in response to events, together with its responses to those events. A state diagram is drawn for every object to demonstrate how it responds to interactions.

State -

Set of values (attributes) that describe an object at a specific point in an object's life.

Event -

Something that takes place and changes a value of of an object (change state)

Term and Definition	Symbol
A State Is shown as a rectangle with rounded corners.	
An Initial State Is shown with a small filled in circle	
A Final State Is shown with as a circle surrounding a small solid filled-in circle. This represents the completion of activity.	
An Event Is an occurrence that triggers a change in state. It is used to label a transition.	**Event Name**
A Transition Indicates that an object in the first state will enter the second state. Is triggered by the occurrence of the event labeling the transition. Is shown as a solid line.	

5 Parts to a Transition -

Source state -

State affected by the event

Event trigger -

Interaction which made the object to change state

Guard condition -

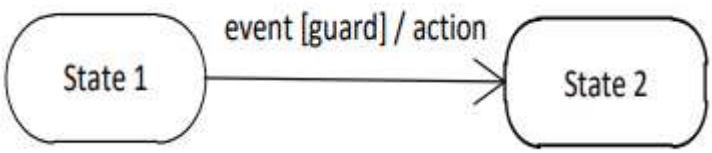
Optional expression that evaluates if action should change state of an object if conditions are true (eg. if pin entered correctly)

Action -

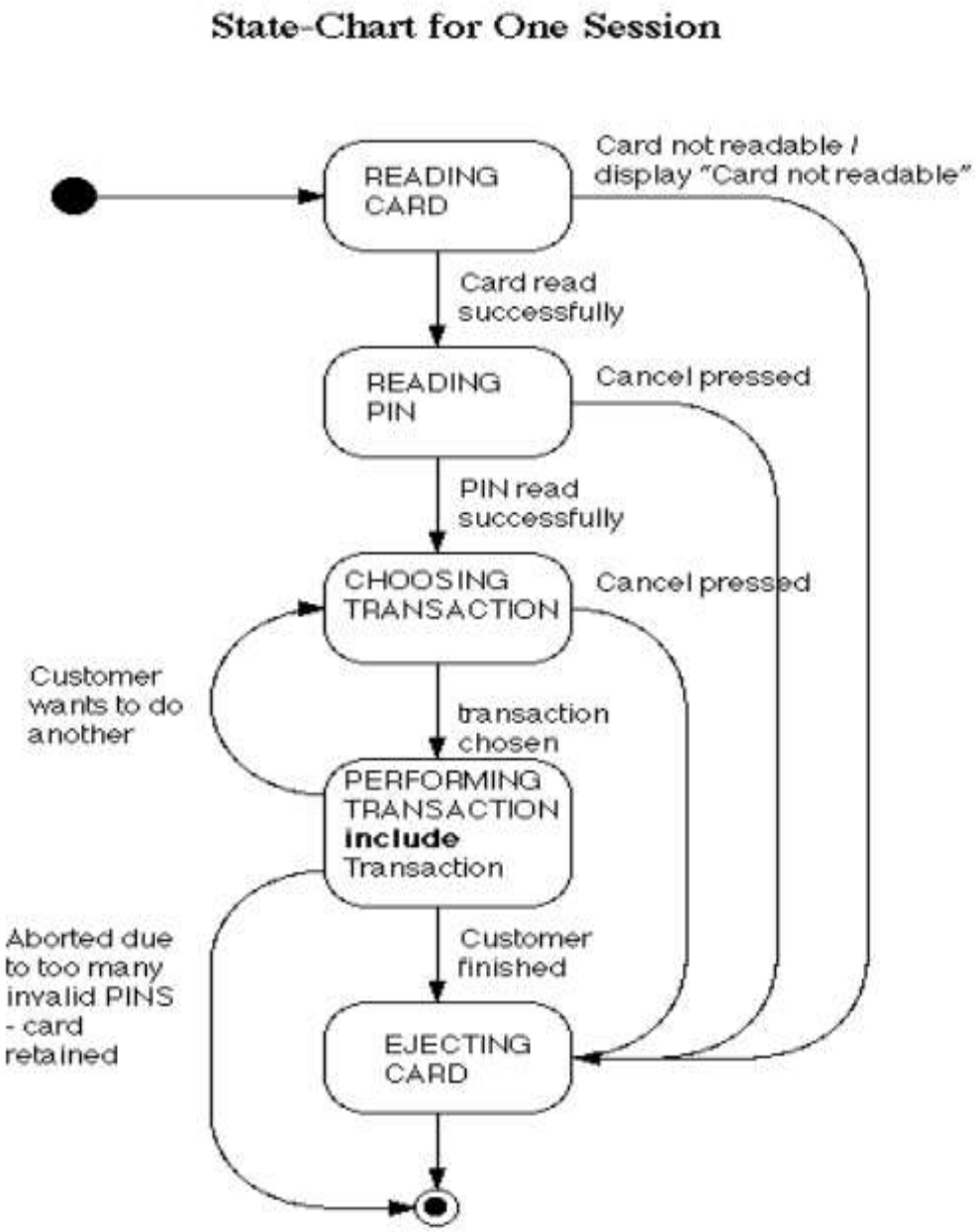
Executable atomic computation that may act directly on the object and indirectly on other object

Target state -

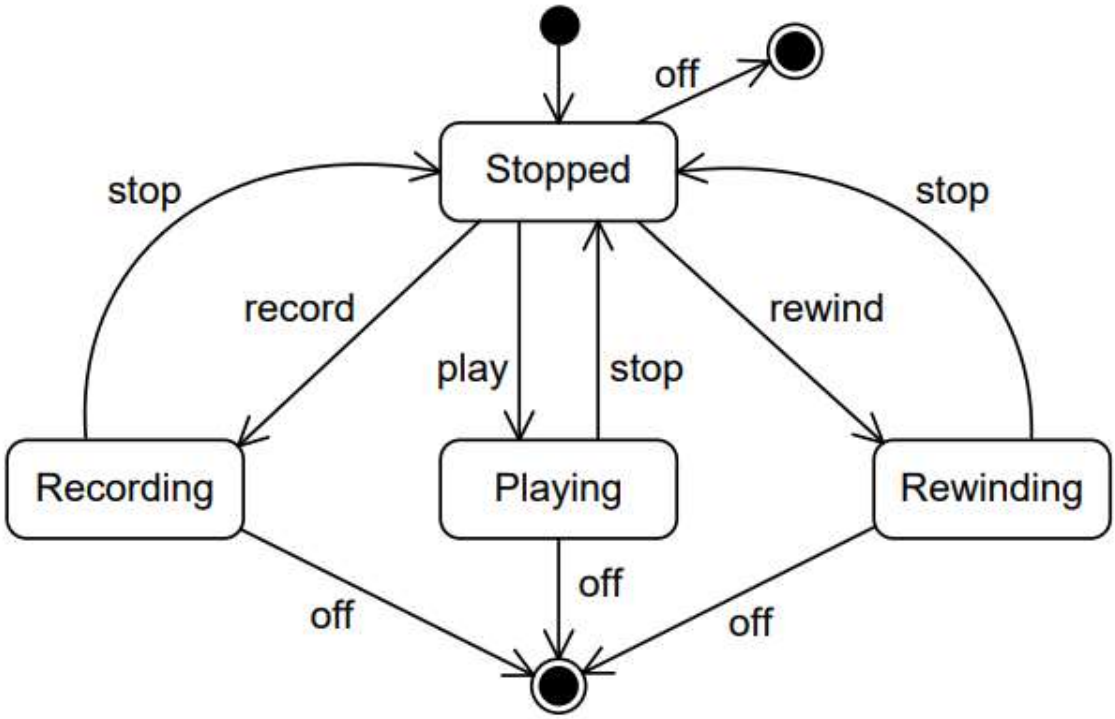
State after being affected by the event and action.



Eg. ATM state diagram



Eg. Tape recorder state diagram



Sequence Diagram -

Generally describe one use case per diagram. Shows how classes collaborate with each other in a sequential order.

Message exchange

Objects send messages to each other which leads to operations.

Objects

Shows instance name and or class name (instance_name : class_name)

Dotted line is the lifeline of the object.

X shows the point at which the object is deleted

Focus of control

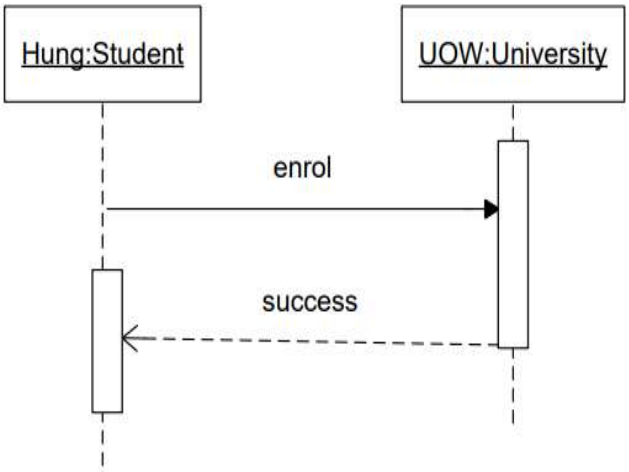
The narrow hollow rectangle shows which object has control (sending or receiving messages)

Simple message

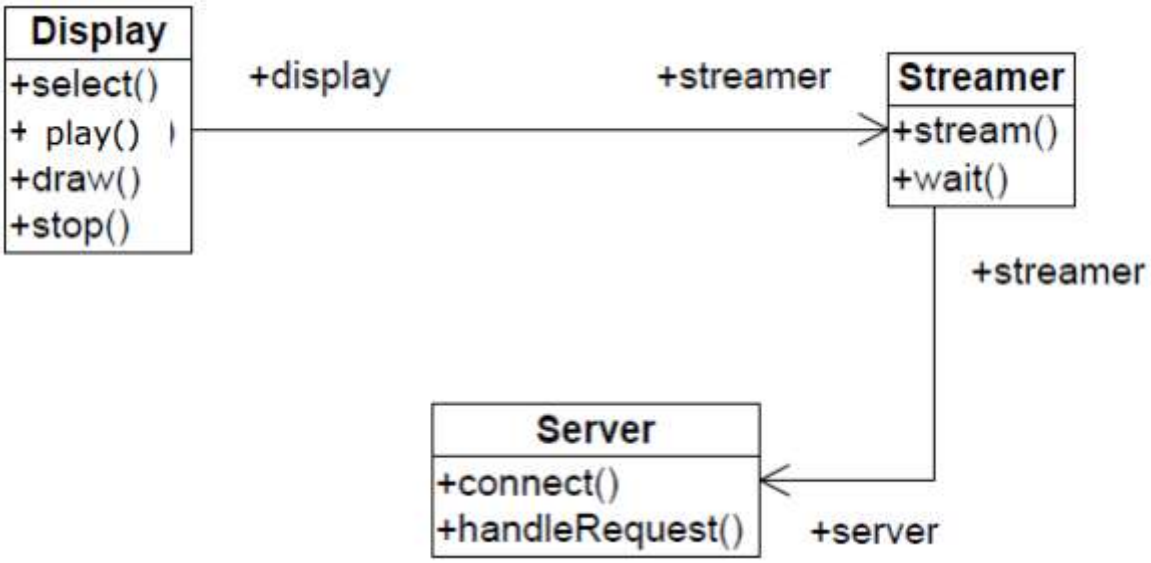
Shown by a line with a filled arrow head

Message return

Shown by dotted line with empty arrow head (not required, and the diagram will quickly become busy if show all returns)



Exercise - Video streamer



Use case: Watch a movie

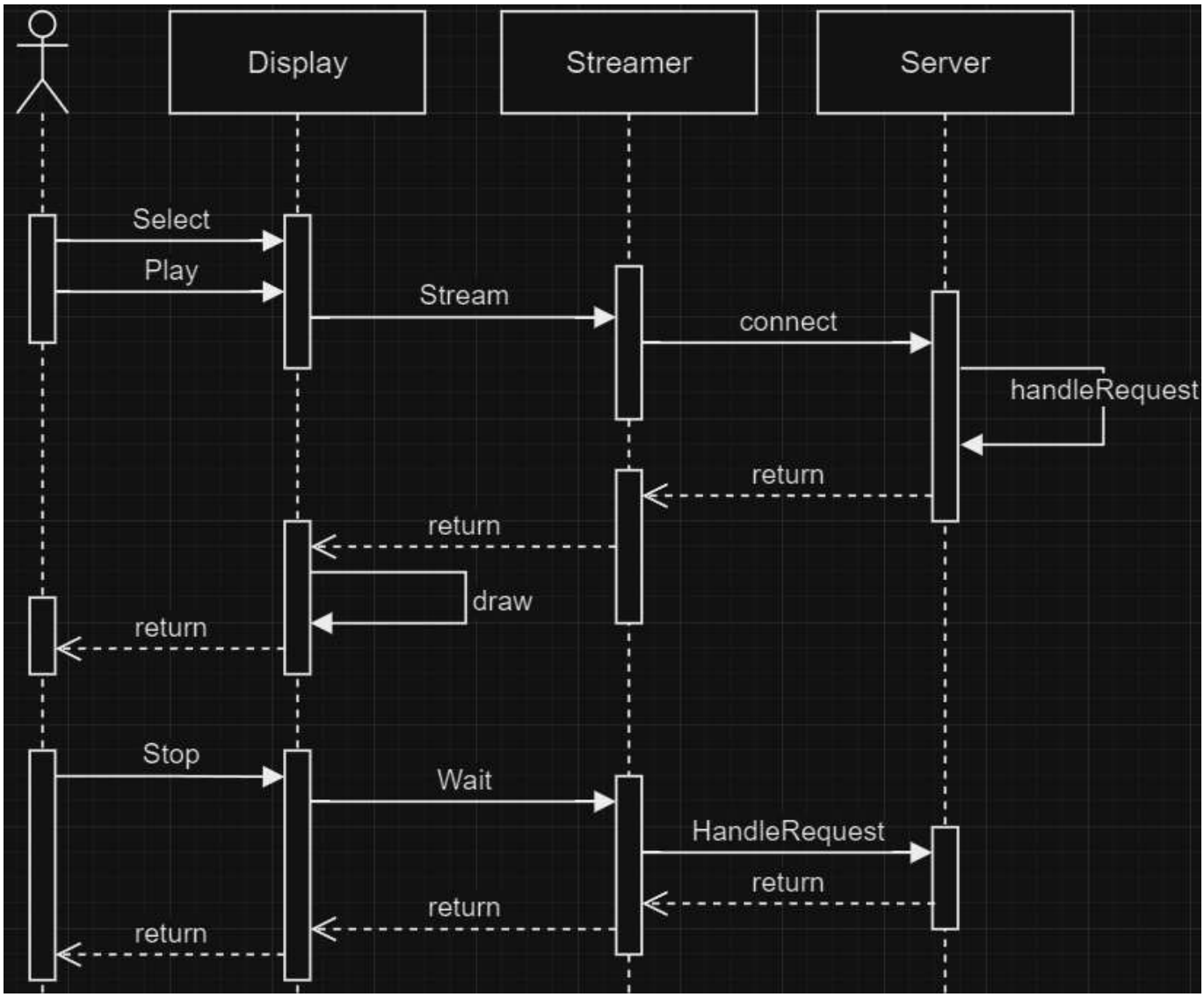
Actors: User

Triggering event: User turn on TV

Normal Flow of events:

1. The system displays a list of movies
2. The user selects a movie that she wants to see
3. She then starts playing the selected movie
4. The system retrieves the movie stream and renders the movie
5. The user stops viewing the movie.
6. The system stops retrieving the movie stream

Draw a sequence diagram.


Display
Streamer
Server
Select
Play
Stream
connect
handleRequest
return
return
draw
return
Stop
Wait
HandleRequest
return
return
return

B-C-E design

Entity-layer : (Database)

1. Holds data and is responsible for persistence
2. Represents business domain concepts (Domain class diagram)
3. Entity classes used to store and manage information in the system (Map directly to database)
4. Some entity class may be transient (do not get stored in DB only temporary)

Control-layer : (Controller)

1. Contains the rule about how to combine info and how to deal with interactions
2. Coordinates all the messaging between the entity class
3. They are the “glue” between boundary and entity classes
4. They contain the application - specific business rules (eg. password should have 8 character)
5. Control classes usually have one per use case (some use case may not need one created when system performs the use case and usually die when use case is performed)

Boundary-layer : (GUI)

1. Responsible for representing info to users and to receive user interaction.
2. Used to model interaction between the environment and the system
3. Actors interact with only boundary

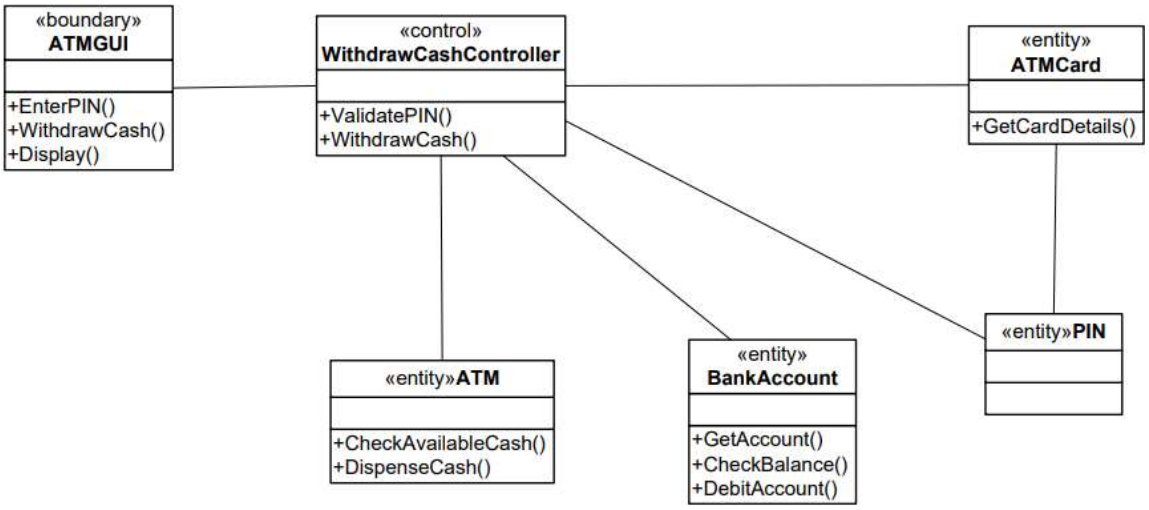
Basic rules of B-C-E

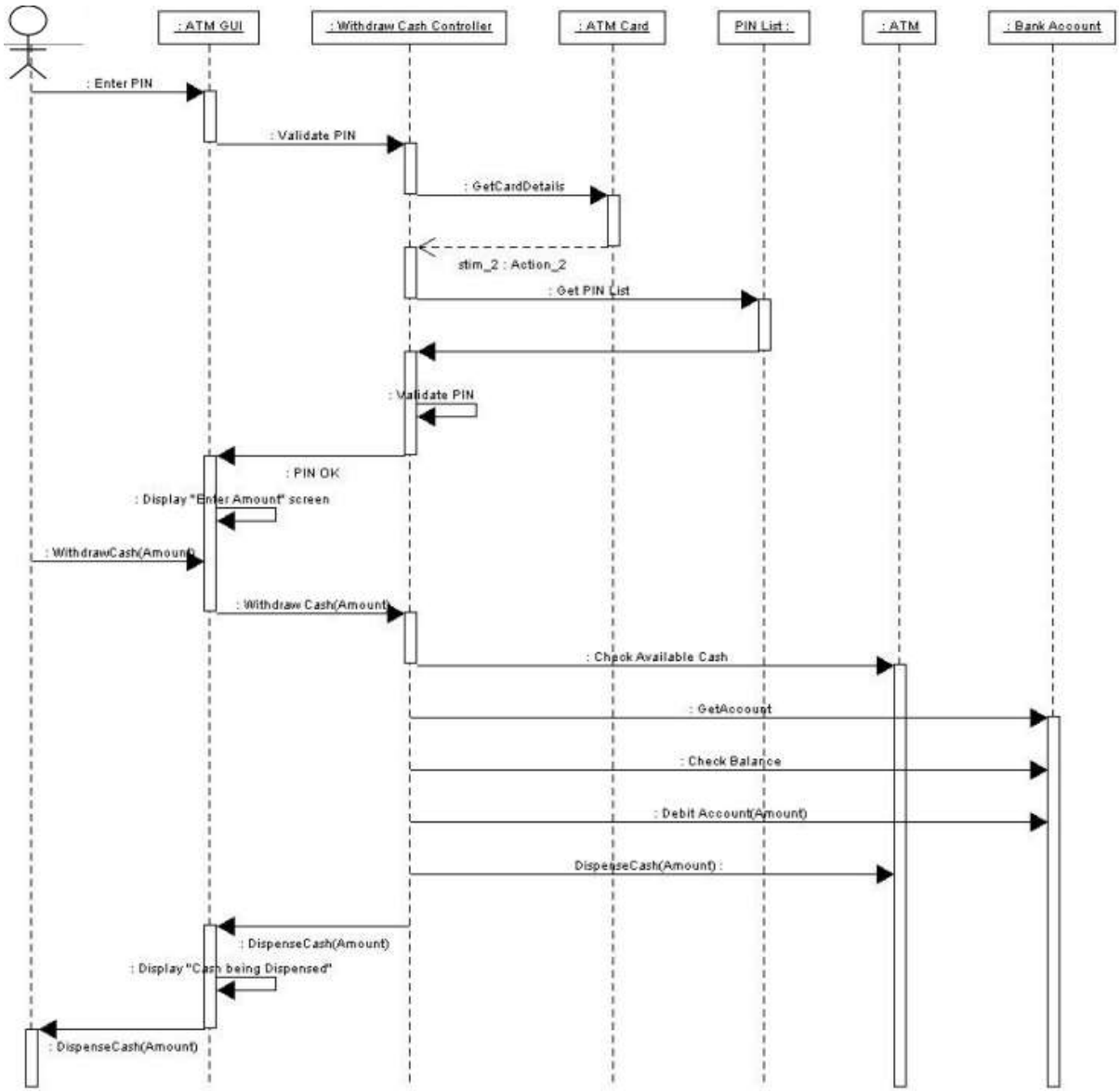
Actors can only talk to boundary classes

Boundary class can only talk to controller and actors

Entity classes can only talk to controller

Controller can talk to boundary and entity classes and to other controllers but not to actors


«boundary»
ATMGUI
+EnterPIN()
+WithdrawCash()
+Display()
«control»
WithdrawCashController
+ValidatePIN()
+WithdrawCash()
«entity»
ATMCard
+GetCardDetails()
«entity»ATM
+CheckAvailableCash()
+DispenseCash()
«entity»
BankAccount
+GetAccount()
+CheckBalance()
+DebitAccount()
«entity»PIN

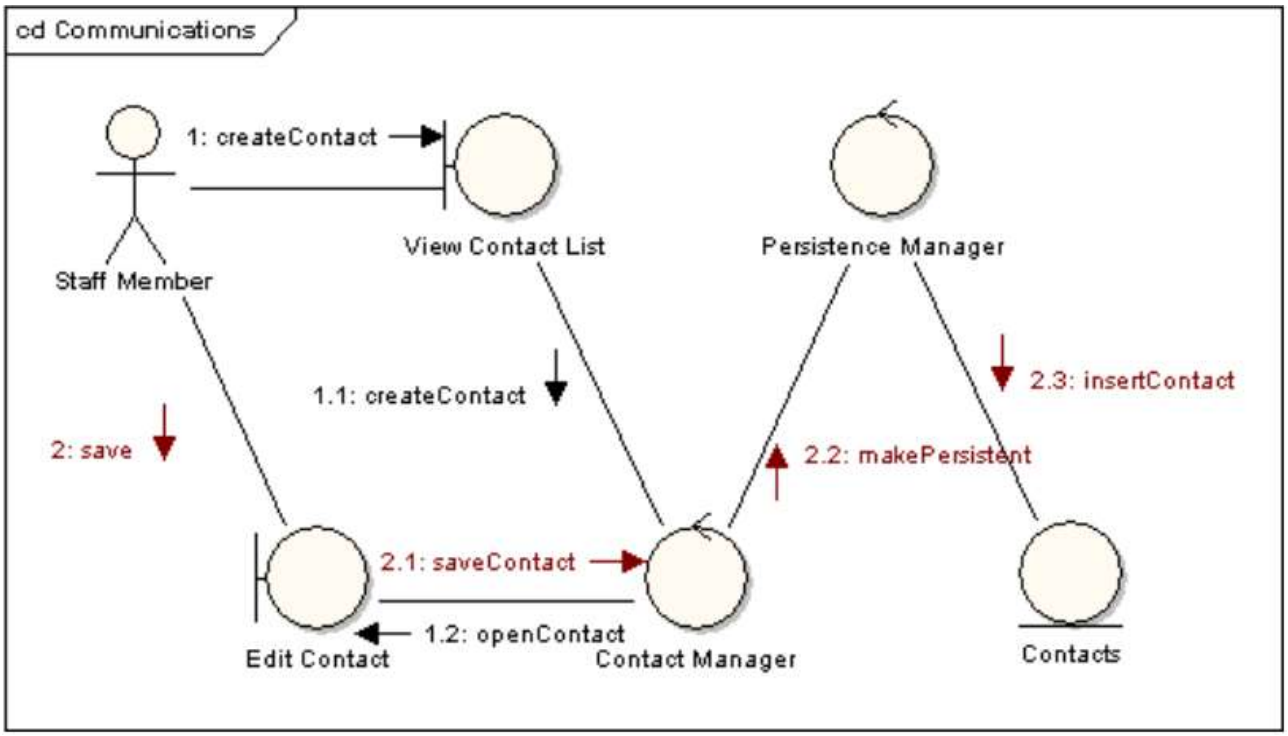

: ATM GUI
: Withdraw Cash Controller
: ATM Card
PIN List :
: ATM
: Bank Account
: Enter PIN
: Validate PIN
: GetCardDetails
stim_2 : Action_2
: Get PIN List
: Validate PIN
: PIN OK
: Display "Enter Amount" screen
: WithdrawCash(Amount)
: Withdraw Cash(Amount)
: Check Available Cash
: GetAccount
: Check Balance
: Debit Account(Amount)
DispenseCash(Amount) :
: DispenseCash(Amount)
: Display "Cash being Dispensed"
: DispenseCash(Amount)

Process View -

Communication Diagram

Convey same information as a sequence diagram with a difference in style

- Sequence diagram focus on message occurring over a timeline
- Communication diagram focus more on links between participating objects
    - Sequence of messages is still shown but sequence is less clear and relationship is more clear



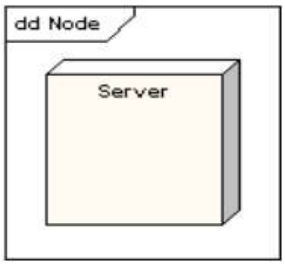
Deployment View

Deployment Diagram -

Models the run-time architecture of a system. It shows the configuration of the hardware elements (nodes) and show how software elements and artifacts are mapped onto those nodes.

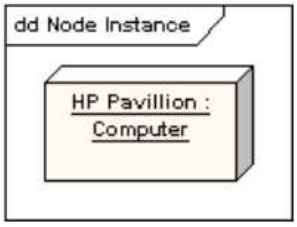
Node

A hardware (or software) element, shown as a three-dimensional box shape



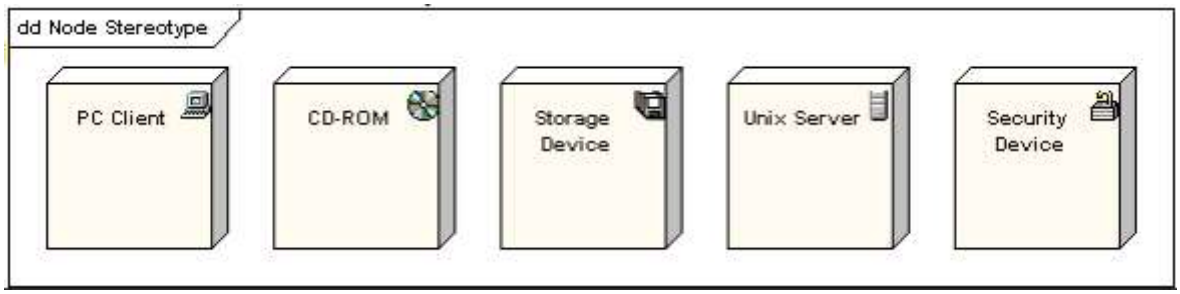
Node Instance

Name and base node type



Node Stereotypes

Standard stereotypes are provided for nodes : <<cdrom>>, <<computer>>, <<disk array>>, <<pc>>, <<pc client>>

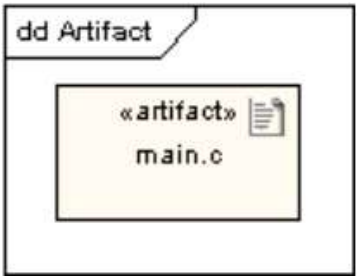


Artifact

Any product of the software development process : source files, executables, design documents, test reports, prototypes, etc.

An artifact is denoted by a rectangle showing the artifact name, keyword and document icon.

Appear in Deployment diagrams as contents of the nodes on which they are deployed

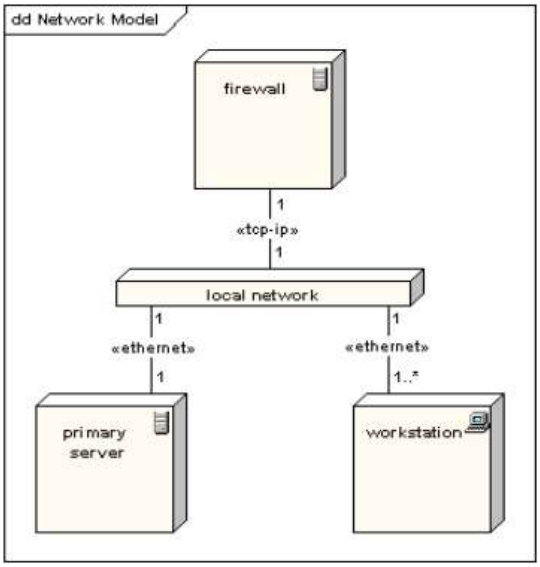


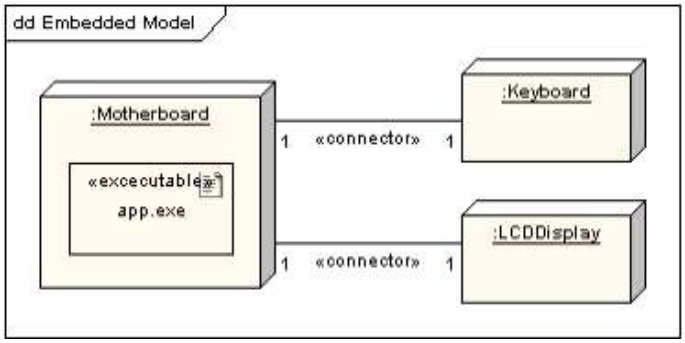
Association

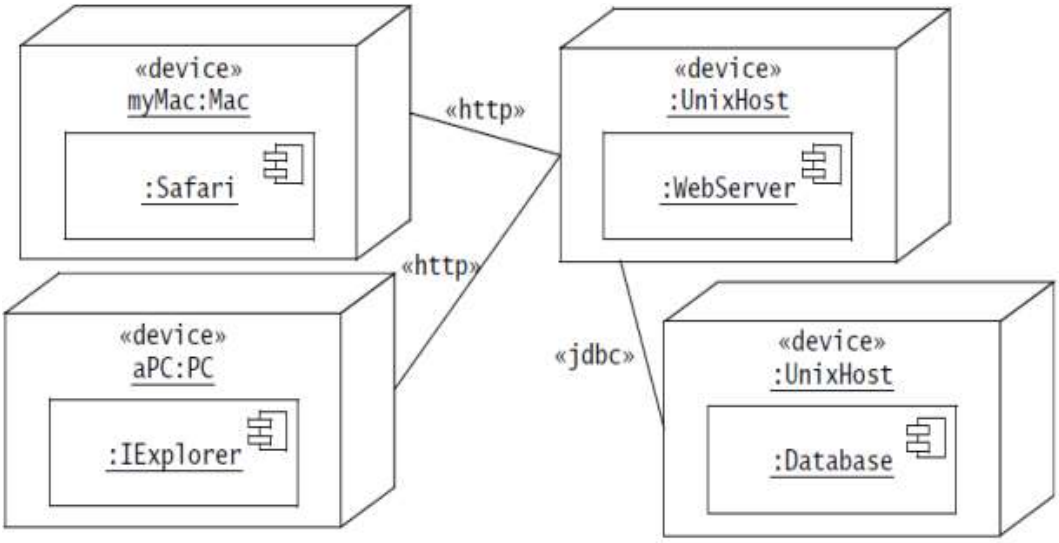
An association represents a communication path between nodes

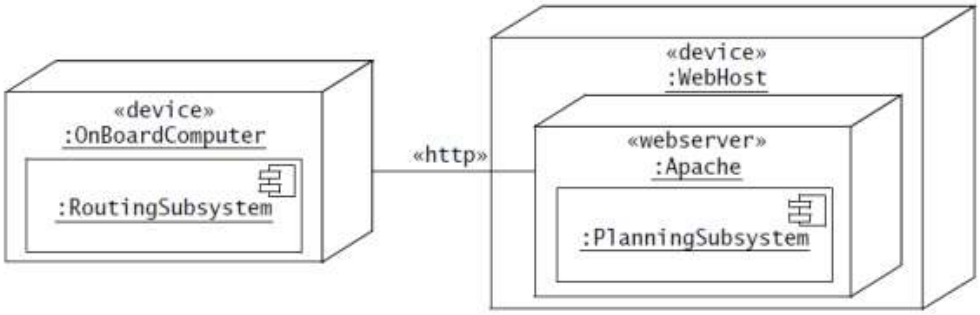
Node as Container

A node can contain other elements, such as components or artifacts.

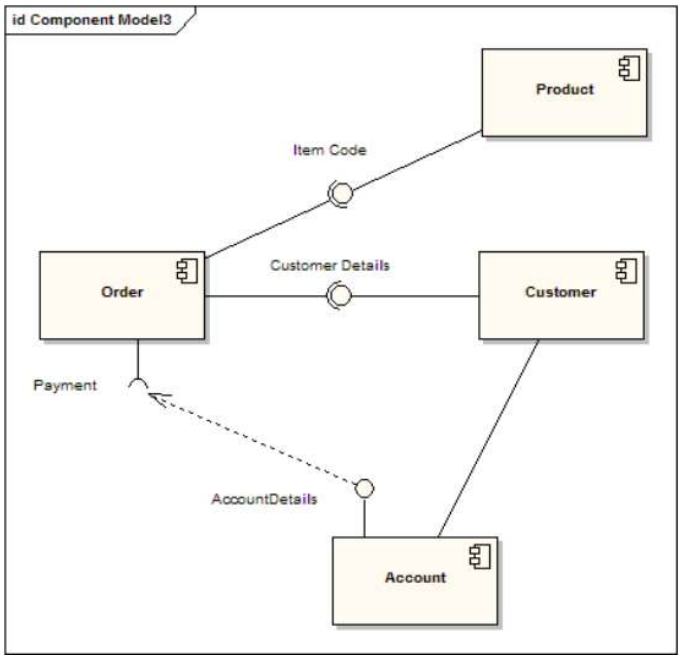








Implementation View



Component diagram

Component diagrams illustrate the piece of **software, embedded controllers,** etc., that will make up a system.

A component is usually implemented by one or more classes (or objects) at runtime.

Assembly connector

It bridges a component's **required interface** (Component1) with the **provided interface** of another component (Component2); this allows one component to provide the services that another component requires

Chapter 5 Testing -

Verification vs Validation

Verification

- The software should conform to its specification

Validation

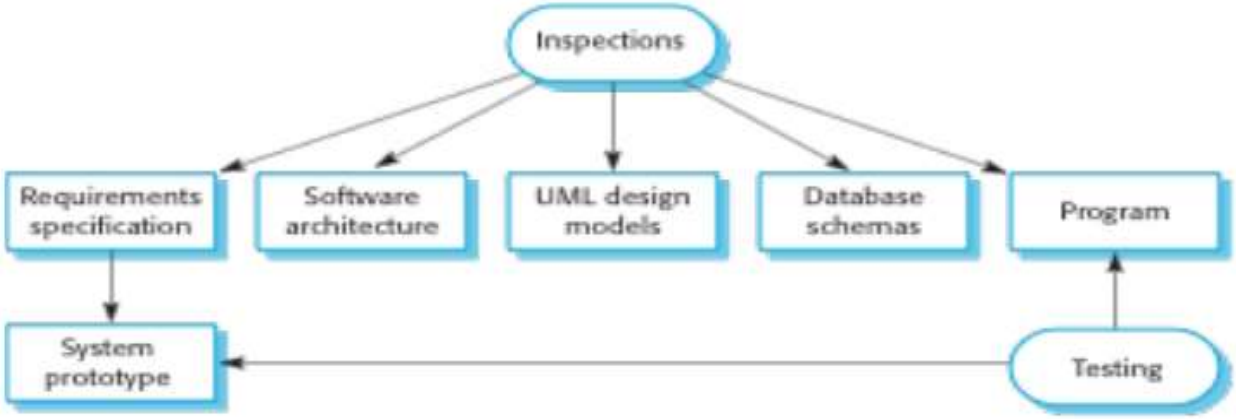
- The software should do what the user require

V & V must be applied during each phase.

2 Principal objectives -

1. To discover and rectify defects in systems
2. To access if the system is usable in an operational situation

Static and Dynamic Verification



Software inspection (Static Verification)

Checking of code, UI, without the need to execute the program

- Concerned with analysis of the static system representation to discover problems
- Aim of discovering anomalies and defects

- Inspection do not require execution of system so may be done before implementation
- Can be applied to any representation of the system (requirement, design, test data, etc)
- Effective technique for discovering program error

Fault class	Inspection check
Data faults	Are all program variables initialised before their values are used? Have all constants been named? Should the lower bound of arrays be 0, 1, or something else? Should the upper bound of arrays be equal to the size of the array or Size -1? If character strings are used, is a delimiter explicitly assigned?
Control faults	For each conditional statement, is the condition correct? Is each loop certain to terminate? Are compound statements correctly bracketed? In case statements, are all possible cases accounted for?
Input/output faults	Are all input variables used? Are all output variables assigned a value before they are output?
Interface faults	Do all function and procedure calls have the correct number of parameters? Do formal and actual parameter types match? Are the parameters in the right order? If components access shared memory, do they have the same model of the shared memory structure?
Storage management faults	If a linked structure is modified, have all links been correctly reassigned? If dynamic storage is used, has space been allocated correctly? Is space explicitly de-allocated after it is no longer required?
Exception management faults	Have all possible error conditions been taken into account?

Reviewer: Joe Bloggs
Team: DT1
Part/Level: WCMS-HLD-1-03
Cycle: 1

Date: 10 March 2003
Owner: M. Bean
Moderator: M. Python

Time Information

Start Time:	10:15
End Time:	11:40
Interruption:	25 min
Delta Review Time:	60 min

Defect Information

Location	Description	Severity (maj/min)	Classification*
1.0.1	TBD indicates incomplete specification	maj	
1.0.2 (L3)	Incorrect spelling of **scenario**	min	
UCD	Missing use case for adding resources to courses	maj	
UCD	Term course used inconsistently	maj	
SD 4.3	Message to list students incorrectly passed to instance of Student	maj	
SD4.10	Name does not match UCD	maj	
SD4.11	Message to list courses incorrectly passed to instance of Course	maj	
SD4.23	Sequence diagram describes interaction not defined in requirements – no corresponding use case	maj	
SD4.23	**Course** spelled incorrectly	min	
SD4.24	Sequence diagram describes interaction not defined in requirements – no corresponding use case	maj	

Summary
Number of major defects: 8
Major / Total defects: 8 / 10 = 80%
Number of minor defects: 2

Size of reviewed artefact: 15 pages
Rate of review: 15 pages / hour

Defect detection rate (maj/hr): 8 defects / hour

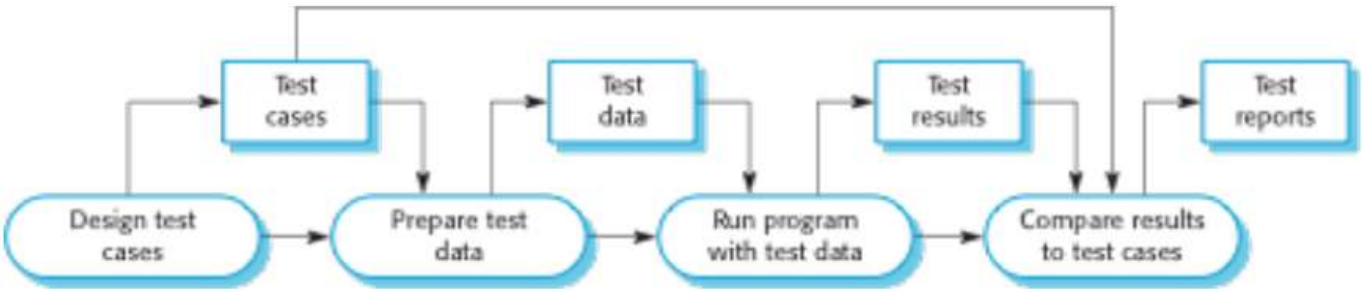
8

Advantages of inspections

1. Inspection is a static process so errors will not interact with each other.
2. Incomplete version of a system can be inspected
3. Inspection can also consider broader quality attribute such as compliance with standards, portability and maintainability

Software testing (Dynamic Verification)

- Concerned with executing and observing product behavior
    - The system is executed with test data and observed
- **Process of executing a program is to**
    1. **Force it to work incorrectly**
        - A good test is one that has a high probability of finding an undiscovered error
    2. **Demonstrate that it works correctly**
        - To show certain types of fault do not exist
- Typically, testing takes about 30-40% of budget for system development



Test Case Design

Testing consist of a set of test cases which are systematically planned and executed

1. Each test case is uniquely identified
2. Purpose of each test case is stated
3. Expected result are listed in the test case and should include all outputs, resultant changes and error messages

7. Test for call status	
Actions	a. Before placing a call b. On placing a call c. On the user ending the testing session by hanging-up the call d. On occurrence of any exception during the call
Expected Result	a. The call status must be "Normal Close" b. The call status must be "In Session" c. The call status must be "Normal Close" d. The call status must be "Exception Close"
Pass/Fail	
Observations (if any)	

Testing is conducted by 2 - 3 groups:

- Software developer
- An independent test group (for larger projects)
- The customer

Development Testing

Includes all testing activities that are carried out by the team developing the system

1. **Unit testing, where individual program units or object classes are tested**
    a. Unit testing should focus on testing the functionality of objects or methods
    b. Unit may be individual function or method within an object or object classes with several attributes and methods
    c. Tests all operations associated with an object, setting and interrogating all object attributes, exercising the object in all possible states

    Automated testing

    - Whenever possible, unit testing should be automated so that test are run and checked without manual intervention
    - In automated unit testing, make use of a test automation framework (JUnit, cppUnit) to write and run program tests
    - Unit testing framework provide generic test classes (eg. cppUnit::TestFixture) that you extend to create specific test cases.

    Types of unit test case:

    - First should reflect normal operation of program and should know component works as expected
    - Other test should be based on testing experience where common problems arise with the use of abnormal inputs to check that it is properly processed and not crash the component

2. **System testing, where some or all of the components in a system are integrated and the system is tested as a whole**
    a. System testing should focus on testing component interactions
3. **Integration testing, where components are combined and 2 modules will interact with each other**

Testing techniques

1. **White box testing - Examine source code of each module**
    a. Applied to increase logic coverage
    b. Basic forms
        i. Statement coverage: each statement is executed at least once
        ii. Decision (branch) coverage: each statement, each decision takes on all possible outcomes at least once
        iii. Condition coverage: each statement, each decision, each condition in a decision takes on all possible outputs at least once
        iv. Path coverage: each statement, all possible combinations of condition outcomes in each decision occur at least once

After reading code, doing tests like passing in float values to int variables to see if the system crashes.

2. **Black box testing - Testing the product functionalities**
    Testers evaluate the functionality of the software without looking at the internal code structure. Designed to validate functional requirements without regards to source code. Generally, a set of results is derived to ensure that modules produce correct results
    Types of blackbox testing
    - Behavioral - Test only functional part of an application to make sure the behavior of the software is expected
    - Specification-Based - Create test scenarios/cases based on software requirements and specifications

    - Input-Output Testing - Passes input data to make sure the actual output data matches the expected output data

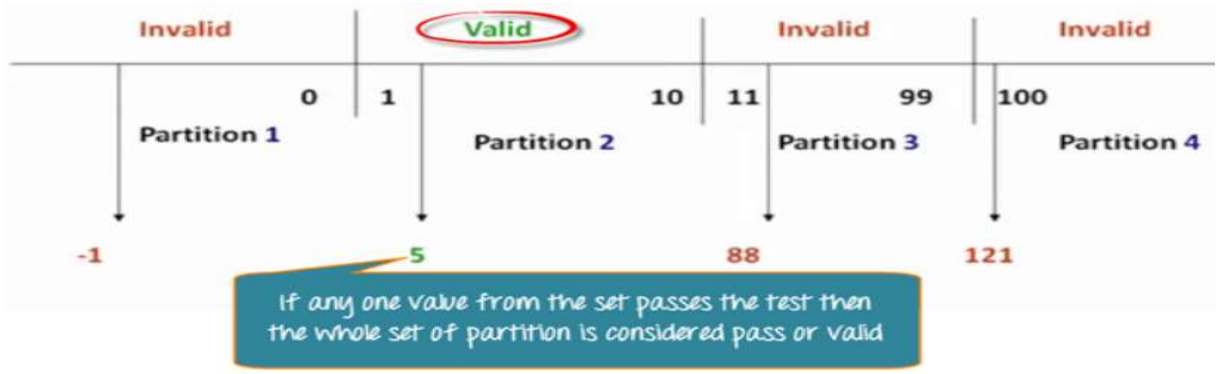
Black box testing systematic vs Random testing -

- Random (Uniform)
    - Picks possible inputs uniformly, avoid designer bias, treats all inputs equally valuable
- Systematic (Non-uniform)
    - Try to select inputs that are especially valuable
    - Usually by choosing representatives of classes that are apt to fail often or not at all
    - Create test case scenarios like negative, positive, zeroes, large numbers

Random testing will take too long therefore black box testing (systematic) will be used instead.

Black box testing strategies

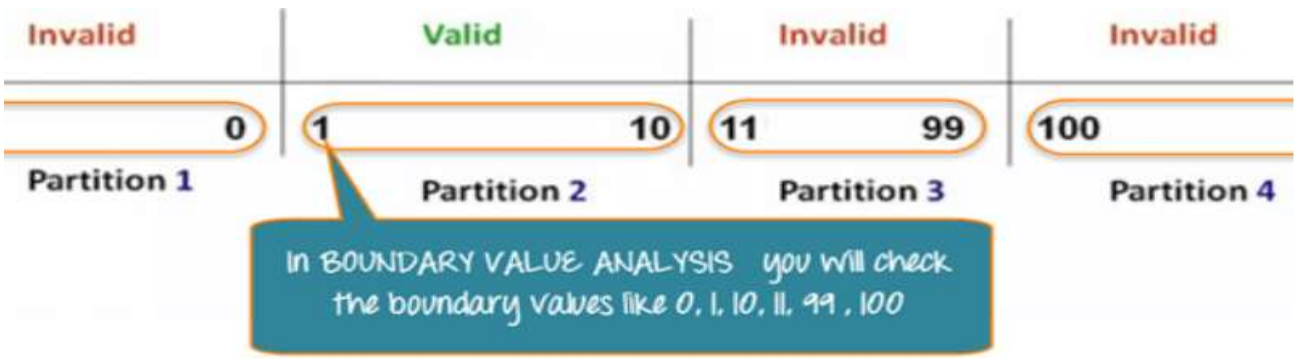
- **Equivalence Partitioning**
    - Testing method that divides the input domain of a program into sets of data from which test cases can be derived
    - Strives to define a test case that uncovers a class of errors , thereby reducing the total number test cases that must be developed.
    - Eg. Pizza website that gives free pizza has a website that asks user to input value 1-10. While values 11-99 are considered invalid, and an error message will appear, “Only 10 Pizza can be ordered”



- **Boundary Value Analysis**

  A test case design technique that complements equivalence partitioning.

  - Rather than selecting any elements of an equivalence set, it selects test cases at the edges of the set.
  - Boundary value analysis leads to a selection of test cases that exercise boundary values.



Integration testing

Test complete systems or subsystems composed of integrated components. (Modules are combined together and then the interaction between 2 modules are tested)

If unit testing has been effective, then faults that remained to be found in integration testing will be primarily interface faults.

Testing effort should focus on interfaces between units rather than their internal details

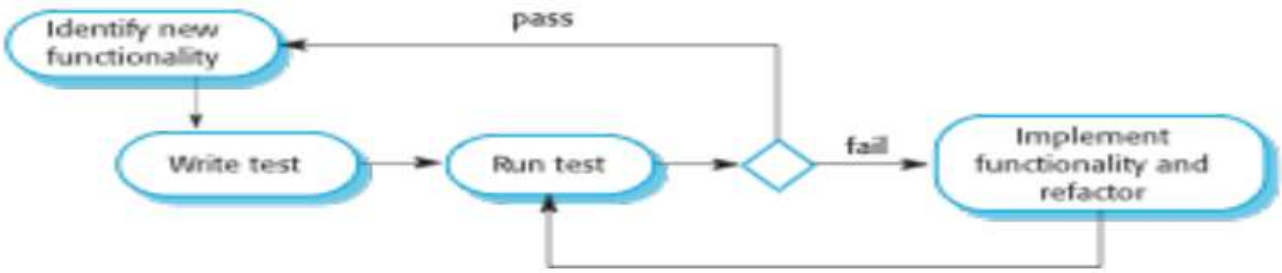
System testing

System test encapsulate the environment as well as the product and include:

1. Function Testing
    a. Test functions of system
2. Performance Testing
    a. Test non functional requirement, e.g reliability, availability, etc
3. Acceptance Testing
    a. Alpha testing and Beta testing

Test-driven Development

1. TDD is an approach to program development in which you interleave testing and code development
2. Test are written before code and 'passing' the test is the critical driver or development
3. Code is developed incrementally, along with a test for increment. You dont move to the next increment until the code that you developed passed its test
4. TDD is introduced as part of agile methods such as Extreme programming. However it can also be used in other development cases



Identify function > Write test > Run test > Fail test > Code/Debug > Repeat until pass > Then identify another function.

Benefits

Code coverage - Every code segment has one associated tests so all code written has at least once test

Regression testing - Checking if changes have not broken previously written code. Developed incrementally as program is developed

Simplified debugging - When a test fails, it should be obvious where the problem lies. The newly written code needs to be checked and modified

System documentation - Tests are a form of documentation that what the code should do.

Test-driven development with JUnit

Provides 4 important features -

1. Test fixtures
2. Test suites
3. Test runners
4. JUnit classes

Each test should be derived from abstract class TestFixture

Override 2 functions:

1. Setup (using the annotation @Before)
    a. Operations that would be invoked prior to each individual test, eg. create new clean instances of class ready for next test
2. Teardown (using the annotation @After)
    a. Operations that would be invoked after each individual test, eg. deletes instances

TestSuite

Groups a number of tests (using TestFixture) so that they cal all be run

TestRunner

Driver Program

JUnit Assert

Provide a set of assertion methods useful for writing tests.

Chapter 6 Continuous integration and delivery (CI/CD) -

Integration

1. Software teams often have multiple developers working on the same codebase at the same time (independently)
    a. Dev A work on feature 1 while Dev B work on feature 2
2. Once they have finished, they need to integrate their work into the main codebase.

Continuous Integration

1. // Practice where developers in a team integrate their work frequently
2. Developers usually integrates several times a day
3. Each integration is verified by an automated build: compile the code and also run automated test

Practices of Continuous Integration

1. Maintain a single source repository using a version control system such as GitHub
2. Everything required to build the software app should be in the repository (code, test scripts, test date, properties files, database schema, third-party libraries, etc.)
3. Automate the process of turning the source code into a running software app, issuing a single command and the whole build process will run automatically.
4. Make your build self-testing, including automated tests as part of the build process. All automated unit tests and acceptance tests should be run as part of the continuous integration practice.
5. Everyone commits (integrate their changes) to the main codebase everyday
    a. Integration is a way in which developers can “inform” other developers about the changes they have made
    b. By integrating their changes frequently, developers quickly find out if there is conflict between the changes.
    c. Errors, conflict, bugs, etc. can be detected and early and rectified quickly
6. Every commit should build the mainline (the main codebase) on an integration machine

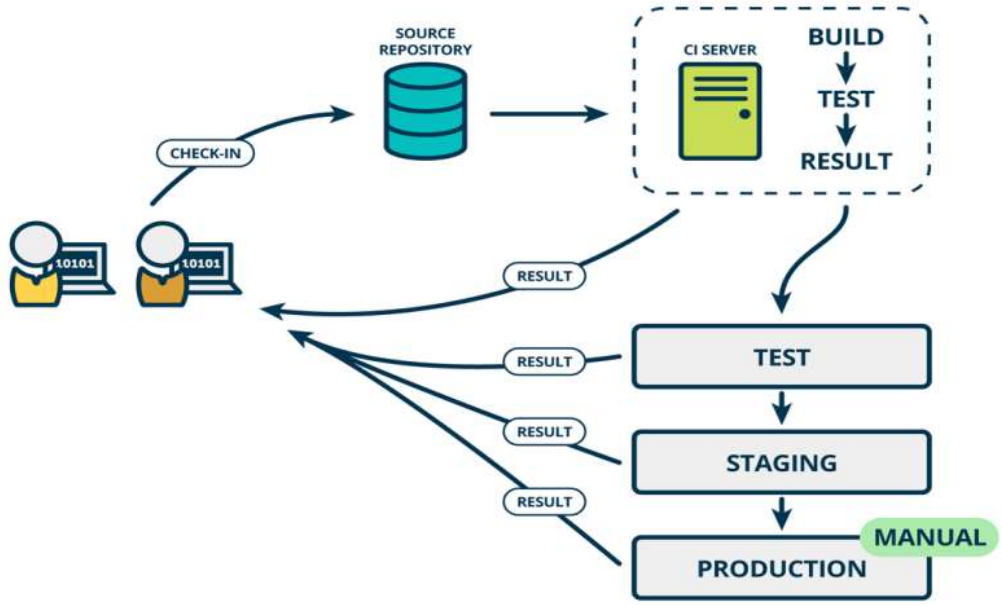
    a. Regular builds happen on an integration machine and only when this integration build succeeds, the commit will then be considered to be done.
7. Fix broken builds immediately
    a. A key part of doing a continuous build is that if the mainline build fails, it needs to be fixed right away
8. Keep the build fast
    a. The whole point of continuous integration is to provide rapid feedback
    b. If the build is not fast, developers will commit less often and will be provided feedback on problems less often
9. Test in a clone of the production environment

Continuous Delivery

CD is a further step beyond Continuous integration.

Each time changes are pushed to the codebase, the new code is automatically built and tested in the staging environment.

The staging environment addresses non-functional requirements such as security, load-balancing, redundancy, and scalability. These may not be covered in the development environments



The deployment of the software application to a production environment is still done manually in continuous delivery.

Continuous Deployment

An additional step to automate deployment

The software application is deployed automatically. Everytime code changes are pushed to the codebase, it will be automatically built and tested and if successful, it will be pushed to production.

Infrastructure as code

IaC is the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.

Infrastructure as code: the whole IT infrastructure can be treated as if they are software. (Code the software to control the hardware) The whole IT environment can be set up, configured and changed automatically through writing code.

Infrastructure as code is prerequisite for common CI/CD practices

Chapter 7 Devops -

Traditional development models consists of 2 teams:

Development Team - Design, Code, Test

IT Operations Team - Deploy, Support, Sys Admin, Maintenance

And there is a wall of confusion between the 2 teams as they have different mindset and use different tools and environments.

Cons

1. **Different Objectives -**
    a. Operations wants the software to be stable and views each change and new release as a potential danger to stability
    b. Development team is paid to change and add new features to the software
2. **Different Perspective -**
    a. Operations view the application as a black box and monitor it with suitable tools (eg. CPU utilization, kernel behavior)
    b. Development team knows the internal working of the application
3. **Different Environment -**
    a. Operations run application in production environment
    b. Developers run application on localhost
4. **Different Requirements-**
    a. Operations is driven by non-functional requirements (Stability, Reliability)
    b. Development is driven by functional requirements (Features)
5. **Client perspective -**
    a. When there are errors, clients do not know who to approach

DevOps

Software engineering methodology which aims to unify software development (Dev) and IT operation (Ops)

Development and operations team are no longer separate (i.e. no confusion wall)

- Increased Collaboration
- Shared Responsibility
- Support Autonomous Team
- Value Feedback
- Automation

Lifecycle

- Development
    - Agile, incremental and interactive development
- Testing
    - Test-driven development, acceptance testing
- Integration
    - New functionality is integrated with existing code and testing takes place
- Deployment
    - Takes place continuously
- Monitoring
    - Bugs and performance reporting in real time

Devops also facilitates the implementation of CI/CD and feedback from operation can be used directly to optimize further development of the application

Tools used in DevOps

Source Code Repository (Git) -

Developers check in and update code => Major component of continuous integration

Build Server (Jenkins, SonarQube, Artifactory) -

Automatically compiles the code in the source code repository into executable code base

Configuration Management (Puppet, Chef) -

Configuration of a server or an environment

VIrtual Infrastructure (AWS, Microsoft Azure) -

Programmatically create new virtual machine provided by cloud vendor

Test Automation (Selenium, Water) -

Automatically performs tests

Benefits

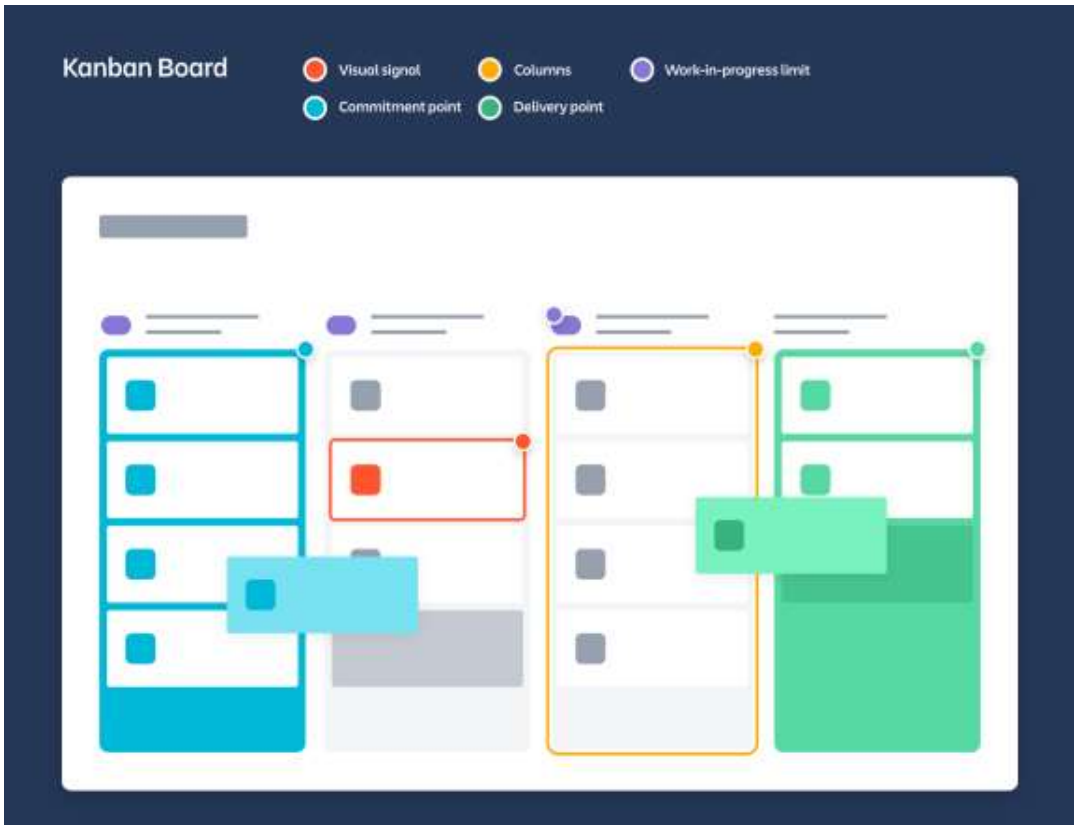
1. **Rapid delivery** - Innovate and improve product faster (New features and bug fixes)
2. **Reliability** - Ensure the quality of application updates and infrastructure changes
3. **Scale** - Operate and manage your infrastructure and development processes at scale

Kanban & Extreme Programming -

Kanban means billboard or signboard in Japanese.

Provides:

- Real time team communication
- Full transparence of work
- Visually tracking progress



Represents the overall project and has 5 components -

1. Visual signals - Visual cards to represent user stories
2. Columns - Represents the “stage” in the workflow (“To do”, “In progress”, “Code review”, “Done”)
3. Work-in-progress limits - Maximum number of cards in one column at any given time to make team focus on the work in those card and a quick way to identify bottleneck

4. Commitment point - The time point when an item is selected from the backlog and moved into the kanban board
5. Delivery point - The end of the workflow and product is delivered to client. Lead time = Delivery point - Commitment point

//

Benefits

- Easy to use
- Visual interface
- Instantly see what everyone is working on
- Visibility to task progress and bottlenecks

Kanban vs Scrum

	Scrum	Kanban
Cadence	Regular Fixed length sprints	Continuous flow
Release Methodology	At the end of each sprint	Continuous delivery
Role	Product owner, scrum master, development team	No required roles
Key Metrics	Velocity	Lead time, cycle time, WIP
Change Philosophy	Teams should not make changes during the sprint	Change can happen at anytime

Extreme Programming

Extreme programming is one several popular agile methods

// Key characteristics:

- Stress customer satisfaction
- Empower developers to confidently respond to changing requirement
- Strong emphasis on teamwork and peers
- Deliver as early as possible and get early feedback

Simplicity :

Do what is needed and take small simple steps to the goal and mitigate failures as they happen

Communication :

Everyone is part of the team and we communicate face to face daily and work together on everything from requirements to code

Feedback :

Listen to feedbacks and make changes after demonstrating software early and talk about the project and adapt process to it

Respect :

Everyone gives and feels the respect they deserve as a valued team member.

Courage :

Tell the truth about progress and estimates. Plan to succeed, works together and adapt to changes when it happens

Extreme Programming Process Planning

Release Planning

User stories are written and represent a desired feature and specification of the system. Customers prioritize user stories based on business value. Together developers and customers create a set of stories to be implemented as the first release. Make frequent small release and plan by time or by scope

Iteration Planning

Iteration planning takes place before each iteration.

Unimplemented user stories are selected from the iteration plan for the upcoming iteration. User stories implemented in the previous iterations but did not pass acceptance tests are also selected. The total user stories selected should have total estimates up to the project velocity from last iteration

Project velocity is often used to measure how much work is getting done on your project

The user stories and failed tests are broken down into the developing tasks

Rules of Extreme Programming -

Testing -

All codes must have unit tests and pass it before it can be released. When a bug is found, tests are created.

Acceptance tests are run often

- Acceptance test are created from user stories
- Customer specifies scenarios to test when a user story has been correctly implemented
- A story can have one or many acceptance tests, whatever it takes to ensure the functionality works.
- Acceptance tests are black box systems tests. Each acceptance test represents some expected results from the system
- User story is not considered completed until it has passed its acceptance test

Differences of Scrum & XP -

Does not allow changes into their sprints. XP team allows for user stories to be removed or added in an ongoing iteration.

XP teamwork in a strict priority order. By contrast, the scrum product owner prioritizes the product backlog but the team determines the sequence in which they will develop the backlog items

Scrum does not prescribe any engineering practices. XP does (eg. test-driven development, pair programming, etc.)

Rational Unified Process -

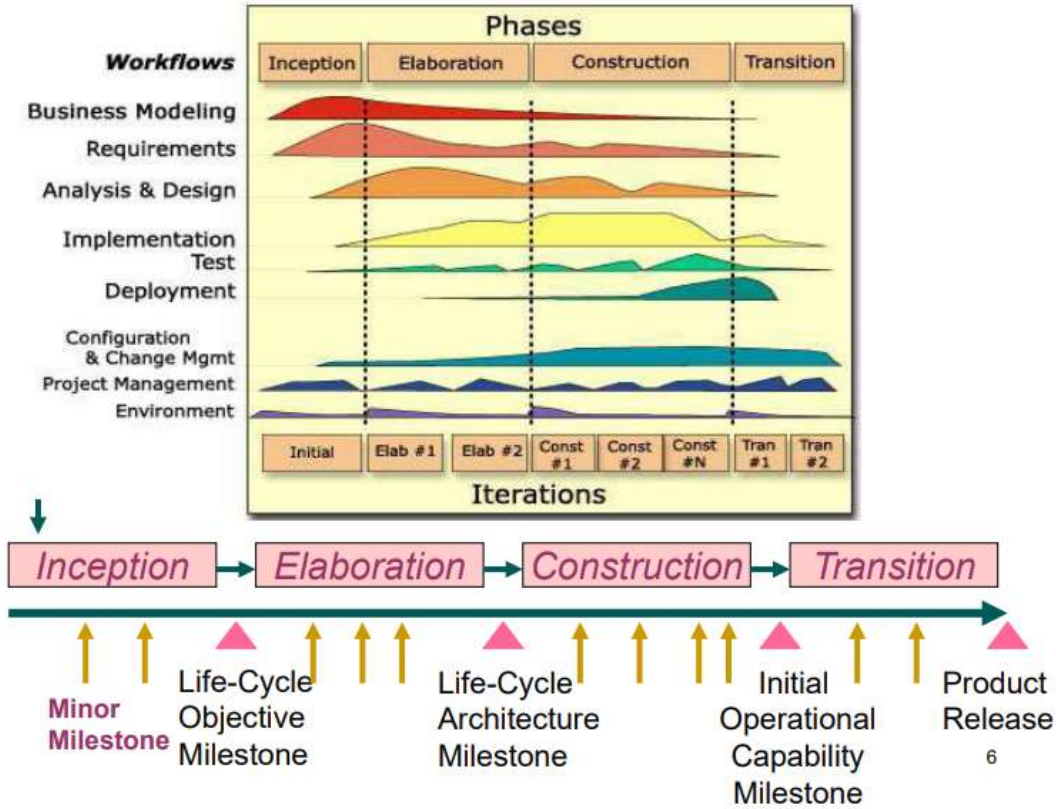
IECT

Model for software development and is deliberately flexible.

- Four phases
- Multiple activities
- Iterations

Principles -

1. **Develop iteratively**
2. **Manage requirements**
    a. Always keep in mind the requirement set by customer and will be changed by users
3. **Use components**
    a. Test individual components and reusability
4. **Model visually**
    a. Use diagrams to represent all major components, users, and their interaction UML
5. **Verify quality**
    a. Always make testing a major part of the project at any point of time
6. **Control changes**
    a. Projects may be created by multiple teams, at various locations with different platforms used. As a result, changes are made to a system is synchronized and verified constantly



Roles (Who) - Defines a set of related skills, competencies and responsibilities

Work products (What) - Represents something resulting from a task

Task (How) - Describe a unit of work assigned to a role that provides meaningful result

Phases -

1. Inception

1. **Objectives**
    a. Establish the project's software **scope**, **acceptance criteria**, **main use-case** scenarios that define the **core functionality** of the project
    b. Evaluate **alternative** architectures
    c. Estimate overall **cost**, **schedule** and **risk**
    d. Produce the "business case" (Business justification and benefits of having functionality provided)
    e. Create a work plan for the following elaboration phase

2. **Activities**
    a. Formulate the scope of the project: capture context and important requirements and constraints
    b. Plan and prepare a business case
    c. Evaluate alternatives for risk management, staffing, project plan, and trade-offs among cost, schedule, and profitability
    d. Synthesize a candidate architecture
3. **Outcomes**
    a. A vision document
        i. The code project requirement
        ii. Key features
        iii. Main constraints
    b. List all use cases and actors
    c. Initial business case which includes
        i. Business context
        ii. Success criteria
        iii. Financial forecast
    d. An initial risk assessment
    e. An initial project plan which shows the phases and iterations
4. **Criteria**
    a. Stakeholder concurrence on scope and costings
    b. Requirements understanding evident through use cases
    c. Credibility of cost & schedule estimates, priorities, risks and development process
    d. Depth and breadth of prototype architecture
    e. Actual vs planned expenditure

2. Elaboration

1. **Objectives**
    a. Define, validate, and baseline the system architecture

b. Baseline the vision
c. Baseline a plan for the construction phase
d. Demonstrate that the baseline architecture will support this vision for a reasonable cost in a reasonable time.

2. **Activities**
    a. Finalize the “vision”
    b. Develop solid understanding of critical use cases
    c. Put in place a development environment, tool, test automation systems etc.
    d. Elaborate the system architecture:
        i. Select subsystems that can be purchased (off the shelf software)
        ii. Integrate and assess the selected components against primary use cases.
3. **Outcomes**
    a. Use-case model (at least 80% complete) in which all use case have been identified in the use-case model survey. All actors have been identified
    b. Supplementary requirements that capture the non-functional requirements
    c. A software architecture description
    d. An executable architectural prototype
    e. A revised risk list and a revised business case
    f. A development plan for the overall project showing both major and minor iterations and stating deliverables at the completion of each iteration
    g. A preliminary user manual (optional)
4. **Criteria**
    a. The product vision and architecture are stable
    b. The executable demo shows that major risk elements have been addressed and resolved
    c. The construction plan sufficiently detailed
    d. All stakeholders agree that the current vision can be achieved with the current development plan

3. Construction

1. **Objectives**
    a. Develop remaining components and application features
    b. Integrate all parts into the product
    c. Test all features thoroughly.
    d. Achieve useful versions (alpha, beta)
    e. Minimize development cost by optimizing resource usage
    f. Achieving adequate quality, practical software and useful versions as quickly as possible
2. **Activities**
    a. Resource management, resource control, and process optimization
    b. Complete component development and testing against the defined evaluation criteria
    c. Assessment of product releases against acceptance criteria for the vision
3. **Criteria**
    a. The product is stable and ready to be exposed to user environment
    b. Stakeholders ready for the product to transition into user environment
    c. Actual resource expenditures are still acceptable relative to projected expenditures

4. Transition

1. **Objectives**
    a. Move software product to the user
    b. Phase starts when the system is mature enough to be deployed in the end-user domain and user documentation is available
    c. Phase includes
        i. Beta testing to validate the new system against user expectation
        ii. Parallel operation with the legacy system that the project is replacing
        iii. Training of users and maintainers
        iv. Rollout of the product to the marketing, distribution and sales team

Capability Maturity Model Integration (CMMI) Model -

CMMI (Capability Maturity Model Integration) is a proven industry framework to improve product quality and development efficiency for both hardware and software.
Used in process improvement activities as a collection of **best practices**, framework for **organizing** and **prioritizing activities**, support for the coordination of **multi-disciplined activities** that might be required to successfully build a product and means to emphasize alignment of the **process improvement objectives** with **organizational business objectives**

5 Levels -

1. Initial - Processes are unpredictable, poorly controlled, reactive
2. Managed - Processes are planned, documented, performed, monitored, and controlled at the project level. Often reactive. (Previous Experience, Basic management policies)
3. Defined - Processes are well characterized and understood. Processes, standards, procedures, tools, etc. are defined at the organizational level. Proactive (Documentation)
4. Quantitatively Managed - Processes are controlled using statistical and other quantitative techniques. (Quantitative and Quality goals are set)
5. Optimizing - Process performance continually improved through incremental and innovative technological improvements

CMMI Process Areas

There are Process areas (basic functions) within each of the 5 Maturity Level
Each process area is defined by a set of goals and practices.
There are 22 process areas:

1. **Project Management**
    a. Quantitative Project Management
        i. To achieve the project's established quality and process-performance objectives
    b. Integrated Project Management

        i. Establish and manage the project and the involvement of the relevant stakeholders according to an integrated and defined process that is tailored from the organization's set of standard processes
    c. Risk Management
        i. Identify potential problems before they occur so that risk-handling activities can be planned and invoked as needed across the life of the product or project to mitigate adverse impacts on achieving objectives
    d. Project Planning
        i. Establish and maintain plans that define project activities
    e. Project Monitoring & Control
        i. Provide an understanding of the project's progress so that appropriate corrective actions can be taken when the project's performance deviates significantly from the plan
    f. Supplier Agreement Management
        i. Manage the acquisition of products from suppliers for which there exists a formal agreement
2. **Engineering**
    a. Requirement Development
        i. Produce and analyze customer, product and product-component requirements.
    b. Technical Solution
        i. Design, develop, and implement solutions to requirements. Solutions, designs, and implementations encompass products, product components, and product-related life-cycle processes either singly or in combinations as appropriate
    c. Product Integration
        i. Assembly of the product from components and ensure that it is integrated, functions properly and deliver the product
    d. Verification
        i. Ensure work meets requirement

e. Validation
    i. Demonstrate product fulfill the intended use
f. Requirements Management
    i. Manage requirement of products and components to identify inconsistencies between requirements and plans and products (Track)

3. Process Management

a. Organizational Innovation & Deployment
    i. Select and deploy incremental and innovative improvements that measurably improve the organization's process and technologies. The Improvement support the organization's quality and process performance objectives as derived from the organization's business objective
b. Organizational Process Performance
    i. Establish and maintain quantitative understanding of the performance of the organization's set of standard processes in support of quality and process-performance objectives, and to provide the process performance data, baselines and models to quantitatively manage the organization's project
c. Organizational Process Focus
    i. Plan and implement organizational process improvement based on a thorough understanding of the current strength and weaknesses of the organization's processes and process assets
d. Organizational Process Definition
    i. Establish and maintain a usable set of organizational process assets
e. Organizational Training
    i. Develop the skills and knowledge of people so they can perform their roles effectively and efficiently

4. Support

a. Causal Analysis & Resolution

i. Identify causes of defects and other problems and take action to prevent them from occurring in the future

b. Decision Analysis & Resolution

i. Analyze possible decisions using a formal evaluation process that evaluates identified alternatives against established criteria

c. Measurement & Analysis

i. Develop and sustain a measurement capability that is used to support management information needs

d. Process & Product Quality Assurance

i. Provide staff and management with objective insight into processes and associated work product

e. Configuration Management

i. Establish and maintain the integrity of work products using configuration identification, configuration control, configuration status accounting, and configuration audits

Maturity Level	Project Managment	Engineering	Process Management	Support
5 Optimizing			Organizational Innovation & Deployment	Causal Analysis & Resolution
4 Quantitatively Managed	Quantitative Project Mngt		Organizational Process Performance	
3 Defined	Integrated Project Mngt Risk Management	Requirements Development Technical Solution Product Integration Verification Validation	Organizational Process Focus Organizational Process Definition Organizational Training	Decision Analysis & Resolution
2 Managed	Project Planning Project Monitoring & Control Supplier Agreement Mngt	Requirements Mngt		Measurement & Analysis Process & Product Quality Assurance Configuration Mngt
1 Initial				

Data Driven Development

Developing software using data and AI

Random forest - Build multiple decision trees with the use of random subset at each node split, and forms into a result through voting

Neural network

DDD Lifecycle

1. **Model Requirement**
    a. Identify components of existing or new product that are feasible to implement with machine learning
    b. Identify the requirements of the ML component and determine if it is a supervised model or an unsupervised model
2. **Data Collection**
    a. Find or build datasets. A use of mixed dataset is common to be done. (pre training on public dataset and post training on company dataset)
3. **Data Cleaning**
    a. Removes inaccurate or noisy data records (Errors created during collection process, Outliers, Incomplete data)
    b. Fill in missing data
4. **Data Labeling**
    a. // Assign correct target values to each instance in the dataset. (Eg, Labeling spam mail and not spam mail)
5. **Feature Engineering**
    a. Selecting relevant information to be extracted by AI while discarding the redundant information so that it can predict accurately
6. **Model Training**
    a. Split training or test data
    b. Selection of a model or a set of models
    c. Tuning hyper-parameters

7. **Model Evaluation**
    a. Assess model performance using test data and measuring how accurate the prediction is. Then choose which model to deploy based on accuracy
8. **Model Deployment**
    a. Deploy on targeted devices
9. **Model Monitoring**
    a. Monitor performance and errors

Ethics

Volkswagen scandal regarding software adjusting carbon emission indicator during testing.

1. Principle 1: Public - act consistently with public interest. Accept full responsibility for their work. Approve software only if it is safe, meet specification and passes appropriate tests. Disclose any danger to anyone or anything. Consider issues of disability
2. Principle 2 : Client and Employer - Provide service in their areas of competence, being honest about limitations of experience. Keep confidential information private
3. Principle 3 : Product - Ensure that their product and related modification is high quality, with acceptable cost and reasonable schedule. Identify and address ethical issues. Ensure adequate testing, debugging and review.
4. Principle 4 : Judgment - Only endorse documents prepared under supervision or within an area of competence. Maintain professional objectivity with respect to any software or documents they are asked to evaluate. Disclose to all concerned parties the conflicts of interest that cannot be avoided or escaped
5. Principle 5 : Management - Ensure good management that includes effective procedures for promotion of quality and reduction of risk. Ensure there is a fair agreement concerning ownership of any software, processes, research, writing or other intellectual property to which a software engineer has contributed.
6. Principle 6 : Profession - Take responsibility for detecting, correcting and reporting errors in software and associated documents.
7. Principle 7 : Colleague - be fair and supportive. Credit their work and review their work in an objective way.
8. Principle 8 : Self - Participate in a lifelong learning mindset. Further knowledge of development and improved ability to create safe, reliable and useful quality software at reasonable cost and within a reasonable time.

Ethics Practices

1. **Fairness and inclusiveness -**
    a. Treat all people fairly regardless of race, age, gender. Software/team should empower everyone and engage them
2. **Reliability and safety -**
    a. Software should perform task reliably and safely
3. **Transparency -**
    a. Software should be understandable, so that others can understand how the output is derived.
4. **Privacy and security -**
    a. Software should be secure and sensitive information should be stored appropriately. Personal information should not be collected without consent or used without the knowledge of the user.
5. **Accountability -**
    a. Parties responsible for the outcomes produced by the software is accountable

Tools -

https://app.diagrams.net/ (Tool to create UML diagrams)

https://tree.taiga.io/ (Tool for scrum)