Terms

Program	• A computer program is a sequence of instructions written to perform a specified task in a computer
Software	• A software is a set of programs, procedures and its documentation concerned with the operation of a data processing system
Software Process	• A Process is a series of definable, repeatable, and measurable tasks leading to a useful result

Software Development

Ad-hoc Software Development (till 1960's)

- The Software was developed on a Trial & Error basis
- No Specific Process was followed during the development of the Product
- Defects were detected only after the product was delivered to the external Users

Terms

Software Engineering is defined as

- Software engineering is the application of a systematic, disciplined, quantifiable approach to the design, development, operation, and maintenance of software
- An establishment and use of sound engineering principles in order to obtain an economical software that is reliable and works efficiently on real machines

Process

A Process is a series of definable, repeatable, and measurable tasks leading to a useful result.

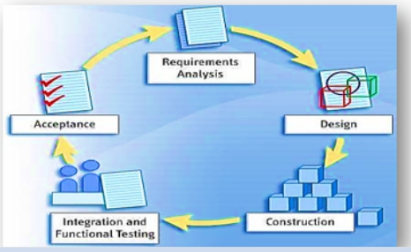
Software Development Process

Software Development process involves transformation of user needs into an effective software solution.

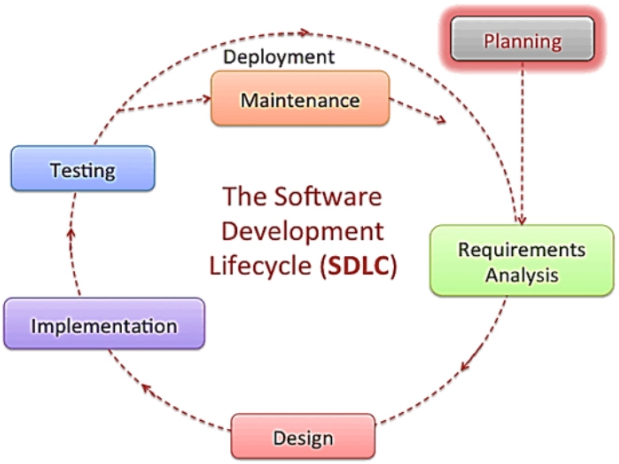
SDLC

Software Development Life Cycle is the process used in a project to develop a software product

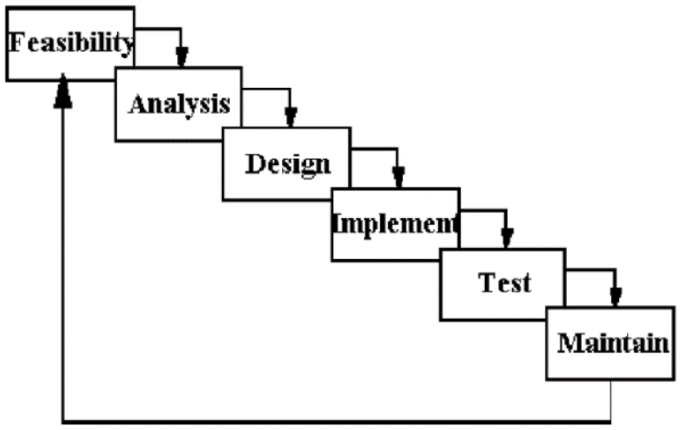
It describes how the development activities are performed and how the development phases follow each other



Software Development Life Cycle



Phases in SDLC



Analysis

The goal of the system analysis is to define the requirements of the system

Requirement gathering requires client as well as the service provider to get the detailed and accurate requirements

SRS(Software Requirement Specification) is the primary artifact of Analysis phase

Activities in Analysis Phase

Requirements gathering and analysis

Preparing Requirements Specification(SRS)

Design

Software design deals with transforming the customer requirements into a set of documents that is suitable for implementation in a programming language

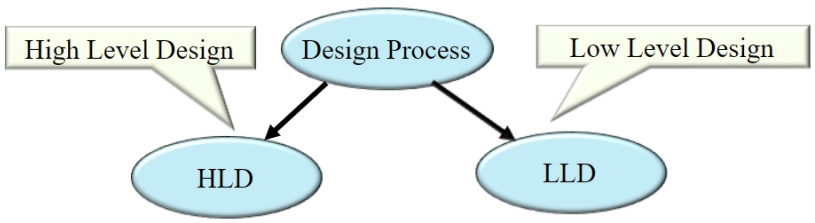
It is the process of defining the architecture, interface, component and other characteristics of a system

The design stage takes the requirements identified in the approved requirements document (SRS) as its initial input

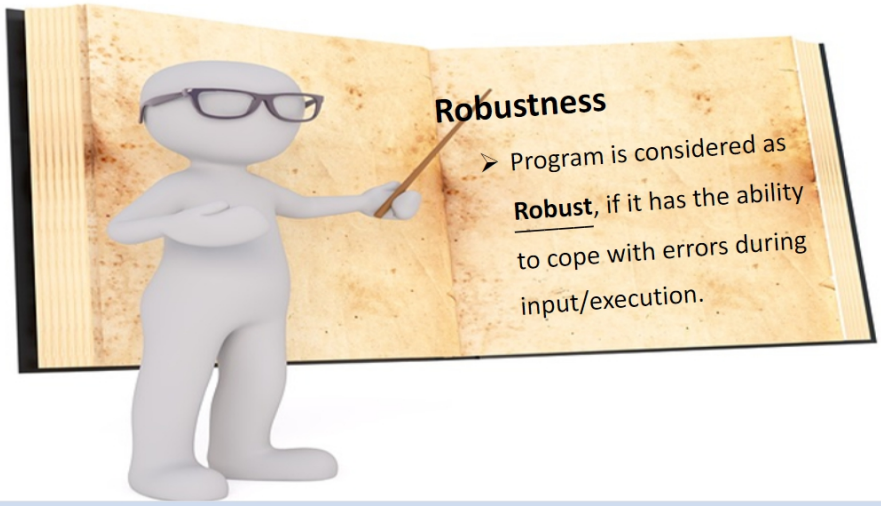
Levels of a Design

Decompose the entire project into units / modules and identify the system architecture, data structure and processing logic

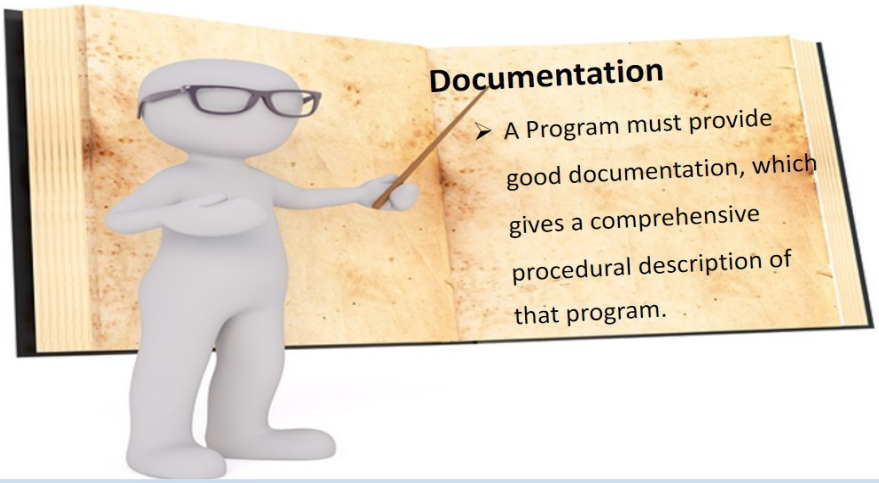
- DD(Design Document)= HLD + LLD



CHARACTERISTICS OF A GOOD PROGRAM



CHARACTERISTICS OF A GOOD PROGRAM



Correctness Vs. robustness

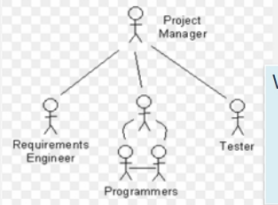
Here, your program is correct, but **not robust**. It is not handling unexpected situations. You need to modify your program to support robustness.

If your program supports robustness, then it should accept only numbers. Otherwise the output should be: "Only Numbers are allowed".

Coupling and Cohesion

- Modules in themselves are not "good" – We must design them to have good properties
- Properties of good Design
  - Component independence
  - Fault prevention and fault tolerance
  - Design for change

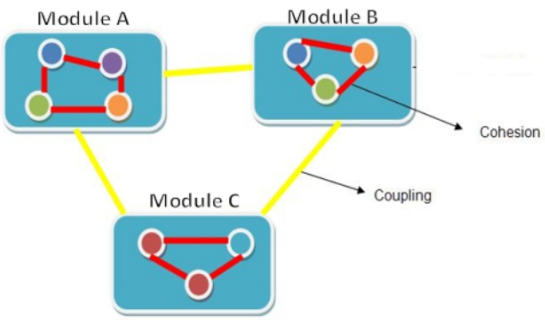
With poor module design:
- Hard to understand
- Hard to locate faults
- Difficult to extend or enhance



With good module design:
- Maximal relationships within modules (cohesion)
- Minimal relationships between modules (coupling)
- This is the main contribution of structured design

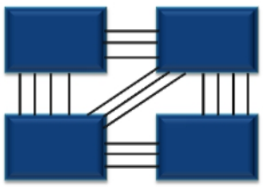
Coupling and Cohesion

- The goal of design is to divide the system into modules and assign responsibilities among the components so that they have
  - High cohesion within the modules
  - Loose coupling between modules
- The principle of coupling and cohesion are the most important design principles



Coupling – Degree of interaction between two modules

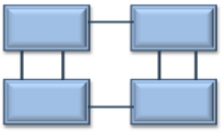
Two modules are tightly coupled when they depend a great deal on each other



Tightly coupled - many dependencies

Uncoupled modules have no interconnections at all; they are completely un related

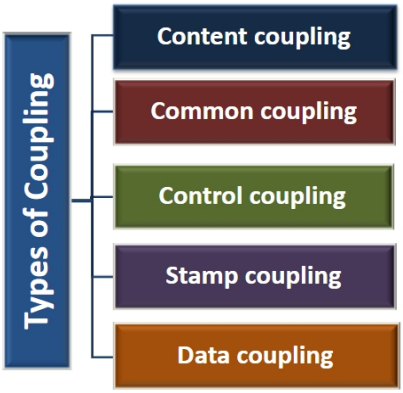
Loosely coupled modules have some dependence, but their interconnections are weak

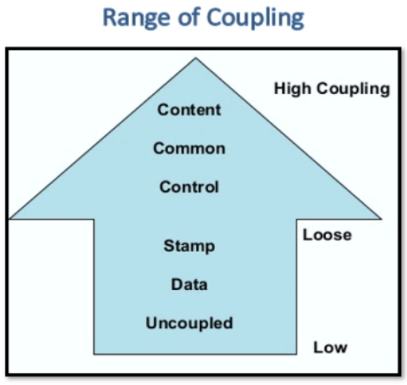


Loosely coupled - some dependencies

Uncoupled - no dependencies

Types of coupling

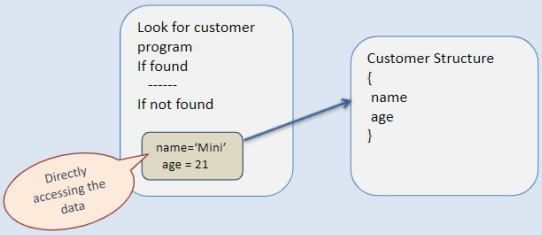




Types of coupling

Content coupling

- Occurs when one component modifies an internal data item in another component, or when one component branches into the middle of another component

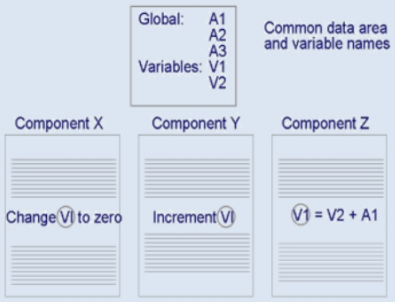


- To reduce content coupling
  - Hide the data so that it can be accessed only by calling the method that can access or modify the data

Types of Coupling

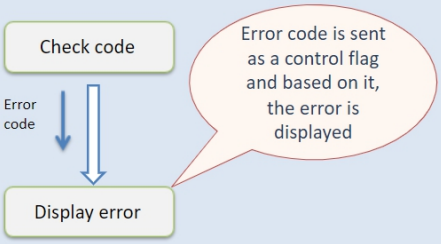
Common Coupling

- Two modules have write access to the same global data



Control coupling

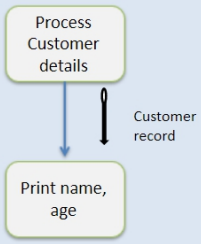
- One module passes an element of control to the other
- It is impossible for the controlled module to function without some direction from the controlling module



Types of Coupling

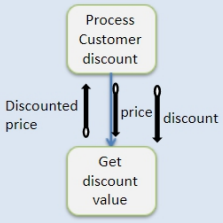
Stamp Coupling

- Data structure is passed as parameter, but the called module operates on only some of individual components



Data Coupling

- Every argument is either a simple argument or a data structure in which all elements are used by the called module

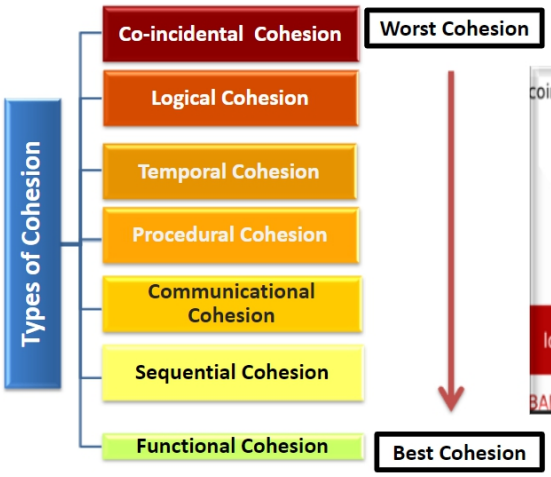


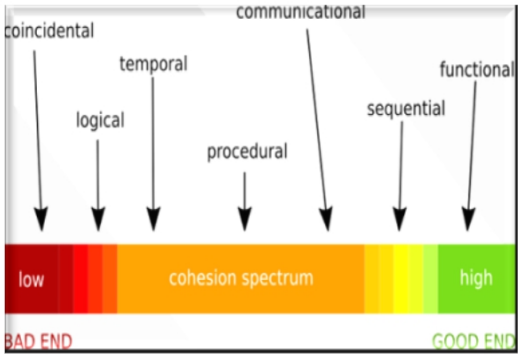
Cohesion

Cohesion refers to the dependence within and among a module's internal elements (e.g., data, functions, internal modules)

Greater the cohesion, the better is the program design

Types of Cohesion





Types of Cohesion

TEKNOTURF

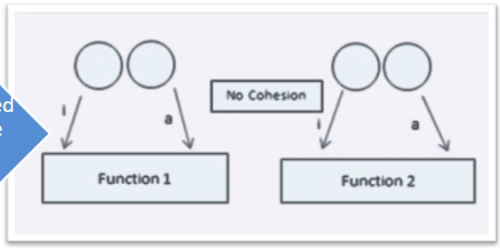
Co-incidental Cohesion

Module performs multiple, completely unrelated actions

Module 1

- Process customer details
- Calculate total sales
- Perform transaction deletion
- Read sales details

Has unrelated tasks inside module



Types of Cohesion

TEKNOTURF

Logical Cohesion

Here, elements perform similar tasks and the activities to be executed are chosen from outside the module.
Operations are related, but the functions are significantly different.

Module

```
Module(data[] , type)
  if type is bar
     display as bar chart
  else if type is pie-chart
     display as pie-chart
  else if type is graph
     display as graph
  .....
```

Task is the same which is "display", but the 'how to be displayed' is decided by the calling function.

Temporal Cohesion

Module's data and functions are related because they are used at the same time in an execution.
Elements are grouped by when they are processed.

Process Error Module

```
processError()
  Release the database connection
  Open error file
  Write error log
  Send error message to user
```

All the activities of the module are performed when an error occurs in the software.

Types of Cohesion

TEKNOTURF

Procedural Cohesion

Similar to temporal, and functions pertain to some related action or purpose

Module

```
processCustomer()
  create customer id
  add new record to customer file
  display customer id to user
```

All the activities of the module related to creating a customer record

Communicational Cohesion

Module which has activities executed sequentially and work on same data

Module

```
processCustomer(customer id)
  get customer details
  get customer purchaser details
  display customer id, customer name,
                    purchase items
```

All the activities of the module act on the customer id

Types of Cohesion

TEKNOTURF

Sequential Cohesion

Elements are involved in activities such that output data from one activity becomes input data to the next activity

CalculateGrade Module

```
CalculateGrade(marks) {
  calculateSum
        ↓ sum
  calculateAverage
        ↓ average
  calculateGrade
}
```

The output of calculateSum is given as input to calculateAverage

Functional Cohesion

Functionally cohesive module performs exactly one action.
Highly recommended Cohesion

Example: `convertCelciusToFarenheit`

Advantages

- More reusable
- Easier corrective maintenance
  - Fault isolation
  - Reduced regression faults
- Easier to extend product

Construction(Code + Unit Testing)

TEKNOTURF

Modular and subsystem programming code will be accomplished during this stage

Unit testing /module testing is done in this stage by the developers

This stage produces the source code, executable, and databases applicable

Testing

TEKNOTURF

Testing is the process of executing the program with the intent of finding errors

Software testing is a process of verifying and validating that a software application or program meets the business and technical requirements

Levels of Testing

- Unit Testing
- Integration Testing
- System Testing
- Acceptance Testing

Testing

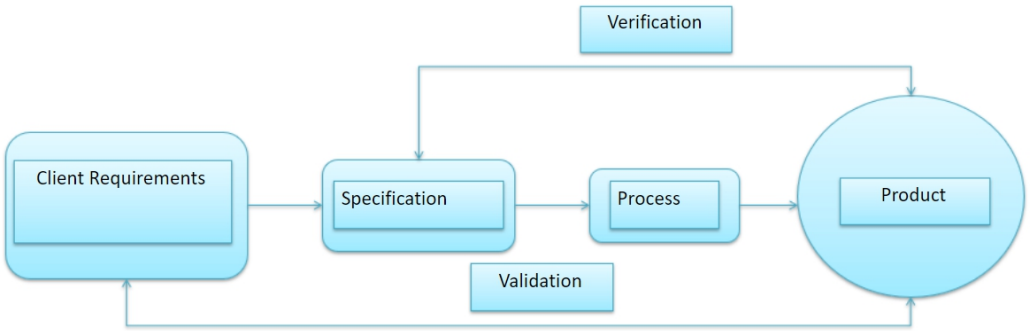
TEKNOTURF

Software testing includes

Verification	• Confirms that the software meets its technical specifications
Validation	• Confirms that the software meets the business requirements
Defect	• Variance between the expected and actual result

Verification, Validation and Defect Finding

TEKNOTURF

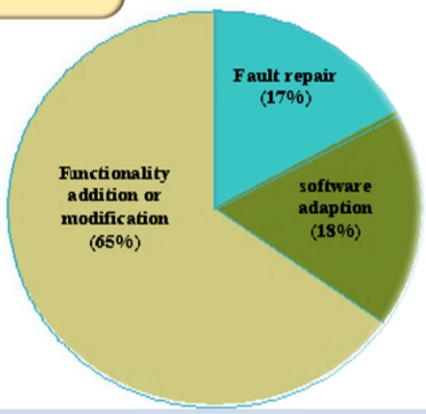


Levels of Testing

TEKNOTURF

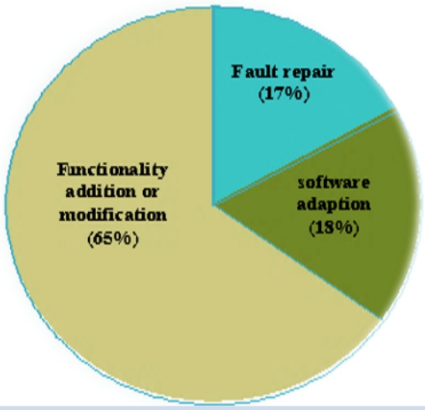
- Unit testing
- System testing
- Integration Testing
- Acceptance Testing

Maintenance



Maintenance

Changes or enhancements happen everywhere. Software is no exemption. Any change that is made to the software after it is deployed is known as maintenance.



Software Development Life Cycle Models

- Waterfall model
- V-model
- Prototype model
- RAD (Rapid Application Development)
- Incremental model
- Spiral model

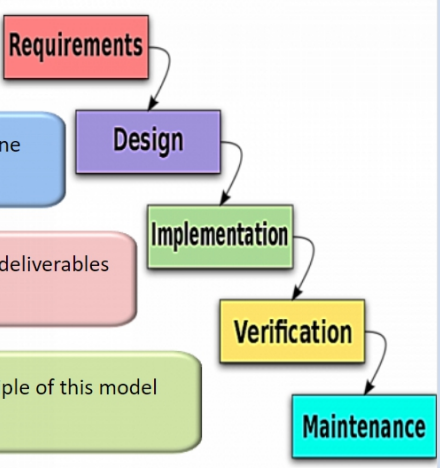
Waterfall Model

This model was proposed by Winston Royce in 1970

Waterfall model derives its name due to the cascading effect from one phase to the other as depicted in the diagram

Each phase has a well defined start and end point, with identifiable deliverables to the next phase

It is a linear sequential model, systematic in approach and the principle of this model suggests - "Can't retrieve to previous phase"



Advantages of Waterfall Model

- Simple and easy to use.
- Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process
- Phases are processed and completed one at a time
- Works well for smaller projects where requirements are very well understood
- Linear approach
- Equivalent importance to all the phases
- Contract Related issues can be addressed effectively

Limitations of Waterfall Model

- This model is suitable if the requirements are well-defined and stable
- User gets a feel of the system only at the later stages of development
- Backtracking cost is high in case of a problem
- Increased development of time and cost
- Systems must be defined up front
- Rigidity
- Hard to estimate costs & project overruns

Waterfall Model

TEKNOTURF

Suitable when

- Software requirements are clearly defined and known
- Product definition is stable
- Software development technologies and tools are well known
- New version of the existing software system is created

V-Model

TEKNOTURF

Verification and Validation Model commonly known as V-Model evolved from waterfall Model

Each phase must be completed before the next phase begins

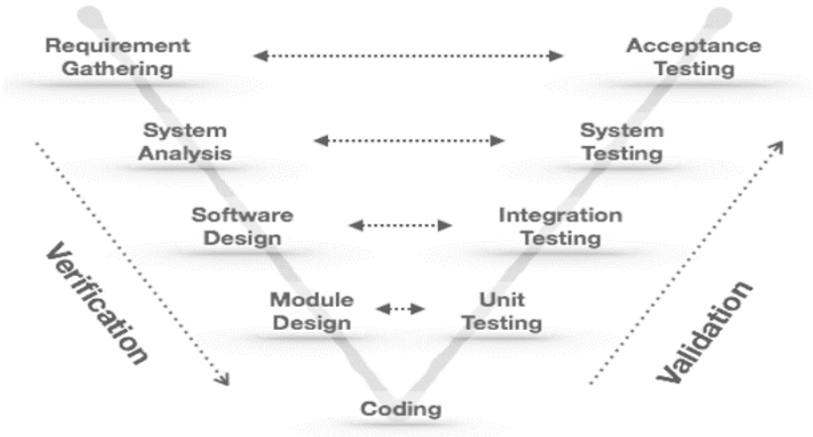
Testing is emphasized in this model more than in the waterfall model

It is a structured approach to testing

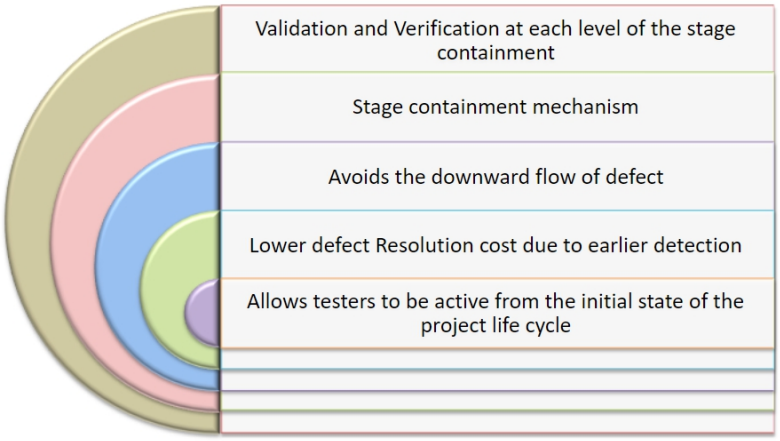
Testing is done from the earlier stage thereby bringing high quality into the development of our products

V-Model

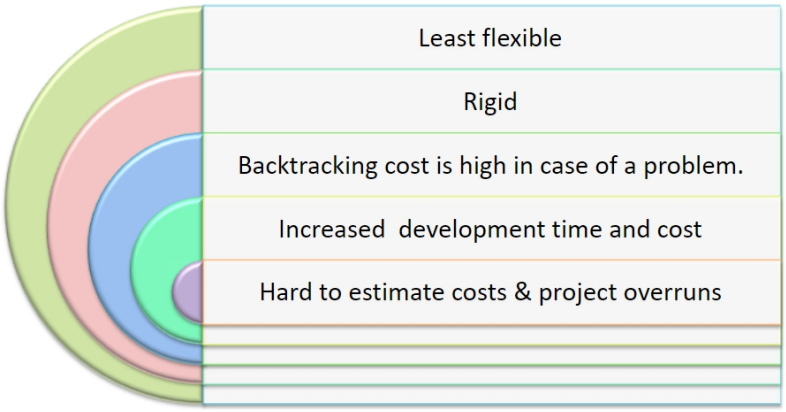
TEKNOTURF



Advantages of V-Model



Limitations of V-Model



Prototype Model

Creates prototypes, which is an incomplete version of the software program being developed.

Simulates only few aspects of the features of the System to be built

Prototype Model

TEKNOTURF

Prototyping can also be used by the end users to describe and prove requirements that the developers have not considered.

Developers build a prototype during the requirements phase.

Prototype is evaluated by the end users to provide corrective feedback

Developers further refine the prototype based on feedback.

When the user is satisfied, the prototype code is brought up to the standards needed for the final product.

Prototype Model

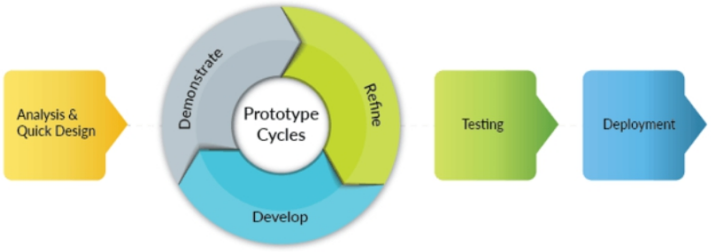
TEKNOTURF

The process of prototyping involves the following steps

- Identify basic requirements
- Develop Initial Prototype
- Review
- Revise and Enhance the Prototype

Prototype Model

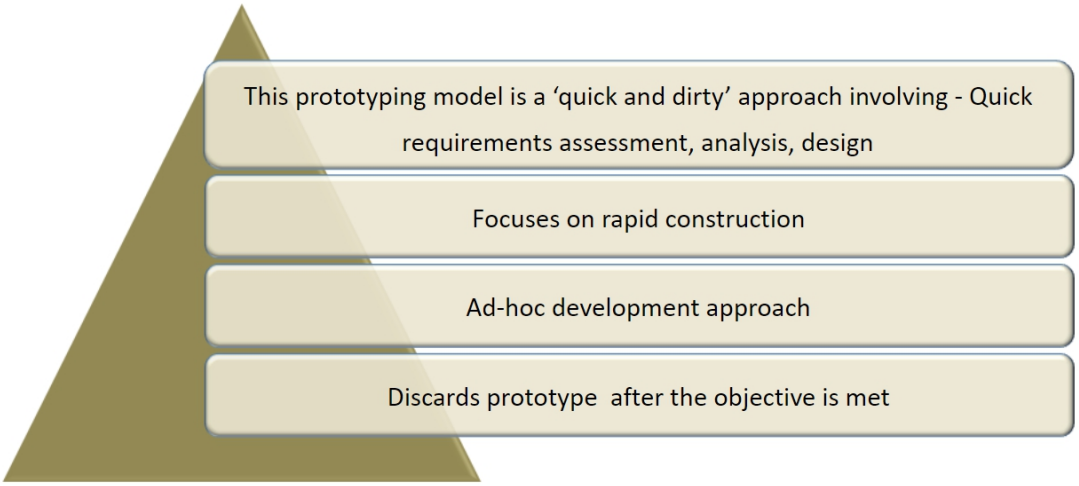
TEKNOTURF



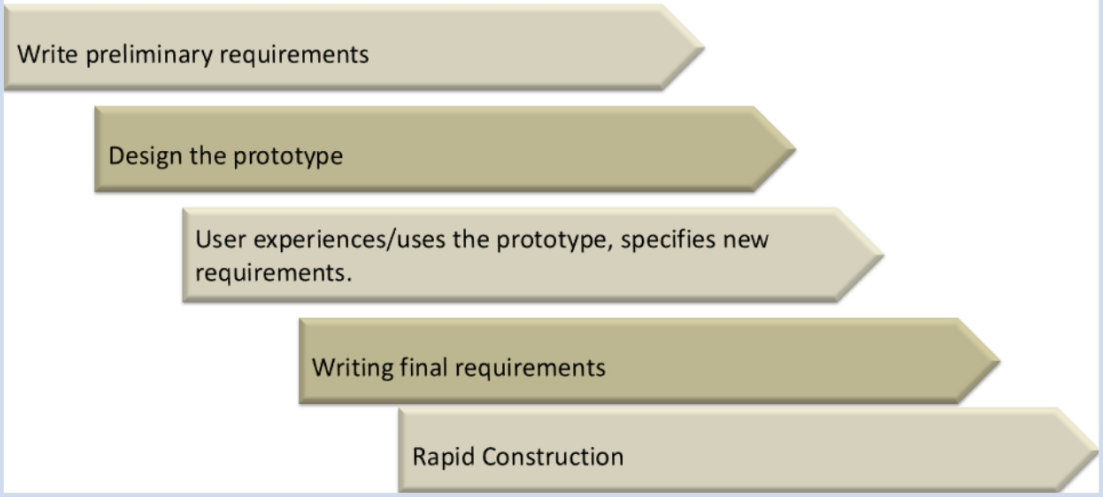
What are the types of prototypes?

- Throw away
- Evolutionary

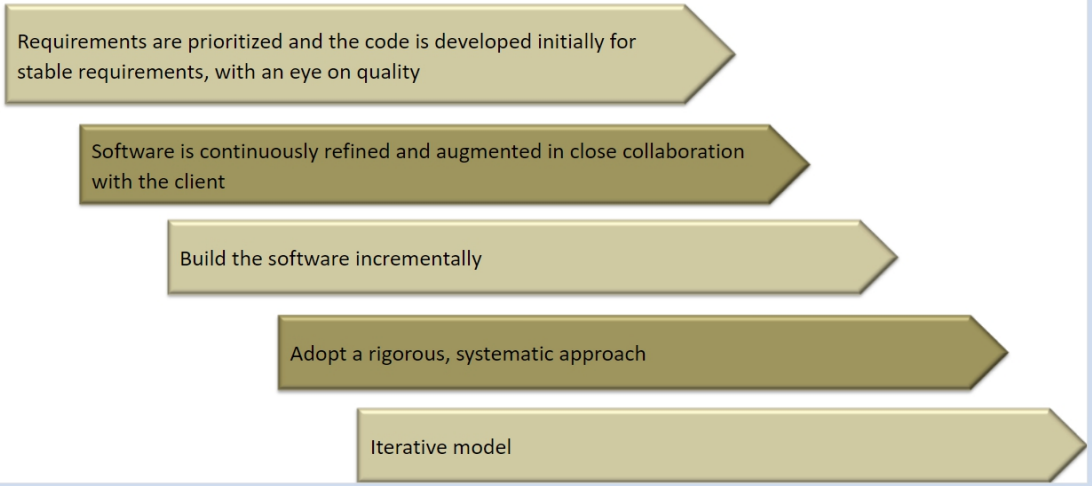
Throw away Prototype Model



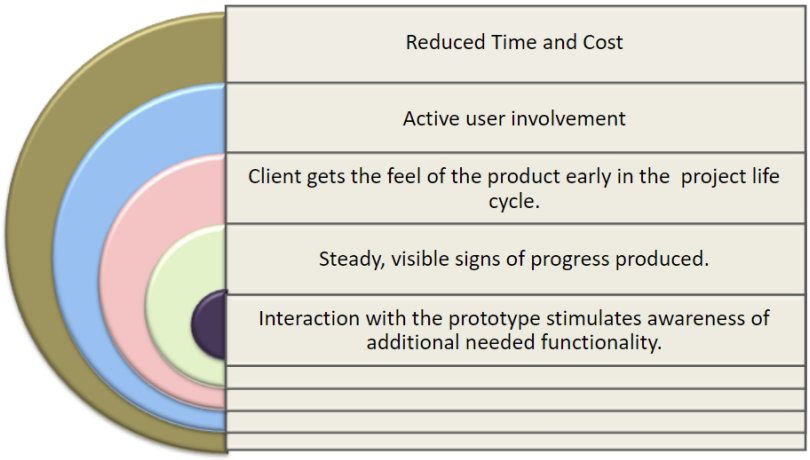
Throw away prototyping-Steps



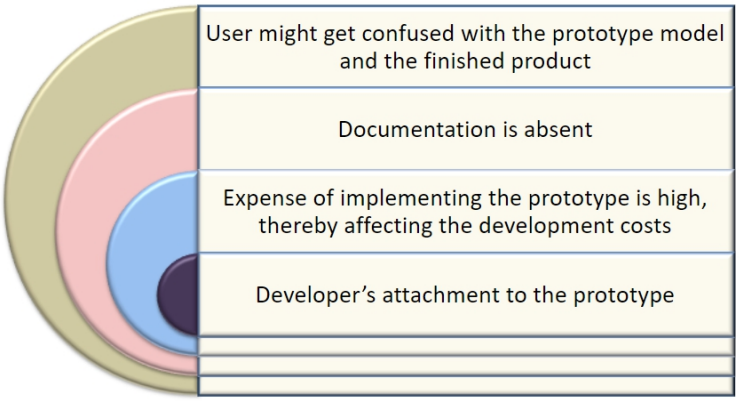
Evolutionary Prototype Model



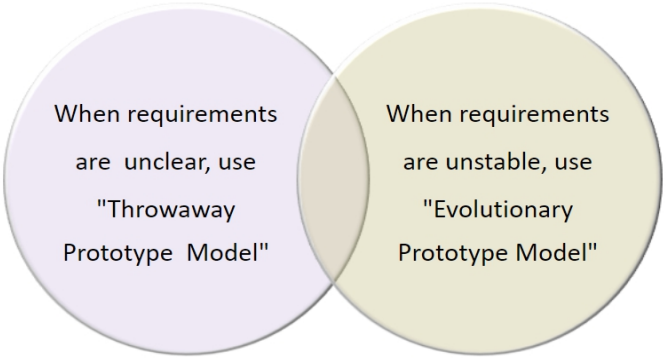
Advantages of Prototype Model

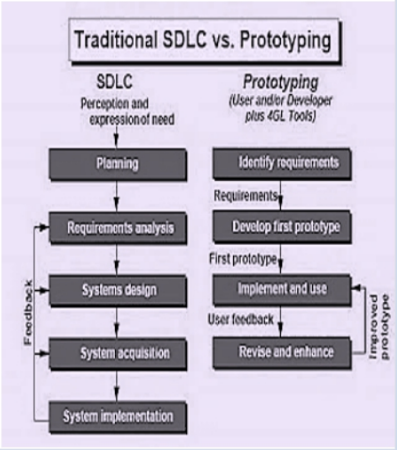


Limitations of Prototype Model



When to use Prototype Model?





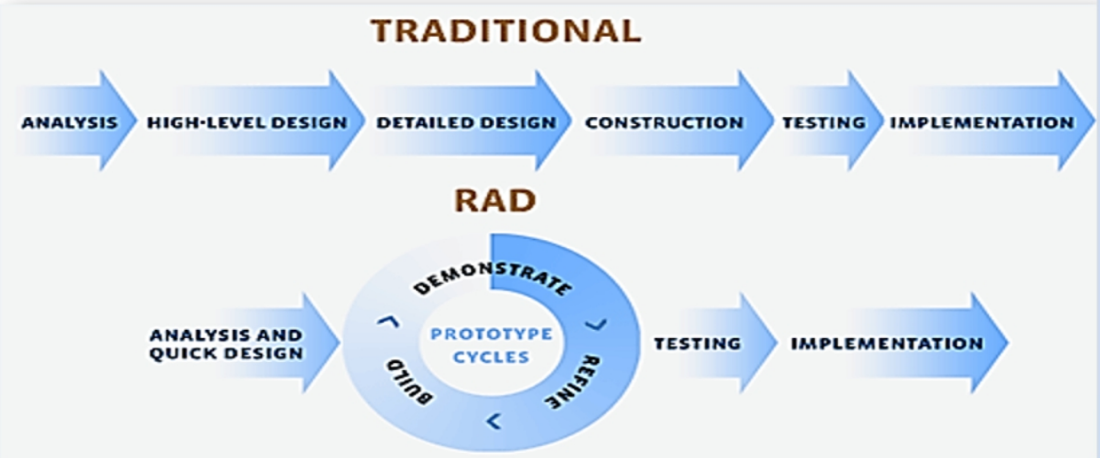
Rapid Application Development

TEKNOTURF

- RAD is a high speed version of the linear sequential model
- Characterized by a very short development life cycle
- The RAD model follows a component based approach
- Individual components are developed by different people and assembled to develop a large software system

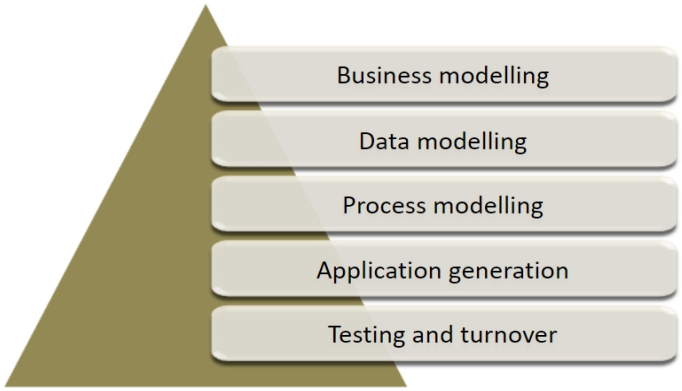
Traditional vs RAD

TEKNOTURF



Rapid Application Development phases

TEKNOTURF



Advantages and Limitations of RAD

TEKNOTURF

Advantages:

- Due to the emphasis on rapid development, it results in the delivery of a fully functional project in short period.
- Facilitates Parallel Development.

Limitations:

- Developers and clients must be committed to rapid-fire activities in an abbreviated time frame.
- If either party is indifferent in needs of other, the project will run into serious problem.
- It is not suitable for large projects.

When to use RAD?

TEKNOTURF

When requirements are not fully understood

User is involved throughout the life cycle

System can be modularized

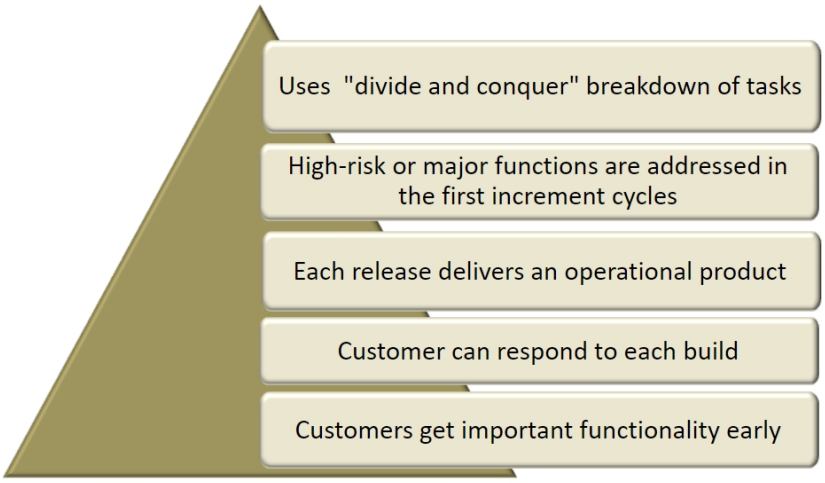
Incremental Model

TEKNOTURF

The incremental model prioritizes requirements of the system and implements them in groups

Each subsequent release of the system adds function to the previous release, until all designed functionalities have been implemented

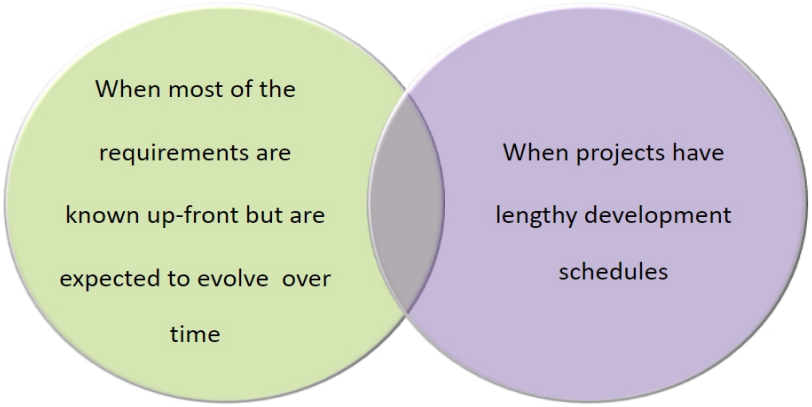
Advantages of Incremental Model



Limitations of Incremental Model

- Requires early definition of a complete and fully functional system to allow for the definition of increments
- Requires good planning and design as basis for the system
- Absence of a well-defined module interface is a major obstacle for this model of development

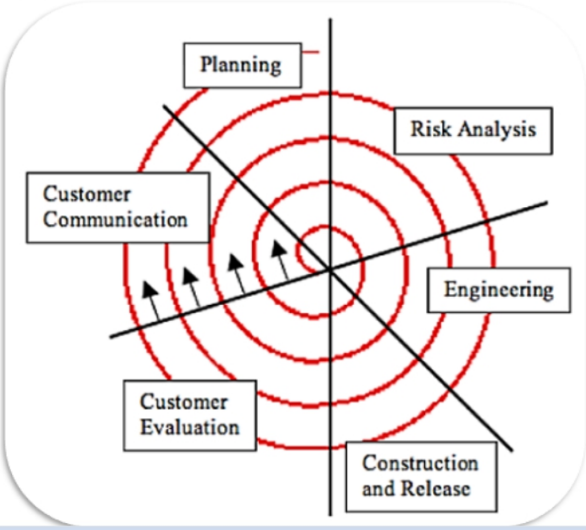
When to use Incremental Model?



Spiral Model

- Proposed by Barry Boehm in 1986
- Diagrammatic representation of this model appears like a spiral with many loops
- Suitable for technically challenging software products that are prone to several kinds of risks
- Accommodates prototyping. This model combines the features of the prototyping model and the waterfall model
- It is favoured for large, expensive, and complicated models
- Suggested for High-Risk Scenarios based projects

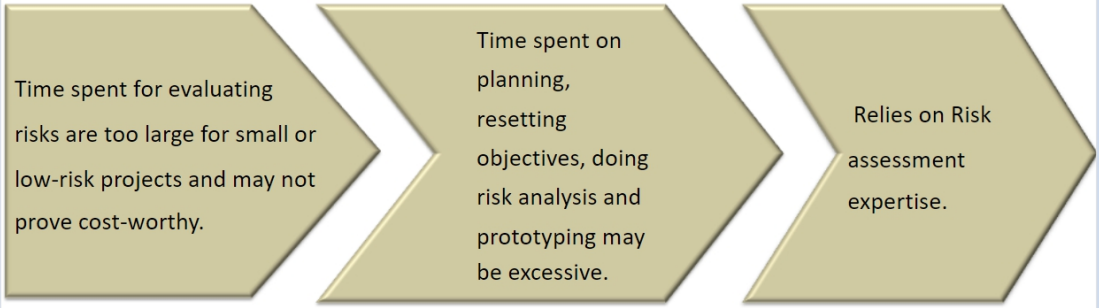
Spiral Model



Advantages of Spiral Model

- Provides early indication risk.
- Users see the system early because of rapid prototyping tools.
- Critical high-risk functions are developed first.
- Early and frequent feedback from users.

Limitations of Spiral Model



When to use Spiral Model?