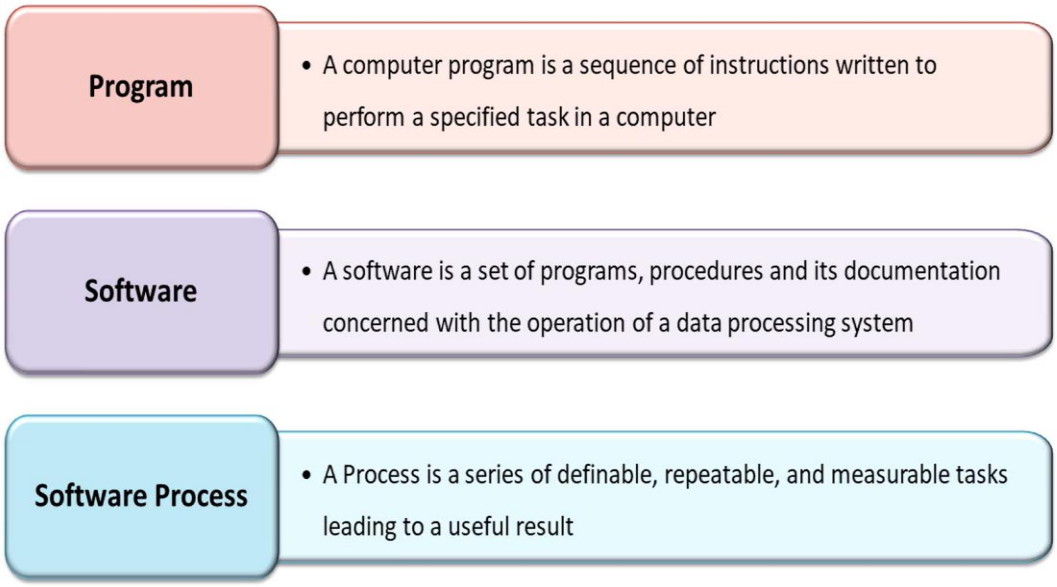
Terms



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

TEKNOTURF

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

Software Crisis

TEKNOTURF

- Software fails to meet the user requirements
- Software crashes frequently
- Development of Software became expensive
- Difficult to alter, debug, and enhance the software
- Software was often delivered late
- Software used resources non-optimally

Process

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

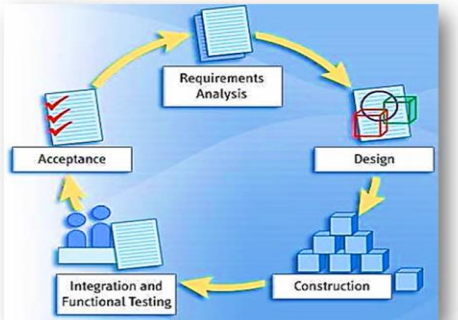
Software Development Process

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

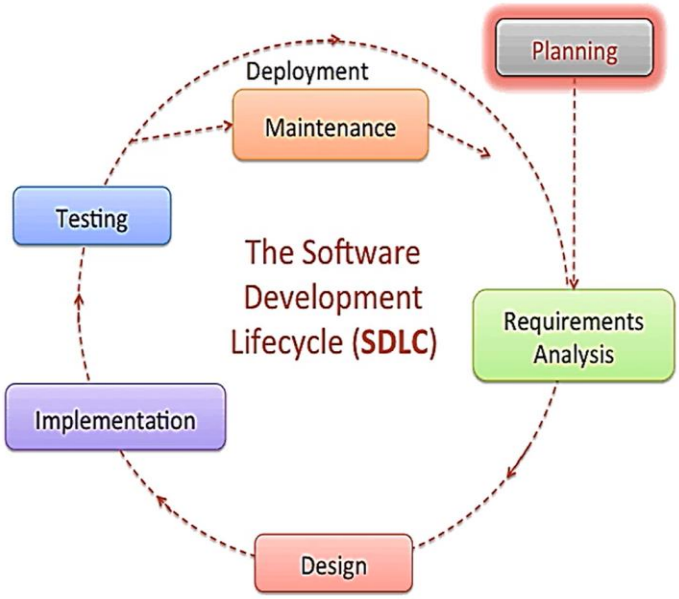
SDLC

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

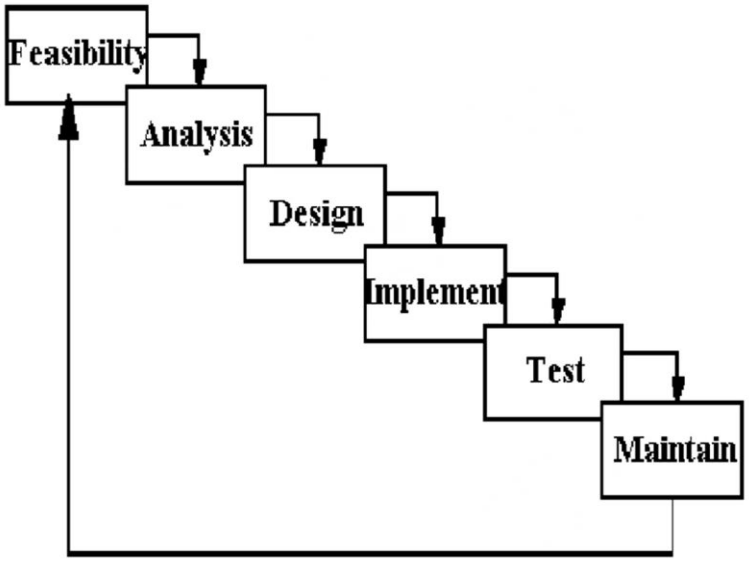
It describes how the development activities are performed and how the development phases follow each other



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

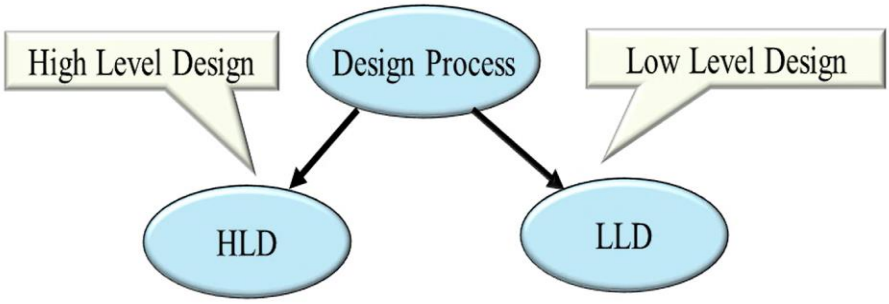
It is the process of defining the architecture, interface, component and other characteristics of a system

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



Construction(Code + Unit Testing)

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

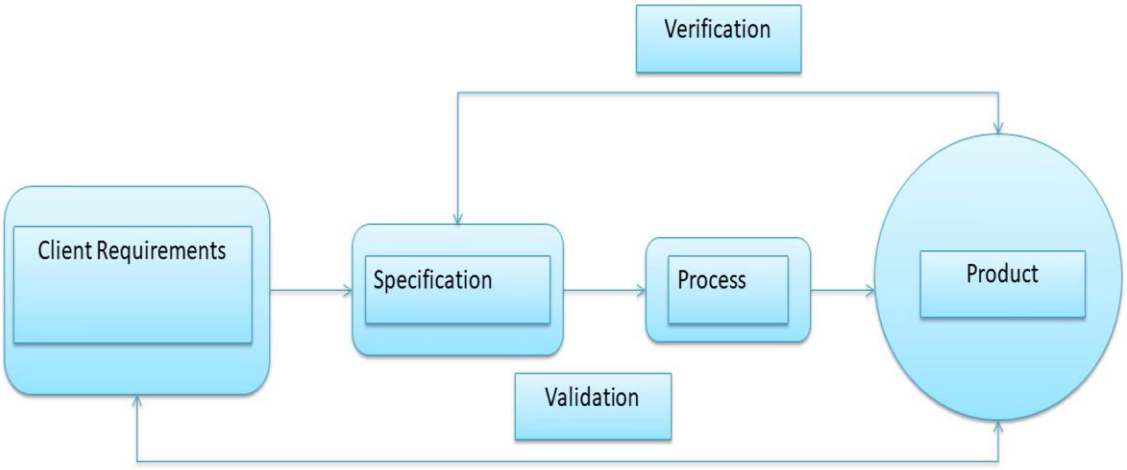
Testing

TEKNOTURF

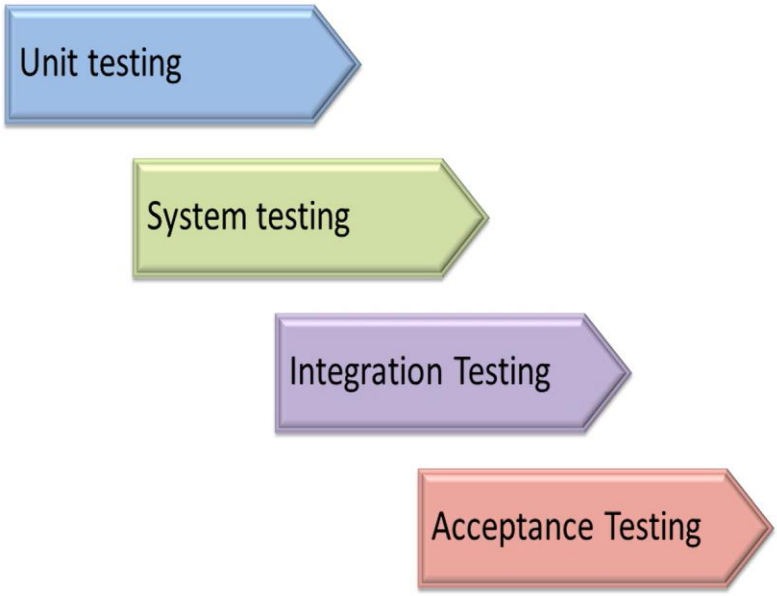
Software testing includes

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

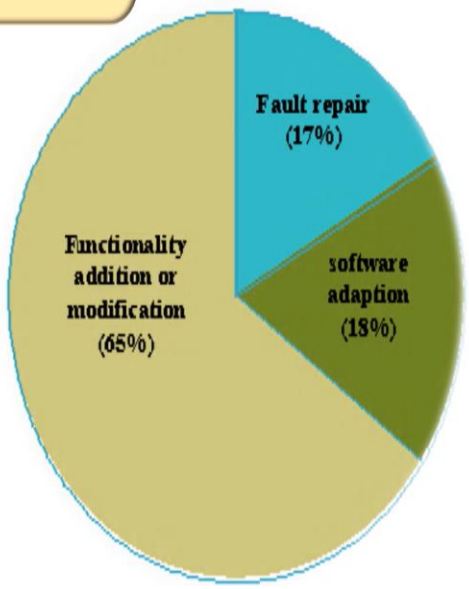
Verification, Validation and Defect Finding



Levels of Testing

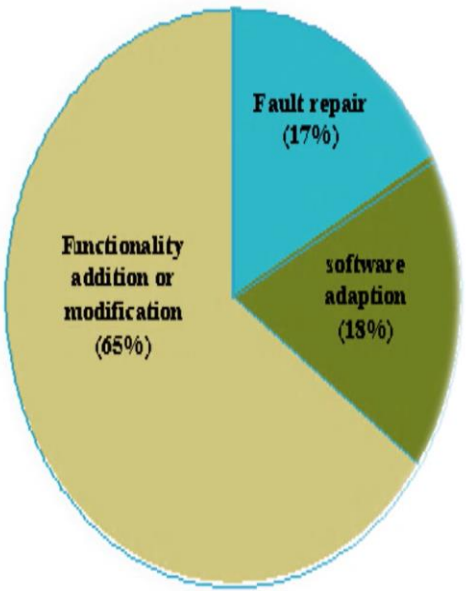


Maintenance



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

TEKNOTURF

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

Waterfall Model

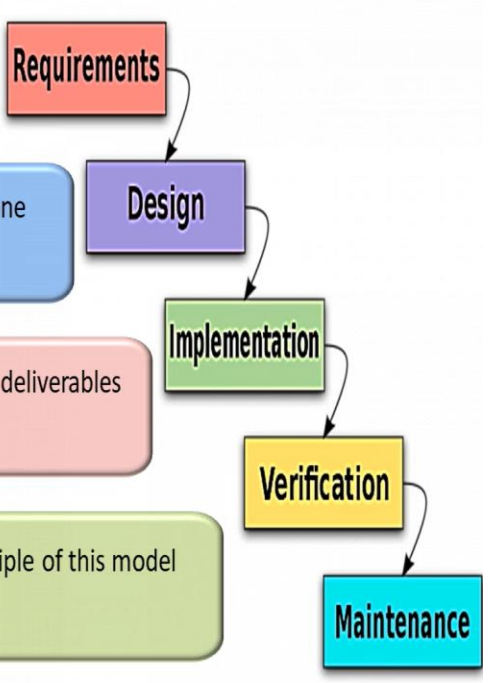
TEKNOTURF

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

TEKNOTURF

- 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

TEKNOTURF

- 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

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

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

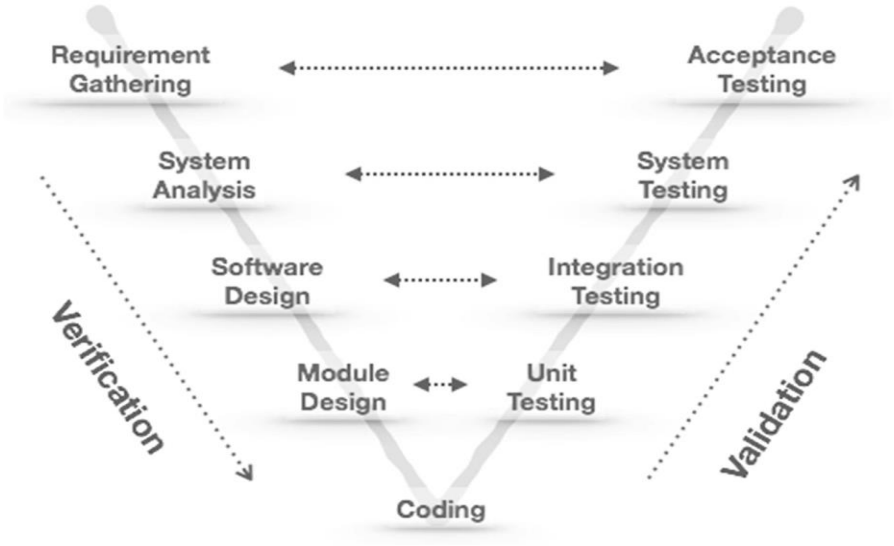
Each phase must be completed before the next phase begins

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

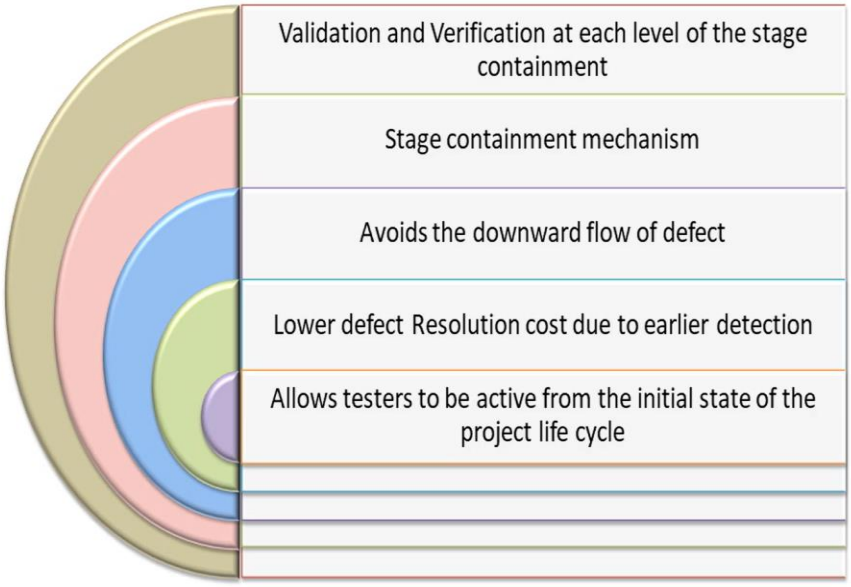
It is a structured approach to testing

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

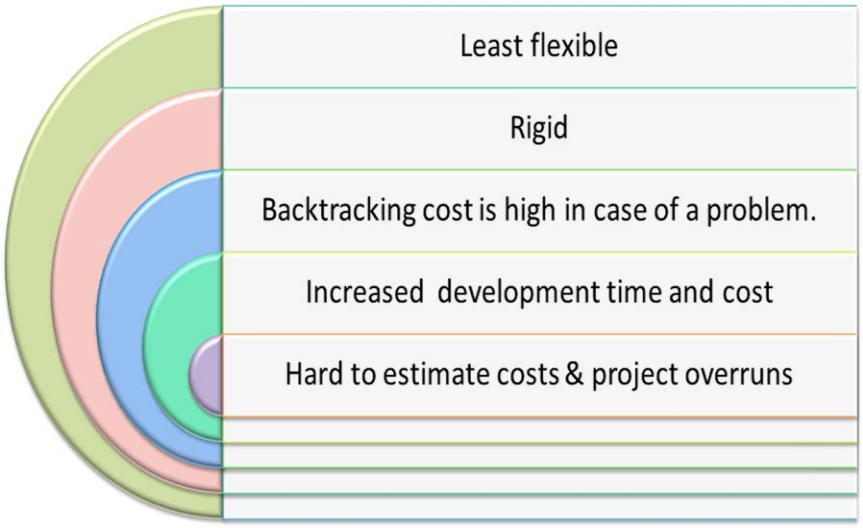
V-Model



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.

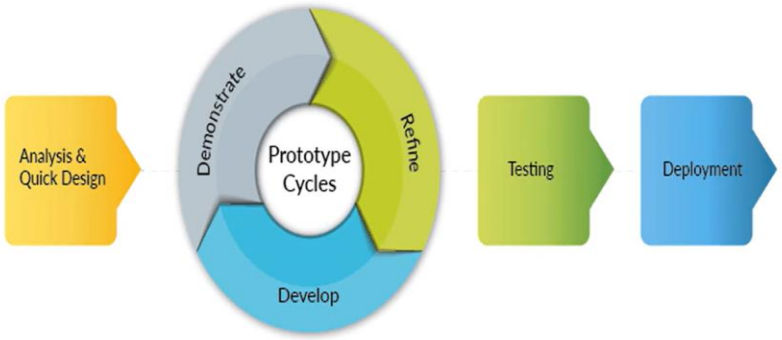
Prototype Model

TEKNOTURF

The process of prototyping involves the following steps

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

Prototype Model



What are the types of prototypes?

- Throw away
- Evolutionary

Throw away Prototype Model

This prototyping model is a 'quick and dirty' approach involving - Quick requirements assessment, analysis, design

Focuses on rapid construction

Ad-hoc development approach

Discards prototype after the objective is met

Throw away prototyping-Steps

TEKNOTURF

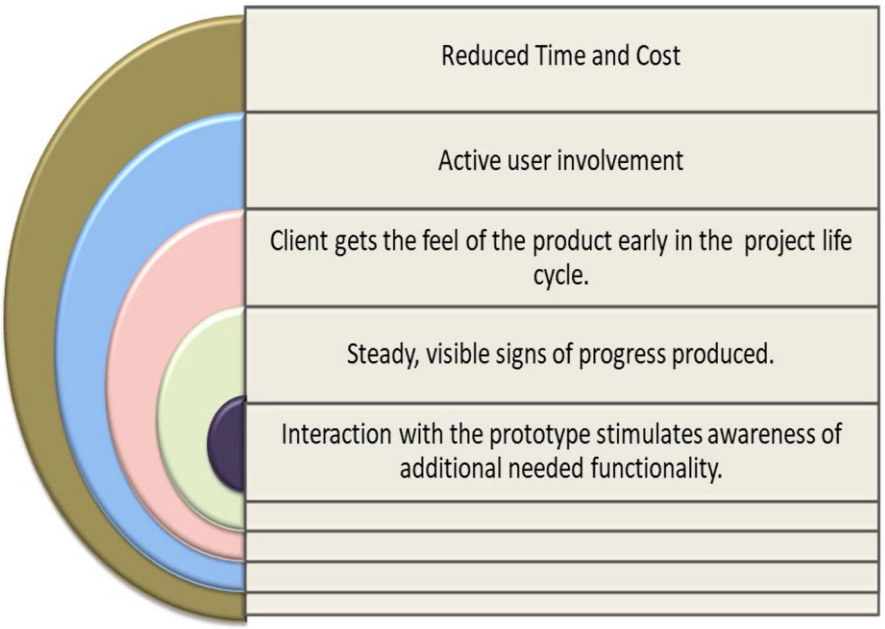
- Write preliminary requirements
- Design the prototype
- User experiences/uses the prototype, specifies new requirements.
- Writing final requirements
- Rapid Construction

Evolutionary Prototype Model

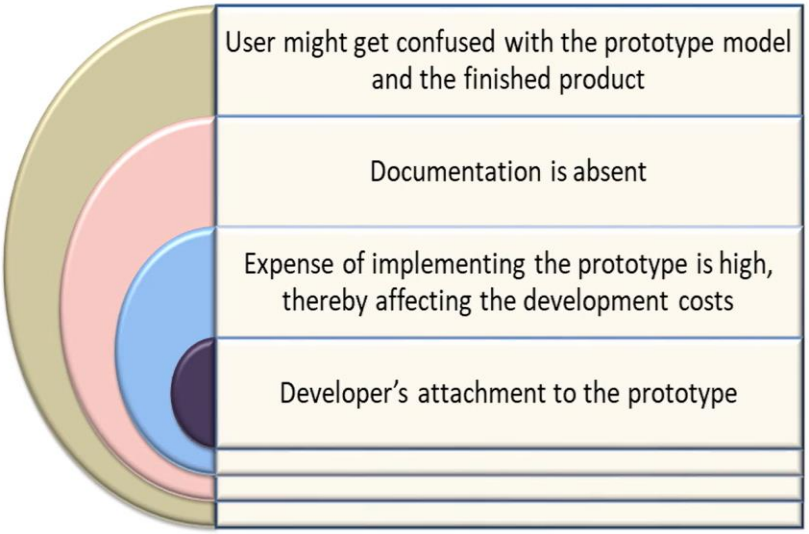
TEKNOTURF

- Requirements are prioritized and the code is developed initially for stable requirements, with an eye on quality
- Software is continuously refined and augmented in close collaboration with the client
- Build the software incrementally
- Adopt a rigorous, systematic approach
- Iterative model

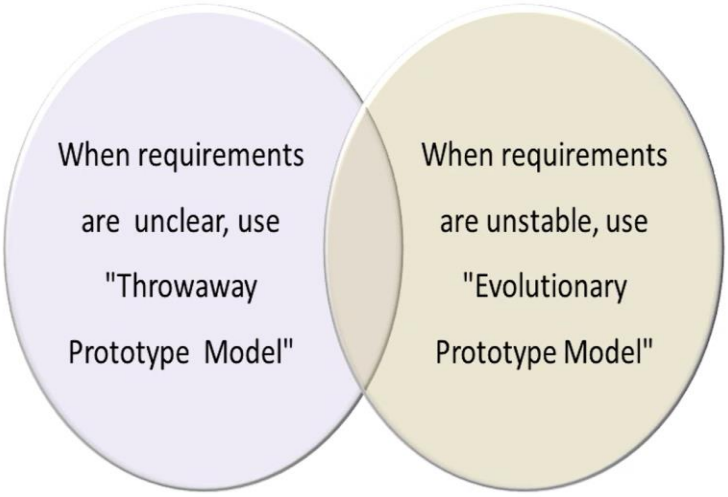
Advantages of Prototype Model

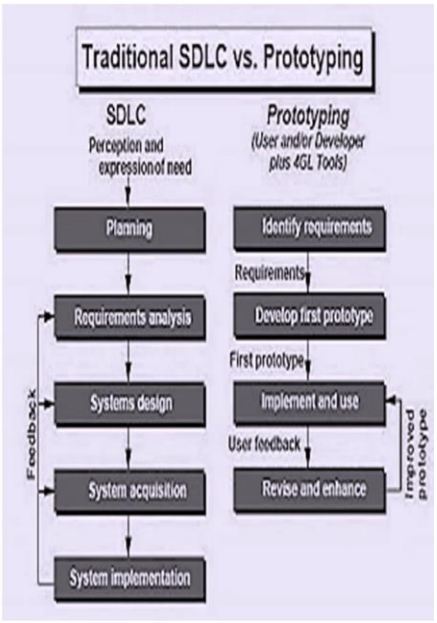


Limitations of Prototype Model



When to use Prototype Model?





Rapid Application Development

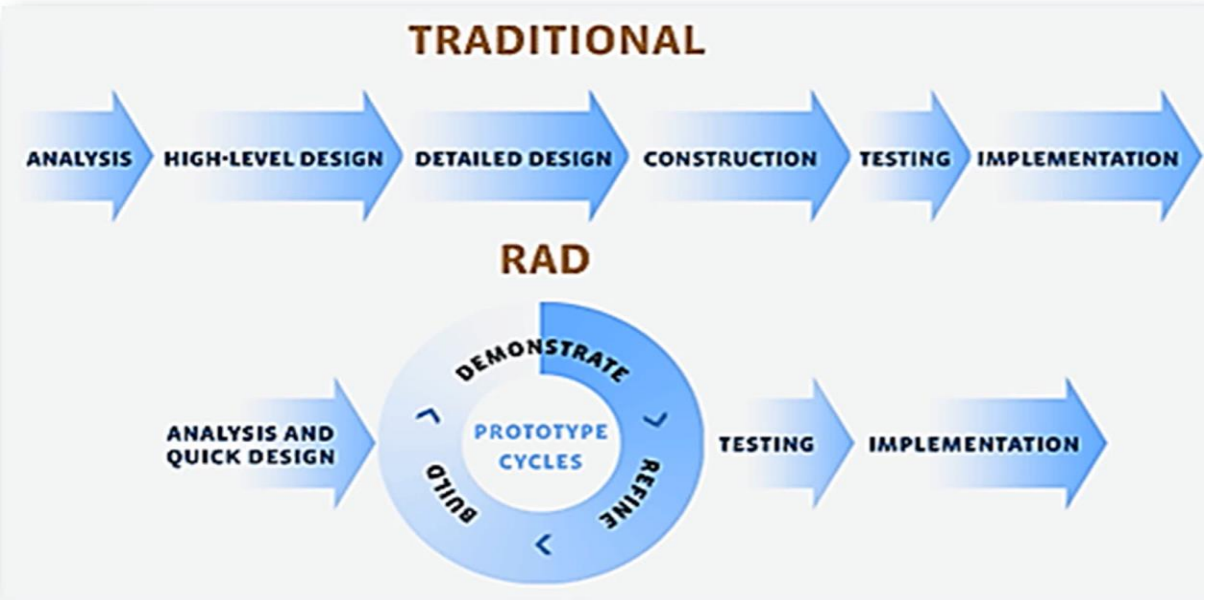
RAD is a high speed version of the linear sequential model

Characterized by a very short development life cycle

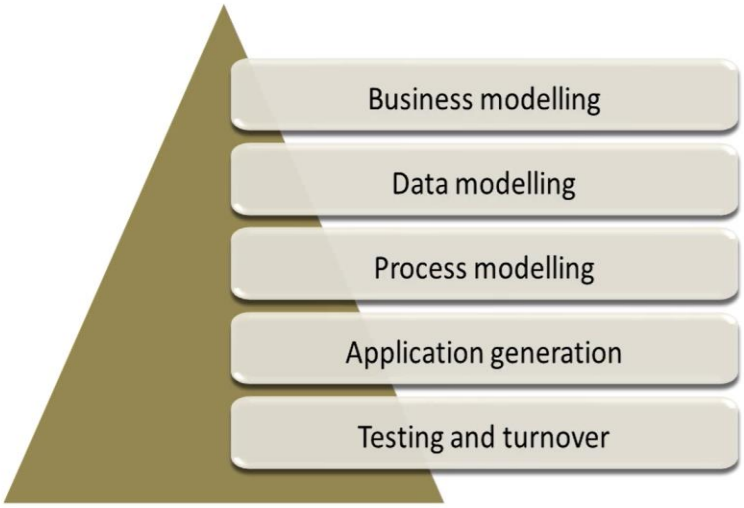
The RAD model follows a component based approach

Individual components are developed by different people and assembled to develop a large software system

Traditional vs RAD



Rapid Application Development phases



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

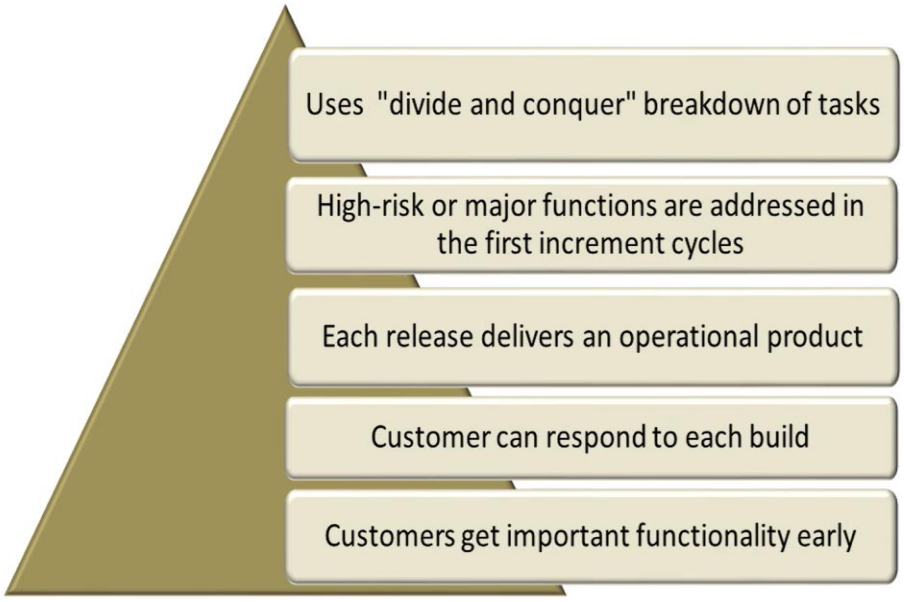
System can be modularized

Incremental Model

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

TEKNOTURF

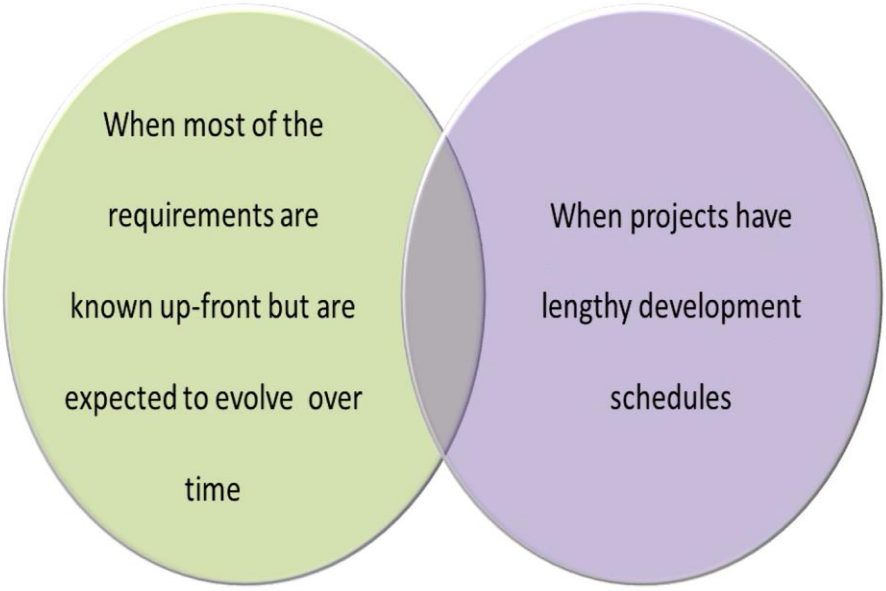
Requires early definition of a complete and fully functional system to allow for the definition of increments

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?

TEKNOTURF



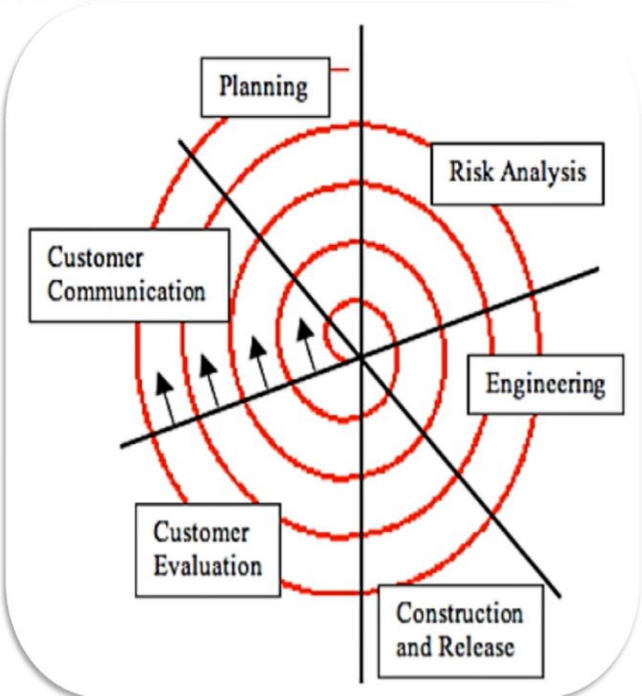
Spiral Model

TEKNOTURF

- 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

TEKNOTURF



Advantages of Spiral Model

TEKNOTURF

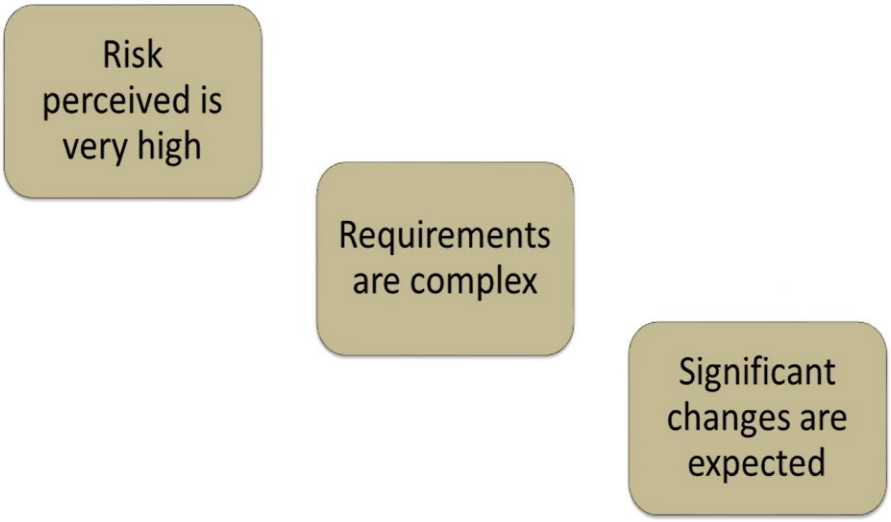
- Provides early indication risk.
- Users see the system early because of rapid prototyping tools.
- Critical high-risk functions are developed first.
- Early and frequent feedback from users.

Limitations of Spiral Model

TEKNOTURF

- Time spent for evaluating risks are too large for small or low-risk projects and may not prove cost-worthy.
- Time spent on planning, resetting objectives, doing risk analysis and prototyping may be excessive.
- Relies on Risk assessment expertise.

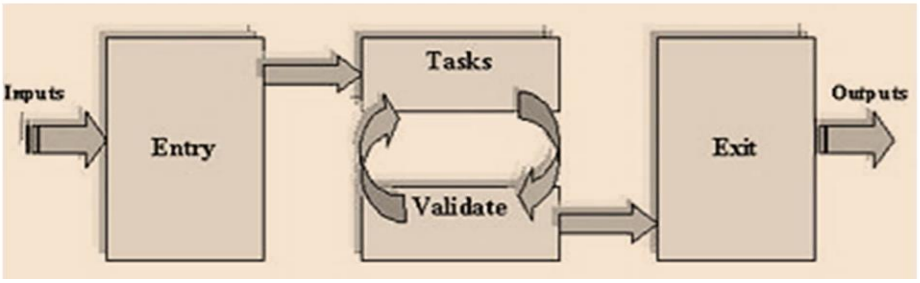
When to use Spiral Model?



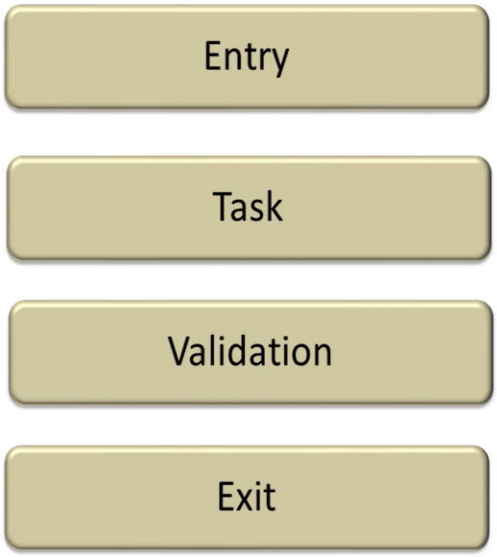
Process Representation Technique – ETVX Model

IBM introduced the ETVX model to document their process.

Each phase in a process performs a well-defined task and generally produces an output.



ETVX Model



ETVX Model Example

Entry

- Hall Ticket

Task

- Show Hall Ticket
- Get Question Paper
- Get Answer Sheet
- Write Answer for Respective questions

Verification

- Verify whether questions are written for appropriate questions
- Review the answers written

Exit

- Submission of the answer sheet to the invigilator

ETVX Model Example for Analysis Phase

TEKNOTURF

Entry

- Feasibility Report

Task

- Collect requirement
- Analyze requirement
- Write SRS

Verification

- Review SRS

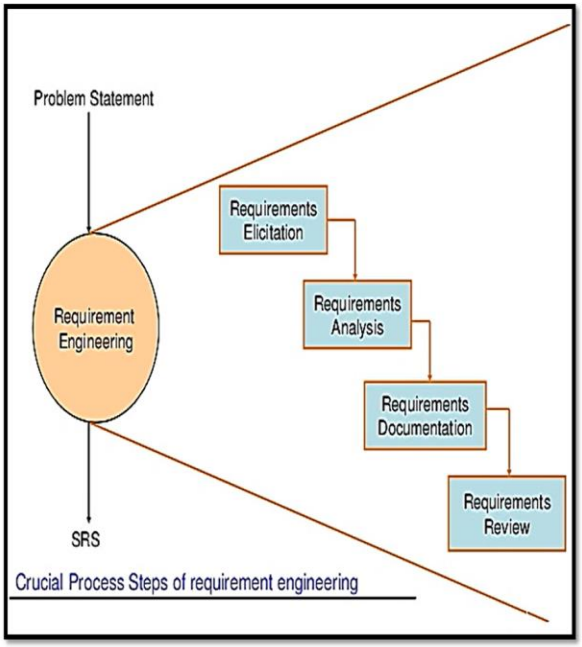
Exit

- SRS

Requirement Engineering

TEKNOTURF

Here you go Tom! You have the requirements for a Matrimonial website in hand. This is the problem statement.



Crucial Process Steps of requirement engineering

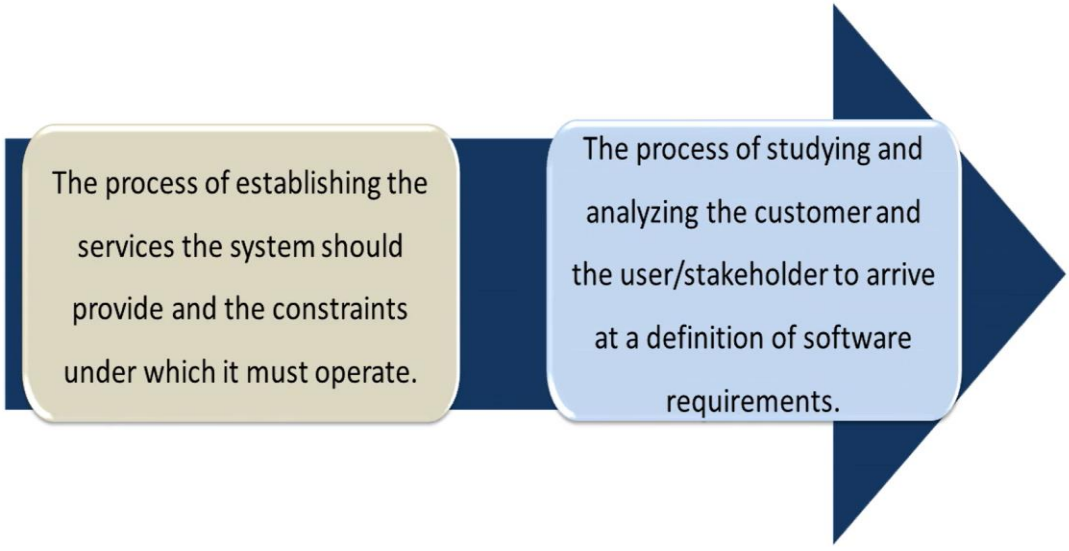
Requirement Elicitation

TEKNOTURF

- Elicit requirements from all stakeholders
- Address problems of scope
- Address problems of understanding
- Address problems of volatility (changing requirements)

Requirement Analysis

TEKNOTURF



Requirement Analysis

TEKNOTURF

Why is Requirement Analysis difficult?

Different "worlds"

- Bridging the gap between the client and the software developer
- Knowing what should be done VS knowing what to let a computer do

Users/stakeholders are not a uniform group

- conflict between cost and usability / performance / features
- conflicting demands from different departments

Getting the good (ideal) system
Vs
possibility of building it well

Requirement Analysis

TEKNOTURF

Other factors

- Expectations, the final solution is difficult to imagine by the users
- Scope of the system
  needs well defined boundaries
- Current vs. future system
  resistance to change
- Process of negotiation between users and designers

Requirement Analysis

TEKNOTURF

Goals of requirement Analysis and specification phase	• Understand the user's requirements • Remove inconsistencies, anomalies, etc. from requirements • Document requirements properly in an SRS document

Requirement Analysis

TEKNOTURF

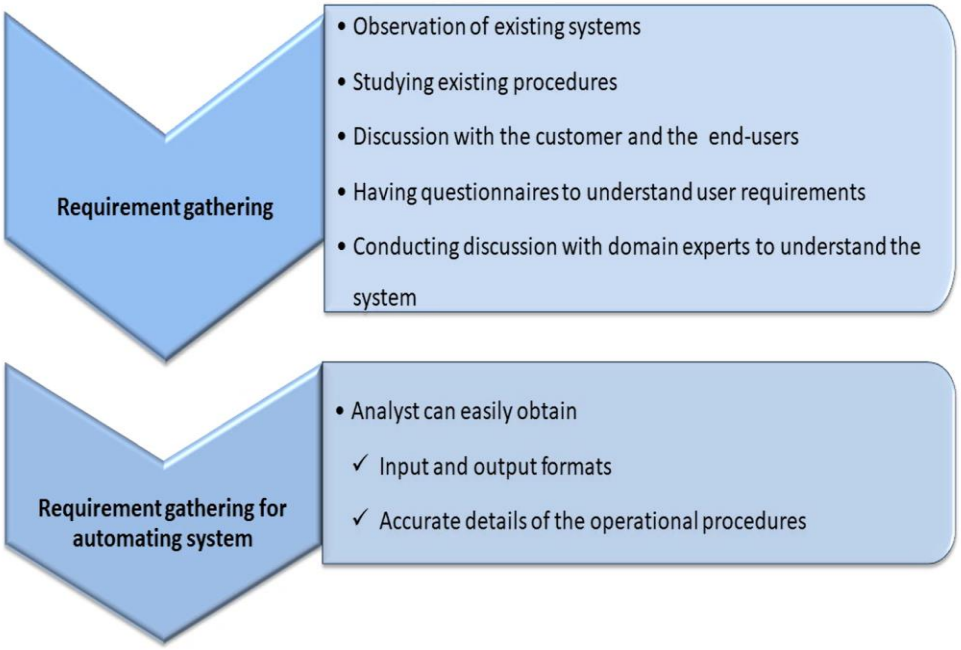
A person who performs requirement analysis is called a system analyst

- Understands User requirement
- Collects data needed for the user requirement
- Writes the software requirement specification

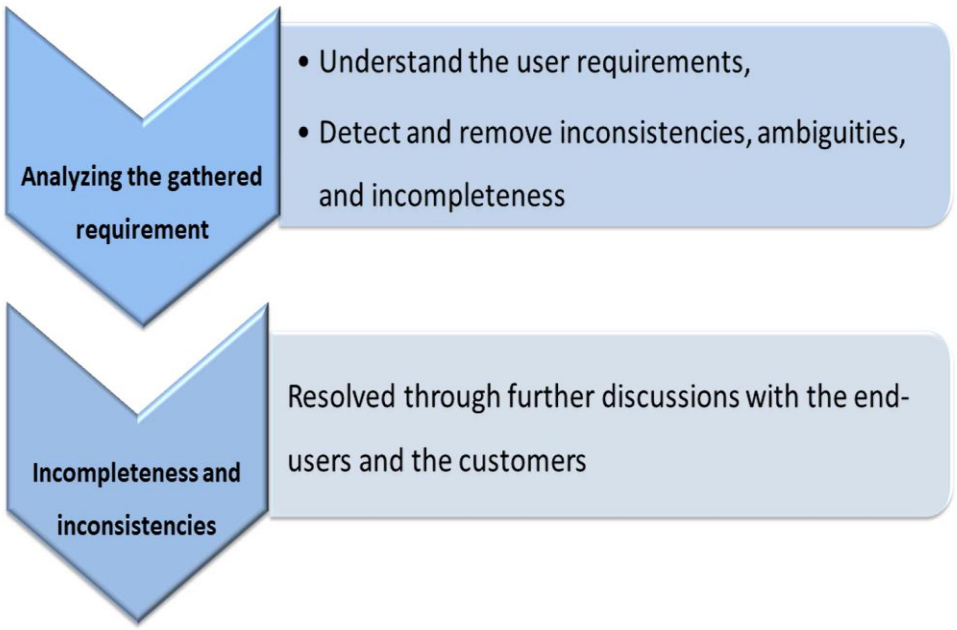
Requirements analysis consists of two main activities:

- Requirements gathering
- Analysis of the gathered requirements

Requirement Analysis



Requirement Analysis



Contradicting Requirement

TEKNOTURF

One requirement conflicts with another requirement.

Tom have you come across any contradicting requirements? Can you give an example?

Sure!

When pouring hot filling into chocolate candy shells,

- the filling should be hot enough to pour quickly
- but it should also be cold enough to prevent melting of the chocolate.

Incomplete Requirement

TEKNOTURF

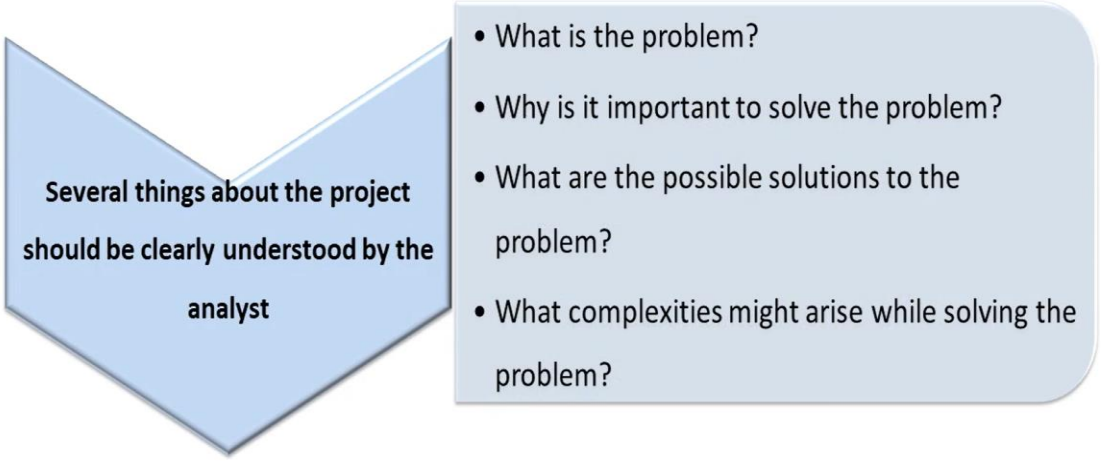
Some requirements have been omitted:

- **Due to oversight.**

Example:

- A solar heater specifies what it will do on a 'sunny day' but it fails to specify what it will do on a 'rainy day'
- The analyst has not recorded when temperature falls below 90 degrees.
  - heater should be turned ON
  - water shower turned OFF.

Requirement Analysis



Software Requirements Specification

Outcome of the analysis phase

Contains all the information about the requirements gathered

Main aim of requirements specification:

- Systematically organizes the requirements arrived during requirements analysis
- Documents requirements properly.

SRS

TEKNOTURF

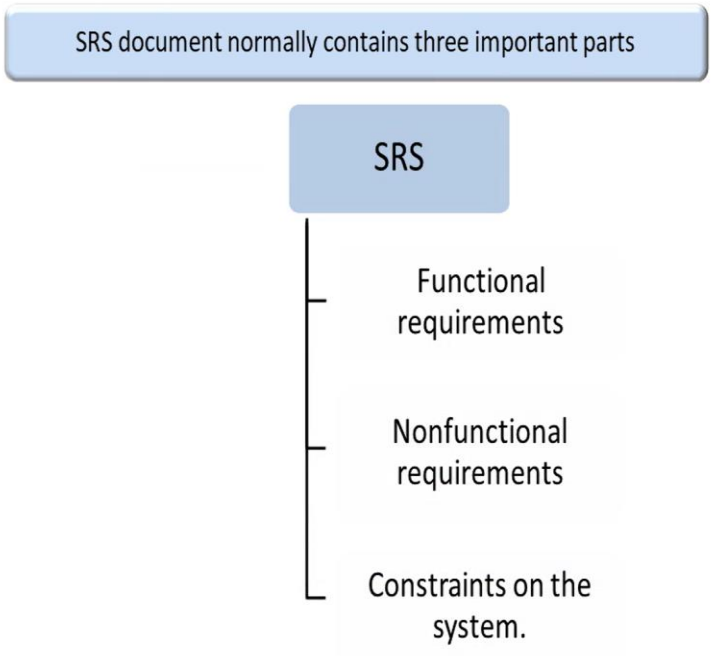
SRS document concentrates on	The SRS document is useful in various contexts
• What needs to be done • Carefully avoids the solution ("how to do") aspects.	• Statement of user needs • Contract document • Reference document • Definition for implementation

Properties of a good SRS

TEKNOTURF

- Concise
- Specify what the system must do
- Easy to change
- Consistent
- Complete
- Traceable
- Verifiable

SRS Document

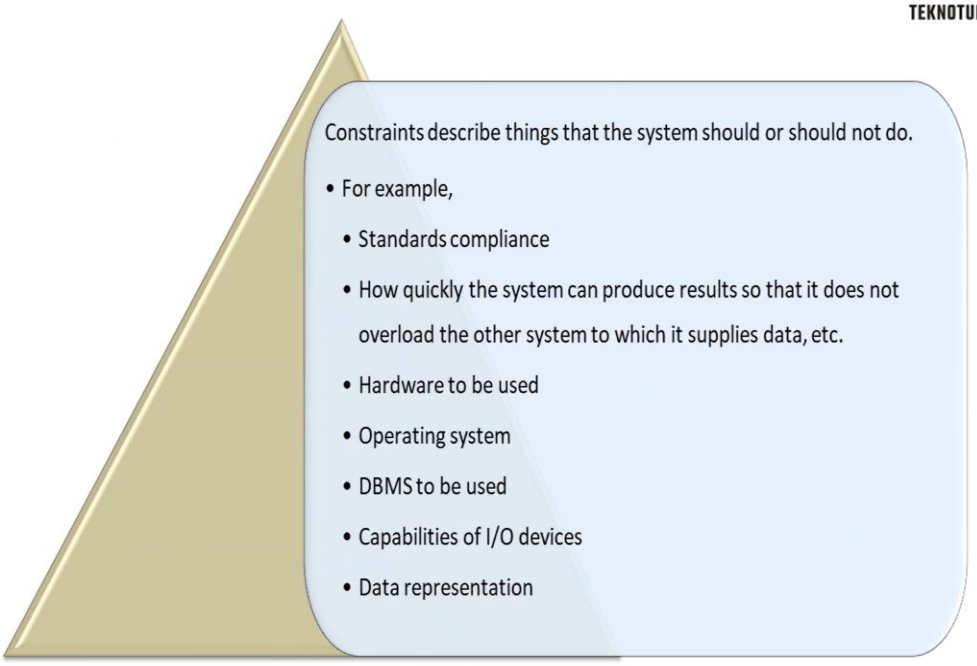


Functional Requirement

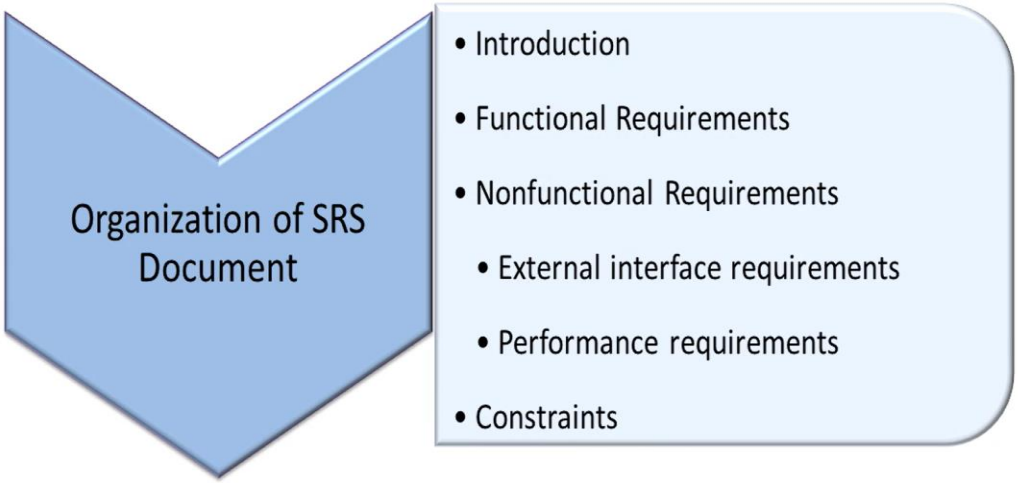
Functional requirements describe

- A set of high-level requirements
- Each high-level requirement
- Takes in some data from the user
- Outputs some data to the user
- Might consist of a set of identifiable functions which process the input

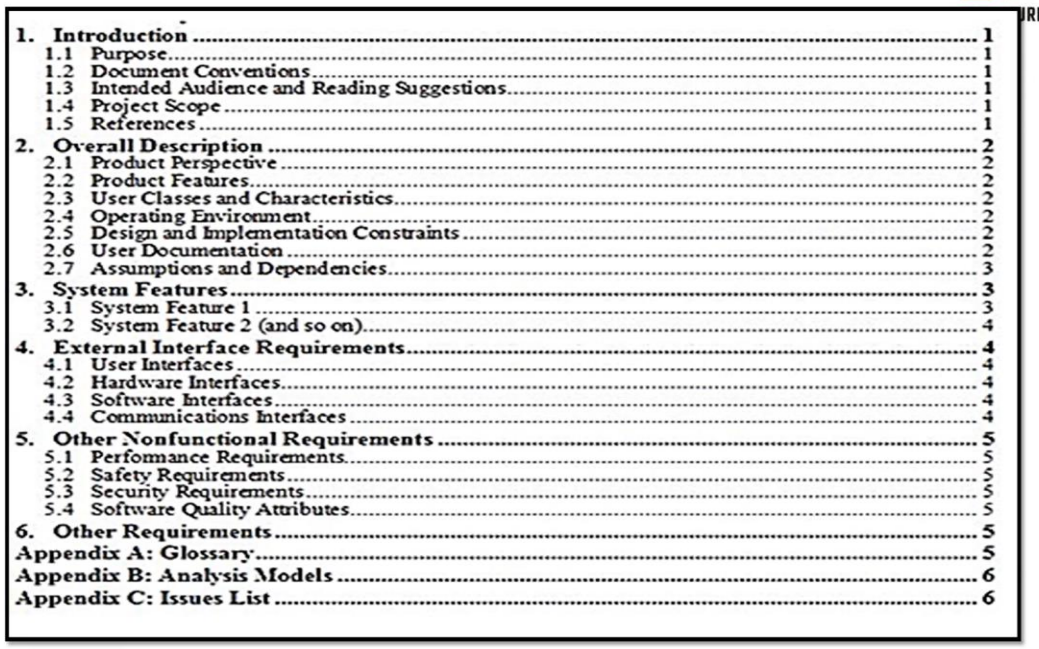
Constraints



SRS Document



SRS Document Structure



ERD

Entity Relationship Diagram

ER diagram is widely used in database design

- Represents conceptual level of a database system
- Describes things and their relationships in high level

ERD

Entity

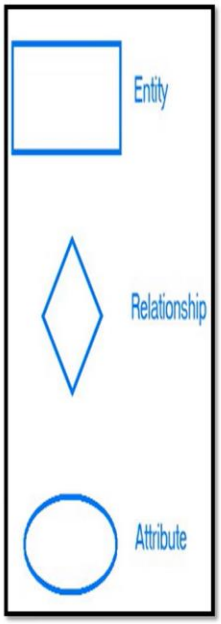
- An entity is a business object that represents a group, or a category of data.

Attribute

- Properties of an entity.

Relationship

- specifies the relations among entities



ERD

Relationship specifies association between two entities.

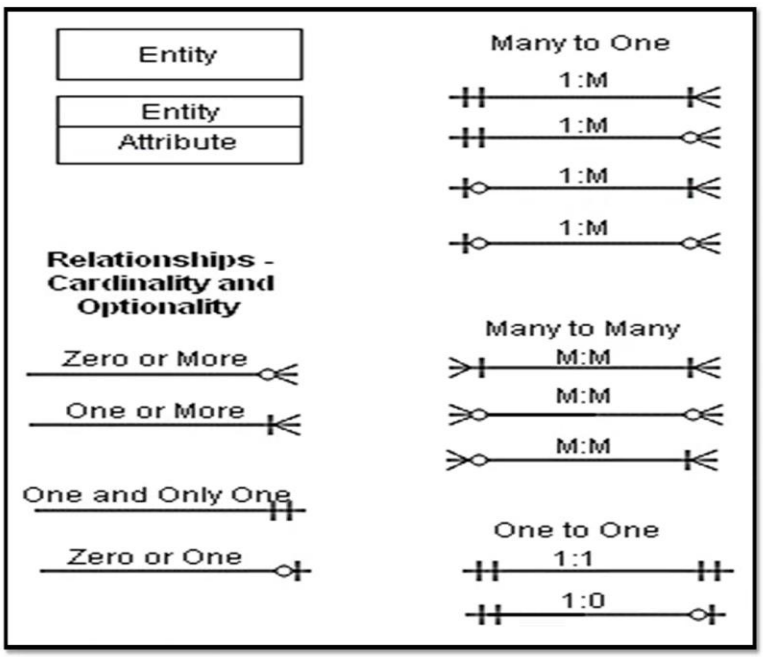
Cardinality

- One instance of an entity maps to how many instance of other entity
- Many-to-Many Relationships
- One-to-Many Relationships
- One-to-One Relationships
- Recursive Relationships

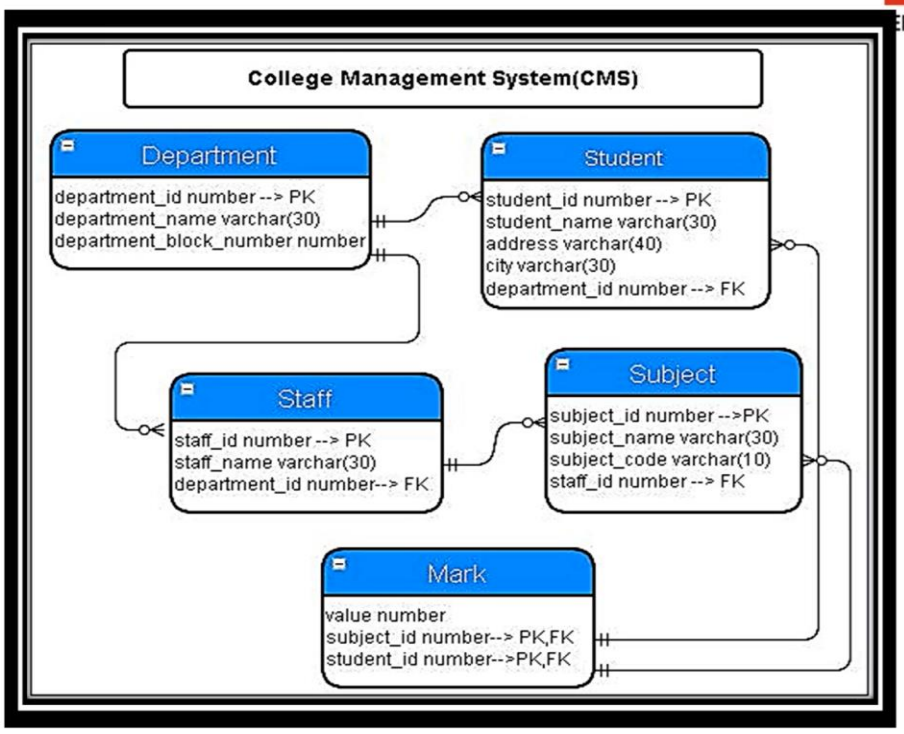
Optionality

- Is the relationship mandatory or optionality
- Mandatory Relationships
- Optional Relationships

ERD Example



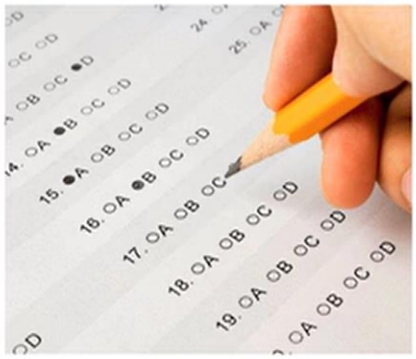
ERD Example



Software Testing

Process of executing the program with the intent of finding errors

Process of verifying and validating so that a software application or program meets the business and technical requirements that guide its design and development work as expected.



Fundamentals of Testing

Testing is the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements, or to identify differences between expected and actual results.

Testing can be used to show the presence of defects, but never their absence!

Importance of Testing

TEKNOTURF

Most cost effective approach to build confidence in most of the software systems

Detects the faults in the software

Improves the quality and reliability by removing the faults from the software

Testing Profession

TEKNOTURF

The testing profession is not about just creating test related documents.

- It is understanding the application functionality and coding
- Testers are the ones who are the certifying authority for the software itself
- Testers destruct the software to give better solutions

Who will test a Software ? What makes one a good Tester?

Who performs testing?

- Programmers and testers

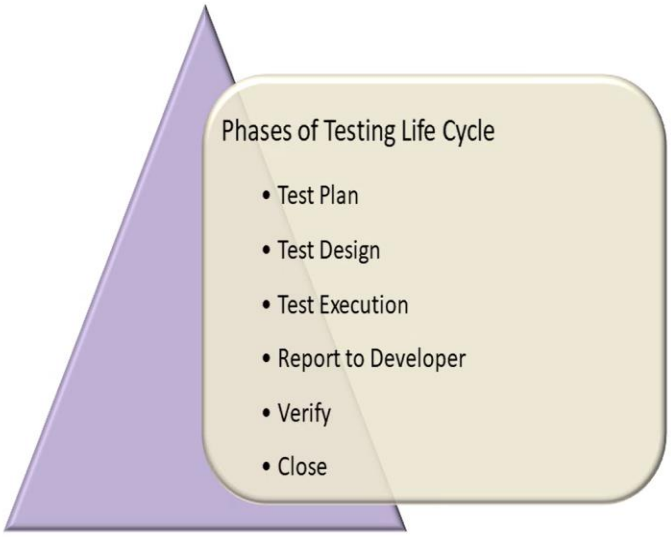
What should a good tester possess?

- Creative thinking
- Questioning skills
- Never give up attitude

Testing Life Cycle

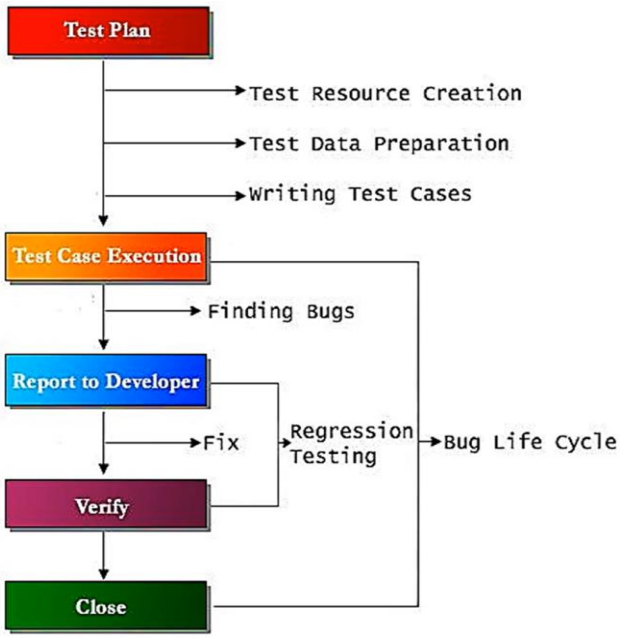
TEKNOTURF

- Software testing life cycle identifies what test activities to carry out and when (what is the best time) to accomplish those test activities



Testing Life Cycle

TEKNOTURF



Methodology of Testing

TEKNOTURF

Manual Testing

What are the different methods of Testing?

Automated Testing

- Example
  - Java – JUNIT
  - Dot Net – N UNIT

Bugzilla

- Defect Tracking Tool

Levels of Testing

TEKNOTURF

What are the different Levels of Testing?

Unit Testing

Integration Testing

System Testing

Acceptance Testing

Unit Testing

TEKNOTURF

Unit Testing is used to check whether a particular module is implementing its specification

The cost of fixing a defect detected during unit testing is lesser in comparison to that of defects detected at higher levels.

Who performs testing?

- normally performed by software developers themselves or their peers
- In rare cases it may also be performed by independent software testers

Integration Testing

TEKNOTURF

Integration Testing is a level of the software testing process where individual units are combined and tested as a group.

The purpose of Integration Testing is to expose faults in the interaction between integrated units.

Who performs this testing?

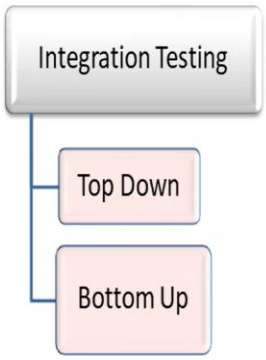
- Either Developers themselves or independent Testers

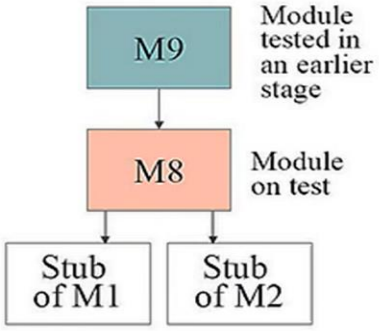
Integration Testing Approaches

TEKNOTURF

Top Down

- Top level units are tested first and lower level units are tested step by step after that
- Test Stubs are needed to simulate lower level units which may not be available during the initial phases



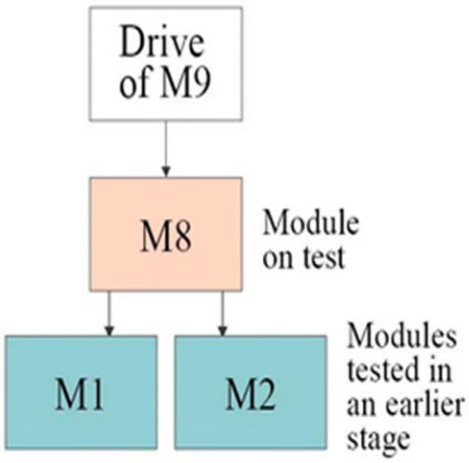


Integration Testing Approaches

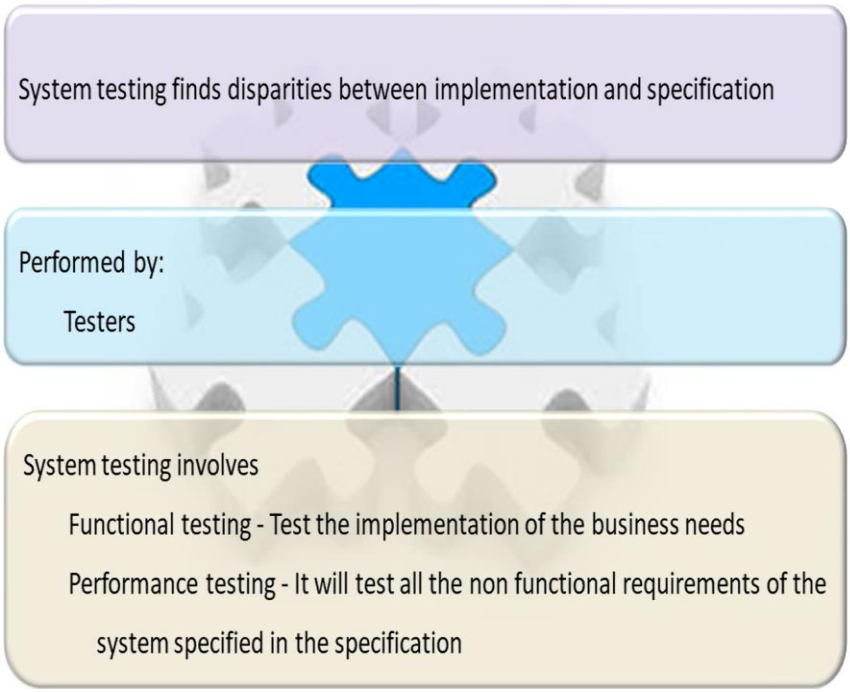
TEKNOTURF

Bottom Up

- Bottom level units are tested first and upper level units step by step after that
- Test Drivers are needed to simulate higher level units which may not be available during the initial phases



System Testing



Performance Testing - Types

Stress tests

- evaluates the system when stressed to its limits

Regression tests

- required when the system being tested is replacing an existing system

Usability test

- testing characteristics related to user friendliness

User Acceptance Testing

TEKNOTURF

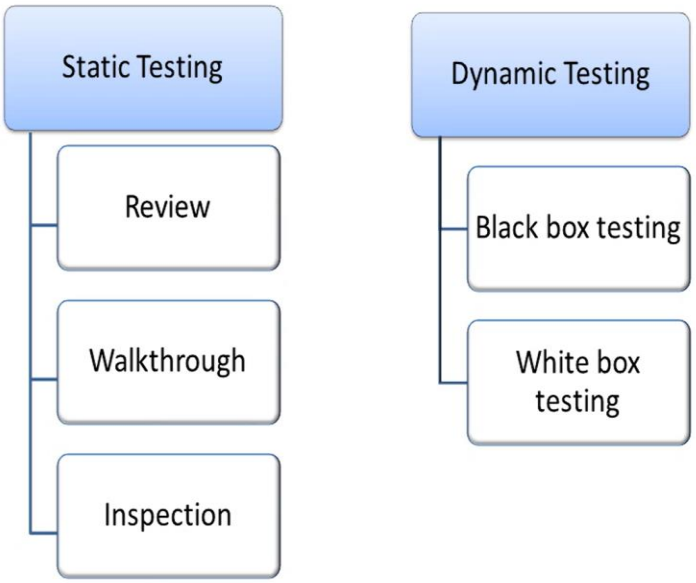
Done by the client to test whether the system meets the specified requirements

Types

- **Alpha testing** - Done by the client in developer's environment
- **Beta testing** - Done by the client in real world environment

Types of Testing

TEKNOTURF



Static Testing

Static testing is the testing of the software work products manually to find errors

Work product means all the documents related to software as well as the code

Execution of the Code is not performed. Sanity of the code is checked.

Members of Static Testing

- Author
- Moderator
- Reader
- Recorder/Scribe
- Inspector

Techniques in Static Testing

- REVIEW
  - It is a process in which the product is re-examined or re-evaluated for possible corrections
- WALK THROUGH
  - Mostly done on the code that is developed
  - The author gives sample test data. The test data examined with the code and intermediate results are recorded
- INSPECTION
  - Inspection involves step by step reading of the product, with each step checked against a predefined list of criteria(historical common errors, standards)

Review

- Review is a static testing technique.
- During review the artifacts are manually examined and the findings are recorded.
- Artifacts for which reviews are performed are
  - ❖ SRS
  - ❖ Design specification
  - ❖ Source code
  - ❖ Test plans
  - ❖ Test specification
  - ❖ Test cases
  - ❖ Test scripts
  - ❖ User guides
  - ❖ web pages

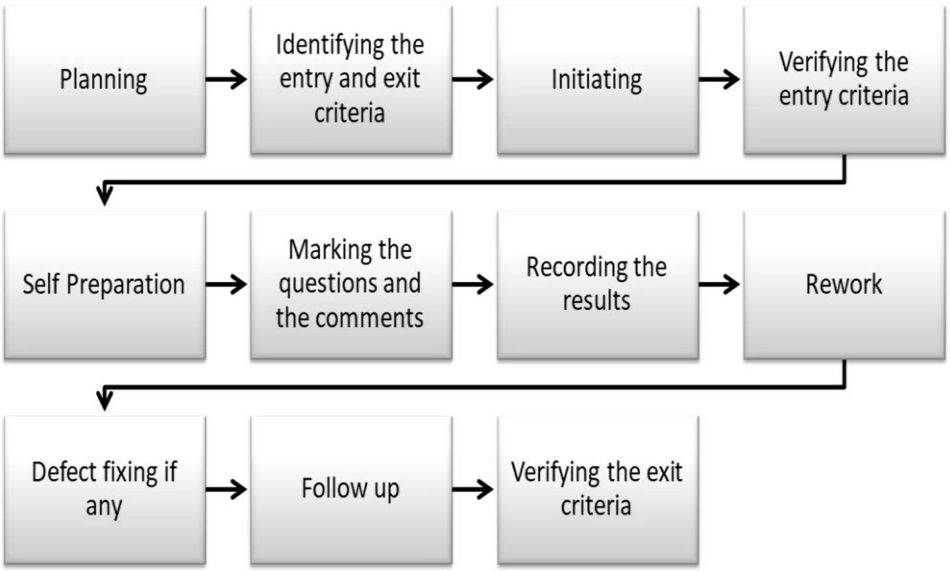
Review Advantages

TEKNOTURF

- Early defect detection and correction
- Fewer defects
- Helps in identifying omissions
- Saves testing cost and time

Review Process

TEKNOTURF



Types of Review

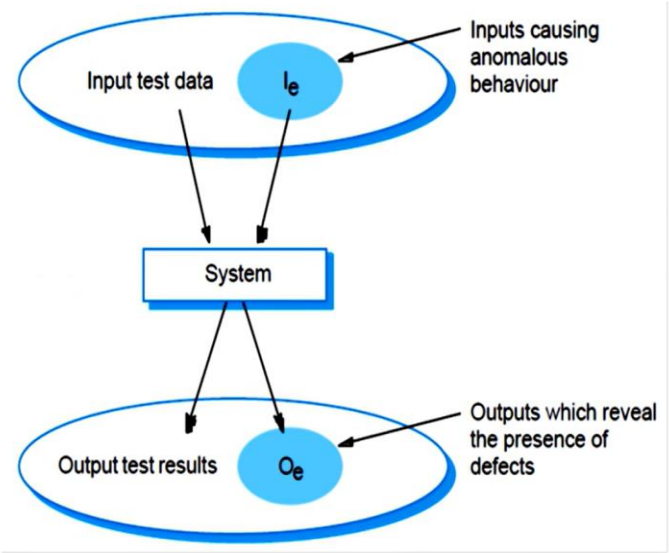
TEKNOTURF

- Informal Review
- Walkthrough
- Technical Review
- Inspection

Dynamic Testing – Black Box Testing

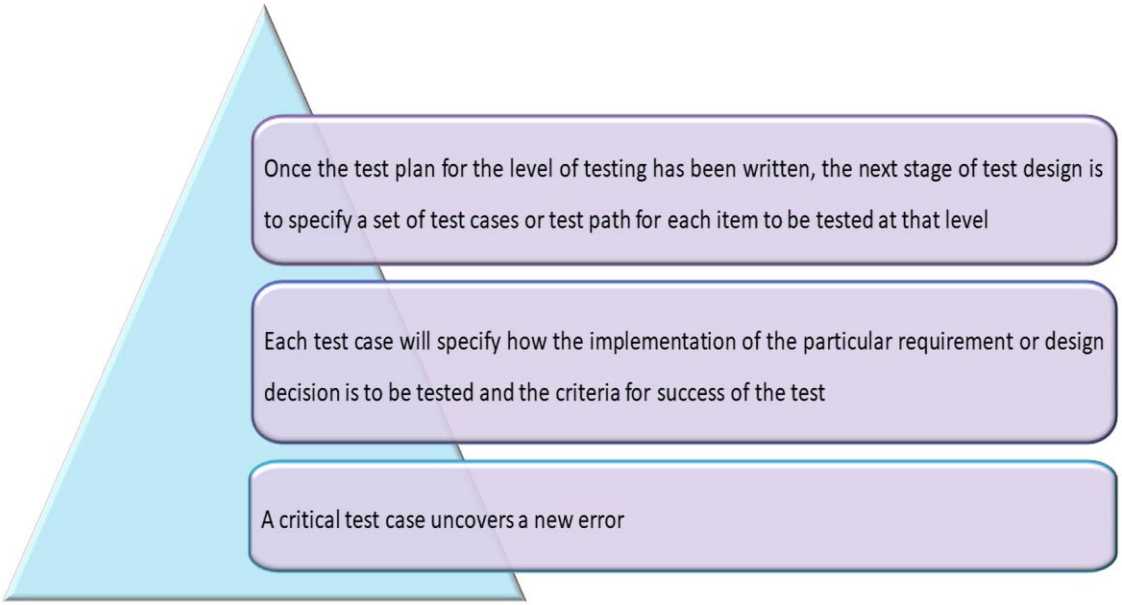
TEKNOTURF

It is testing that ignores the internal mechanism of a system or a component and focuses solely on the outputs generated in response to selected inputs and execution conditions.



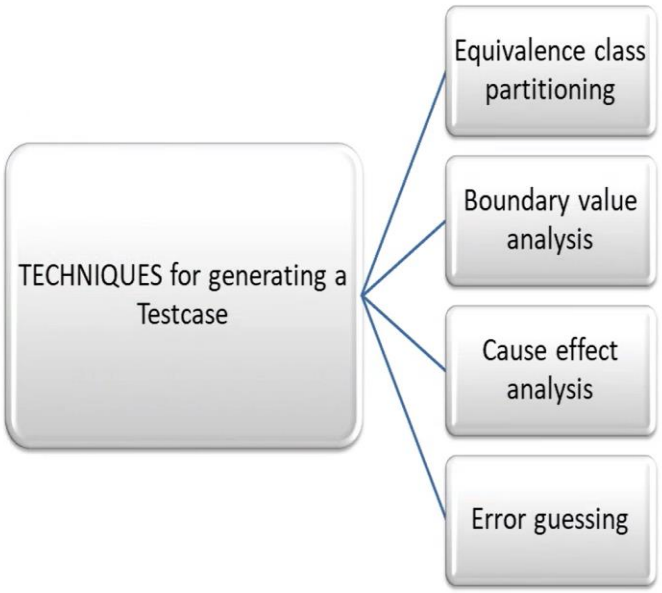
Test Case

TEKNOTURF



Black Box Testing

TEKNOTURF



Equivalence Class Partitioning

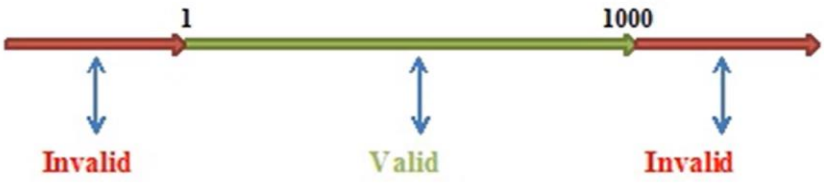
TEKNOTURF

Black-box technique divides the input domain into classes of data from which test cases can be derived.

For each of these equivalence classes, the set of data should be treated the same by the module under test and should produce the same answer.

Equivalence class guidelines:

- For a range of input choose three values such that,
  - One value is above the range
  - One value is below the range
  - One value is within the range



Equivalence Class Partitioning

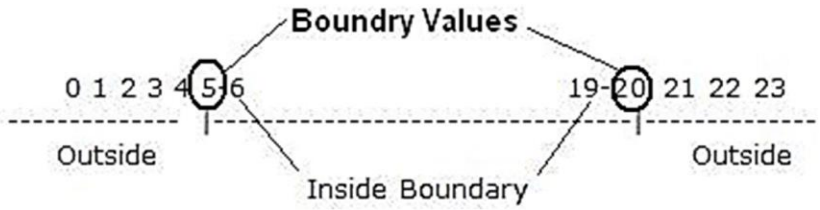
TEKNOTURF

- EXAMPLE SCENERIO
  - The component "generate_grading" is passed an exam mark (out of 75) and a coursework (c/w) mark (out of 25), from which it generates a grade for the course in the range 'A' to 'D'. The grade is calculated from the overall mark, which is calculated as the sum of the exam and c/w marks, as follows:
    - greater than or equal to 70 - 'A'
    - greater than or equal to 50, but less than 70 - 'B'
    - greater than or equal to 30, but less than 50 - 'C'
    - less than 30 - 'D'

    - When a mark is outside its expected range, then a fault message ('FM') is generated. All inputs are passed as integers.

Boundary Value Analysis

- Black-box technique focuses on the boundaries of the input domain rather than its center
- Boundary value analysis guidelines:
  - If input condition specifies a range bounded by values a and b, test cases should include a and b, values just below a and just above b



Cause Effect Analysis

The cause effect analysis helps in identifying the causes and their effects (and identify how they are linked) associated with a particular problem or situation.

A cause is an input condition or an equivalence class of input conditions. An effect is an output condition or a system transformation.

It is suitable for applications in which combinations of input conditions are few.

Cause Effect Analysis

EXAMPLE:

- A module for calculating increment, that decides how much increment percent will be assigned to every employee. The decision based on the experience and the designation
  - For exp (valid input range is 1 to 50)
    - Between 1 – 3 then inc% 10
    - Between 4 – 10 then inc% 20
    - Greater than 10 then inc% 30
  - For designation
    - Developer then inc% 30
    - Tester then inc% 40

Error Guessing

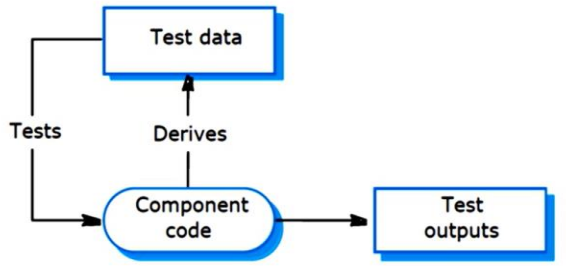
- It is an ad hoc approach guided by intuition and experience
- Example:
  - Consider a program is written into a file. The following test cases can be derived
    - Having an empty file
    - Not at all having a file
    - Having a file with read permission

Dynamic Testing – White Box Testing

It deals with the internal logic and structure of the code

Derivation of test cases according to the program structure

It is also called as glass box testing, structural testing, clear box testing



Basis Path Testing

- Basis Path Testing is white box testing method
- Design test cases to cover the following in the code written
  - ❖ every statement(Statement coverage)
  - ❖ every predicate (condition) in the code(branch coverage)
  - ❖ loops (loop coverage)

Basis Path Testing

Derive a logical complexity measure of procedural design

- Break the module into blocks delimited by statements that affect the control flow (eg condition, method call, loop)
- Mark out these as nodes in a control flow graph
- Draw connectors (arcs) with arrow heads to mark the flow of logic
- Identify the number of regions (Cyclomatic Number) which is equivalent to the McCabe's number

McCabe's Number (Cyclomatic complexity)

- It defines the number of independent paths
- Provides the number of tests that must be conducted to ensure that all the statements are executed at least once.

Basis Path Testing

Complexity of a flow graph 'G', v(G), is computed in one of three ways

- V(G) = No. of regions of G +1
- V(G) = E - N +2 (E: No. of edges & N: No. of nodes)
- V(G) = P + 1 (P: No. of predicate nodes in G or No. of conditions in the code)

Define a basis set of execution paths

Determine independent paths

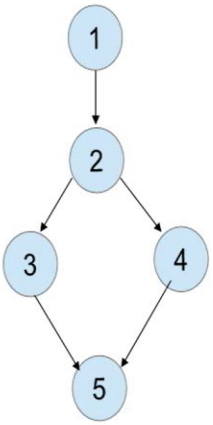
Eliminate infeasible paths

Derive test case to exercise (cover) the basis set

Basis Path testing

```
• int main() {
  – int a,b;
  – scanf("%d%d",&a,&b);
  – if(a > b)
  –   printf("\na is greater");
  – else
  –   printf("\n b is greater");
  – }
```

1
2
3
4
5



Mc cab's No
P+1 = 2
E-N+2 = 5-5+2 = 2
R+1 = 2

Individual Path	testcase
1-2-3-5	a=10 , b=5
1-2-4-5	a=5 , b=10

Basis Path Testing

```
• int main() {
  – int n,i;
  – scanf("%d",&n);
  – for(i=1;  i<=n;  i++){
  –   printf("\n%d",i);
  – }
• }
```

1
2
3
4
5
6

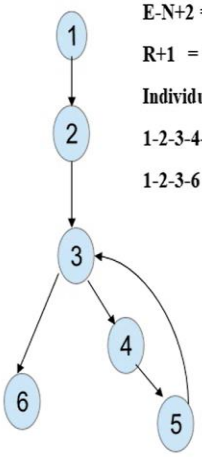
Mc cab's no

P+1 = 2

E-N+2 = 6-6+2 = 2

R+1 = 2

Individual Path	
1-2-3-4-5-3-6	n=2
1-2-3-6	n=0



Black Box vs. White Box Testing

TEKNOTURF

BLACK BOX TESTING

- Test cases are designed depending on the functionality
- Identifies hidden functionality
- Techniques
  - Equivalence class partitioning
  - Boundary value analysis
  - Cause effect analysis and graphing
  - Decision table

WHITE BOX TESTING

- Test cases are designed depending on the logic of the code
- Identifies unreachable code and checks for code coverage
- Techniques
  - Basis path testing

Static Testing Vs Dynamic Testing

TEKNOTURF

STATIC TESTING

- manual examination (reviews) of the code or other project documentation
- Testing methodologies
  - Review
  - Inspection
  - Walkthrough
- Bugs discovered at this point are less expensive to fix

DYNAMIC TESTING

- the examination of the physical response from the system to variables that are not constant and change with time
- Testing methodologies
  - Black box testing
  - White box testing
- Depends on the type of bug identified.

When to Stop Testing?

TEKNOTURF

Some of the common factors to stop Testing

- Testing budget of the project
- Resources available and their skills
- Project deadline and test completion deadline

Debugging

TEKNOTURF

- Debugging is the process of finding the source of already identified bugs and fixing it.
- Performed by Developers.
- Steps in debugging
  - Find the defect in the code
  - Identify the root cause of the problem
  - Identify the exact place in the code that is the cause of the problem
  - Fix the defect
  - Recheck to ensure that the defect fixed is working as expected

Software Evolution

TEKNOTURF

It is impossible to produce a system of any size which does not need to be changed.

Parts of the software may have to be modified to correct the errors that are found in operation, or to improve its performance or other non-functional characteristics.

After delivery, software systems always evolve in response to demand for change.

Software Evolution Approaches

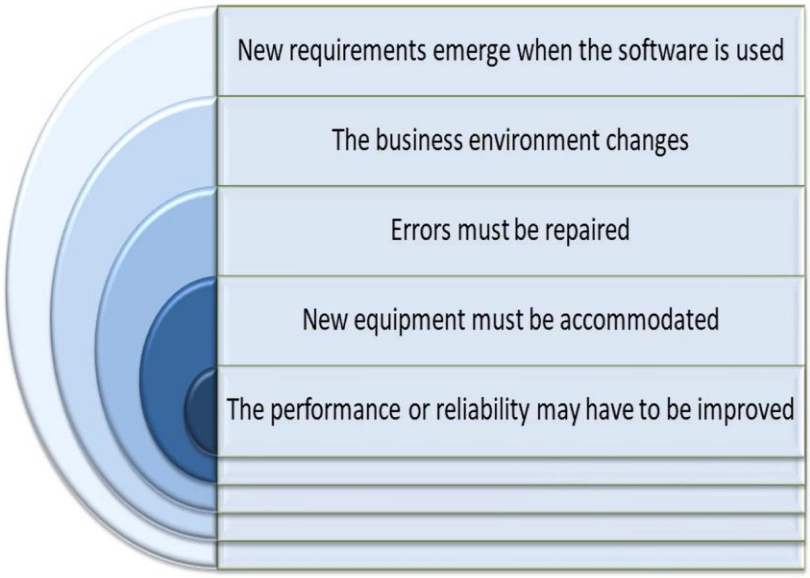
TEKNOTURF

Software maintenance

Architectural transformation

Software re-engineering.

Software Change



Software change strategies

Software Maintenance

- Changes are made in response to changed requirements but the fundamental software structure is stable

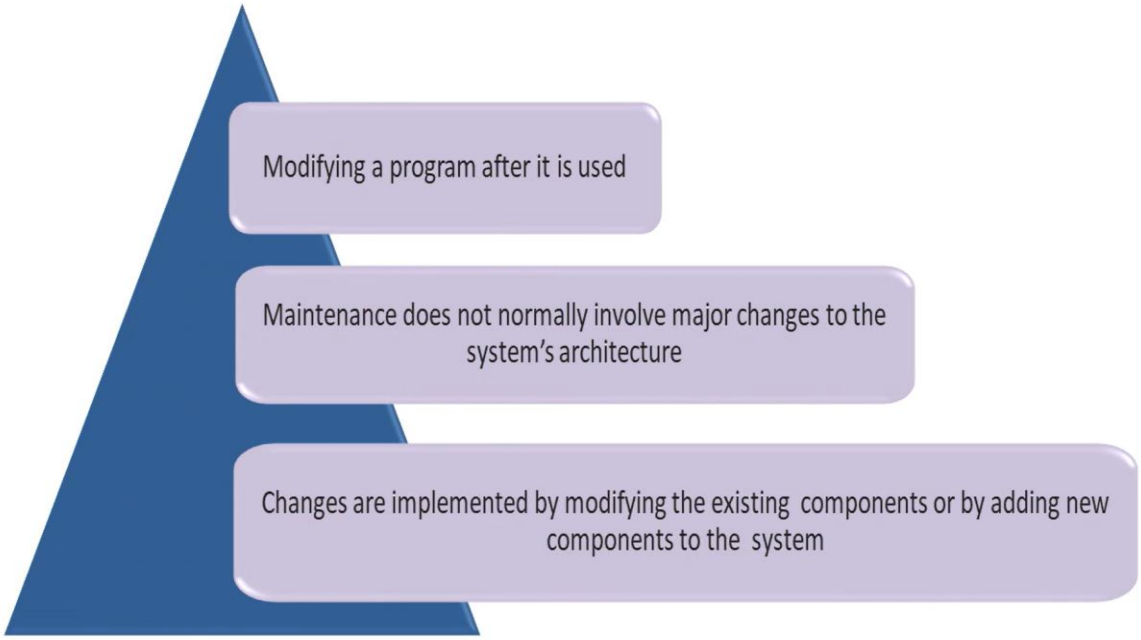
Architectural Transformation

- The architecture of the system is modified
- Generally from a centralized to a distributed architecture

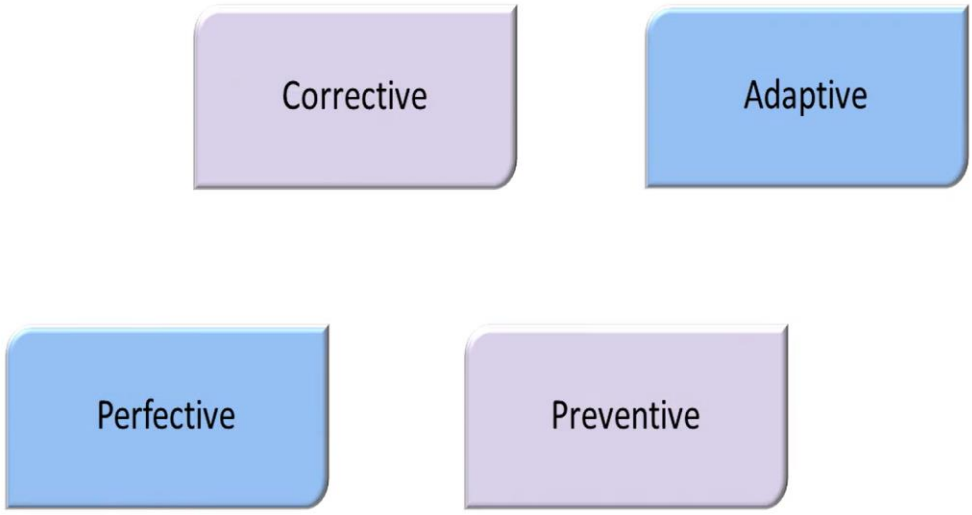
Software Re-engineering

- New functionality is not added to the system but it is restructured and reorganized to facilitate future changes

Software Maintenance



Types of maintenance



Types of maintenance

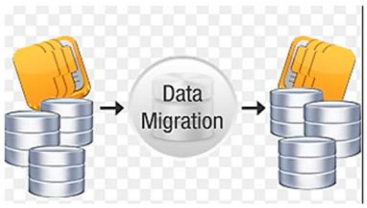
TEKNOTURF

Corrective maintenance

- Maintenance that includes the repair of defects in an existing system
  - Defects can stem from
    - Requirements specification errors
    - Design errors
    - About 80% of all problems stem from requirements and design
    - Coding errors

Adaptive maintenance

- Maintenance to adapt software to changes in the working environment
- Invoked by:
  - Internal needs
- External requirements e.g. changes in law



Types of maintenance

TEKNOTURF

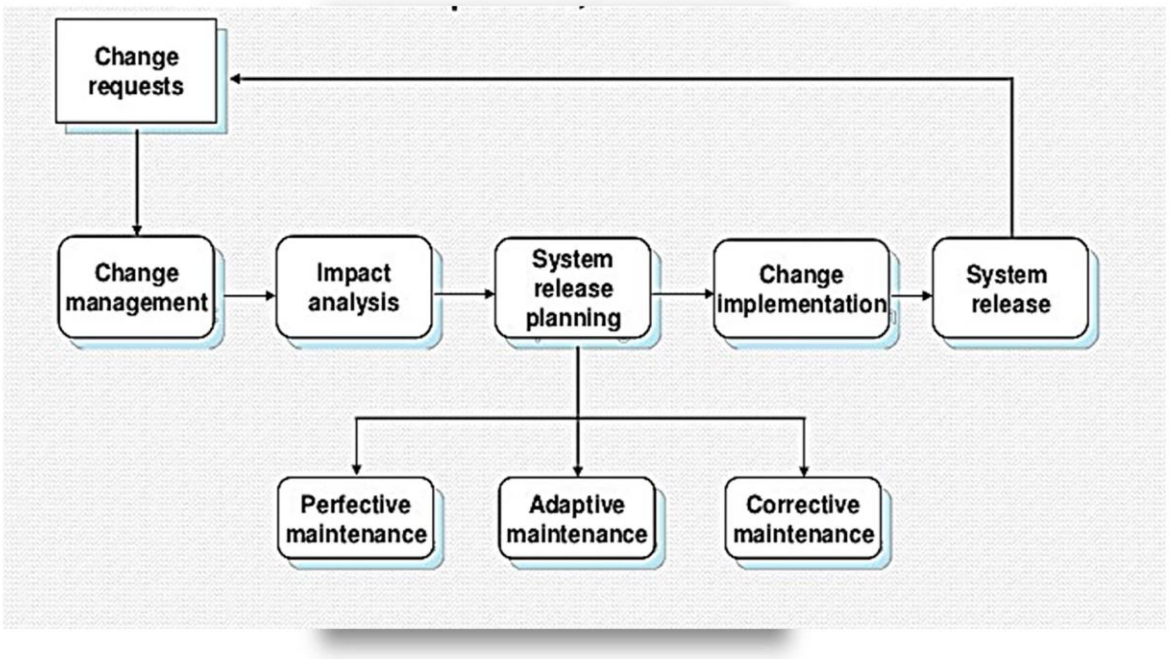
Perfective maintenance

- Includes all efforts to polish or refine the quality of the software or the documentation
- Important that the improvement reduces the system maintenance costs

Preventive maintenance

- Changes made to the system to avoid any software fault in the future

Software Maintenance



Maintenance costs

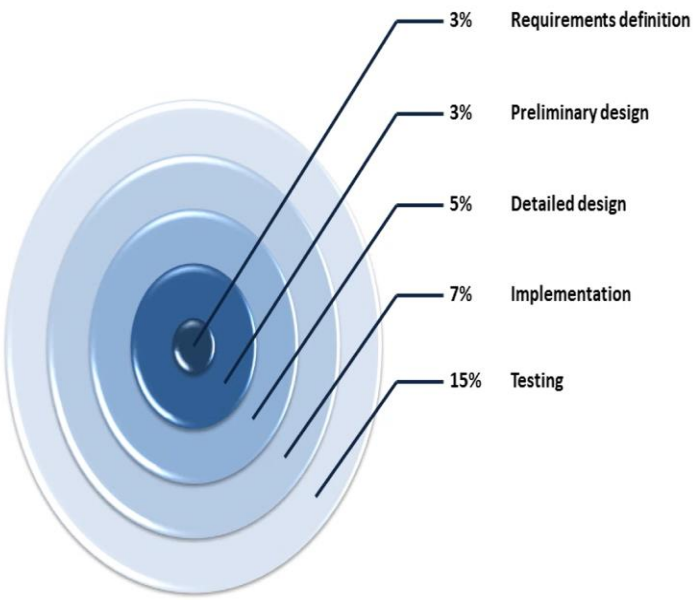
Usually greater than the development costs

Affected by both technical and non-technical factors

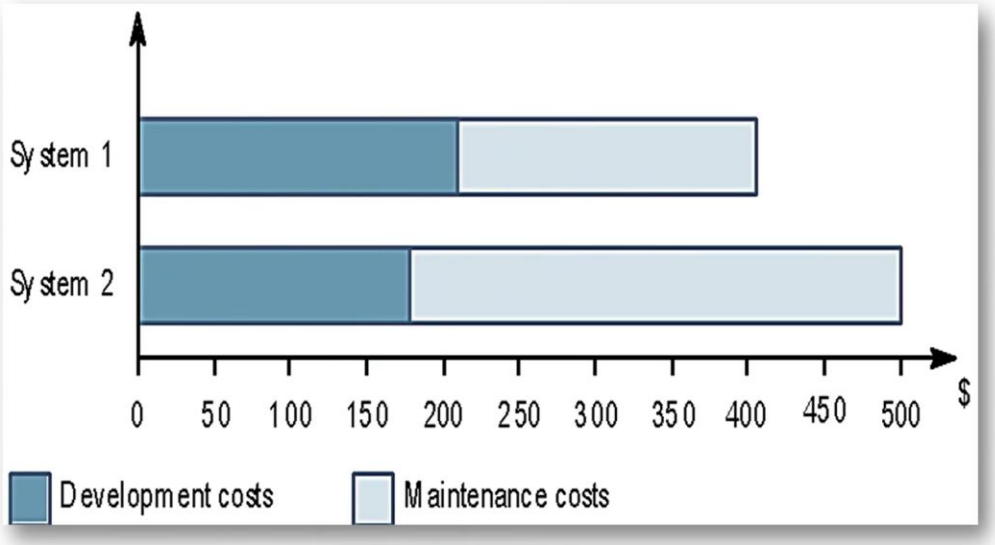
Increases as software is maintained

Ageing software can have high support costs

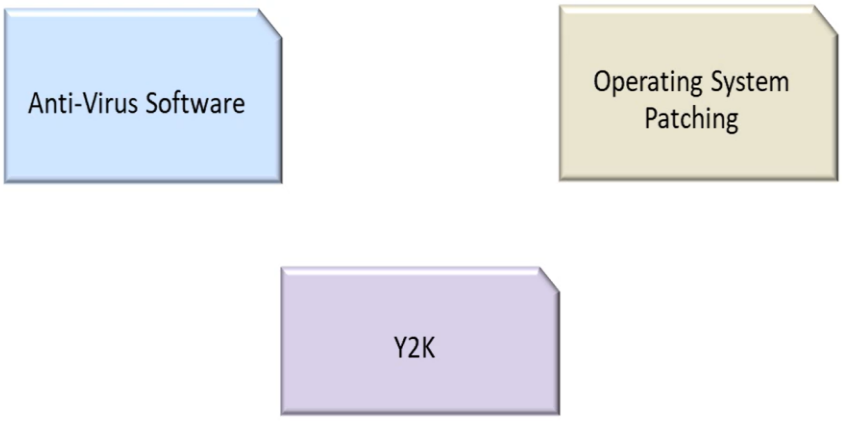
Relative Costs of Maintenance



Development / Maintenance cost



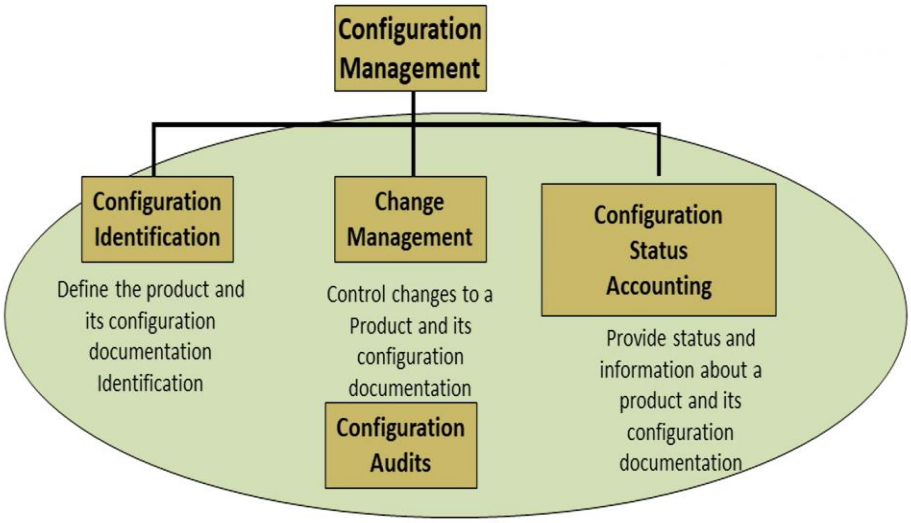
Maintenance Example



Software Configuration Management

What is configuration

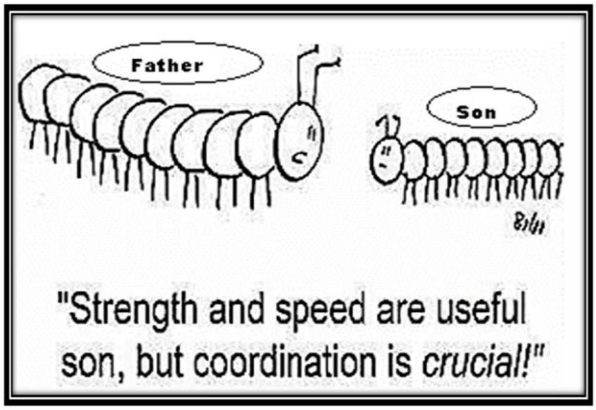
A configuration is the functional and physical characteristics of a hardware or software as set forth in technical documentation or achieved in a product.



Software Configuration Management

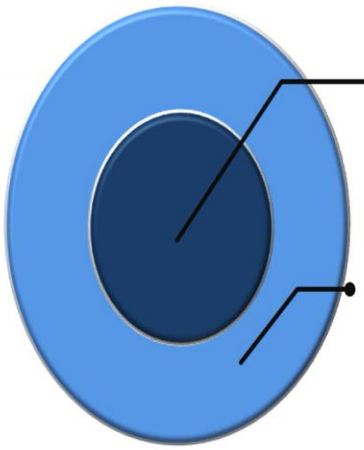
TEKNOTURF

The art of coordinating S/W development activities, minimizing confusion by identifying, modifying and controlling modifications to software



Software Configuration Management

TEKNOTURF



New versions of software systems are created as they change:

- For different machines/OS
- Offering different functionality
- Tailored for particular user requirements

Configuration management is concerned with managing evolving software systems:

Software Configuration Management

SCM defines

- the types of documents to be managed and a document naming scheme
- who takes responsibility for the CM procedures and creation of "baselines"
- policies for change control and version management
- the CM records which must be maintained

Describes the tools which should be used to assist the CM process and any limitations to their use

Effectiveness of Software Configuration Management

Software entities that SCM is expected to manage include :

- Plans
- Specifications (SRS, Design)
- User Documentation
- Test data
- Support Software Tools, Source Code, Executable, and Libraries

SCM is said to be effective:

- when every work product can be accounted for
- when every work product or change made to it can be tracked and controlled

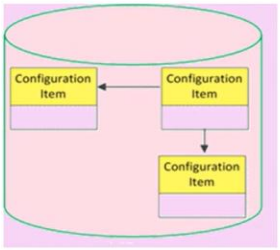
Configuration Item

A Configuration Item is an aggregation of hardware, software, or both, that is designated for configuration management and treated as a single entity in the configuration management process.

Large projects typically produce thousands of documents which must be uniquely identified.

Some of these documents must be maintained for the lifetime of the software.

Document naming scheme should be defined so that related documents have related names.



Types of Configuration Objects

To manage SCIs they must be separately named and organized using object-oriented approach.

There are two types of Objects

- Base Object
- Aggregate

Types of Configuration Objects

TEKNOTURF

Base Object

A base object is the "unit of text" that has been created by a software engineer during analysis, design, code, or test.

Example

- section of a requirement specification
- a source listing for a component
- a suite of test cases that are used to exercise the code.

Types of Configuration Objects

TEKNOTURF

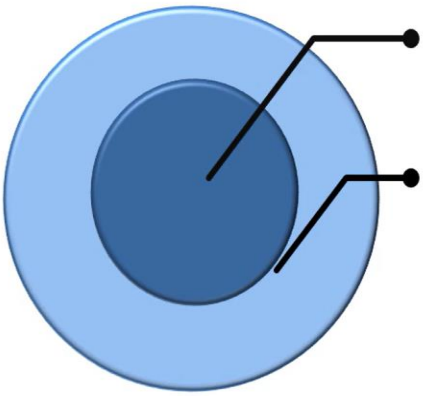
Aggregate Object

- An Aggregate Object is a collection of basic objects and other aggregate objects

Example

- Design Specification is an aggregate object which comprises of Data Design, Architectural Design, Module Design, Interface Design

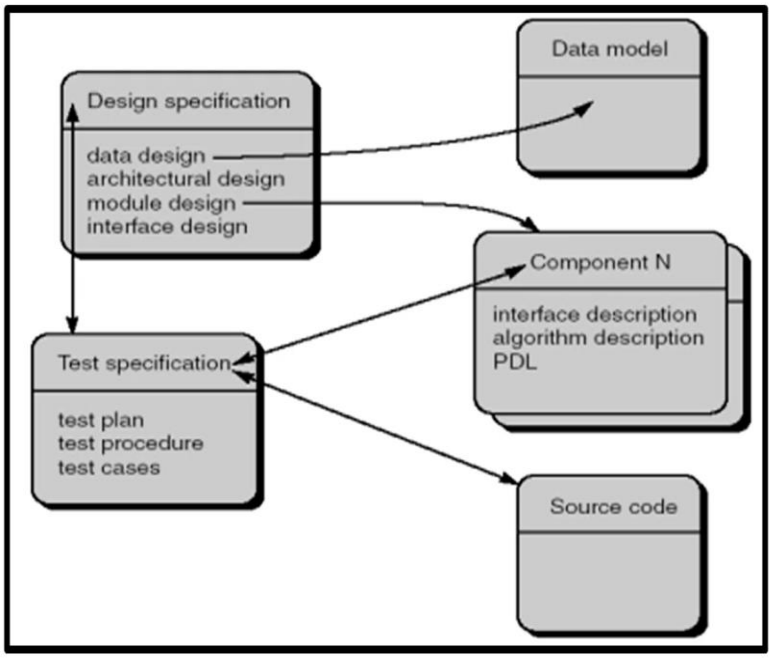
Relationship between Configuration Objects



A curved arrow indicates a compositional relation.

A double-headed straight arrow indicates an interrelationship.

Relationship between Configuration Objects



Baseline and Evolution Graph

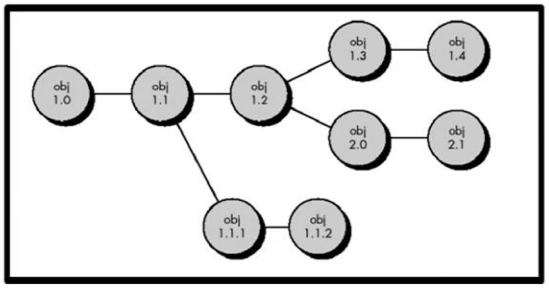
TEKNOTURF

Baseline

"Specification or product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal Change Control Procedures"

The evolution graph

Describes the change history of an object.

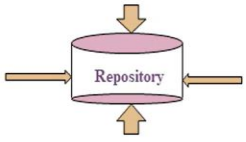


Configuration Repository

TEKNOTURF

All CM information should be maintained in a configuration database

This should allow queries about configurations to be answered



- Who has a particular system version?
- What platform is required for a particular version?
- What versions are affected by a change to component X?
- How many reported faults in version T?

The CM database should preferably be linked to the software being managed

Check -in and Check-out

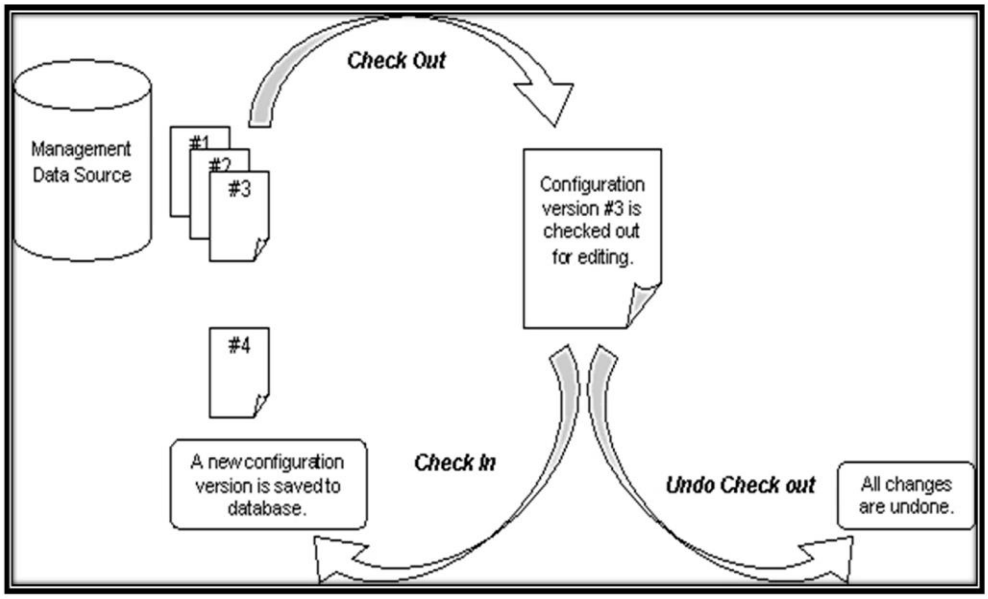
The Configuration items available in the Configuration Management system will be in read-only mode by default

Check in — An operation used to make a developer's object version available to other users.

Check out — A process that creates a new version of an object from an existing version stored in the database.

Developers check out objects so they can work on them.

Check -in and Check-out



Change management

TEKNOTURF

Change management (or change control) is the process during which the changes of a system are implemented in a controlled manner by following a pre-defined framework / model with some reasonable modifications.

Activities in Change Management:

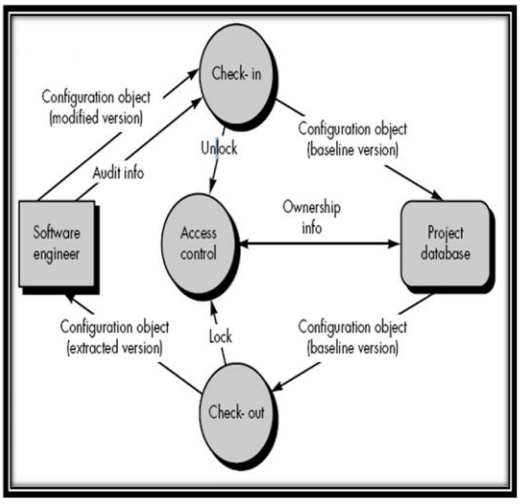
- Filtering changes
- Managing changes and the change process
- Reviewing and closing of Requests for Change (RFCs)
- Management reporting and providing management information

Synchronization Control

TEKNOTURF

Synchronization Control

- helps to ensure that parallel changes, performed by two different people, don't overwrite each other.
- locks the object in the project database so that no updates can be made to it until the currently checked out version has been replaced.



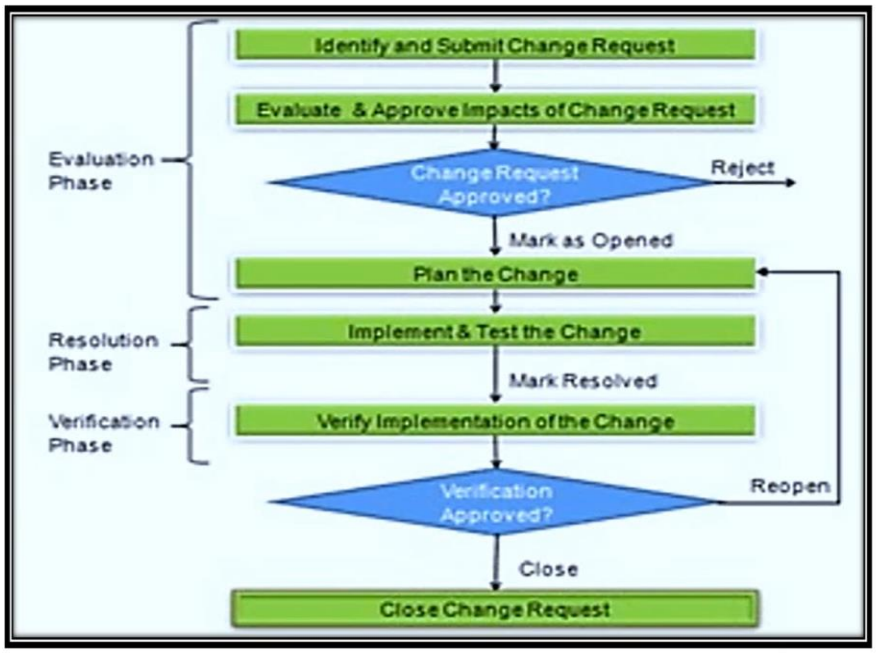
Change Control Board(CCB)

Change Control Board (CCB) or Software Change Control Board (SCCB)

- is a committee that makes decisions regarding whether or not proposed changes to a software project should be implemented.
- is constituted of project stakeholders or their representatives.

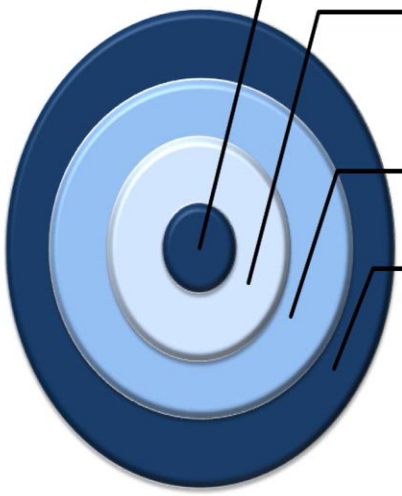
The authority of the change control board may vary from project to project, but decisions reached by the change control board are often accepted as final and binding.

Change Control Process



Version Management

TEKNOTURF



Version control is a mechanism used to manage multiple versions of computer files and programs

It allows users to

- lock files so they can only be edited by one person at a time
- track changes to files

Version Control allows you a place to store your work as it progresses, and allows you instant recall of any work from any point in time.

It also allows others to recall communal (or just your) work at any time, in a read only fashion, or also to make changes themselves.

Benefits of Version Management

TEKNOTURF