TERMS

TEKNOTURF

Let us re-confirm our understanding of a few 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
PROCESS	• A Process is a series of definable, repeatable, and measurable tasks leading to a useful result

SOFTWARE DEVELOPMENT

TEKNOTURF

Ad-hoc Software Development (till 1960s)

Software was developed on a Trial & Error basis

No Specific Process was followed during the development of the Product; No proper testing was done

Defects were detected only after the product was delivered to external Users

SOFTWARE ENGINEERING

TEKNOTURF

PLANNING & ANALYSIS

DESIGN

DEVELOPMENT & TESTING

MAINTENANCE

Software engineering is the application of a systematic, disciplined, quantifiable approach to the design, development, operation and maintenance of software.

https://ieeexplore.ieee.org/document/159342

PROCESS & SOFTWARE DEVELOPMENT PROCESS

TEKNOTURF

Let us define what a Process is

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

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

Benefits:

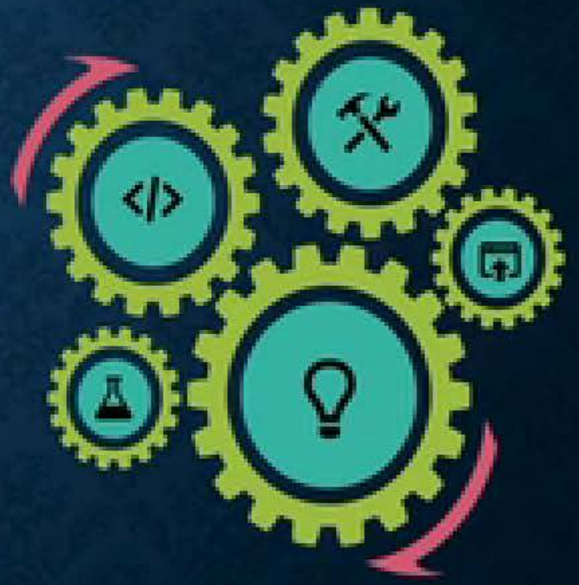
1. Helps to develop software smoothly
2. Allows the development within the given budget, time and scope
3. Mitigate risk

SOFTWARE DEVELOPMENT LIFE CYCLE (SDLC)

TEKNOTURF

What is SDLC?

1. SDLC is the process used in a project to develop a software product.
2. Defines the phases, and tasks to be performed with the objective of developing software that meets requirements
3. Describes how development activities are performed, and their appropriate sequence



SOFTWARE DEVELOPMENT LIFE CYCLE

TEKNOTURF

The different phases of SDLC - planning, analysis, design, implementation , testing, deployment and maintenance

5-DEPLOYMENT & MAINTENANCE

Deployment – Release the product / installing at client's end

Maintenance – Any changes done to software from deployment till retirement of software

4-TESTING

Testing the software product developed to find defects

1-PLANNING & REQUIREMENT ANALYSIS

Planning - project objective and resource estimation

Analysis – study the system requirements

3-DEVELOPMENT (IMPLEMENTATION)

Converting the design into a complete information system

2-DESIGN

How to deliver the actual functionality

PHASES IN SDLC

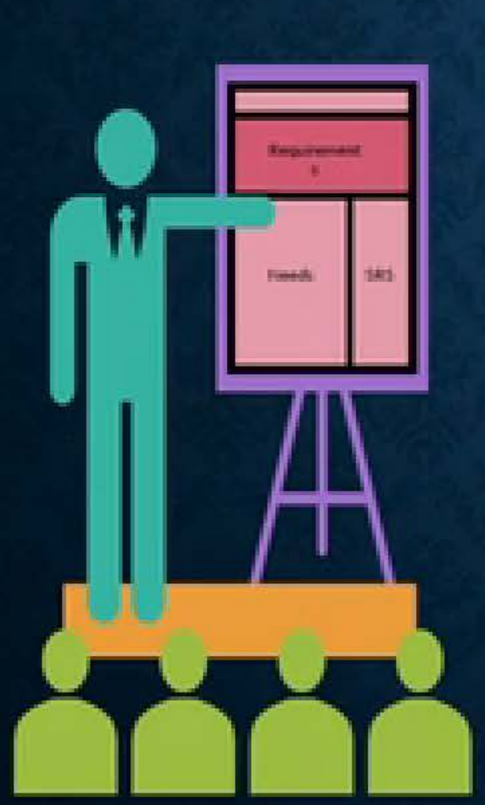
TEKNOTURF

Let us learn the various phases of SDLC

ANALYSIS

TEKNOTURF

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



01

Requirement analysis

- The analyst is responsible for defining the requirements of the system
- The main focus is to identify the problem that the customer is trying to solve.

The activities are

- ✓ **Requirement gathering and analysis** 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

DESIGN

TEKNOTURF

Design converts the software requirement specification document into system specification.

System Design

- 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
- The design documents and the artefacts are produced at the end of the design phase

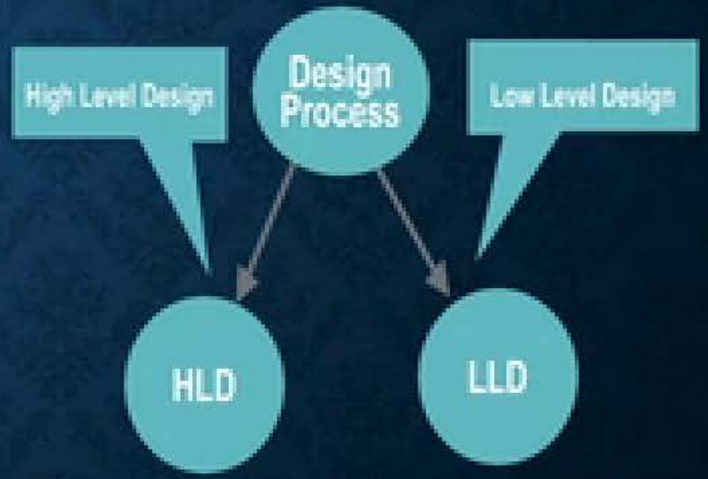
02

LEVELS OF A DESIGN

TEKNOTURF

Design process

- Decomposes the entire project into units / modules and identify the system architecture, data structure and processing logic
- The two levels of design are High Level Design and Low-Level Design
- High Level Design focuses on
  - what modules are required
  - what each module performs, and
  - how these modules communicate with one another
- Low Level Design focuses on
  - writing a detailed algorithm
- DD(Design Document)= HLD + LLD



CONSTRUCTION (CODE + UNIT TESTING)

TEKNOTURF

Modular and subsystem programming code is accomplished during this stage

Development

- The goal of the coding phase is to convert the design into a workable solution using any programming language
- Unit testing /module testing is done in this stage by the developers
- This stage produces the source code, executable, and databases applicable

03

TESTING

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

TEKNOTURF

Software testing

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

04

Levels of Testing

1. Unit Testing
2. Integration Testing
3. System Testing
4. Acceptance Testing

LEVELS OF TESTING

TEKNOTURF

The levels of testing explained

UNIT TESTING

Done by the developer - individual module is for functional correctness

INTEGRATION TESTING

Checks for the interface errors between the integrated components

SYSTEM TESTING

The system is tested as a whole to check the functional and nonfunctional correctness

ACCEPTANCE TESTING

Done by the end user for the system acceptance

09:18

TESTING (V & V)

TEKNOTURF

Software Testing includes Verification , Validation & Defects finding

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

MAINTENANCE

TEKNOTURF

Changes or enhancements happen everywhere. Software is no exemption.

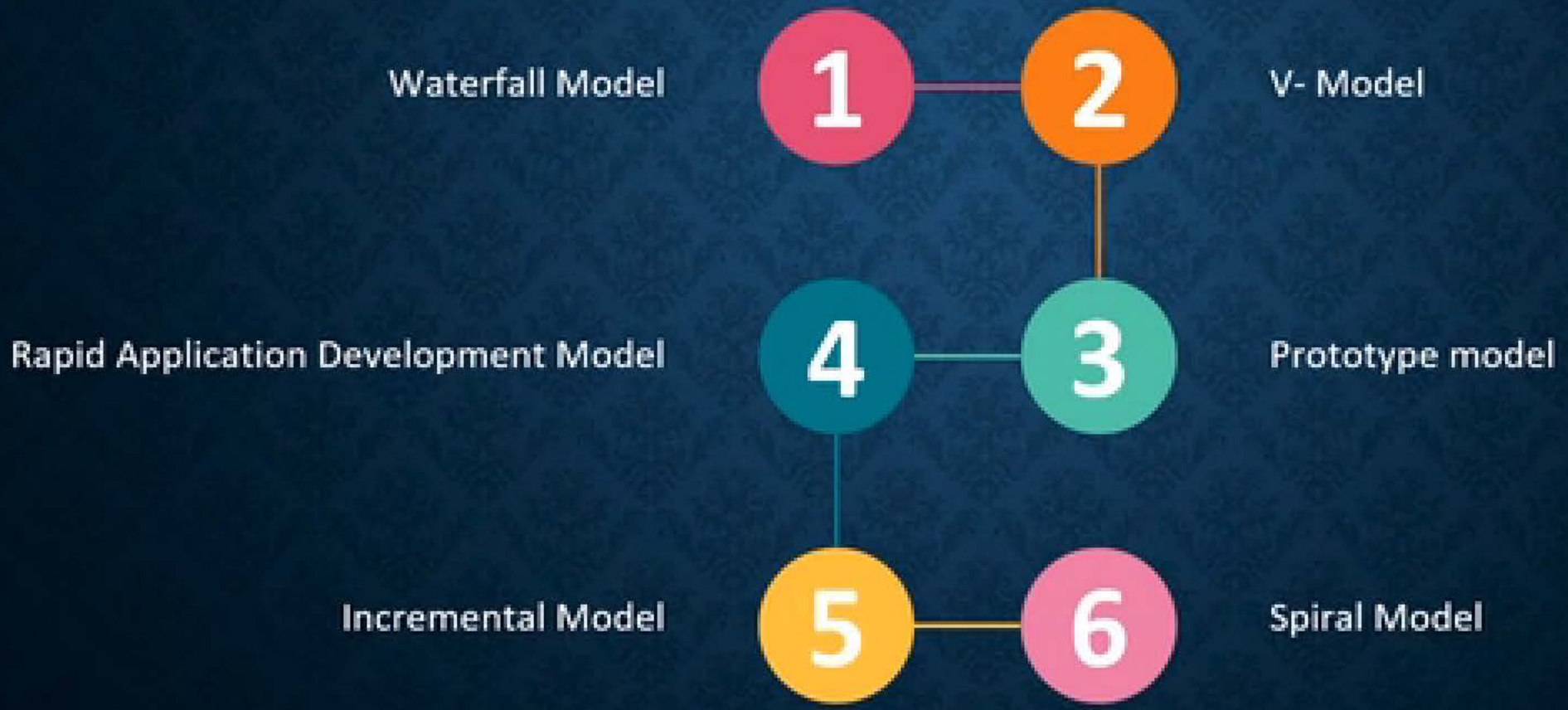
SOFTWARE MAINTENANCE

05

Any change that is made to the software after it is deployed is known as maintenance.

SOFTWARE DEVELOPMENT LIFE CYCLE MODELS

TEKNOTURF



WATERFALL MODEL

TEKNOTURF

Also known as the linear sequential model

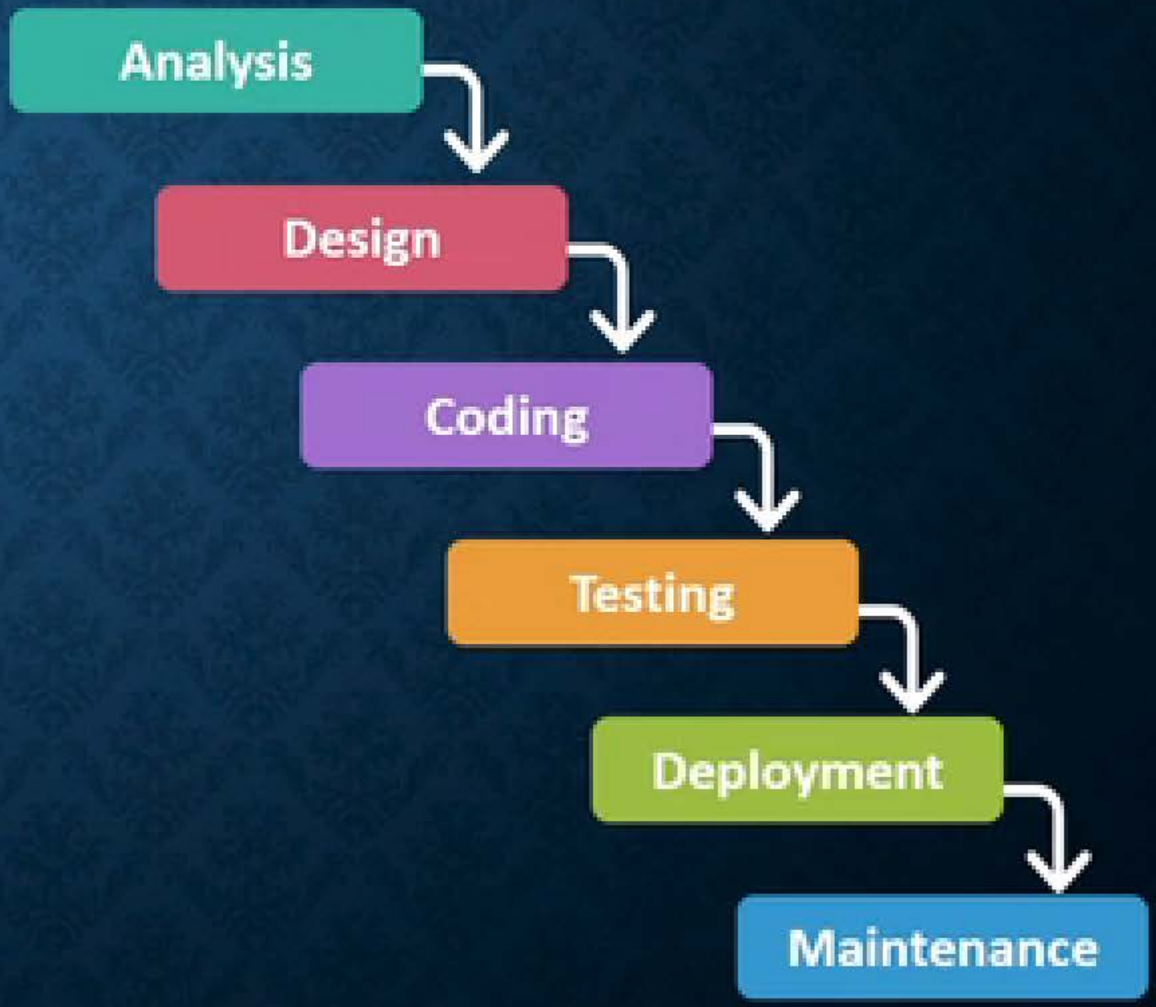
WATERFALL MODEL

TEKNOTURF

Key Features:

1. Each phase has a well defined end identifiable deliverables to the next phase
2. It is a linear sequential model and is systematic in approach
3. The main consequence of this model is - "Can't iterate to previous phase"

i.e. this model follows a "no going back" style between phases



ADVANTAGES OF WATERFALL MODEL

TEKNOTURF

1

Simple and easy to use

The amount of resources required for it is minimal

2

Easy to manage

As each phase has specific deliverables and therefore provides high visibility

3

Equal importance to all the phases

Phases are processed and completed one at a time

4

Well suited for smaller projects

Is well suited for smaller projects where requirements are very well understood.

5

Follows linear approach

Each development phase proceeds in strict order

LIMITATIONS OF WATERFALL MODEL

TEKNOTURF

Is suitable if the requirements are well-defined and stable

Not desirable for large and complex projects where the requirements are expected to change frequently

Systems must be defined up front ; Rigidity

- Does not allow to modify the requirements once the development process is triggered.
- To start a phase, all the deliverables related to the previous phase should be 100% complete.
- Suppose a project is in coding phase and a requirement needs to be changed. Then the entire process needs to be done from scratch to incorporate the changes

Client gets a feel of the system only when the final product is delivered

- Client will get the preview of the application only after the final version of the software is developed. There is no feed back loop.
- If client finds any mismatch in requirements, then it is a hectic process to incorporate the changes

Delayed Testing

Testing phase starts only after the entire implementation is complete. Until then, Testers are not involved in the process leading to the blocked state.

Hard to estimate costs & schedule Cost of change or correction is very high

No way to perform testing prior to testing phase. Also client interacts with the system only when delivered. If any changes are to be incorporated, there will be an overrun in cost and schedule because these changes should go through the entire process again..

WATERFALL MODEL

TEKNOTURF

Waterfall model is suitable when

Software requirements are clearly defined and known	
Product definition is stable	To write an application to play chess, all the rules are well defined and are stable. Hence waterfall model can be used.
New version of the existing software system is created	An application already exists in Dotnet Technology. Client wants the same application to be converted to Java technology
Software development technologies and tools are well known	

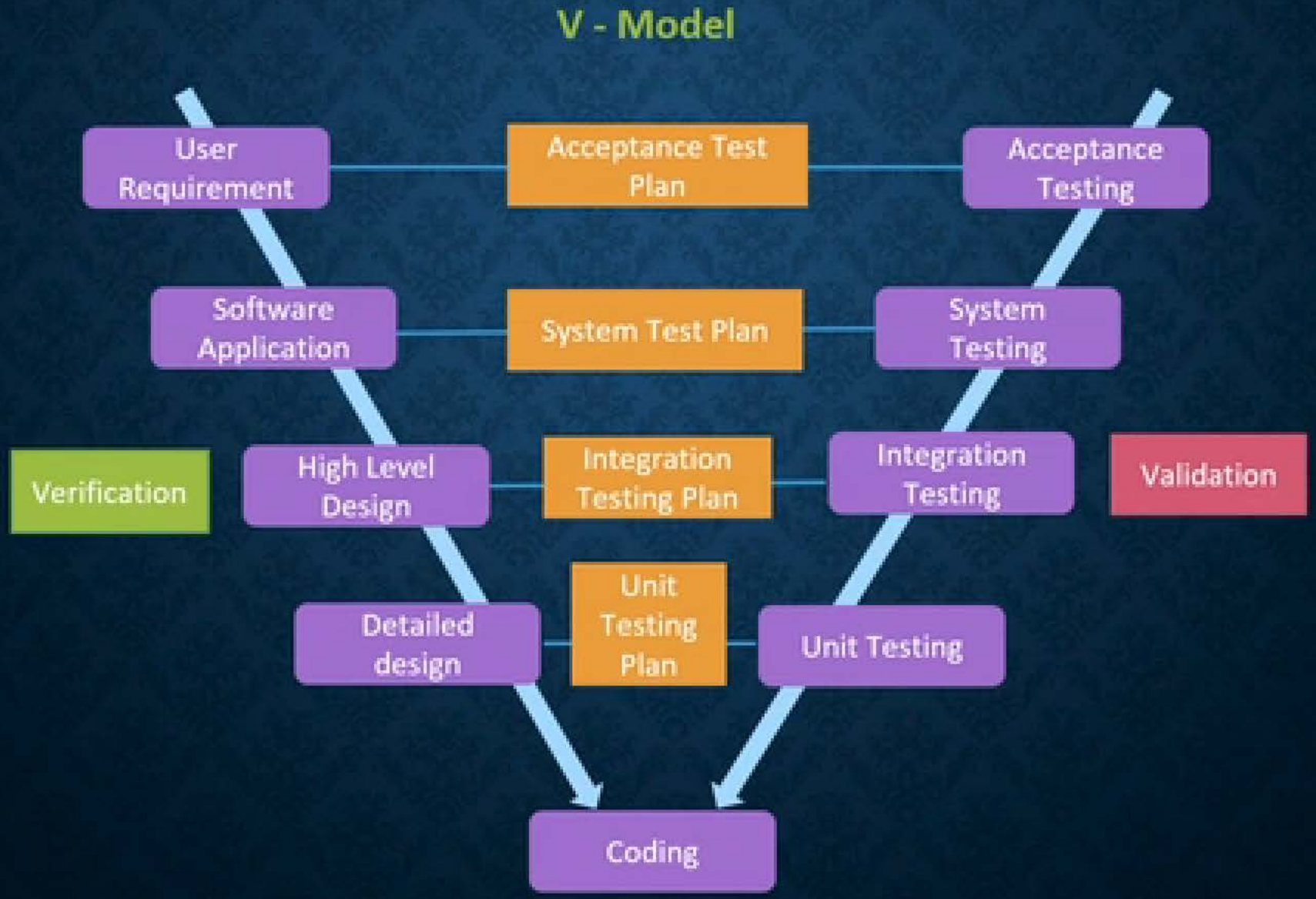
V-MODEL

TEKNOTURF

Verification and Validation Model

1. Verification and Validation Model commonly known as V-Model evolved from waterfall Model
2. It is a structured approach to testing
3. Testing is emphasized in this model more than in the waterfall model
4. Testing is done from the earliest stages. This ensures that Quality of software becomes a focus-area right from the start.
5. Each phase must be completed before the next phase begins

V-MODEL


V - Model
User Requirement
Acceptance Test Plan
Acceptance Testing
Software Application
System Test Plan
System Testing
Verification
High Level Design
Integration Testing Plan
Integration Testing
Validation
Detailed design
Unit Testing Plan
Unit Testing
Coding

ADVANTAGES OF V-MODEL

TEKNOTURF

1. Validation and Verification at each level of the stage containment
2. Stage containment mechanism
3. Avoids the downward flow of defect
4. Lower defect resolution cost due to early detection
5. Allows testers to be active from the initial state of the project life cycle

LIMITATIONS OF V-MODEL

TEKNOTURF

1. Least flexible
2. Rigid
3. Backtracking cost is high in case a problem occurs
4. Hard to estimate costs & project overruns
5. No Prototype guidance

PROTOTYPE MODEL

TEKNOTURF

Prototyping is the concept of developing a model for the system to be built

Prototyping is the concept of developing a model for the system to be built.

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

ACTUAL

PROTOTYPE MODEL

TEKNOTURF

The process of prototyping involves the following steps

1. Identify basic requirements
2. Develop Initial Prototype
3. Review
4. Revise and Enhance the Prototype

PROTOTYPE MODEL - WHY

TEKNOTURF

One of the major reasons why prototyping is done is to help software architects / designers make choices.

Another reason is to help communicate ideas within the Team or outside – to make clear what one is talking about, to understand what someone else's ideas, and to pitch for support

Another reason is to test if something works.

Prototyping can also be used by the end users to describe and prove requirements that the developers ha

PROTOTYPE MODEL

TEKNOTURF

There are two types of prototyping - Throwaway, and Evolutionary

THROW AWAY

- Used when the requirement is unclear.
- A small part of the system is developed and then given to the end user for review.
- The prototype is then discarded or thrown away.

EVOLUTIONARY

- Used when the requirements are unstable.
- An initial prototype is presented to the user for feedback and suggestions for improvements, and refined by the developer and the process is repeated.
- In Evolutionary prototype the prototype is converted into the final system.

THROWAWAY PROTOTYPE MODEL

TEKNOTURF

A 'quick and dirty' approach involving - Quick requirements assessment, analysis, design

- Focuses on rapid construction and has an Ad-hoc development approach; Discards prototype after the objective is met

The detailed steps for the throw away prototype are

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

WHY prototype is THROWN AWAY

- When working with this prototype its not necessary that the developer should strictly follow the documentation
- Also, not necessary to adopt the same technology. Developer may use some easy to use tools or technology for his ease.
- Hence this prototype will be thrown away and the actual product will be developed with proper documentation and in the technology required.

EVOLUTIONARY PROTOTYPE MODEL

TEKNOTURF

An incremental and iterative model

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

ADVANTAGES OF PROTOTYPE MODEL

TEKNOTURF

WHEN TO USE PROTOTYPE MODEL

TEKNOTURF

SCENARIO 1:

- Peter needs to develop an application for Azure online shopping.
  Client requires color changes depending on the product displayed.
- Peter is not clear with the client requirement.

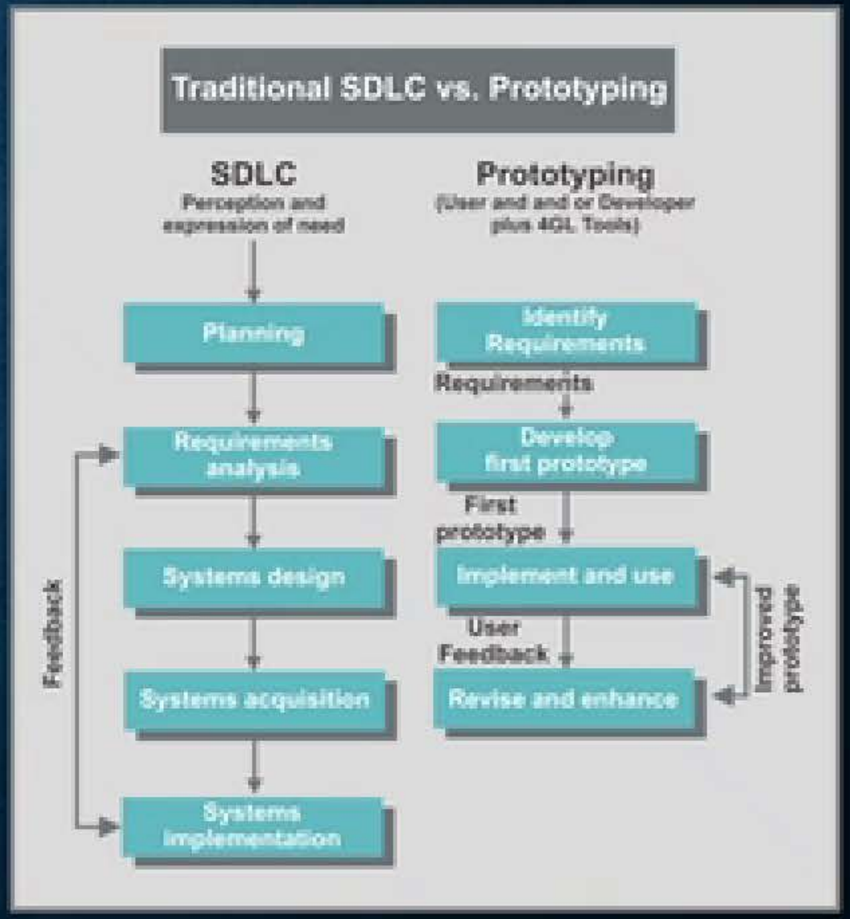
What type of prototype should be adopted for this scenario?

SCENARIO 2:

Stakeholders explain Peter the process for creating the Balance sheet.

Their explanation is not stable and has multiple variations.

What type of prototype is best suited for this scenario?



RAPID APPLICATION DEVELOPMENT (RAD)

TEKNOTURF

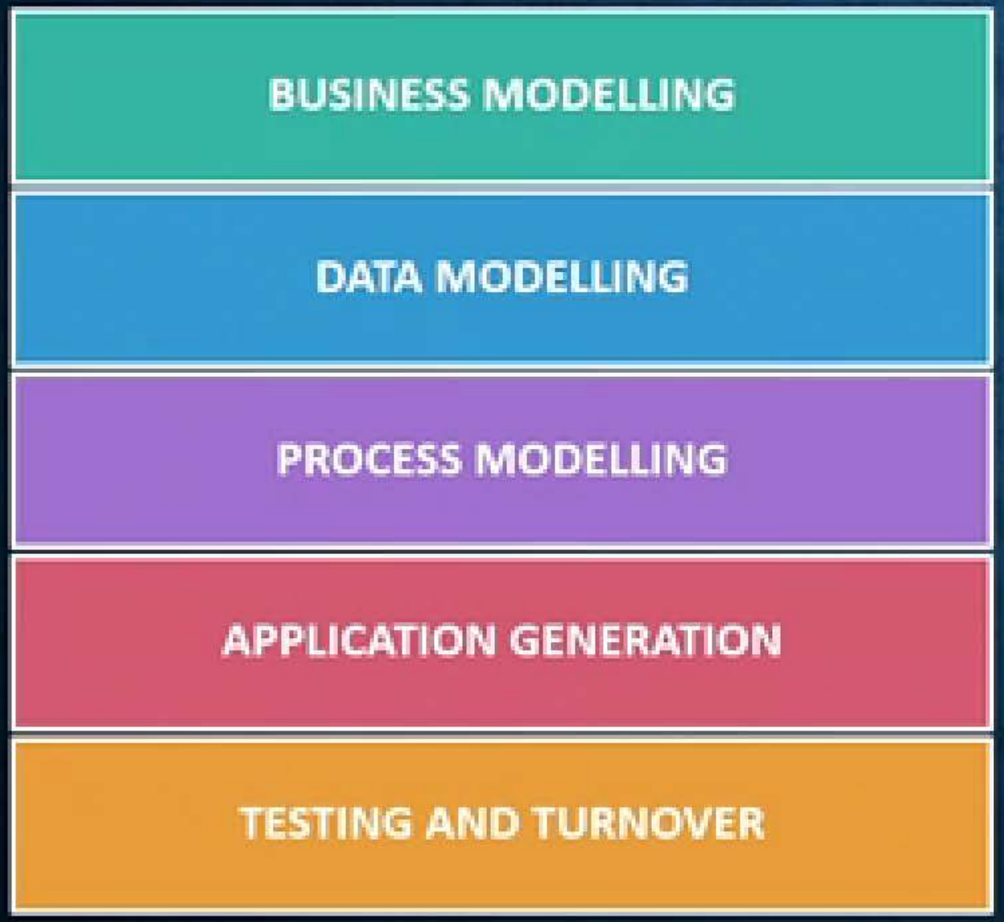
RAD model is an incremental model

1. In RAD model the modules are developed in parallel as if they were mini projects
2. RAD is a high speed, version of the linear sequential model
3. Characterized by a very short development life cycle; Customer can provide feedback regarding the requirements
4. The RAD model follows a component based, approach
5. Individual components are developed by different people and assembled to develop a large software system

RAPID APPLICATION DEVELOPMENT (RAD) PHASES

TEKNOTURF

The phases of RAD model are as follows



RAD model is used

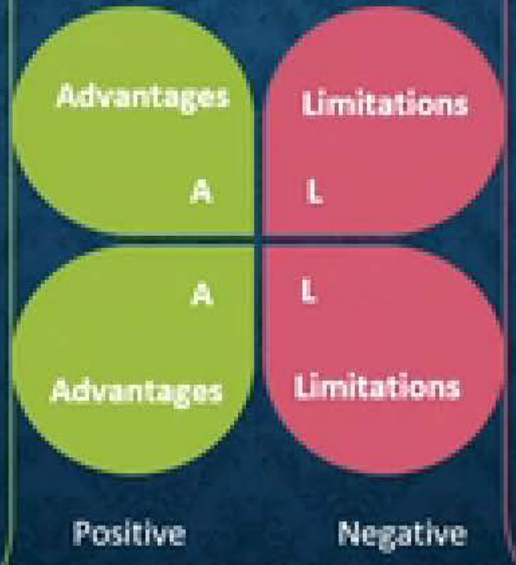
1. When requirements are not fully understood
2. When User is involved throughout the life cycle
3. When System can be modularized

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

INCREMENTAL MODEL

TEKNOTURF

This model uses incremental cycles

- The incremental model prioritizes requirements of the system and implements them in small manageable units called module. Each module undergoes analysis, design, coding and testing.
- Each subsequent release of the system adds function to the previous release, until all designed functionalities have been implemented

INCREMENTAL MODEL IS USED

- When most of the requirements are known up-front but are expected to evolve over time
- When projects have lengthy development schedules

INCREMENTAL MODEL – ADVANTAGES AND LIMITATIONS

TEKNOTURF

INCREMENTAL MODEL – ADVANTAGES AND LIMITATIONS

ADVANTAGES

- Uses "divide and conquer" breakdown of tasks
- High-risk or major functions are addressed in the first increment cycles; Each release delivers an operational product
- Customer can respond to each build and
- Customers get important functionality early

LIMITATIONS

- Requires early definition of a complete and fully functional system to allow for increments
- Requires good planning and design
- Absence of well-defined module interfaces are major obstacles

SPIRAL MODEL

TEKNOTURF

Proposed by Barry Boehm in 1986; Suggested for High-Risk Scenarios based projects

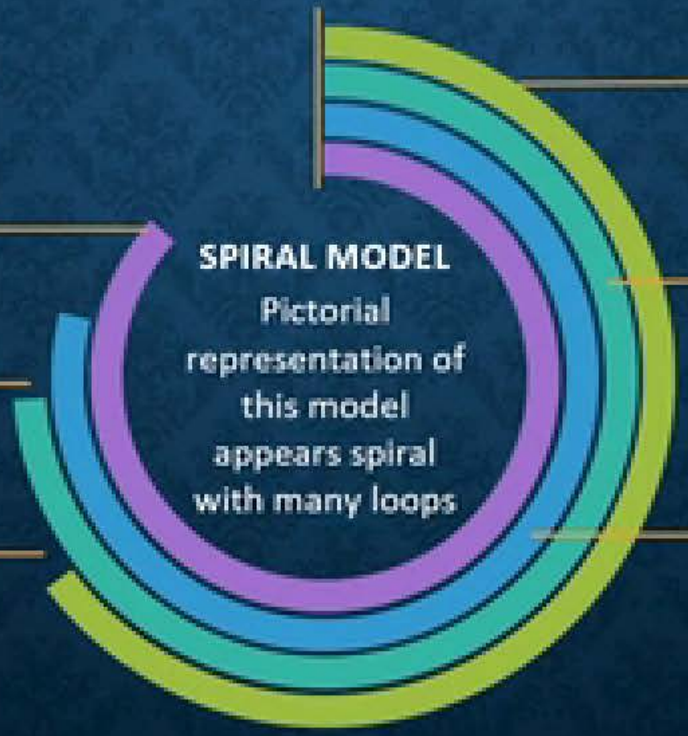
The count for number of loops of the spiral is unknown.
It varies from project to project.

Each loop of the spiral represents a Phase of the software development process.

Risk-driven software development process model.

Has four phases – Planning, Risk Analysis, Engineering and Evaluation; It is favourable for large, expensive, and complicated models

Combination of waterfall and iterative model.



Accommodates prototyping. This model combines the features of the prototyping model and the waterfall model

SPIRAL MODEL IS USED WHEN

- Risk perceived is very high
- Requirements are complex
- Significant changes are expected

ADVANTAGES AND LIMITATIONS OF SPIRAL MODEL

TEKNOTURF

ADVANTAGES

- Provides early indication of risk as here high risk analysis is done at all phases.
- Users see the system early because of rapid prototyping tools.
- Critical high-risk functions are developed first.
- Early and frequent feedback from users.

LIMITATIONS

- 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.

REQUIREMENTS ENGINEERING

TEKNOTURF

Helps in a clear understanding of the requirements / needs of a Software project

Requirement:a capability that the system must possess to meet a need, solve a problem, or offer a service

Types of Requirement:

Functional Requirement: Specifies the system behavior that includes the features and various functions of the system. Fo example, if a pen is considered its functionality is it should write.

Non-Functional Requirement: Specifies the overall attributes and constraints imposed on the system.

DEFINE

Requirement Engineering is the process of defining, documenting and maintaining the requirements

INITIAL PHASE

One of the initial phases in the software development is Requirements Analysis

REQUIREMENTS ENGINEERING

Helps in a clear understanding of the requirements / needs of a Software project

DEFINE

Requirement Engineering is the process of defining, documenting and maintaining the requirements

IMPORTANCE

For a software project to be successful this phase is very critical.

OUTPUT

The Software Requirements Specification (SRS) document is the outcome of the Requirements Analysis phase

INITIAL PHASE

One of the initial phases in the software development is Requirements Analysis

GOALS

To understand, analyze and make the requirements very clear

STEPS INVOLVED

The steps in Requirements Engineering are

- Requirements Elicitation,
- Requirements Analysis and Requirements

REQUIREMENTS ELICITATION

Gathers requirements from the users

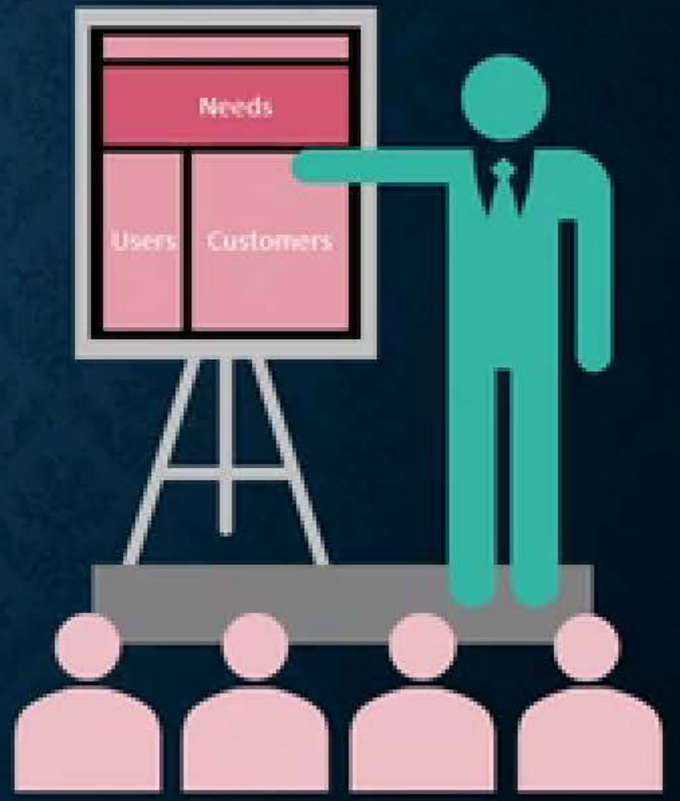
TEKNOTURF

PROCESS

Requirements elicitation is the process of gathering requirements from the users, customers and other stake holders

Elicitation helps to

- To make clear what the problem(s) are that the system aims to solve
- Make clear what we will build
- Make clear what stakeholder expectations are and so on.



Requirements Elicitation is used to

- ✓ Address problems of scope - The boundary of the system is ill defined
- ✓ Address problems of understanding - Users himself is not clear about his needs
- ✓ Address problems of volatility (changing requirements

REQUIREMENTS ELICITATION TECHNIQUES

TEKNOTURF

Elicitation aims to pull out the ideas about the project, from the SME's and stake holders

TECHNIQUES

Document Analysis
Review of the existing documents

Interviews
Interviewer asks the relevant question to the stake holders to get the information

Workshops
Workshops are one of the most resource efficient technique in requirement elici

Brainstorming
The SME's and the stake holders involved brainstorm to get new ideas

Focus groups
Providing feedback by discussing the topic to refine the ideas obtained as a result of the earlier elicitation

Surveys
SME's and stakeholders conduct survey/questionnaire for the requirement elicitation

REQUIREMENTS ANALYSIS (RA)

TEKNOTURF

Analyze the requirements

PROCESS

1. Requirements analysis is the process of establishing the services the system should provide and the constraints under which it must operate

2. The process of studying and analyzing the customer and the user/stakeholder to arrive at a definition of software requirements

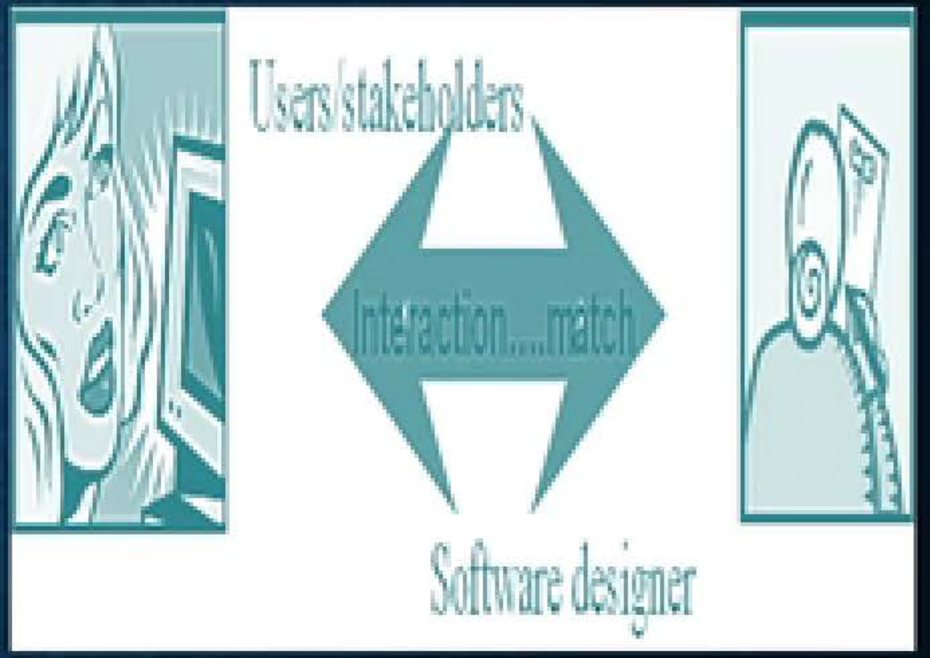
3. Requirement analysis helps in obtaining requirements that are
   - Clear
   - Complete
   - Consistent
   - Unambiguous

REQUIREMENTS ANALYSIS (RA)

TEKNOTURF

Why is Requirement Analysis difficult?

1. **Learn 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
2. **Users/stakeholders are not a uniform group**
   - ✓ Conflict between cost and usability / performance / features
   - ✓ Conflicting demands from different departments
3. **Other factors**
   - ✓ It is sometimes not easy to visualize the solution or system as it will ultimately be. There may also be differences between how stakeholders visualize the system
   - ✓ Scope of the system needs well defined boundaries
   - ✓ Current vs. future system - resistance to change
   - ✓ Process of negotiation between users and designers



Getting the good (ideal) system

Vs

possibility of building it well

REQUIREMENT ANALYSIS (RA)

TEKNOTURF

Elicitation is followed by the Requirement Analysis, where the requirements are scrutinized and analysed for unambiguous , consistent and clear requirements.

RA helps to

- Understand the requirements
- Remove inconsistencies, anomalies, etc. from requirements
- Document requirements properly in an SRS document

Try out!!!

An elicitation was performed, and the below requirements are obtained from the customer. Can you perform an analysis and identify whether the requirements are clear or does it have any ambiguity or inconsistency in it.

- The UI Screens should be highly user-friendly
- System should send a confirmation email after the employee details are successfully added into the database.
- A requirement is that the response time should be not more than 5 seconds. Another requirement indicates that the response time may be delayed.


Ambiguous

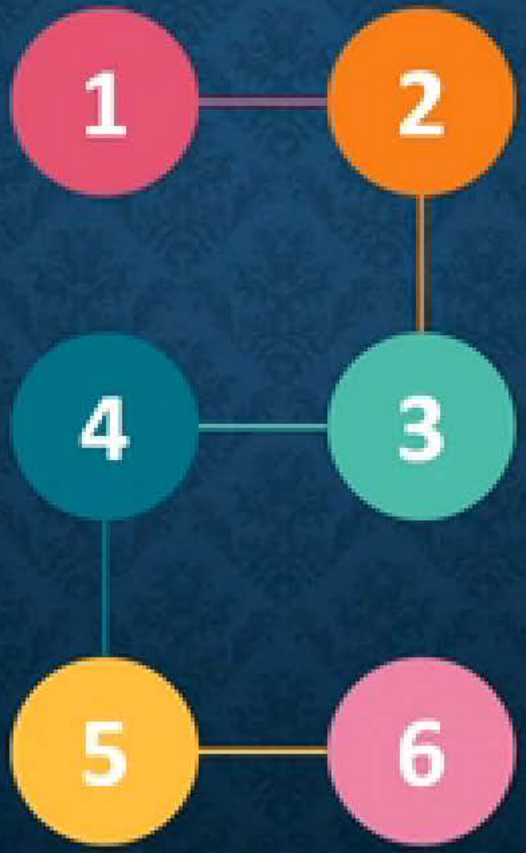

Correct Requirement


Inconsistent Requirement

REQUIREMENT ANALYSIS – GATHERING REQUIREMENTS

TEKNOTURF

Observation of existing systems



Studying existing procedures

Having questionnaires to understand user requirements

Discussion with the customer and the end-users

Conducting discussion with domain experts to understand the system

Finally, through these steps, the Analyst can easily obtain

- ✓ Input and output formats
- ✓ Accurate details of the operational procedures

RA–ANALYZING THE GATHERED REQUIREMENTS

TEKNOTURF

What is to be built in the software and not how it is to be built

1

Analyzing the gathered requirement

- ✓ Understands the user requirements
- ✓ Detects and removes inconsistencies ambiguities, and incompleteness

2

Incompleteness and inconsistencies – Resolved

- ✓ Resolved through further discussions with the end- users and the customers

3

Analyst should understand the following

- ✓ What is the problem?
- ✓ Why is it important to solve the problem?
- ✓ What are the possible solutions to the problem?
- ✓ What complexities might arise while solving the problem?

RA – CONTRADICTING AND INCOMPLETE REQUIREMENTS

TEKNOTURF

Contradicting requirements

- One requirement conflicts with another requirement
- Example: The Hiring Team asks that the system capture the marital status of potential hires. The HR Legal Team states that personal information such as marital status must not be recorded by the system

Incomplete requirements

- Some requirements have been omitted:
- Due to oversight
- A simple 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 that when temperature falls below 90 degrees.
    - ✓ heater should be turned ON
    - ✓ water shower turned OFF

RA –PRIORITIZATION OF REQUIREMENTS

TEKNOTURF

1. Analyst must prioritize the requirements so that it allows all stake holders to focus on the most important functionalities first .
2. Prioritization also gives an insight that allows all the stakeholders involved to understand and agree on the important features of the product
3. MoSCoW is one of the requirement prioritization technique that helps to classify the need based on the four possible priority classification
4. Must: Mandatory requirement which must be there in the product to be considered as a success
5. Should: Not mandatory to be there, but, if possible, it could be incorporated in the solution but even without that still it is considered as success
6. Could: Desired capability, can be considered with respect to the available time and resources
7. Wont: Not required at all at this point of time, can be a thought for the future upgrades

SOFTWARE REQUIREMENTS SPECIFICATION

TEKNOTURF

1

Main aim of requirements specification

- Systematically organizes the requirements gathered during requirements analysis
- Documents all the requirements properly

2

Software Requirements Specification

- Specifies what the software is supposed to do without specifying how it is implemented
- The implementation details will not be present in the SRS
- Outcome of the analysis phase
- Contains all the information about the requirements gathered

SOFTWARE REQUIREMENT SPECIFICATION (SRS) – IMPORTANCE

TEKNOTURF

SRS DOCUMENT

- SRS is a reference document
- It is a contract between the development team and the customer
- SRS document concentrates on - What needs to be done
- Carefully avoids the solution (“how to do”) aspects

THE SRS DOCUMENT IS USEFUL IN MANY CONTEXTS

- Statement of user needs
- Contract document sentiments.
- Reference document
- Definition for implementation

PROPERTIES OF A GOOD SRS

02:34

SRS DOCUMENT

TEKNOTURF

SRS document normally contains three important parts

1

Functional requirements

Specifies the input, the task and the output

2

Non-Functional requirements

Specifies the overall quality attributes and constraints

3

Constraints on the system

Specifies the restrictions

FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

TEKNOTURF

FUNCTIONAL REQUIREMENT

- A functional Requirement describes a service that the software system must offer
- Example : For a service provider application, few functional requirements are
  - Add a New Plan
  - Register Customer
  - Open a new Connection
  - Generate Bill for Postpaid connection
  - Make Payment
  - Change Plan for a connection
  - Deactivate Connection

NON - FUNCTIONAL REQUIREMENT

- Non-functional Requirements describe qualitative attributes of a software system.
- Few non-functional requirements are
  - Maintainability
  - Portability
  - Usability
  - Reliability
  - Robustness
  - Security
  - Performance

FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

TEKNOTURF

EXAMPLES FOR NON-FUNCTIONAL REQUIREMENT

Portability - Tom was having Windows 10 on his laptop. Now he changed to MacOS laptop. He installs the game that he had on the old laptop. If that game works as efficiently as it worked on the old device, then that application is portable.

Notification should be sent to the user within 5 second from the transaction time and this should happen till 5000 users use the application simultaneously

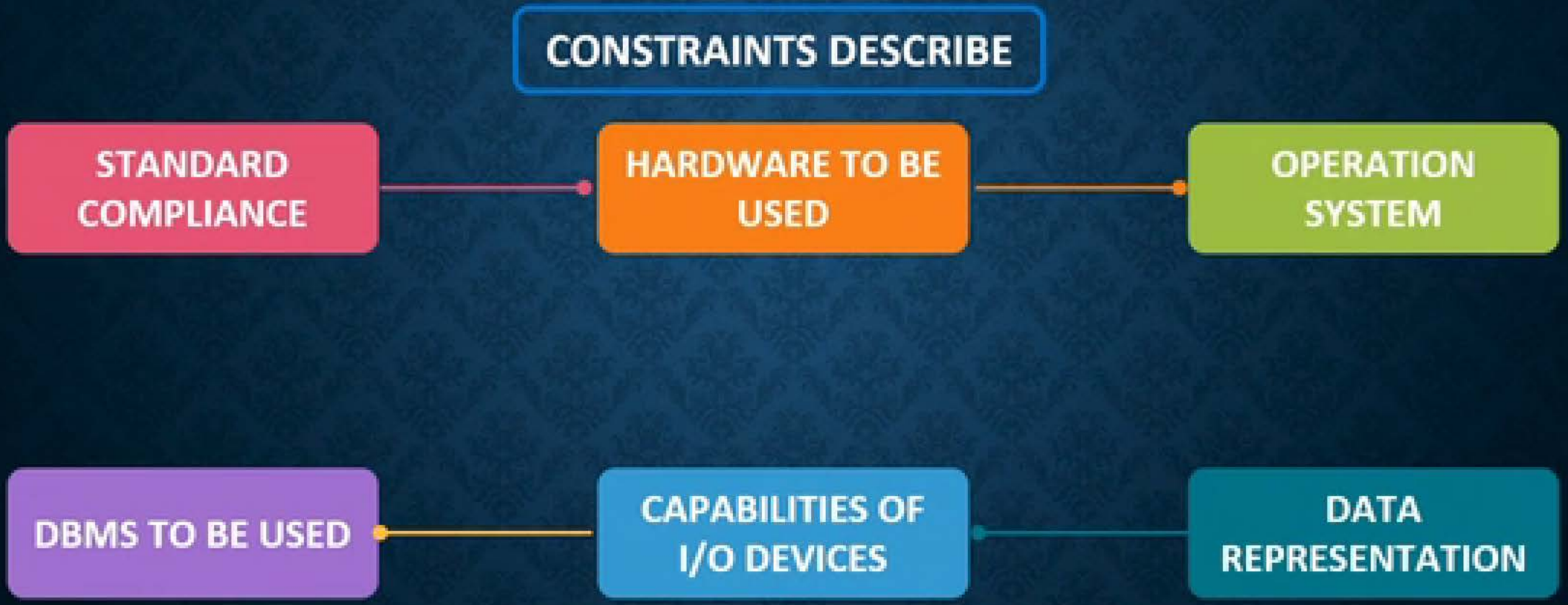
Security - 1. User needs to be prompted to provide a digital signature when a new page is loaded.
2. For a Banking website, back button should not work.

Maintainability - Due to change in the government norms changes in Income tax calculation is to be done. These changes should be easily incorporated in the code with minimal changes.

Usability - 1. In an online shopping website, when a user selects a product, only that product should be displayed on the screen and to view few other related products there should be a hyperlink "Click here for more Products"
2. To login in a website, entered a username and password. An error message is displayed as Username or password is incorrect. It is not specific whether the username or the password is incorrect.

Robustness - When a product is added by providing wrong data like product name with special characters, price with a value exceeding the range, discount with a negative value it should prompt with appropriate error messages.

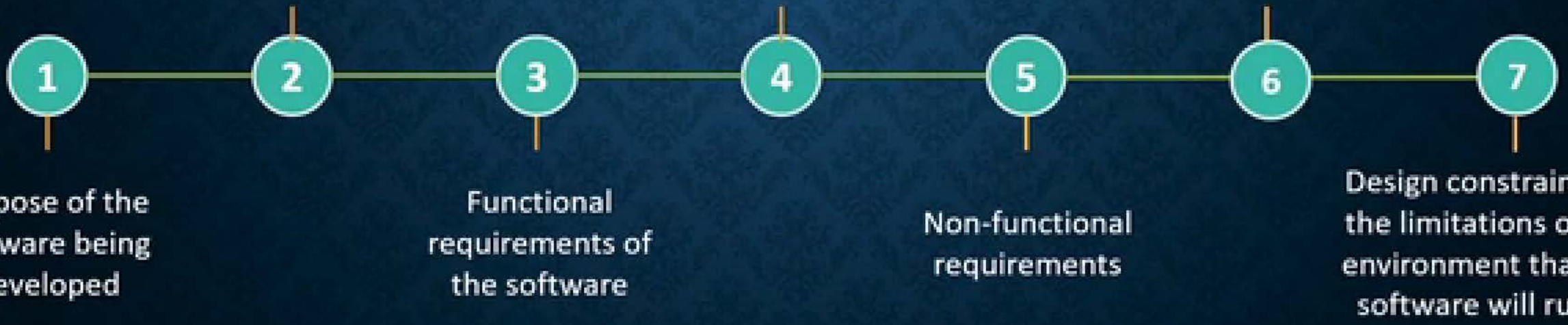
SRS – CONSTRAINTS



ELEMENTS OF SRS DOCUMENT

TEKNOTURF

ELEMENTS OF SRS



ENTITY RELATIONSHIP DIAGRAM (ERD)

TEKNOTURF

ER diagram is widely used in database design

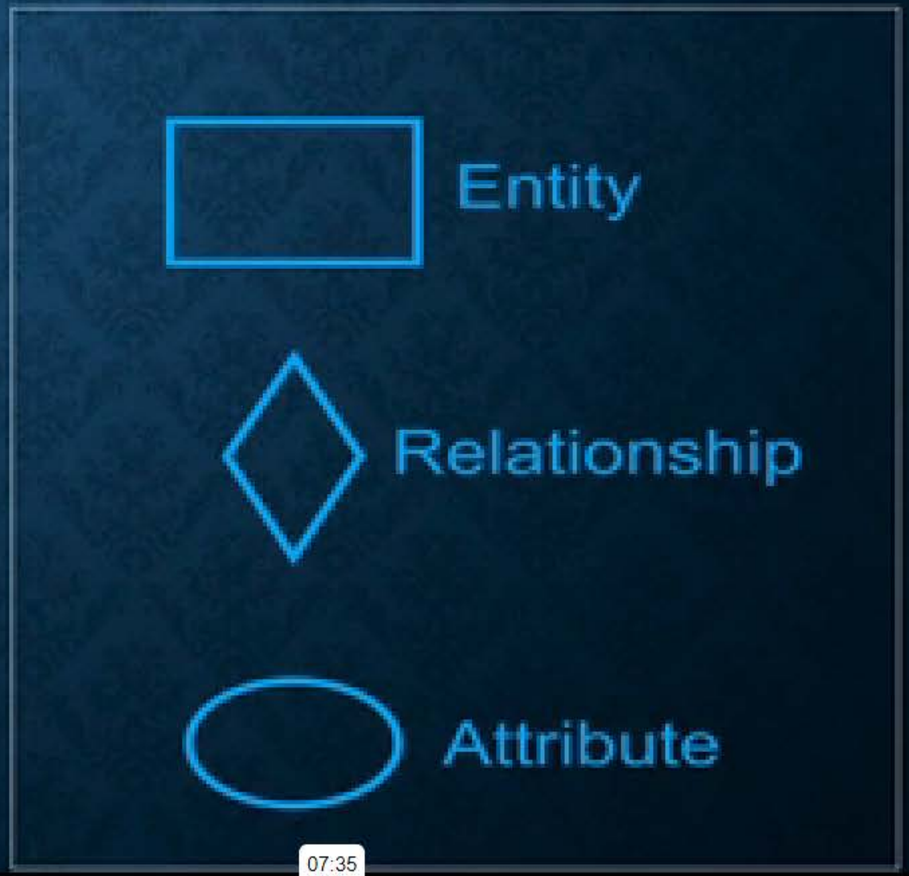
ERD is part of the data modeling that focus on defining and analyzing the data needed to support business.

01 **IMPORTANCE**

Represents conceptual level of a database system

02 **DESCRIBE**

Describes things and their relationships in high level



ERD

TEKNOTURF

WHAT IS AN ENTITY?

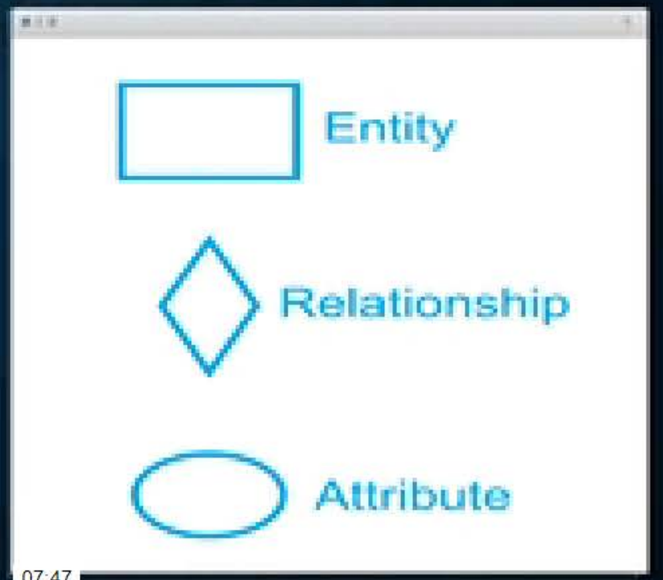
An entity is a business object that represents a group, or a category of data

ATTRIBUTES

Properties of an entity

RELATIONSHIP

Specifies the relations among entities



07:47

ERD

TEKNOTURF

CARDINALITY

One instance of an entity maps to how many instance of other entity

This relationship can be

- One-to-One
- One-to-Many
- Many-to-Many

OPTIONALITY

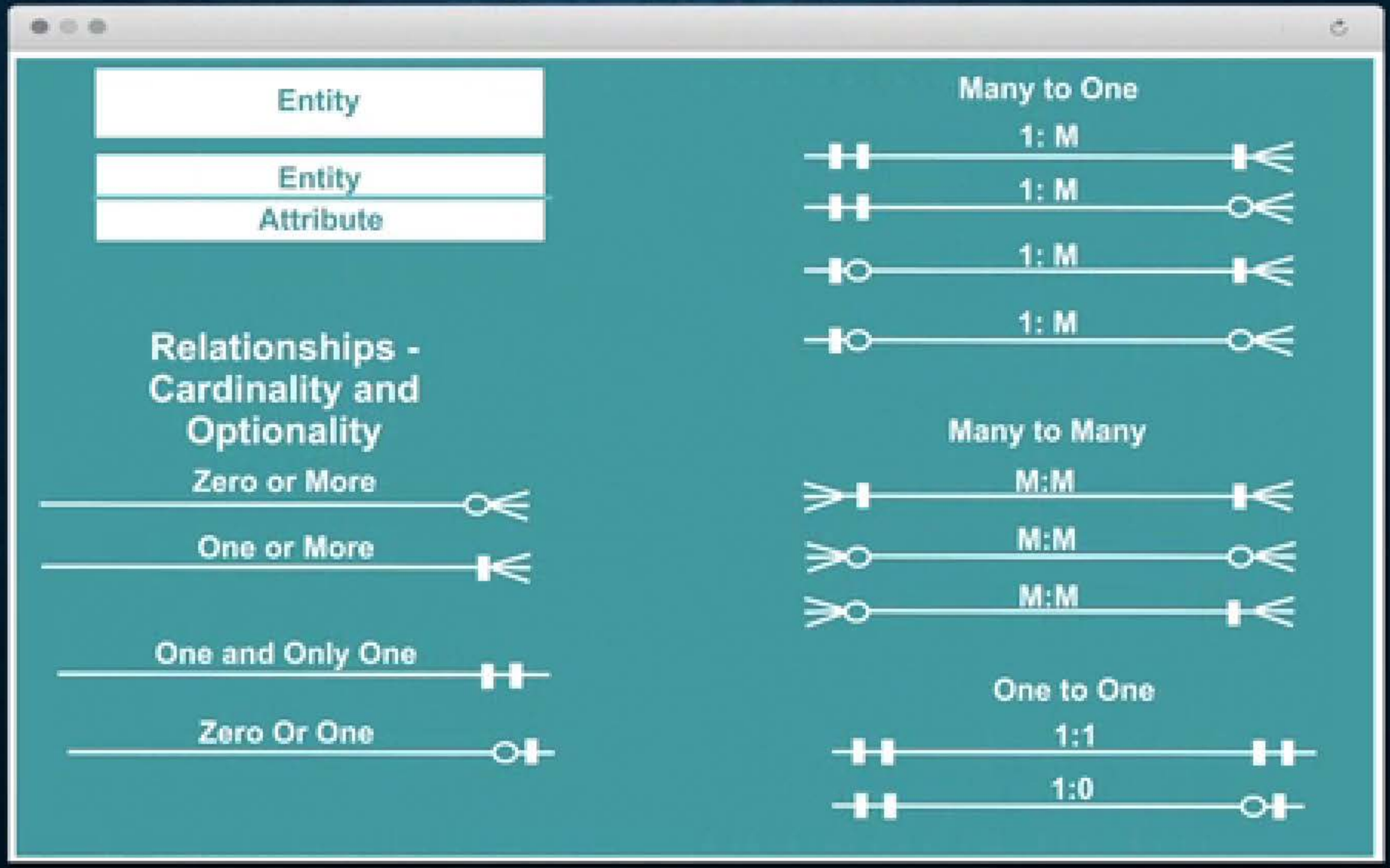
Is the relationship mandatory or optionality

Specifies the relations among entities

- Mandatory Relationships
- Optional Relationships

ERD EXAMPLE

ERD Example



ERD

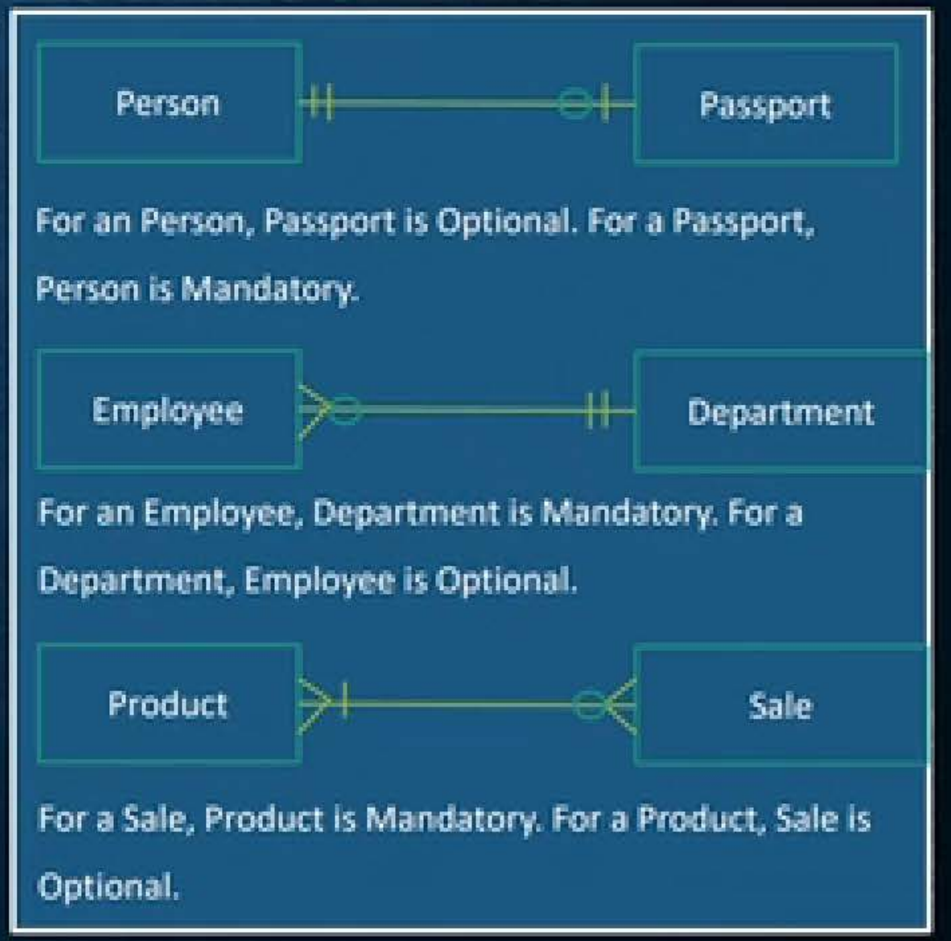
Relationship specifies association between two entities

CARDINALTY

- One Passport is related to one Person. One Person has only one Passport. Hence here the relationship is One-to-One.
- One Department can have many Employees. Hence relationship between Department and Employee is One-to-Many.
- One Product can have many Sales. Also one Sale can have many Products. Here relationship between Product and Sale is Many-to-Many.

OPTIONALITY

- For an Employee, Department is Mandatory. But for a Department, Employee is Optional.



DESIGN

TEKNOTURF

Design converts the software requirement specification document into system specification

System Design

- It is the process of defining the architecture, interface, components and other characteristics of a system Build a sentiment analysis model using this dictionary
- The design stage takes the requirements identified in the approved requirements document (SRS) as its initial input
- The design documents and the artifacts are produced at the end of the design phase

00:54

WHY DESIGN ?

TEKNOTURF

SYSTEM DESIGN

1. Design process helps in understanding the “how to” implement or a representation of the program and not the actual code or program
2. Design helps to enforce standards and guidelines and achieves reusability of modules
3. A good design achieves low coupling between the modules related and high cohesion within a module
4. Makes the system portable to different software or hardware platforms

GOOD DESIGN

TEKNOTURF

Efficiency

helps determine the runtime, response time and throughput. It also limits the memory usage, size of executables to be developed

Reliability

helps in achieving a design based on whether it is mission critical, real time or an online processing

QUALITIES

1. **Functional**

The very basic attribute that makes the system work and can be implemented

2.

3. **Flexible**

helps in building systems that can be easily modified when a requirement change happens

4. **Portability**

helps in designing a system that can be easily migrated to different hardware or software platforms

5.

LEVELS OF A DESIGN

TEKNOTURF

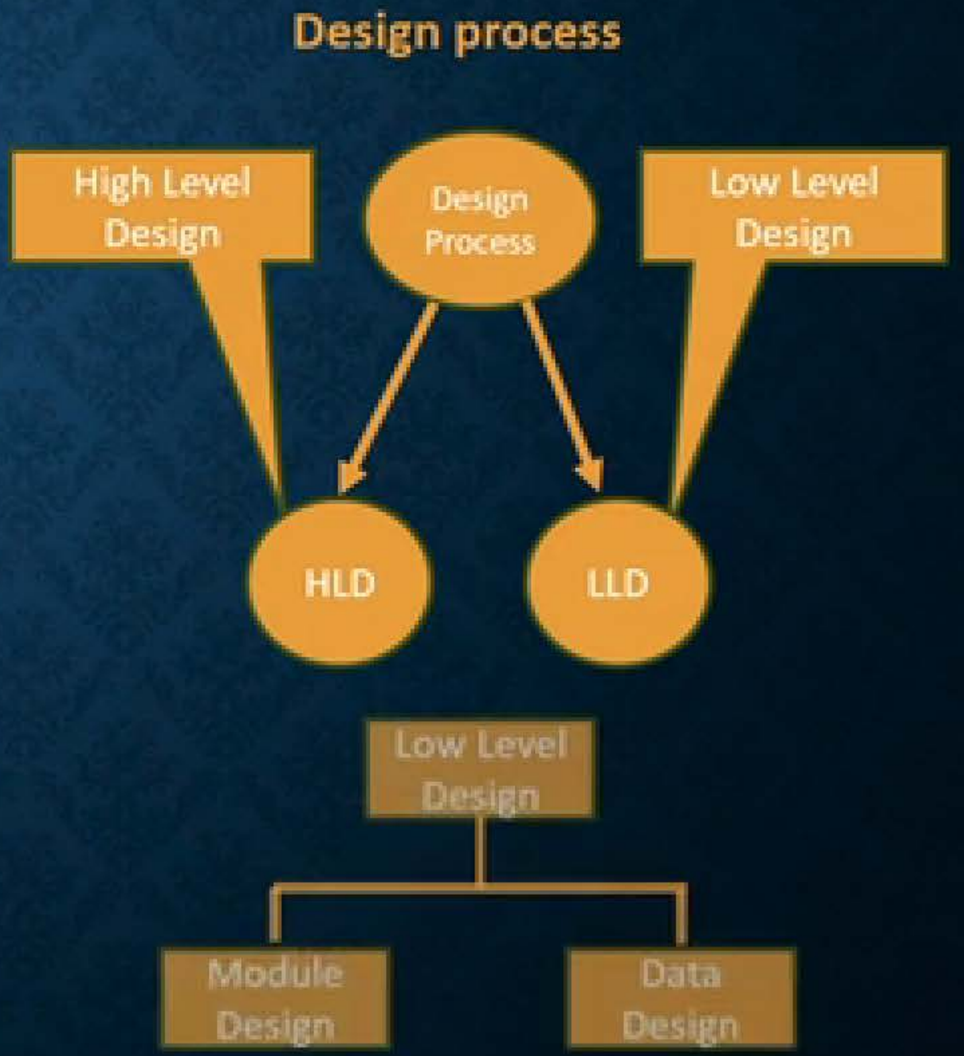
1. Decomposes the entire project into units or modules and identify the system architecture, data structure and processing logic

2. The two levels of design are High Level Design and Low Level Design

3. High Level Design focuses on
   - what modules are required
   - what each module performs, and
   - how these modules communicate with one another

4. Low Level Design focuses on
   - writing a detailed algorithm

5. DD(Design Document)= HLD + LLD



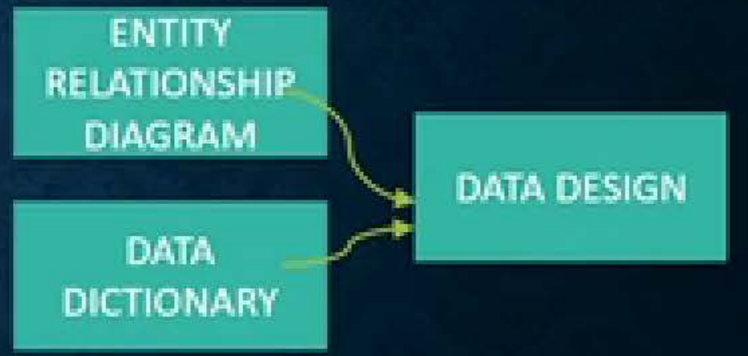
HIGH LEVEL DESIGN (HLD)

TEKNOTURF

DATA DESIGN

High Level Design, Architecture design, data design and interface design are part of high level design

The data design helps in transforming the data model created during requirements analysis phase into the data structures that will be used to implement the software



A database designer helps in creating the data architecture for a system to represent the data components

The Entity Relationship Diagram created during requirements Analysis is mapped to the data structures; the data dictionary is also mapped in the data design

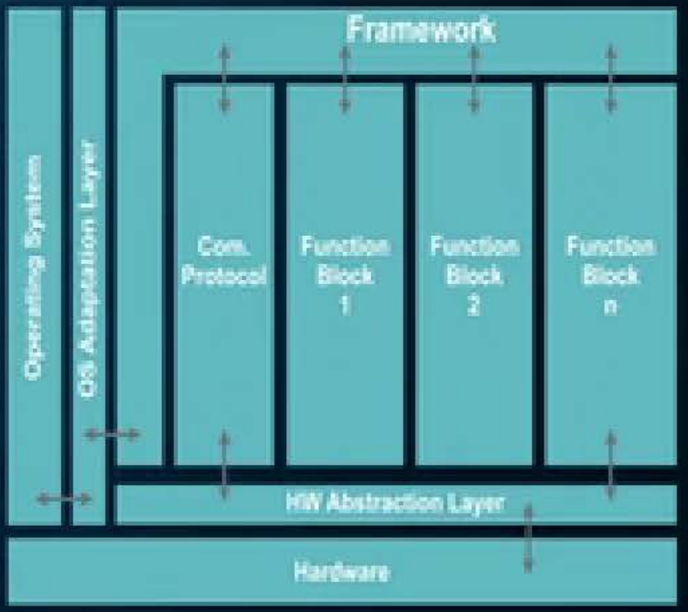
HIGH LEVEL DESIGN (HLD)

TEKNOTURF

ARCHITECTURAL DESIGN

Architecture design defines the modules of the system, the functions that each module perform, how they interact with each other, the design patterns used, and the constraints

The hardware and software used in developing are also specified in the architectural design



A system architect usually does the architectural design at the end of the software requirements analysis

Examples are Layered architecture, Distributed architecture and Client Server architecture

HIGH LEVEL DESIGN (HLD)

TEKNOTURF

INTERFACE DESIGN

1. The interface design focuses on the user interface, general interaction with the software, displaying the data and data inputs

2. The interface design in overall, describes how the software communicates with itself, and with the users interacting with the software

3. For example, selecting from menu options, command level input, display the data and other information such as feedback, Prompting the user for confirmation when a request is placed for the deletion of a file, usage of graphs and charts for voluminous data

4. Audio visual communication between user and interface) is also important; screen designs must focus on presenting relevant information to the user

5. Undo and redo features are also to be included in the interface design phase

LOW LEVEL DESIGN (LLD)

TEKNOTURF

PROCEDURAL DESIGN

1. Low Level Design focuses on writing a detailed algorithm
2. Procedural design or Detailed design also called as LLD focuses on algorithm design and concrete data representation at a higher level of abstraction and notation and it is not the implementing programming language
3. Algorithm is represented by pseudocodes that will map to the statements in the implementation language. The implementation is done in the coding phase
4. The interconnections among functions, and data structure is also focused in the low level design
5. Detailed data structures, algorithms, pseudocodes and procedural specifications are also part of low level design

DATA DESIGN

PERSON

personID : numeric(10) <<primary key>>
name: char(50)
contact: char(10)
city: char(25)

LOW LEVEL DESIGN (LLD)

TEKNOTURF

PROCEDURAL DESIGN - PSEUDOCODE

1. Pseudocodes helps the programmer to write better abstractions of logics

2. Using a simple and plain language to represent the various modules to be implemented and describing what is to be coded; the Implementation follows the low level design

PSEUDOCODE

```
set total to zero

get list of numbers

loop through each number in the list
   add each number to total
end loop

if number more than zero
   print "it's positive" message
else
   print "it's zero or less" message
end if
```

CONSTRUCTION (CODE + UNIT TESTING)

TEKNOTURF

Modular and subsystem programming code is accomplished during this stage

DEVELOPMENT

1

The goal of the coding phase is to convert the design into a workable solution using any programming language

2

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

3

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

CONSTRUCTION (CODE + UNIT TESTING)

TEKNOTURF

GUIDELINES

1. Using standard control constructs for looping and decision making, usage of user defined datatypes, abstracting the data structures, naming variable conventions, indentations
2. Software Verification and Validation (V&V) can be enforced on the software being developed
3. Reviews and tests can be performed at the end of each phase of the software development
4. Reviews and tests ensures that the software requirements are complete and testable and that design code, and documentation satisfy the requirements

TESTING

TEKNOTURF

Scenario 1

An organization conducted test for hiring employees. They used OMR Sheets for the candidates to answer

To evaluate the OMR sheet, the code was written to check if the candidate has shaded the correct option

Example : For Question 1, if the answer was Option C, he code will check if option C was shaded.

Do You find any flaw in the evaluation process ?

What would be the evaluation result if a candidate shades all the options for all the questions ?

The score will be 100 because the code checks if the correct answer is shaded.

Testing should be done to find fault with the application tested rather than to prove that it works fine as expected.

A Good Testers intention should be to find the presence of defects in the code and not its absence.

TESTING

TEKNOTURF

Scenario 2

Consider testing of a Banking application.

Testers provided all possible inputs and tested all functionalities in that application. All worked fine.

When the client tested the application, it works fine for valid and invalid combinations.

After clicking logout, client clicked the back button.

It displayed the previous page and also allowed to perform any other transactions too.

This is a major flaw which should be considered for testing.

TESTING

TEKNOTURF

Testing is the process of executing a program with the intention of finding errors

1. **Define**

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

2. **V&v**

   Verification - Are we building the product right?

   Validation - Are we building the right product?

Example

Peter ordered for a Choco chip cone ice cream

VERIFICATION:

Is it a cone?

Does it have Choco chips around ?

Does it have chocolate flavor?

VALIDATION:

Peter consumes the ice cream to ensure if it is as per his expectation

TESTING

TEKNOTURF

Testing is the process of executing a program with the intention of finding errors

1. **Define**

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

2. **V&v**

   Verification - Are we building the product right?

   Validation - Are we building the right product?

3. **Ensures**

   Software testing is a process that ensures correctness, completeness and quality of the developed software

4. **IEEE 610 STANDARD - DEFINITION**

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

5. **Testing steps**

   Software testing is planned.

   Tests are clearly specified, and after execution, results are recorded

IMPORTANCE AND NEED OF TESTING

TEKNOTURF

Ensures that the software meets requirements and is fit for use.

Saves money by identifying defects before the software is released. Remember, the earlier a defect is identified and fixed, the better

Minimises risk in extending existing software. Less risk the software is, more the software becomes trusted.

Provides information on the limits / constraints of the software application that can be communicated to end-users, and other stakeholders to keep in mind.

For example, in a University, assume there is a student management system. The system shows incorrect marks due to some bug in the software. This is a bug. Rather if the software is properly tested, it would have displayed an error message like " Unable to display marks. Please try later". So, testing plays a vital role in software development and introducing testing early in the life cycle ensures that such errors can be identified and prevented from cascading to later phases

DEBUGGING

TEKNOTURF

Debugging is the process of finding the source of the already identified bugs and fixing it.

Performed by Developers

1. Find the defect in the code
2. Identify the root cause of the problem
3. Identify the exact place in the code that is the cause of the problem
4. Fix the defect
5. Recheck to ensure that the defect fixed is working as expected

SOFTWARE TESTING LIFE CYCLE(STLC)

TEKNOTURF

STLC identifies the activities that are to be carried out and determines when (best time) to accomplish the activities

Phases of STLC

Test plan

- Resource allocation
- Creating test environment
- Test schedule

Test design

Writing test scenario, test cases, test data and test scripts

Test execution

Executing test scripts and Finding bugs

Report to developer

After test execution, if there are any bugs, it is reported to the developer.
Once the developer closes the bug, tester does the regression testing to ensure quality

Test cycle closure

Once testing is completed, testing artifacts are documented.

METHODOLOGIES OF TESTING

TEKNOTURF

Manual Testing

Manual testing is where the tests are executed manually by the QA

Automated

Testers use automated scripts / tools to automate test execution and compare the expected with the actual results

Examples - Automated tools

junit – java

Nunit - dotnet

Defect tracking

Process of logging and monitoring the bugs during testing

Examples – defect tracking tools bugzilla

LEVELS OF TESTING

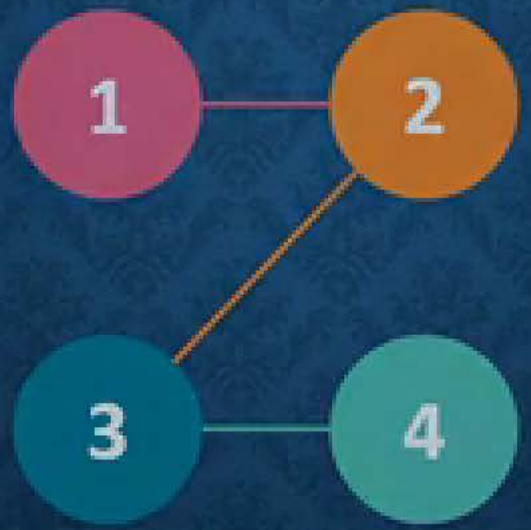
TEKNOTURF

Unit testing

Done by the developer - individual module is for functional correctness

System testing

Done by the end user for the system acceptance



Integration testing

Checks for the interface errors between the integrated components

Acceptance testing

The system is tested as a whole to check the functional and non functional correctness

UNIT TESTING

TEKNOTURF

Done by the developers

Module

- Unit testing is done to test whether a particular module is implementing its specification
- Involves verification of the code that was delivered in coding phase

Who does?

- Unit testing is normally done by the software developers themselves or their peers
- Rarely the unit testing can also be done by the independent software testers

INTEGRATION TESTING

TEKNOTURF

Done by the developers

Integration

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

Who does?

Either the Developers themselves or the independent testers

Expose faults

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

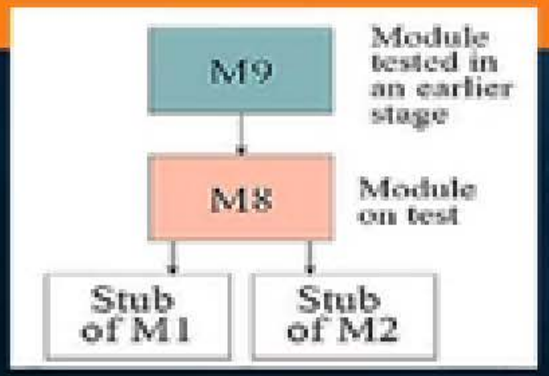
SYSTEM TESTING

TEKNOTURF

Performed by the testers

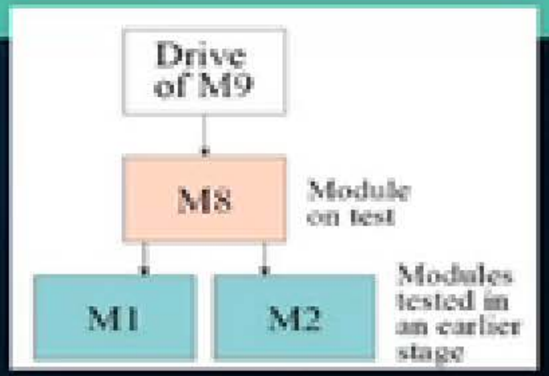
Top down

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



Bottom up

- Bottom level units are tested first and upper-level units' step by step after that
- Drivers are dummy module through which other functions/modules are called
- Test Drivers are needed to simulate higher level units which may not be available during the initial phases



SYSTEM TESTING

TEKNOTURF

Performed by the testers

Aim

- System testing finds disparities between implementation and specification

What does it test?

System testing involves

- Functional testing - Tests the implementation of the business needs
- Performance testing - Tests all the non functional requirements of the system specified in the specification

PERFORMANCE TESTING - TYPES

TEKNOTURF

Methodologies – Manual and Automated

Stress Test	» Stress test evaluates the system when stressed to its limits
Regression Test	» Regression test is required when the system being tested is replacing an existing system
Usability Test	» Usability test is testing characteristics related to user friendliness

USER ACCEPTANCE TESTING

TEKNOTURF

Done by the Client

WHO DOES IT?

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

OBJECTIVE

To confirm if the system is as per the users requirement and certify that it is acceptable for delivery to the customer

Types

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

TESTING PRINCIPLES

TEKNOTURF

SEVEN TESTING PRINCIPLES

Ensures following a right strategy for testing

1. Testing shows the presence of defects
2. Early testing
3. Exhaustive testing is impossible
4. Defect clustering
5. Pesticide paradox
6. Testing is context dependent
7. Absence of error - fallacy

TESTING PRINCIPLES

TEKNOTURF

Testing shows the presence of defects

Testing shows the presence of defects but not their absence

Early testing

Early testing is done to find out early defects in each phase of the SDLC

Exhaustive testing is impossible

100% testing (all possible inputs and preconditions) is not always possible

Absence of error - fallacy

Finding defects on a system that does not match users requirements is helpless

Defect clustering

A few smaller modules may contain many defects and could cause the failure of the software

Pesticide paradox

Avoid repeating same test; new test cases have to be revised; write different testcases for exercising different parts of the code

Testing is context dependent

Testing a mission critical system is different from testing a library management system

TYPES OF TESTING

TEKNOTURF

Static testing

The techniques are

- Review
- Walkthrough
- Inspection

Dynamic testing

The techniques are

- Black box testing
- White box testing

STATIC TESTING

TEKNOTURF

Manually

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

Documents

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

Techniques

Reviews, Walkthrough and Inspection

No Execution

Execution of the Code is not performed. Sanity of the code is checked. This means, we can do static testing of the code, to ensure that addition or modification of any feature in the application has not broken anything in the code

MEMBERS OF STATIC TESTING

TEKNOTURF

Author

Moderator

Reader

Recorder / Scribe

Inspector

TECHNIQUES IN STATIC TESTING

TEKNOTURF

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 is 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

TEKNOTURF

Review is a static testing technique

During review the artifacts are manually examined and the findings are recorded; Led by the moderator

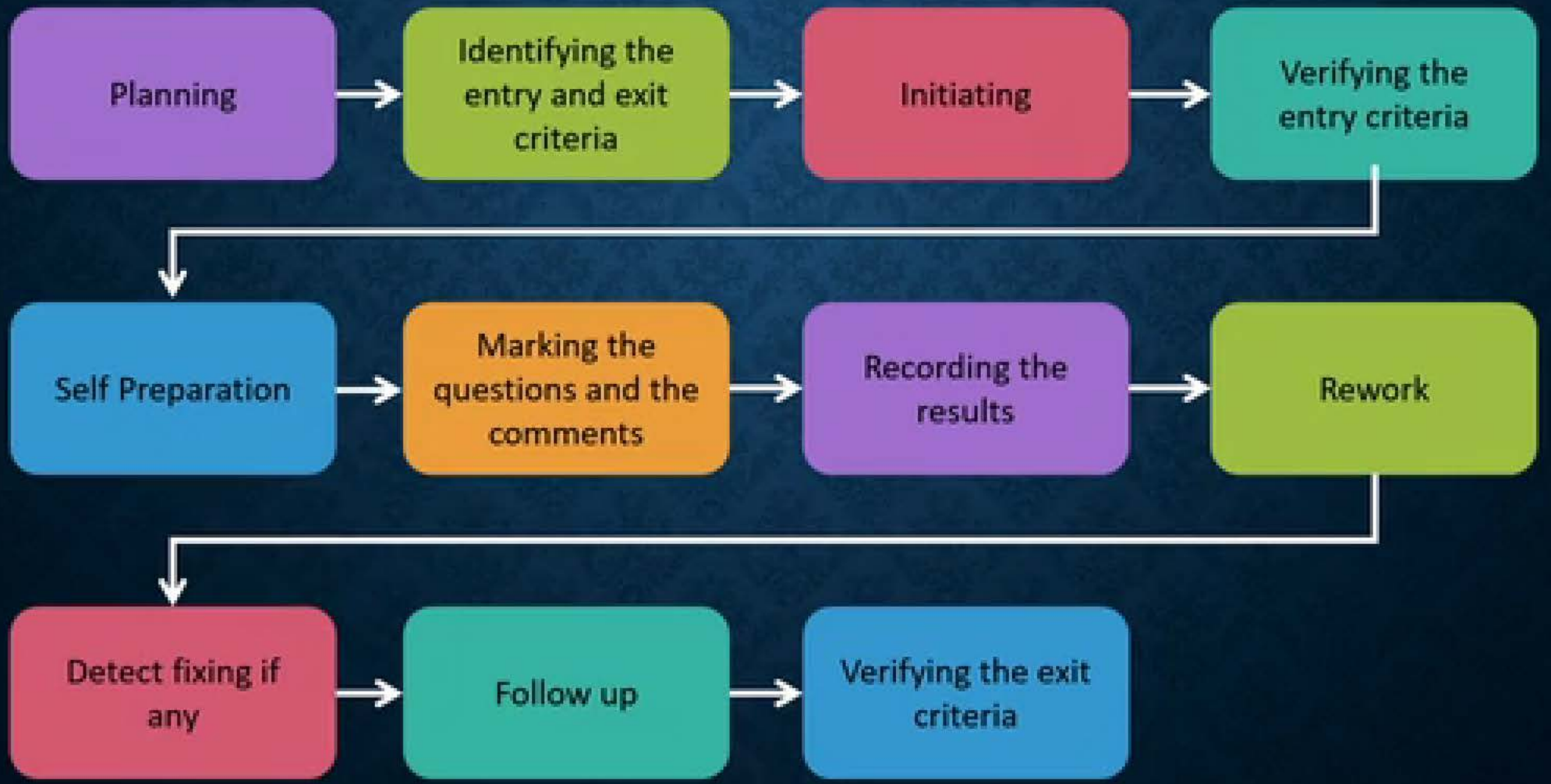
Artefacts for which reviews are performed are

- SRS
- Design specification
- Source code
- Test plans
- Test specification
- Test cases
- Test scripts
- User guides
- web pages

REVIEW PROCESS

TEKNOTURF

Review is a static testing technique



WALKTHROUGH

TEKNOTURF

A meeting which focuses on dry runs of code and scenarios testing; Led by the Author

This meeting can be conducted in an informal way that can end up in a formal way

Peer groups can participate in walkthroughs

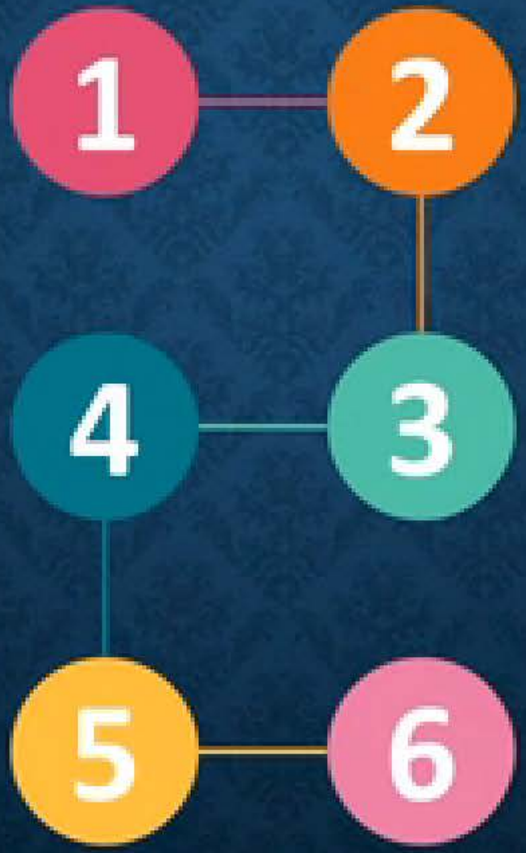
Finding defects is the main aim of a walkthrough; understanding the defects through walkthrough

Review report can be prepared which includes a list of all the findings

INSPECTION

TEKNOTURF

1. Prepared before the meeting; led by a trained moderator
2. Aims at finding defects
3. Review report can be prepared which includes a list of all the findings
4. Specified entry and exit criteria for the product acceptance
5. Inspection reports contains list of findings
6. Follow up process is done in a formal manner



STATIC TESTING ADVANTAGES

TEKNOTURF

- Early defect detection and correction
- Fewer defects
- Improve the development productivity
- Helps in identifying omissions of requirements
- Saves testing cost and time
- Communication within team is improved

TYPES OF TESTING

TEKNOTURF

- Static Testing
  - Review
  - Walk through
  - Inspection

There are two types of testing

- ✓ Static testing – The techniques are Review, Walkthrough and Inspection.
- ✓ Dynamic testing – The techniques are Black box testing and White box testing.

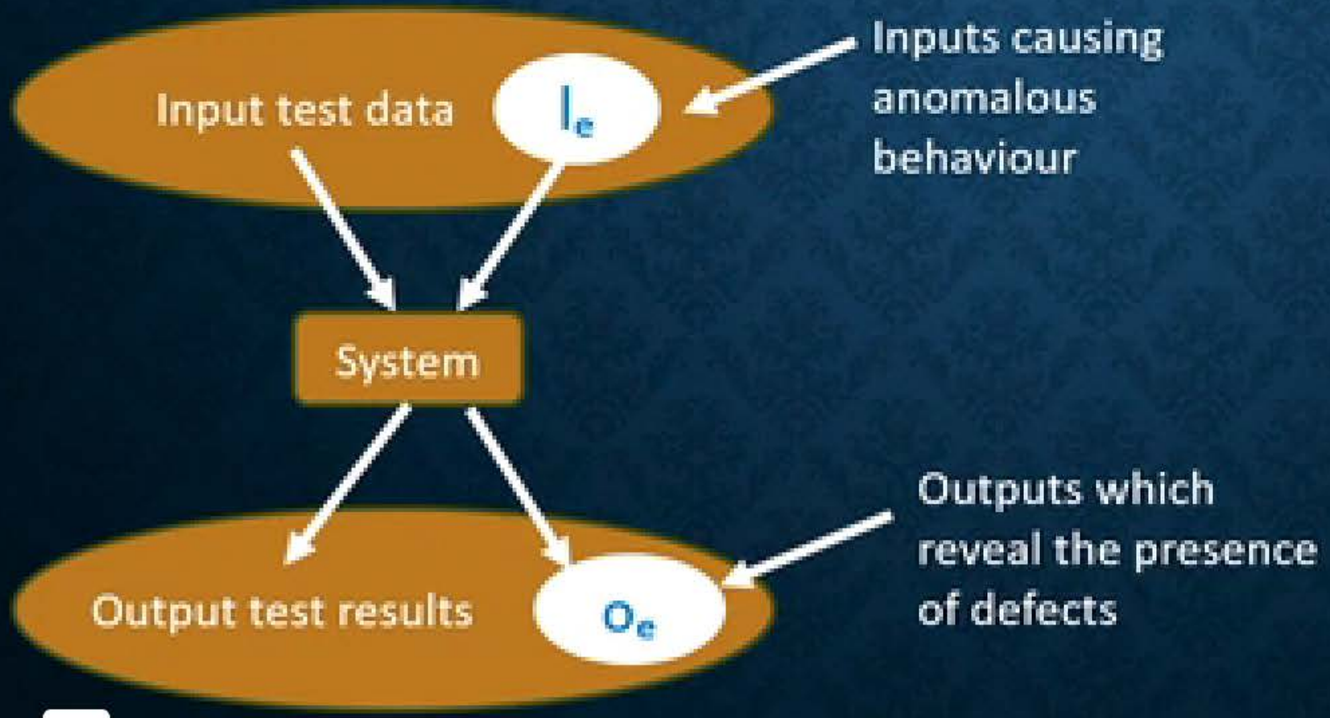
- Dynamic Testing
  - Black box testing
  - White box testing

DYNAMIC TESTING - BLACK BOX TESTING

TEKNOTURF

Testing after the system is done

It is a 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.



Black Box Testing

DYNAMIC TESTING

TEKNOTURF

DYNAMIC TESTING - TEST CASE

Test cases

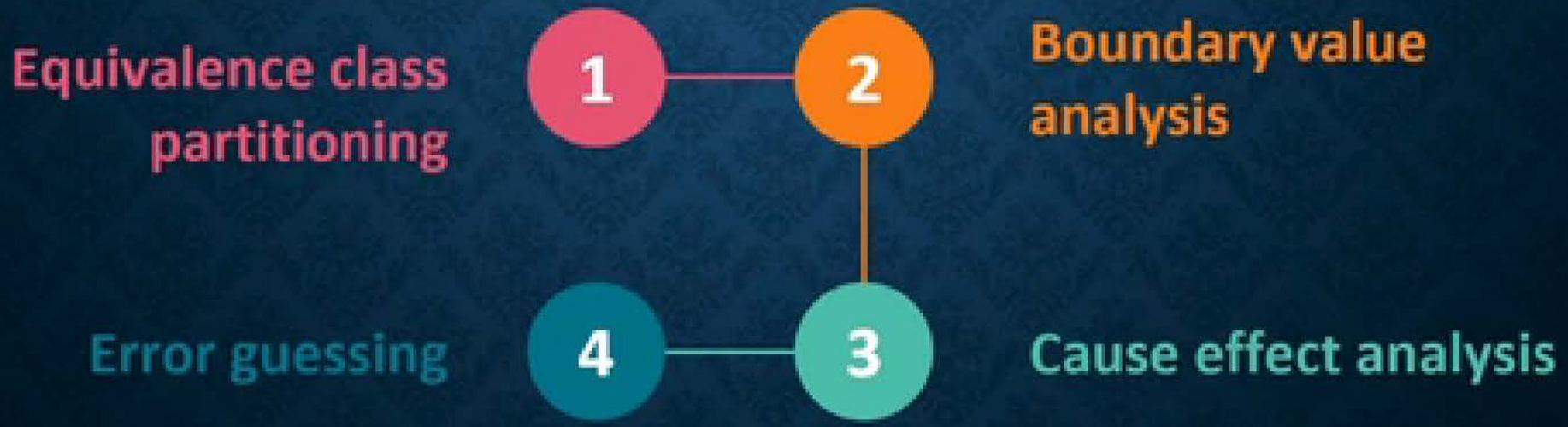
1. Once the test plan for the level of testing has been written, the next stage of test design is to specify a set of test cases or test path for each item to be tested at that level

2. Each test case will specify how the implementation of the particular requirement or design decision is to be tested and the criteria for success of the test

BLACK BOX TESTING – TECHNIQUES FOR GENERATING A TEST CASE

TEKNOTURF

BLACK BOX TESTING TECHNIQUES



EQUIVALENCE CLASS PARTITIONING

TEKNOTURF

EQUIVALENCE PARTITIONING

1. Black-box technique divides the input domain into classes of data from which test cases can be derived
2. 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
3. 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

EQUIVALENCE PARTITIONING

EXAMPLE SCENARIO

- The component "generate_grading" is the scores (out of 75) and a coursework (c/w) scores (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 error message ('FM') is generated

    All inputs are passed as integers

BOUNDARY VALUE ANALYSIS

TEKNOTURF

BOUNDARY VALUE ANALYSIS

1. Black-box technique focuses on the boundaries of the input domain rather than its center

2. For each of the 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

Example : If input condition specifies a range 5 to 20

CAUSE EFFECT ANALYSIS

TEKNOTURF

CAUSE EFFECT ANALYSIS

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

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

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

CAUSE EFFECT ANALYSIS

TEKNOTURF

CAUSE EFFECT ANALYSIS

EXAMPLE SCENARIO

- A module for calculating increment, that decides how much increment percent will be assigned to every employee
- The decision is based on the experience and the designation
  - ✓ Between 1 – 3 then incr% 10
  - ✓ Between 4 – 10 then incr% 20
  - ✓ Greater than 10 then incr% 30

  For designation

  - ✓ Developer then incr% 30
  - ✓ Tester then incr% 40

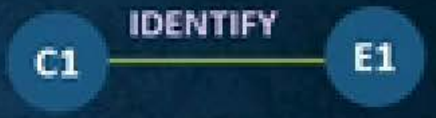
CAUSE EFFECT ANALYSIS

TEKNOTURF

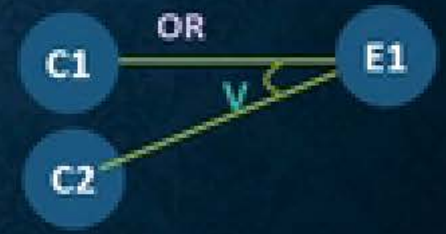
CAUSE EFFECT GRAPHING

1. The cause effect graphing is used to capture the relationship between certain combinations of inputs(causes) and outputs(effects)
2. The causes and effects represent the nodes of a cause effect graph
3. Intermediate nodes are also present, and they represent the linking of causes and effects
4. The steps involved are:
   - Specify the inputs (causes) and actions (effect)
   - Design a cause effect graph
   - Convert the cause effect graph into a decision table
   - Decision table rules is converted to test cases
   - Each column of the decision table represents a test case

NOTATIONS

1. 

2. 

3. 

4. 

CAUSE EFFECT GRAPHING

TEKNOTURF

CAUSE EFFECT GRAPHING - EXAMPLE

1. An employee record has to be updated if an input U1 is received. Else it has to be deleted if an input of D1 is received. If the first character of the input is not U or D, the record is simply printed.

2. 

3. TreeSet internally uses compareTo() method to compare the objects for sorting.

DECISION TABLE

C1 (U)	C2 (D)	C3 (1)	A1 (PRINT)	A2 (UPD /DEL)
1	0	1	0	1
0	1	1	0	1
0	0	1	1	0

STATE TRANSITION DIAGRAM

TEKNOTURF

State transition model is a black box technique to deriving test cases.

A state transition diagram has four components:

- ✓ The states that the software may occupy (Example., A word document being in an open or closed state)
- ✓ The transitions from one state to another (A word document from open to closed state)
- ✓ The events that cause a transition (Closing a file makes the document transition to closed state)
- ✓ The actions that result from a transition (a file being opened or closed).

A transition may not perform the transition to a new state. An invalid input gives an error message as the action, and the transition would be back to the same state as the system was in before.

CREATION OF A PLOT

TEKNOTURF

1 **Ad-hoc approach**

It is an ad hoc approach guided by intuition and experience

2 **Example**

Consider that a program is written into a file. The following test cases can be derived

- ✓ Having an empty file
- ✓ Not having a file at all
- ✓ Having a file with read permission

BLACK BOX TESTING

TEKNOTURF

Scenario

- Azure Solutions is planning to give incentive for the employees based on the appraisal rating obtained by them.
- The rating and the incentive amount are mentioned below.

Appraisal Range	Incentive
7 – 10	1000 USD
4 – 6	500 USD
1-3	300 USD

Test Data Using	
Equivalence Partitioning	**BVA**
3, 9,12	6,7,10,11
2,5,9	3,4,6,7
-2,2,5	-1,1,3,4

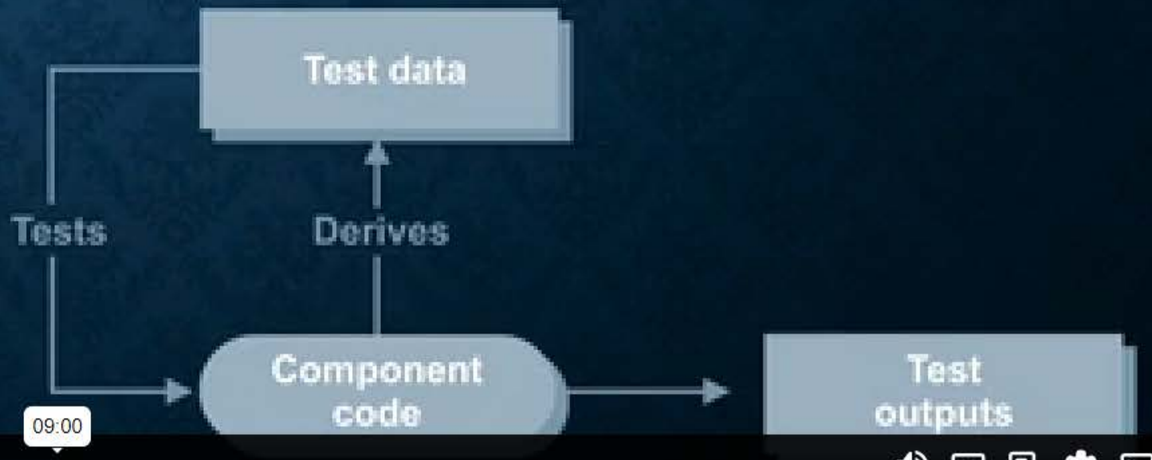
DYNAMIC TESTING – WHITE BOX TESTING

TEKNOTURF

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

WHITE BOX TESTING

1. It deals with the internal logic and structure of the code
2. Derive test cases based on the program structure
3. To perform white box testing testers should have a thorough knowledge of
   - Problem Domain
   - Internal implementation of the code



09:00

BASIS PATH TESTING

TEKNOTURF

Basis Path Testing is a white box testing method

MAIN FEATURES OF BASIS PATH TESTING

- 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 – STATEMENT COVERAGE

TEKNOTURF

Has each and every statement in the code been executed?

MAIN FEATURES OF STATEMENT COVERAGE

The number of executable statements covered in a code determines the statement coverage

Assessment of statement coverage = number of executable statements covered in the code / Total number of executable statements in the code to be tested

BASIS PATH TESTING – STATEMENT COVERAGE

TEKNOTURF

Has each and every statement in the code been executed?

MAIN FEATURES OF STATEMENT COVERAGE

The number of branch statements (e.g., IF / CASE) covered in a code determines the branch or the decision coverage

Assessment of branch coverage = number of all branches or decisions covered in the code / Total number of all possible branch or decision statements in the code

BASIS PATH TESTING – LOOP COVERAGE

TEKNOTURF

How many times is the loop executed?

MAIN FEATURES OF LOOP COVERAGE

The number of times loops are executed in a code determines the loop coverage;

0 times, 1 time or multiple times

For example, for a do while loop, the number of times the loop can be executed is

1 or many

BASIS PATH TESTING

TEKNOTURF

DERIVE A LOGICAL COMPLEXITY MEASURE OF PROCEDURAL DESIGN

- ✔ Break the module into blocks delimited by statements that affect the control flow (e.g., 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

MC CABE'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

BLACK BOX VS. WHITE BOX TESTING

TEKNOTURF

COMPLEXITY OF A FLOW GRAPH 'G', V(G), IS COMPUTED IN ONE OF THESE THREE WAYS

- V(G) = No. of closed 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)

STEPS INVOLVED

- 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

TEKNOTURF

SAMPLE CODE

```
main() {
        int a,b;                    1
        scanf("%d%d",&a,&b);
2       if(a > b)
          printf("\na is greater");  3
        else
          printf("\nb is greater");  4
        }  5
```

MCCABE'S NUMBER

Mc cabe's No

- ✓ P+1= 2
- ✓ E-N+2 = 5-5+2 = 2
- ✓ R+1 = 2
- ✓ 1-2-3-5 a=10 , b=5
- ✓ 1-2-4-5 a=2 , b=6

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 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

SOFTWARE EVOLUTION

TEKNOTURF

1. It is impossible to produce a system of any size which does not need to be changed
2. 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
3. After delivery, software systems always evolve in response to demand for change

SOFTWARE CHANGE

TEKNOTURF

01:06

SOFTWARE EVOLUTION APPROACHES

TEKNOTURF

SOFTWARE EVOLUTION

- Software maintenance
- Architectural transformation
- Software re-engineering

SOFTWARE CHANGE STRATEGIES

TEKNOTURF

1

Software Maintenance

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

2

Architectural Transformation

The Architecture of the system is modified; Generally from a centralized to a distributed architecture.

Architecture may also get modified depending on the requirement / need.

3

Software re-engineering

New functionalities are not added to the system but it is restructured and reorganized to facilitate future changes

SOFTWARE MAINTENANCE

TEKNOTURF

1

Modify a program after it is being used

2

Maintenance does not normally involve major changes to the system architecture

3

Changes are implemented by modifying the existing components or by adding new components to the system

TYPES OF SOFTWARE MAINTENANCE

TEKNOTURF

02:42

TYPES OF MAINTENANCE

TEKNOTURF

CORRECTIVE MAINTENANCE

1. Maintenance that includes the repair of the defects in the existing system

2. Defects can stem from
   - Requirements specification errors
   - Design errors
   - Coding errors
   - A large number of problems stem from Requirements and Design

TYPES OF MAINTENANCE

TEKNOTURF

ADAPTIVE MAINTENANCE

1. Maintenance to adapt software to changes in the working environment

2. Invoked by
   - Internal needs; e.g. run the software on a new operating system
   - External requirement e.g. change in law

TYPES OF MAINTENANCE

TEKNOTURF

Perfective Maintenance

1

Involves making functional enhancements to the system which can increase the system's performance

2

Includes all efforts to polish or refine the quality of the software or the document

3

Important that the improvement reduces the system maintenance cost

Preventive Maintenance

1

Changes made to the system to prevent occurrence of errors in future

2

Changes made to the system to avoid any Software faults in the future

SOFTWARE MAINTENANCE

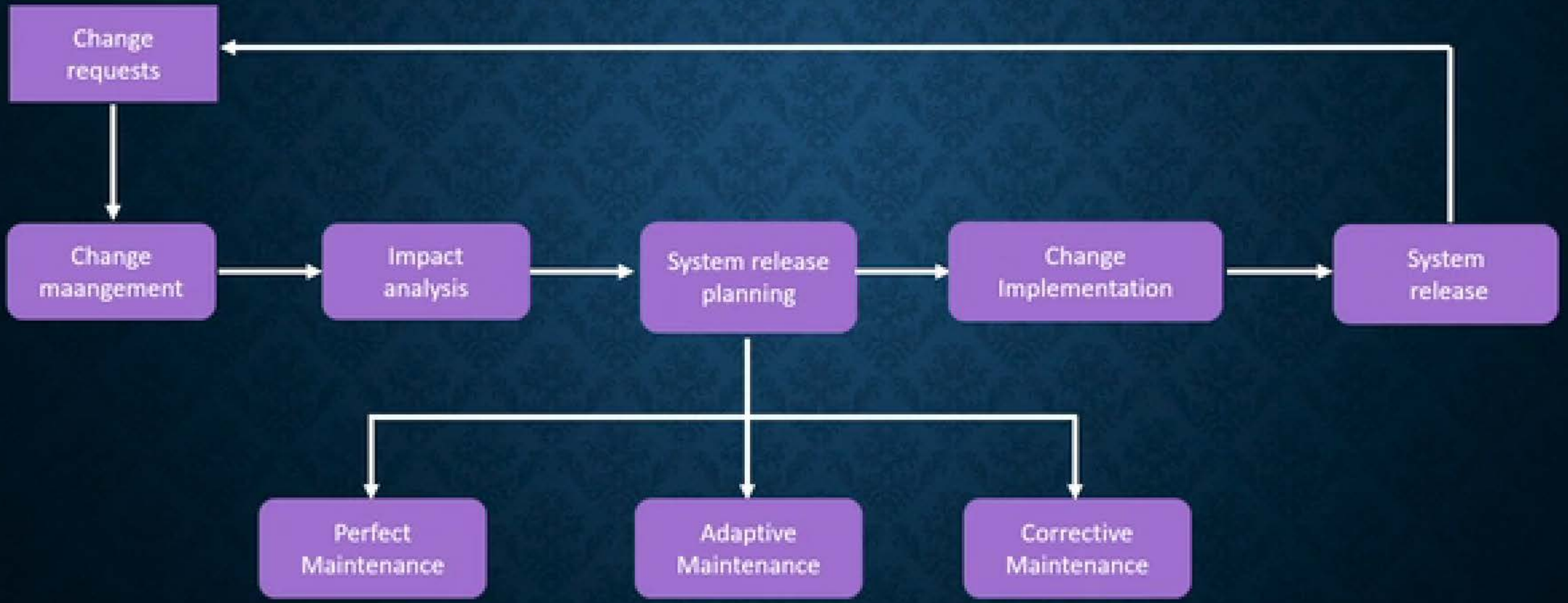
TEKNOTURF

Maintenance activity can be for

- Fault Repair
- Software Adaption
- Functionality Addition or Modification

A study on maintenance effort distribution revealed that Effort for functionality addition or modification of existing components is more compared to Software Adaption and Fault F

SOFTWARE MAINTENANCE



05:17

MAINTENANCE COST

TEKNOTURF

Maintenance Cost

1. Allows a variable or value to be tested for equality against a list of values.
2. Each possible input value is referred to as a case.
3. It can contain an unlimited number of case statements.
4. If more than one match exists, the first matching element is returned.

RELATIVE COST OF MAINTENANCE

TEKNOTURF

1

Cost of maintenance is relatively high.

2

A study on estimating software maintenance resulted as the cost of maintenance is so high that it consumes major percentage (67%) of the cost of entire software process cycle.

SOFTWARE MAINTENANCE EXAMPLES

TEKNOTURF

Corrective Maintenance

Tom developed an application for Global pharmacy. When the client was using that application, they found a defect that the application accepted a date of expiry which was prior to the current date. They wanted this error to be rectified. The application should not allow the user to add medicine with such expiry date. What Maintenance does this scenario depict ?

Adaptive Maintenance

In the Pharmacy application, there was a provision to calculate discount for the total bill amount for a sale. But as the discount may vary for each medicine, the client wanted the provision to calculate discount for each medicine in the bill. What Maintenance does this represent ?

Perfective Maintenance

Reports were generated for the Pharmacy application representing the sales and purchase details of medicines. Developer felt that along with those details which were displayed in a table format, it will be good if the same is displayed in graphical notation too and implemented the same. What maintenance does this scenario represent?

Preventive Maintenance

Grab your product" is an online shopping application for Mobile and its accessories. As it was New year Eve, the client expected more number of customers might access their link for shopping. To prevent the network issue, they decided to make necessary changes in the cloud. What type of maintenance is this?

SOFTWARE CONFIGURATION MANAGEMENT

TEKNOTURF

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

SCM helps to minimize the above confusions by coordinating software development activities, by "identifying, modifying and controlling" modifications to software.

As we saw in the situation with the ABC Bank document, multiple issues crop up when software is collaboratively developed by Teams.
Some of them are :-

- Are all changes done by team members reflected in the repository?
- Who did what change at what time?
- Is it possible to revert back to the previous version of the code/ project?

SCM activities are to

- identify the change,
- control the change,
- ensure that the changes are being properly implemented,
- report the changes to others who are involved in it.

WHY SOFTWARE CONFIGURATION MANAGEMENT

TEKNOTURF

1. Most projects developed in IT industry are complex
2. Lots of people are involved in the project. All need to update their code in the repository
3. There will be conflicts in code when changes are made by all in the team.
4. There will be communication gap when changes are made
5. The solution for these problems - **Software Configuration Management**

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:

- System change is a team activity
- Configuration Management is responsible for managing and controlling the costs and effort involved in making changes to a system.

SOFTWARE CONFIGURATION MANAGEMENT

TEKNOTURF

Software entities that SCM is expected to manage include :

- Specifications (SRS,)
- Design
- Programs
- Test data
- User Documentation
- 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

SOFTWARE CONFIGURATION MANAGEMENT PLAN

TEKNOTURF

Software configuration management plan starts during the early phases of a project.

SCM PLAN DEFINES

- The various artifacts to be managed along with the code 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.

SCM plan describes the tools which should be used to assist the CM process and any limitations to their use.

CONFIGURATION ITEM

TEKNOTURF

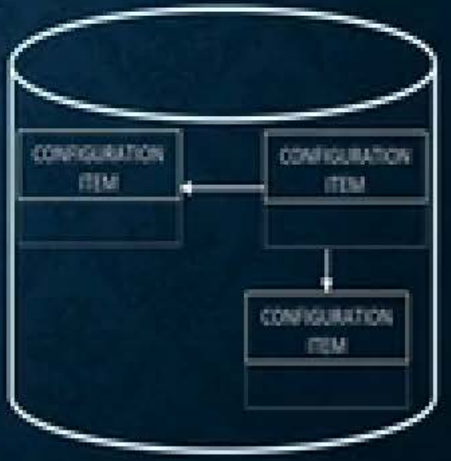
A Configuration Item is an aggregation of hardware, software, or both, that is designated for configuration management and treated as a single entity in the configuration management process.

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.

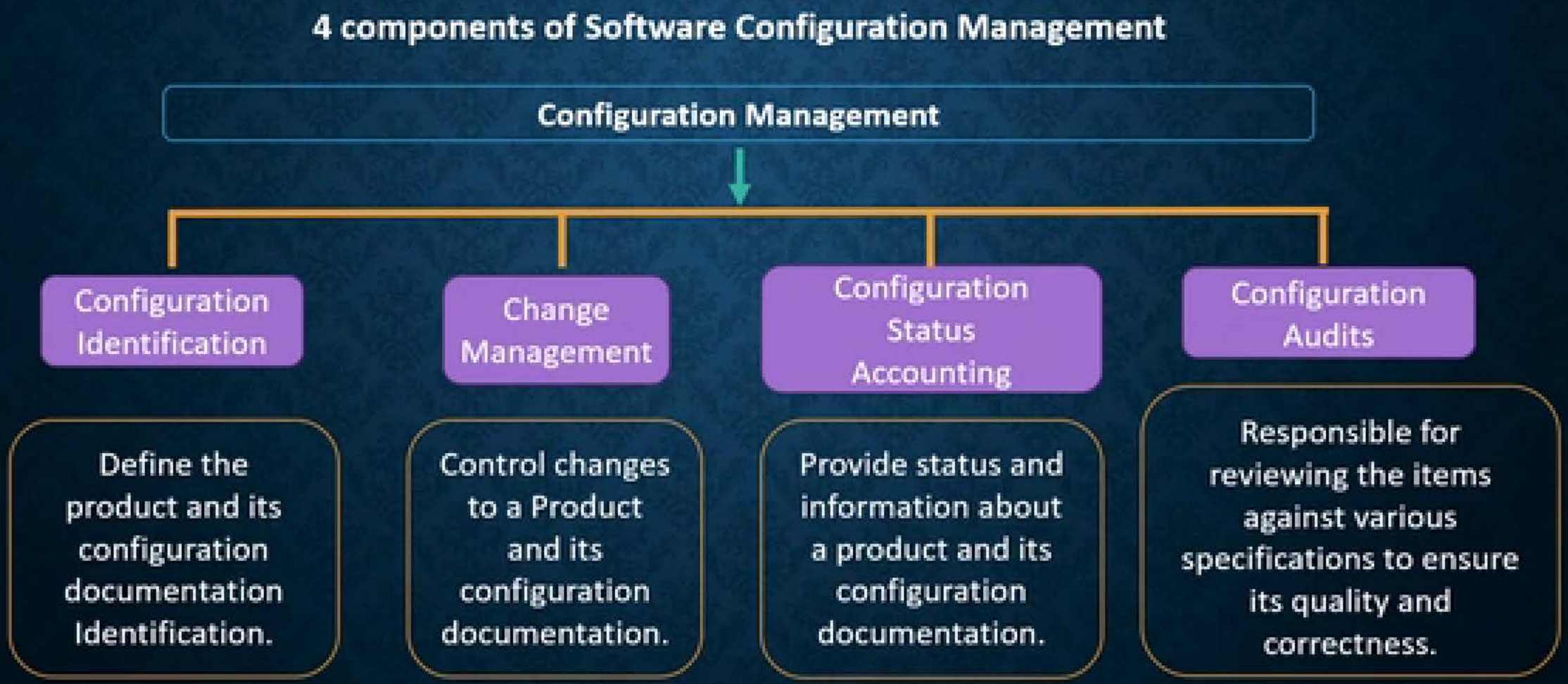
Document naming scheme should be defined so that related documents have related names.

IEEE



SOFTWARE CONFIGURATION MANAGEMENT

TEKNOTURF



TYPES OF CONFIGURATION OBJECTS

TEKNOTURF

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

There are two types of Objects : Base Object and Aggregate Object

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

BASELINE

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"

IEEE Std. No. 610.12-1990

Example for Baseline

Baseline A : The API has been completely been defined; the bodies of the methods are empty.

Baseline B : All data access methods are implemented and tested.

Baseline C : The GUI is implemented.

BASELINE

TEKNOTURF

As system is developed, series of baselines are developed

- Developmental Baseline
- Functional Baseline
- Product Baseline

Many naming scheme exist for baselines (2.0,3.01a, …)

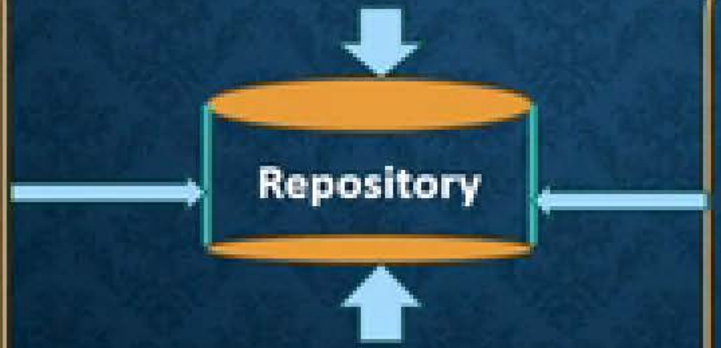
A 3 digit scheme is quite common

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?

CHECK -IN AND CHECK-OUT

TEKNOTURF

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.
	A process that creates a new version of an object from an existing version stored in the database.
CHECKOUT	Developers check out objects so they can work on them.

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:

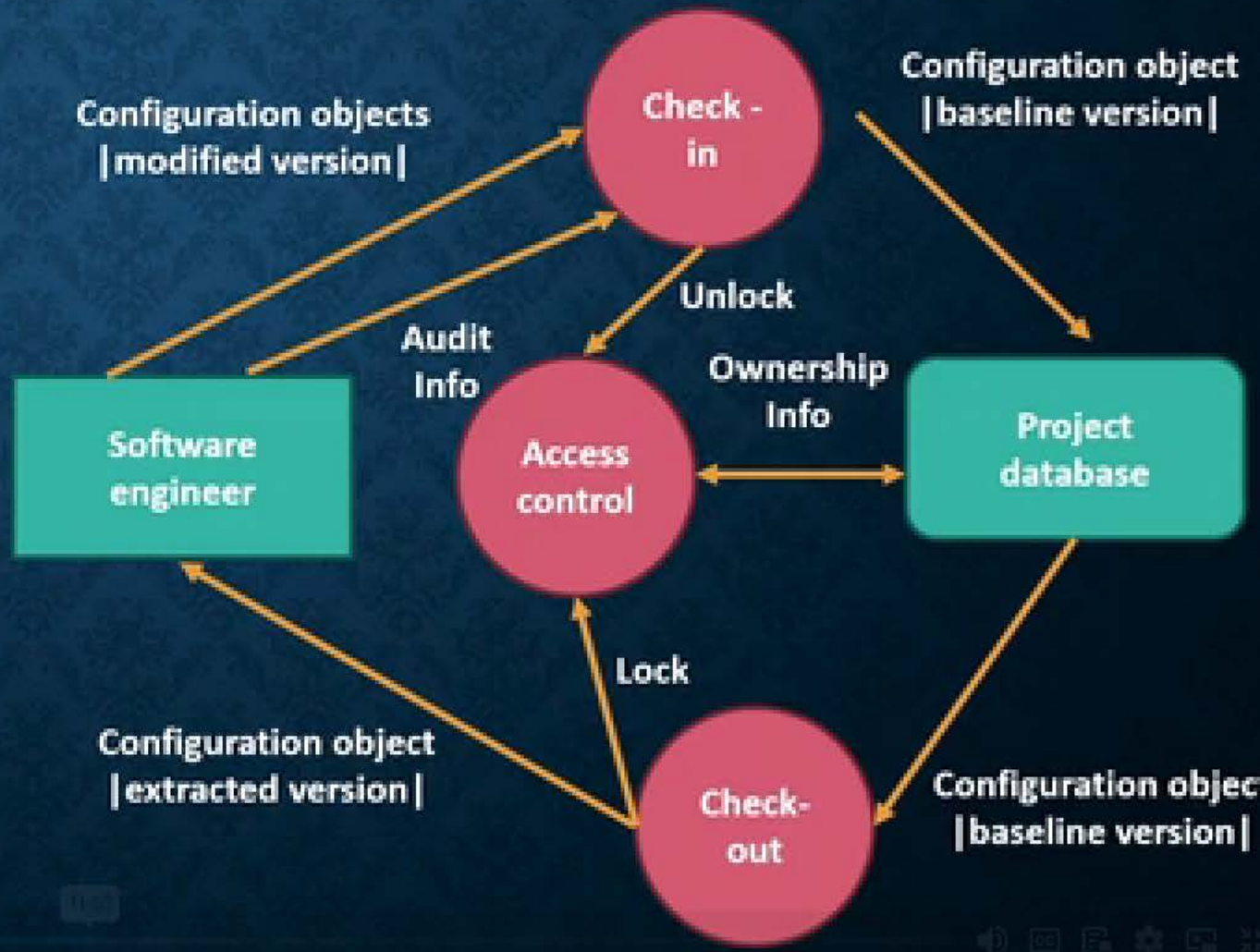
- Filtering changes
- Managing changes and the change process
- Reviewing and closing of Requests for Change (RFCs)

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)

TEKNOTURF

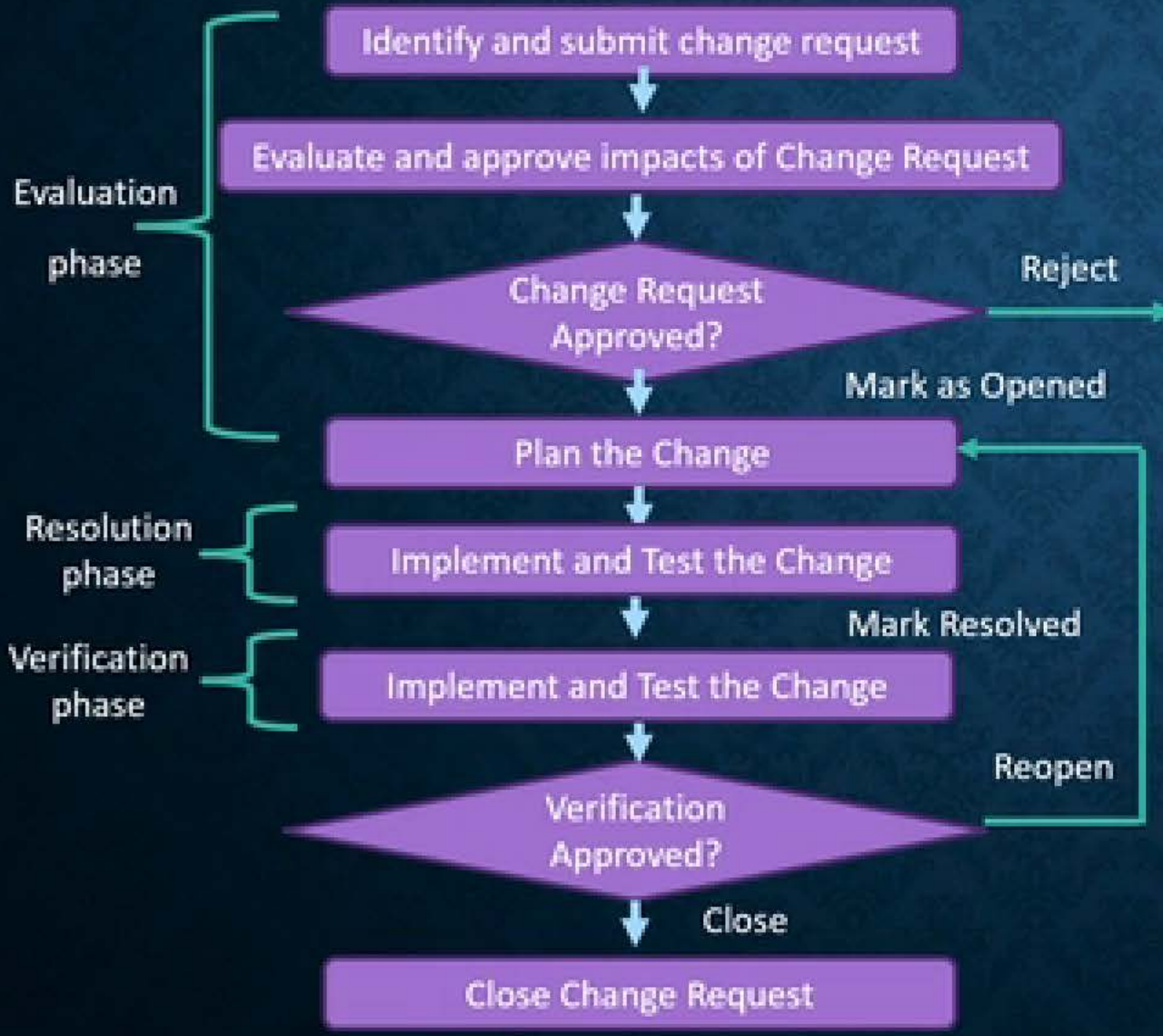
CHANGE CONTROL BOARD (CCB) OR SOFTWARE CHANGE CONTROL BOARD (SCCB)

- 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

TEKNOTURF



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

1. Rolls back to a previous version of a given file
2. Compares two versions of a file, highlighting differences
3. Creates branches that allow for parallel concurrent development
4. Maintains an instant audit trail on each and every file: versions, modified date, modifier, and any additional amount of meta-data your system provides for and whichever you choose

SCM TOOLS

TEKNOTURF

Features supported in any change management tool are

1. Concurrency management
2. Version control
3. Synchronization

SCM TOOLS

TEKNOTURF

RCS

- It is a local repository model
- Saving of revisions is independent of the central repository
- Allows revision of documents, committing changes and merging docs together.

CLEARCASE

- Provides controlled access to software assets, including code, requirements, design documents, models, test plans and test results
- Provides parallel development support, automated workspace management, baseline management, secure version management, reliable build auditing, and flexible access virtually anytime, anywhere.
- Supports storage of binary files and large repositories of files.

SUBVERSION

- Open Source Project

It is centralized (one server)

- Allows revert to the previous version to study and analyze the changes
- Allows concurrent development on same file

SOME SCM TERMS TO BE FAMILIAR WITH

TEKNOTURF

Term	Meaning
Repository (repo)	The database where the files are stored
Server	The system that stores the repository.
Client	The system that connects to the server(repo)
Working Set / Working Copy	Local directory of files, where changes are made.
Trunk / Main	The primary location for code in the repository.
Add	Put a file related to the project into the repo for the first time,

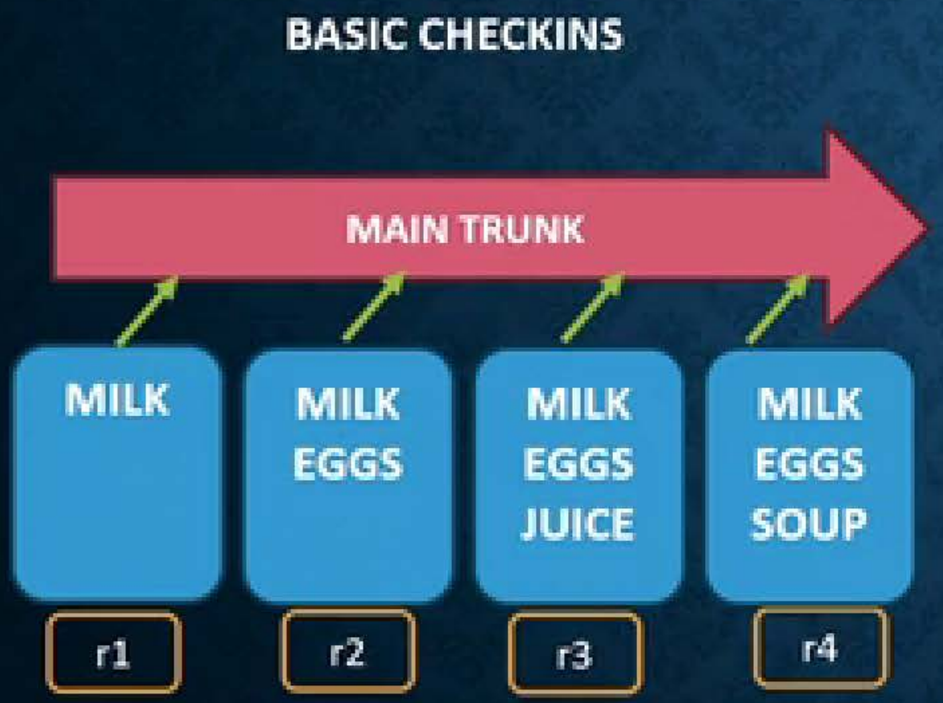
SOME SCM TERMS TO BE FAMILIAR WITH

TEKNOTURF

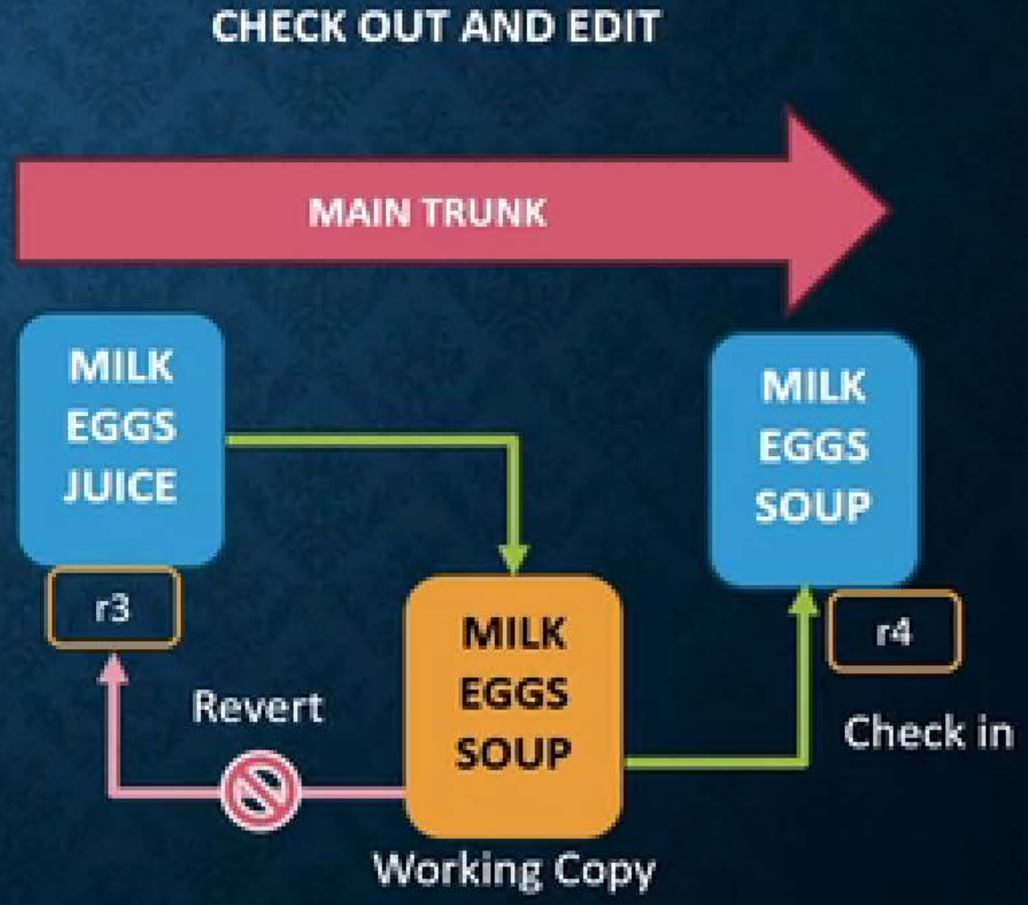
Term	Meaning
Revision	Depicts the version of a file (v1, v2, v3, etc.).
Head	The latest revision in the repo.
Check out	Download a file from the repo to work in local.
Check in	Upload a file to the repository after making necessary addition / modification. The file gets a new revision number, and people can "check out" the latest one.
Check in Message	A short message describing what was changed.

CHECK-IN AND CHECK-OUT

TEKNOTURF



Each time we check in a new version, we get a new revision (r1, r2, r3, etc.).



Check out, by default fetches the latest version

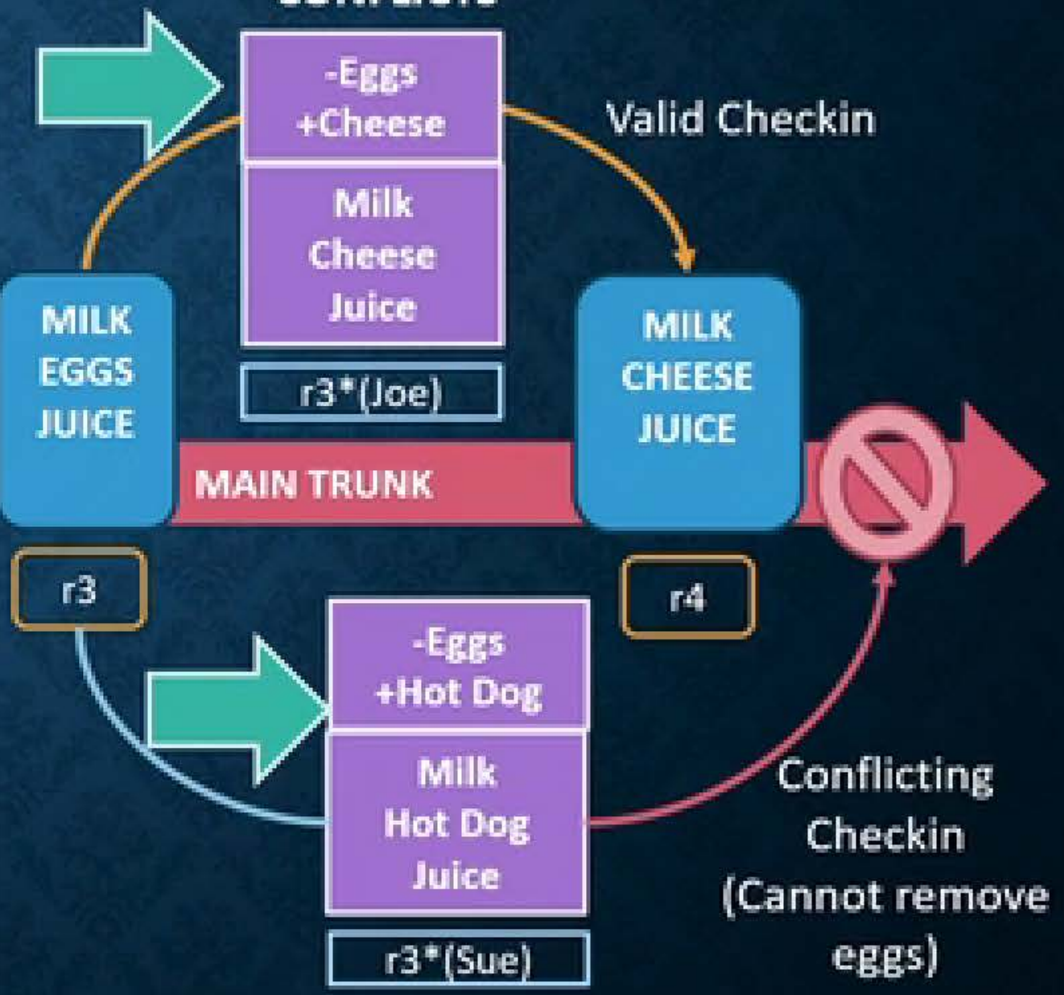
DIFF AND CONFLICT

TEKNOTURF

BASIC DIFFS

Changes made when editing the artifact

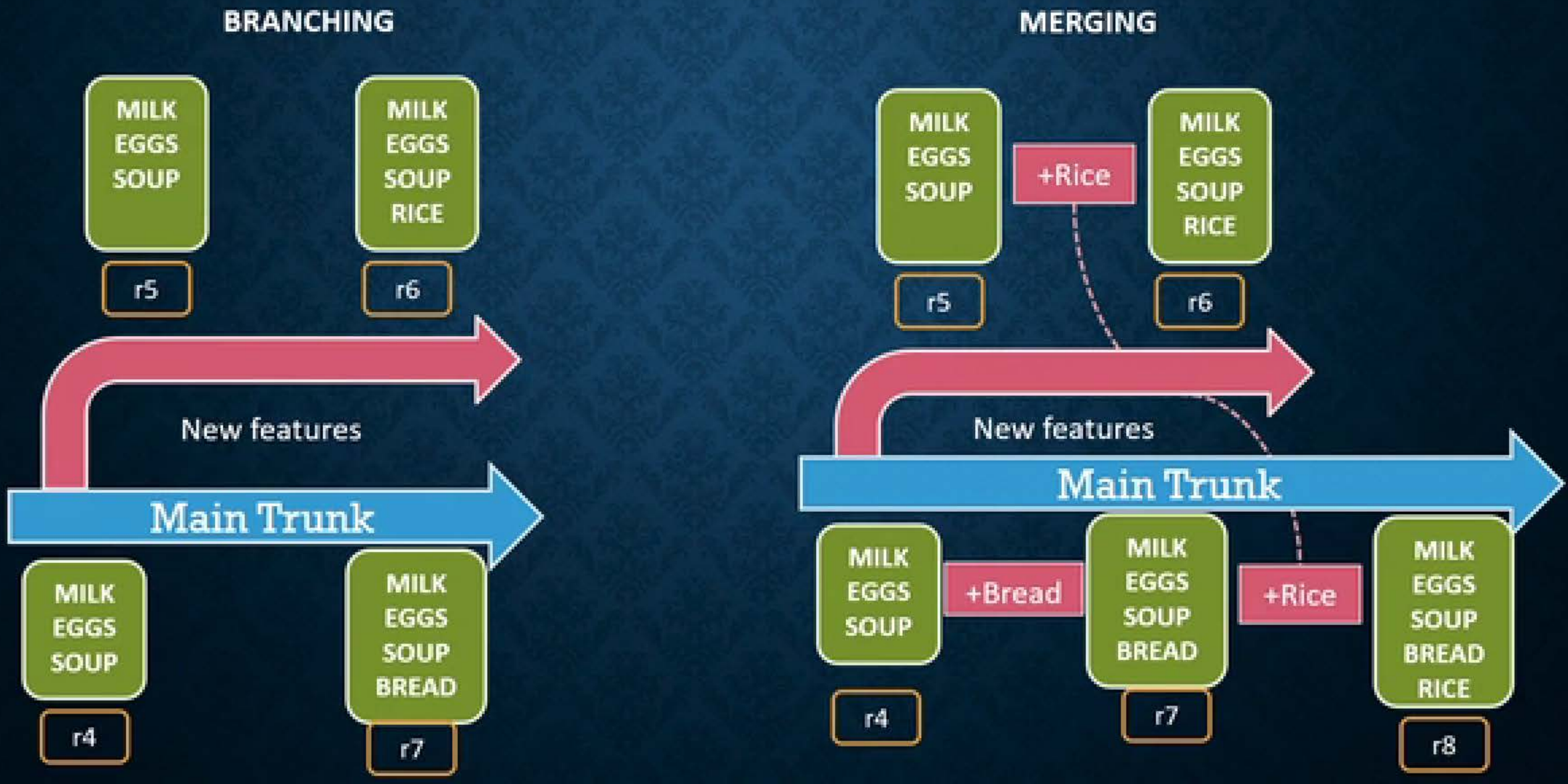
CONFLICTS



Conflicts occur when there is an overlap in the changes made

BRANCH AND MERGE

TEKNOTURF



SOFTWARE CONFIGURATION MANAGEMENT - ACTIVITIES

TEKNOTURF

Code is kept in a common repository

Developers pick up a working copy and work on their local machine

Changes are committed to the repository.

Check if anyone has checked in the same file in the repository

If yes compare the conflicts and merge the changes as per the requirement

Check in the new version of files into the repository

- Details of what, why, when and by whom changes were made to the software is also preserved
- Tracking the changes is a very simple task with SCM