Normalization and Denormalization

Definition of Normalization

Normalization is the method of arranging the data in the database efficiently. It involves constructing tables and setting up relationships between those tables according to some certain rules. The redundancy and inconsistent dependency can be removed using these rules in order to make it more flexible.

Redundant data wastes disk space, increases data inconsistency and slows down the DML queries. If the same data is present in more than one place and any updation is committed on that data, then the change must be reflected in all locations. Inconsistent data can make data searching and access harder by losing the path to it.

There are various reasons behind performing the normalization such as to avoid redundancy, updating anomalies, unnecessary coding, keeping the data into the form that can accommodate change more easily and accurately and to enforce the data constraint.

Normalization includes the analysis of functional dependencies between attributes. The relations (tables) are decomposed with anomalies to generate relations with a structure. It helps in deciding which attributes should be grouped in a relation.

The normalization is basically based on the concepts of normal forms. A relation table is said to be in a normal form if it fulfils a certain set of constraints. There are 6 defined normal forms: 1NF, 2NF, 3NF, BCNF, 4NF and 5NF. Normalization should eliminate the redundancy but not at the cost of integrity.

Normalization is a process of organizing the data in database to avoid data redundancy, insertion anomaly, update anomaly & deletion anomaly.

Anomalies in DBMS

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let's take an example to understand this.

Example:

Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee's id, emp_name for storing employee's

address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:

emp_id	emp_name	emp_address	emp_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

The above table is not normalized. We will see the problems that we face when a table is not normalized.

Update anomaly: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert anomaly: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

Delete anomaly: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

To overcome these anomalies we need to normalize the data. In the next section we will discuss about normalization.

Normalization and Denormalization

Types of Normalization

Here are the most commonly used normal forms:

- First normal form(1NF)
- Second normal form(2NF)
- Third normal form(3NF)
- Boyce & Codd normal form (BCNF)
- Forth Normal Form
- Fifth Normal Form

Primary Key –

A primary key is a column (or columns) in a table that uniquely identifies the rows in that table.

For example -

Employee(Employee_id, Employee_name, Department_id)
In this table *Emp_id* is the primary key.
The value placed in primary key columns must be unique for each row : no duplicates can be tolerated. In addition, nulls are not allowed in primary key columns.

Foreign Key –

Foreign keys are columns that point to primary key columns.

For example -

Employee(Employee_id, Employee_name, Department_id)
Department(Department_id, Department_name, Total_Employee)
Employee_id is the primary key of the table *Employee* and *Department_id* is the primary key of the table *Department* but in *Employee* table *Department_id* is the foreign key that points to the primary key in the *Department* table.

Er. Deepak Ranabhatt

Normalization and Denormalization

FIRST NORMAL FORM (1NF)

- There is no duplicate rows or tuples in the relation.
- Each data field can contain only one value.
- Entries in a column (attribute) are of the same kind (type).

Example -

Table Name : *Office*

Department_id	Department_name	Employee_id	Employee_name	Salary
D-100	PHP	E-1001 E-1005 E-1008	Mr. A Mr. B Mr. C	30000 28000 25000
D-200	JAVA	E-1002 E-1004 E-1006	Mr. X Mr. Y Mr. Z	25000 30000 20000
D-300	Admin	E-1003 E-1007	Mr. D Mr. E	20000 30000

Table "office" not in first normal form, because in this table many data field contain multiple values, so it is require to convert into first normal form.

Table Name : *Office*

Department_id	Department_name	Employee_id	Employee_name	Salary
D-100	PHP	E-1001	Mr. A	30000
D-100	PHP	E-1005	Mr. B	28000

Er. Deepak Ranabhatt

D-100	PHP	E-1008	Mr. C	25000
D-200	JAVA	E-1002	Mr. X	25000
D-200	JAVA	E-1004	Mr. Y	30000
D-200	JAVA	E-1006	Mr. Z	20000
D-300	Admin	E-1003	Mr. D	20000
D-300	Admin	E-1007	Mr. E	30000

SECOND NORMAL FORM (2NF)

A relation is in 2NF if it is in 1NF and every non-key attribute is fully dependent on each candidate key of the relation.
That means in second normal form each table have only one entity which uniquely identify other entities. This particular entity contain only primary key value. In another way we can say that if there is more than one primary key then the table is required to convert into second normal form.

Example -

The "Office" table which shown in First Normal Form is require to convert into Second Normal Form.

Functional Dependecy in "Office" Table
(Department_id, Employee_id) → (Department_name, Employee_name, Salary)

Partial Dependecy in "Office" Table
Department_id → Department_name
Employee_id → (Employee_name, Salary)

After 2NF the "Office" table is divided into two tables which are :

Normalization and Denormalization

Table Name : *Employee*

Employee_id	Employee_name	Salary	Department_id
E-1001	Mr. A	30000	D-100
E-1005	Mr. B	28000	D-100
E-1008	Mr. C	25000	D-100
E-1002	Mr. X	25000	D-200
E-1004	Mr. Y	30000	D-200
E-1006	Mr. Z	20000	D-200
E-1003	Mr. D	20000	D-300
E-1007	Mr. E	30000	D-300

Table Name : *Department*

Department_id	Department_name
D-100	PHP
D-200	JAVA
D-300	Admin

THIRD NORMAL FORM (3NF)

A relation is in third normal form if it is in 2NF and every non-key attribute of the relation is non-transitively dependent on each candidate key of the relation.

Er. Deepak Ranabhatt

Normalization and Denormalization

Example -
Library(Book_id, Book_name, Author_name, Bookshelf_number, Book_category)

Functional Dependecy
book_id → (Book_name, Author_name, Bookshelf_number, Book_category)

Transitive Dependecy
Bookshelf_number → Book_category

Table Name : *Library*

Book_id	Book_name	Author_name	Bookshelf_number	Book_category
B-100	Hacking Sectrets Exposed	Srikanth Ramesh	10	Hacking
B-105	The Complete Reference C++	Herbert Schildt	30	C and C++
B-200	Linux Shell Scripting with Bash	Ken O. Burtch	40	Linux
B-250	The Basics of Web Hacking	Josh Pauli	10	Hacking
B-350	Database System Concepts	Silberschatz Korth Sudarshan	20	DBMS
B-480	Shell Scripting	Steven Parker	40	Linux
B-610	The Complete Reference Java	Herbert Schildt	50	JAVA
B-750	Penetration Testing with the Bash Shell	Keith Makan	40	Linux

Table "Library" not in third normal form, because a transitive dependency present in this table.

Er. Deepak Ranabhatt

Normalization and Denormalization

Table Name : *Book*

Book_id	Book_name	Author_name	Bookshelf_number
B-100	Hacking Sectrets Exposed	Srikanth Ramesh	10
B-105	The Complete Reference C++	Herbert Schildt	30
B-200	Linux Shell Scripting with Bash	Ken O. Burtch	40
B-250	The Basics of Web Hacking	Josh Pauli	10
B-350	Database System Concepts	Silberschatz Korth Sudarshan	20
B-480	Shell Scripting	Steven Parker	40
B-610	The Complete Reference Java	Herbert Schildt	50
B-750	Penetration Testing with the Bash Shell	Keith Makan	40

Table Name : *Bookshelf*

Bookshelf_number	Book_category
10	Hacking
20	DBMS
30	C and C++

Er. Deepak Ranabhatt

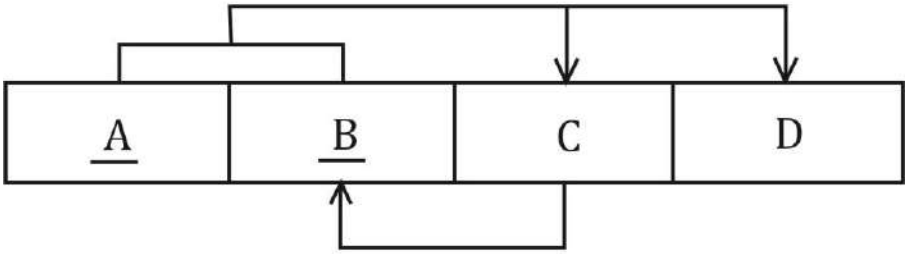
Normalization and Denormalization

40	Linux
50	JAVA

BOYCE CODE NORMAL FORM (BCNF)

A table is in BCNF when every determinant in the table is a candidate key. Clearly when a table contains only one candidate key the 3NF and the BCNF are equivalent. Putting that proposition another way, BCNF can be violated only when the table contains more than one candidate key.

The table is in 3NF but not in BCNF



Functional Dependencies

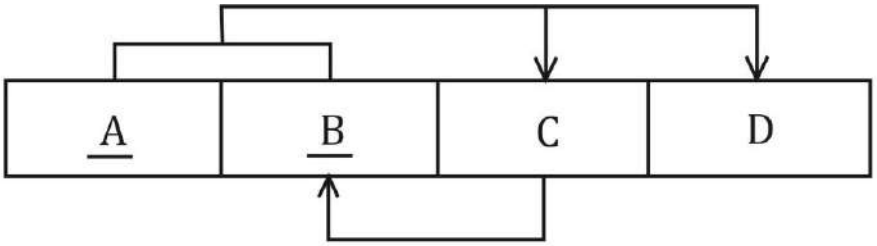
$(A, B) \rightarrow (C, D)$

$C \rightarrow B$

Notice that this structure has two candidate keys : (A,B) and (A,C). The table structure shown in above figure has no partial dependencies, nor does it contain transitive dependencies. (The condition C → B indicated that a non-key attribute determines part of the primary key - and that dependency is not transitive or partial because the dependent is a prime attribute!). Thus the table structure in the above figure is in 3NF but not in BCNF.

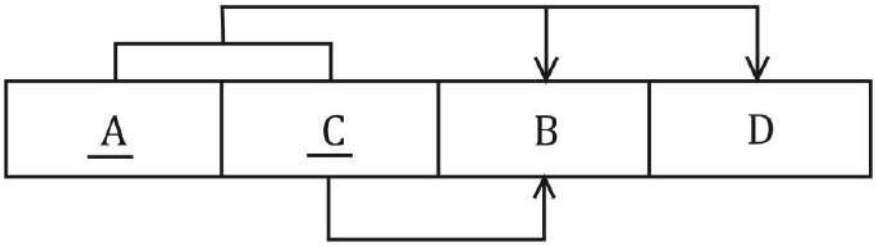
Normalization and Denormalization

Step 1



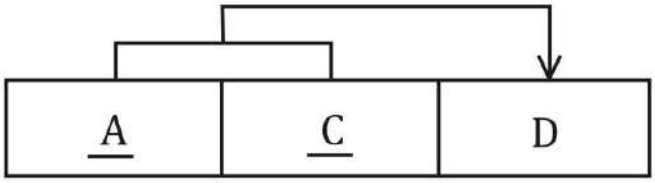
The table is in 3NF but not in BCNF

Step 2

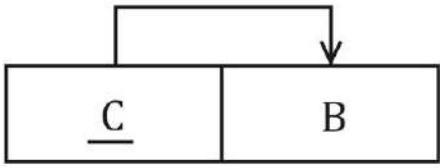


Partial Dependency

Step 3



3NF and BCNF



3NF and BCNF

Example -
Class_Test(Student_id, Professor_id, Class_code, Student_grade)

Functional Dependecy
(Student_id, Professor_id) → (Class_code, Student_grade)
Class_code → Professor_id>

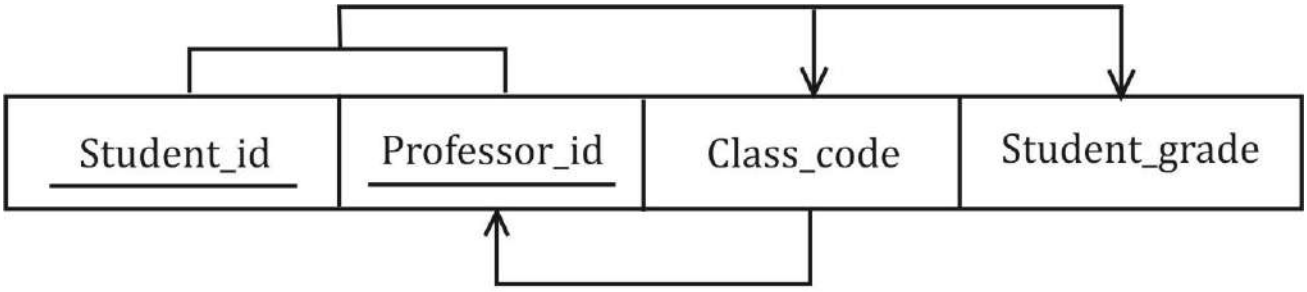
Table Name : *Class_Test*

Student_id	Professor_id	Class_code	Student_grade
S-10001	P-100	C-1001	A
S-10001	P-300	C-1002	B

Er. Deepak Ranabhatt

Normalization and Denormalization

S-10002	P-100	C-1001	C
S-10002	P-200	C-1003	C
S-10003	P-100	C-1005	A
S-10004	P-100	C-1001	B
S-10004	P-200	C-1003	A
S-10005	P-100	C-1004	A



The table is in 3NF but not in BCNF

The table reflects the following conditions :

- Each "Student_id" identifies a student uniquely.
- Each "Professor_id" identifies a professor uniquely.
- Each "Class_code" identifies a class uniquely.
- A student can take many classes. For example - Student "S-10001" attend both classes "C-1001" and "C-1002".
- A professor can teach many classes but each class is tought by only one professor. For example - Professor "P-100" teaches the classes "C-1001" and "C-1004". (So, the "Professor_id" can identify by the "Class_code" not vice versa.)
- If a new professor is assigned to teach class "C-1001", three rows will require updates, thus producing an update anomaly.

Er. Deepak Ranabhatt

Normalization and Denormalization

- If student "S-10003" drops class "C-1005", information about wqho tought class is lost, thus producing a deletion anomaly.

Table Name : *Grade*

Student_id	Class_code	Student_grade
S-10001	C-1001	A
S-10001	C-1002	B
S-10002	C-1001	C
S-10002	C-1003	C
S-10003	C-1005	A
S-10004	C-1001	B
S-10004	C-1003	A
S-10005	C-1004	A

Table Name : *Class*

Class_code	Professor_id
C-1001	P-100
C-1002	P-300
C-1003	P-200
C-1004	P-100
C-1005	P-100

Er. Deepak Ranabhatt

Normalization and Denormalization

FOURTH NORMAL FORM (4NF)

A table is in the fourth normal form (4NF) if:

- It is in BCNF.
- It does not have any independent multi-valued parts of the primary key.

Fourth Normal Form is related to Multi-value Dependency. Under fourth normal form, a record type should not contain two or more independent multi-value facts about an entity. In addition the record must satisfy third normal form.

A multi-value dependency exists when

- There are at least three attributes A, B and C in a relation.
- For each value of A there is a well-defined set of values for B, and a well-defined set of values for C.
- The set of values of B is independent of set C.

If a table in 4NF then -

- All attributes must be dependent on the primary key, but they must be independent of each other.
- No row may contain two or more multivalued facts about an entity.

Example -
Course(Course_code, Professor, Reference_book)

Multivalue Dependency
Course_code →→ *Professor*
Course_code →→ *Reference_book*

Table Name : *Course*

Course_code	Professor	Reference_book
C-100	Mr. X	The TCP/IP Guide (Charles M. Kozierok)

Er. Deepak Ranabhatt

Normalization and Denormalization

C-100	Mr. Y	The TCP/IP Guide (Charles M. Kozierok)
C-200	Mr. A	Linux Shell Scripting with Bash (Ken O. Burtch)
C-200	Mr. A	Shell Scripting (Steven Parker)
C-200	Mr. A	Penetration Testing with the Bash Shell (Keith Makan)
C-200	Mr. B	Linux Shell Scripting with Bash (Ken O. Burtch)
C-300	Mr. C	Hacking Sectrets Exposed (Srikanth Ramesh)
C-300	Mr. D	Hacking Sectrets Exposed (Srikanth Ramesh)
C-300	Mr. D	The Basics of Web Hacking (Josh Pauli)
C-400	Mr. E	Database System Concepts (Silberschatz Korth Sudarshan)

The table reflects the following conditions :

- A course can be tought by one or many professor but each professor can teach only one course. For example - Course "C-100" tought by the professors "Mr. X" and "Mr. Y". (So, *Course_code* → → *Professor*)
- A professor can refer one or many textbook for a particular course.

Er. Deepak Ranabhatt

Normalization and Denormalization

- A textbook can be refer by one or many professor allocated for a particular course.
- Textbooks refer for a particular course can not be refer for another course. (So, *Course_code* → → *Reference_book*)

Table Name : *Instructor*

Course_code	Professor
C-100	Mr. X
C-100	Mr. Y
C-200	Mr. A
C-200	Mr. B
C-300	Mr. C
C-300	Mr. D
C-400	Mr. E

Table Name : *Textbook*

Course_code	Reference_book
C-100	The TCP/IP Guide (Charles M. Kozierok)
C-200	Linux Shell Scripting with Bash (Ken O. Burtch)
C-200	Shell Scripting (Steven Parker)

Er. Deepak Ranabhatt

Normalization and Denormalization

C-200	Penetration Testing with the Bash Shell (Keith Makan)
C-300	Hacking Sectrets Exposed (Srikanth Ramesh)
C-300	The Basics of Web Hacking (Josh Pauli)
C-400	Database System Concepts (Silberschatz Korth Sudarshan)

FIFTH NORMAL FORM (5NF)

A table is in the fifth normal form (5NF), if:

- It is in the fourth normal form.
- Every join dependency in the table is implied by the candidate keys.

A table is in Fifth Normal Form (5NF) or Project-Join Normal Form (PJNF) if it is in 4NF and it can not have a lossless decomposition into any number of smaller tables.
The fifth normal form deals with join-dependencies, which is a generalisation of the multi-value dependency.

Table Name : *Project_details*

Employee_id	Project_name	programming_Language
E-100	Google	Laravel
E-100	Facebook	PHP

Er. Deepak Ranabhatt

Normalization and Denormalization

E-200	Facebook	PHP
E-200	Google	CakePHP
E-200	Yahoo	CakePHP
E-300	Yahoo	Laravel

Table Name : *Project*

Employee_id	Project_name
E-100	Google
E-100	Facebook
E-200	Facebook
E-200	Google
E-200	Yahoo
E-300	Yahoo

Table Name : *Language*

Employee_id	programming_Language
E-100	Laravel
E-100	PHP
E-200	PHP
E-200	CakePHP

Er. Deepak Ranabhatt

Normalization and Denormalization

E-300	Laravel

Table Name : *Project_Language*

Project_name	programming_Language
Google	Laravel
Facebook	PHP
Google	CakePHP
Yahoo	CakePHP
Yahoo	Laravel

BASIS FOR COMPARISON	NORMALIZATION	DENORMALIZATION
Basic	Normalization is the process of creating a set schema to store non-redundant and consistent data.	Denormalization is the process of combining the data so that it can be queried speedily.
Purpose	To reduce the data redundancy and inconsistency.	To achieve the faster execution of the queries through introducing redundancy.
Used in	OLTP system, where the emphasize is on making the insert, delete and update anomalies faster	OLAP system, where the emphasis is on making the search and analysis faster.

Er. Deepak Ranabhatt

Normalization and Denormalization

BASIS FOR COMPARISON	NORMALIZATION	DENORMALIZATION
	and storing the quality data.	
Data integrity	Maintained	May not retain
Redundancy	Eliminated	Added
Number of tables	Increases	Decreases
Disk space	Optimized usage	Wastage

S.NO	NORMALIZATION	DENORMALIZATION
1.	In normalization, Non-redundancy and consistency data are stored in set schema.	In denormalization, data are combined to execute the query quickly.
2.	In normalization, Data redundancy and inconsistency is reduced.	In denormalization, redundancy is added for quick execution of queries.
3.	Data integrity is maintained in normalization.	Data integrity is not maintained in denormalization.

Er. Deepak Ranabhatt

Normalization and Denormalization

4.	In normalization, redundancy is reduced or eliminated.	In denormalization redundancy is added instead of reduction or elimination of redundancy.
5.	Number of tables in normalization is increased.	Denormalization, Number of tables in decreased.
6.	Normalization optimize the uses of disk spaces.	Denormalization do not optimize the disk spaces.

Er. Deepak Ranabhatt

Chapter 1
Software Development Life Cycle

SDLC is a systematic process for building software that ensures the quality and correctness of the software built. SDLC process aims to produce high-quality software that meets customer expectations. The system development should be complete in the pre-defined time frame and cost. SDLC consists of a detailed plan which explains how to plan, build, and maintain specific software. Every phase of the SDLC life cycle has its own process and deliverables that feed into the next phase. SDLC stands for Software Development Lifecycle.

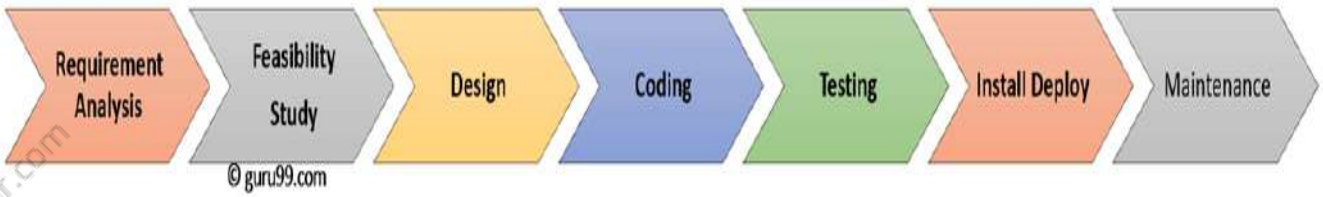
Why SDLC?

Here, are prime reasons why SDLC is important for developing a software system.

- It offers a basis for project planning, scheduling, and estimating
- Provides a framework for a standard set of activities and deliverables
- It is a mechanism for project tracking and control
- Increases visibility of project planning to all involved stakeholders of the development process
- Increased and enhance development speed
- Improved client relations
- Helps you to decrease project risk and project management plan overhead

SDLC Phases

The entire SDLC process divided into the following stages:



- Phase 1: Requirement collection and analysis
- Phase 2: Feasibility study:
- Phase 3: Design:

Er. Deepak Ranabhatt

Chapter 1
Software Development Life Cycle

- Phase 4: Coding:
- Phase 5: Testing:
- Phase 6: Installation/Deployment:
- Phase 7: Maintenance:

In this tutorial, I have explained all these phases

Phase 1: Requirement collection and analysis:

The requirement is the first stage in the SDLC process. It is conducted by the senior team members with inputs from all the stakeholders and domain experts in the industry. Planning for the quality assurance requirements and recognization of the risks involved is also done at this stage.

This stage gives a clearer picture of the scope of the entire project and the anticipated issues, opportunities, and directives which triggered the project.

Requirements Gathering stage need teams to get detailed and precise requirements. This helps companies to finalize the necessary timeline to finish the work of that system.

Phase 2: Feasibility study:

Once the requirement analysis phase is completed the next step is to define and document software needs. This process conducted with the help of 'Software Requirement Specification' document also known as 'SRS' document. It includes everything which should be designed and developed during the project life cycle.

There are mainly five types of feasibilities checks:

- Economic: Can we complete the project within the budget or not?
- Legal: Can we handle this project as cyber law and other regulatory framework/compliances.
- Operation feasibility: Can we create operations which is expected by the client?

- Technical: Need to check whether the current computer system can support the software
- Schedule: Decide that the project can be completed within the given schedule or not.

Phase 3: Design:

In this third phase, the system and software design documents are prepared as per the requirement specification document. This helps define overall system architecture.

This design phase serves as input for the next phase of the model.

There are two kinds of design documents developed in this phase:

High-Level Design (HLD)

- Brief description and name of each module
- An outline about the functionality of every module
- Interface relationship and dependencies between modules
- Database tables identified along with their key elements
- Complete architecture diagrams along with technology details

Low-Level Design(LLD)

- Functional logic of the modules
- Database tables, which include type and size
- Complete detail of the interface
- Addresses all types of dependency issues
- Listing of error messages
- Complete input and outputs for every module

Phase 4: Coding:

Once the system design phase is over, the next phase is coding. In this phase, developers start build the entire system by writing code using the chosen programming language. In the coding phase, tasks are divided into units or modules

and assigned to the various developers. It is the longest phase of the Software Development Life Cycle process.

In this phase, Developer needs to follow certain predefined coding guidelines. They also need to use programming tools like compiler, interpreters, debugger to generate and implement the code.

Phase 5: Testing:

Once the software is complete, and it is deployed in the testing environment. The testing team starts testing the functionality of the entire system. This is done to verify that the entire application works according to the customer requirement.

During this phase, QA and testing team may find some bugs/defects which they communicate to developers. The development team fixes the bug and send back to QA for a re-test. This process continues until the software is bug-free, stable, and working according to the business needs of that system.

Phase 6: Installation/Deployment:

Once the software testing phase is over and no bugs or errors left in the system then the final deployment process starts. Based on the feedback given by the project manager, the final software is released and checked for deployment issues if any.

Phase 7: Maintenance:

Once the system is deployed, and customers start using the developed system, following 3 activities occur

- Bug fixing - bugs are reported because of some scenarios which are not tested at all
- Upgrade - Upgrading the application to the newer versions of the Software
- Enhancement - Adding some new features into the existing software

The main focus of this SDLC phase is to ensure that needs continue to be met and that the system continues to perform as per the specification mentioned in the first phase.

Chapter 1

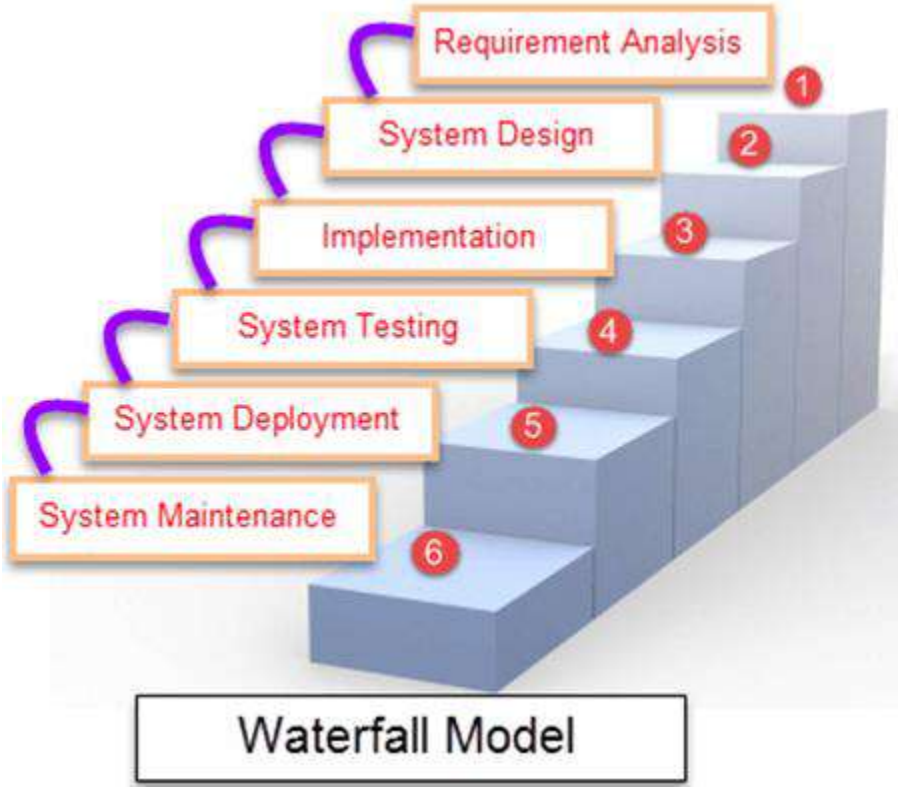
Software Development Life Cycle

Software Development Life Cycle Models

A software life cycle model is a descriptive representation of the software development cycle. SDLC models might have a different approach but the basic phases and activity remain the same for all the models.

1) Waterfall Model

Waterfall Model is a sequential model that divides software development into different phases. Each phase is designed for performing specific activity during SDLC phase. It was introduced in 1970 by Winston Royce.



Er. Deepak Ranabhatt

Chapter 1
Software Development Life Cycle

Different Phases of Waterfall Model in Software Engineering

Different phases	Activities performed in each stage
Requirement Gathering stage	• During this phase, detailed requirements of the software system to be developed are gathered from client
Design Stage	• Plan the programming language, for Example Java, PHP, .net • or database like Oracle, MySQL, etc. • Or other high-level technical details of the project
Built Stage	• After design stage, it is built stage, that is nothing but coding the software
Test Stage	• In this phase, you test the software to verify that it is built as per the specifications given by the client.
Deployment stage	• Deploy the application in the respective environment
Maintenance stage	• Once your system is ready to use, you may later require change the code as per customer request

Waterfall model can be used when

- Requirements are not changing frequently
- Application is not complicated and big

Er. Deepak Ranabhatt

Chapter 1
Software Development Life Cycle

- Project is short
- Requirement is clear
- Environment is stable
- Technology and tools used are not dynamic and is stable
- Resources are available and trained

Advantages and Disadvantages of Waterfall-Model

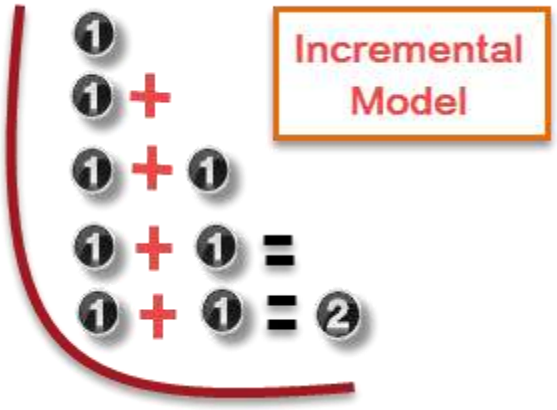
Advantages	Dis-Advantages
• Before the next phase of development, each phase must be completed	• Error can be fixed only during the phase
• Suited for smaller projects where requirements are well defined	• It is not desirable for complex project where requirement changes frequently
• They should perform quality assurance test (Verification and Validation) before completing each stage	• Testing period comes quite late in the developmental process
• Elaborate documentation is done at every phase of the software's development cycle	• Documentation occupies a lot of time of developers and testers

Er. Deepak Ranabhatt

- Project is completely dependent on project team with minimum client intervention
- Clients valuable feedback cannot be included with ongoing development phase
- Any changes in software is made during the process of the development
- Small changes or errors that arise in the completed software may cause a lot of problems

2. Incremental Modal

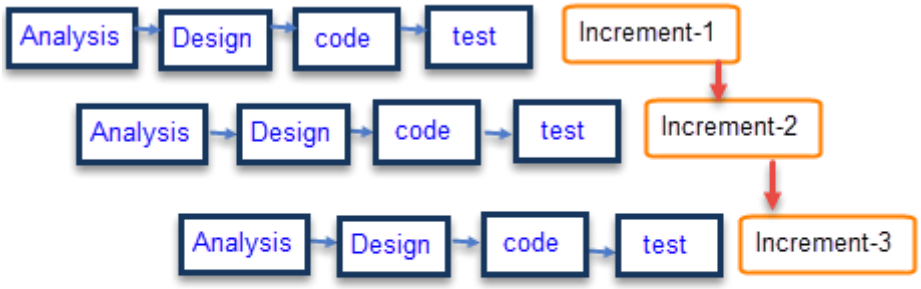
Incremental Model is a process of software development where requirements are broken down into multiple standalone modules of software development cycle. Incremental development is done in steps from analysis design, implementation, testing/verification, maintenance.



Each iteration passes through the requirements, design, coding and testing phases. And each subsequent release of the system adds function to the previous release until all designed functionality has been implemented.

Chapter 1

Software Development Life Cycle



Incremental Model

The system is put into production when the first increment is delivered. The first increment is often a core product where the basic requirements are addressed, and supplementary features are added in the next increments. Once the core product is analyzed by the client, there is plan development for the next increment.

Characteristics of an Incremental module includes

- System development is broken down into many mini development projects
- Partial systems are successively built to produce a final total system
- Highest priority requirement is tackled first
- Once the requirement is developed, requirement for that increment are frozen

Incremental Phases	Activities performed in incremental phases
Requirement Analysis	• Requirement and specification of the software are collected
Design	• Some high-end function are designed during this stage

Er. Deepak Ranabhatt

Code	• Coding of software is done during this stage
Test	• Once the system is deployed, it goes through the testing phase

When to use Incremental models?

- Requirements of the system are clearly understood
- When demand for an early release of a product arises
- When software engineering team are not very well skilled or trained
- When high-risk features and goals are involved
- Such methodology is more in use for web application and product based companies

Advantages and Disadvantages of Incremental Model

Advantages	Disadvantages
• The software will be generated quickly during the software life cycle	• It requires a good planning designing
• It is flexible and less expensive to change requirements and scope	• Problems might cause due to system architecture as such not all requirements collected up front for the entire software lifecycle
• Throughout the development stages changes can be done	• Each iteration phase is rigid and does not overlap each other

• This model is less costly compared to others	• Rectifying a problem in one unit requires correction in all the units and consumes a lot of time
• A customer can respond to each building	
• Errors are easy to be identified	

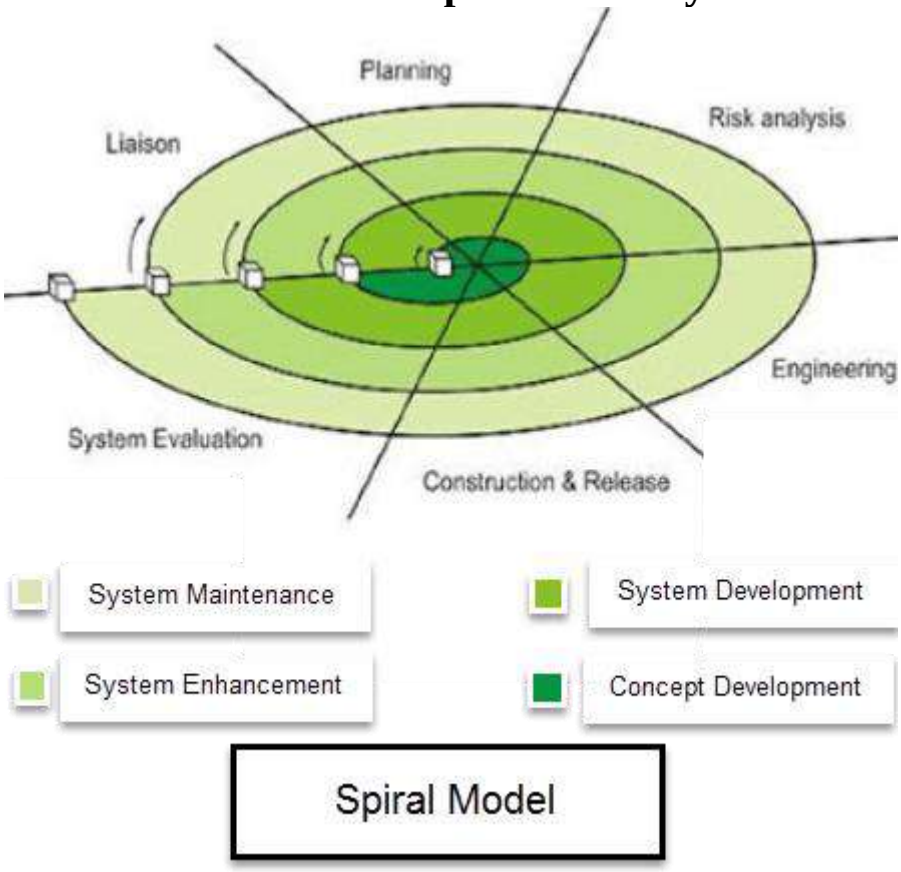
3. Spiral Model

Spiral Model is a combination of a waterfall model and iterative model. Each phase in spiral model begins with a design goal and ends with the client reviewing the progress. The spiral model was first mentioned by Barry Boehm in his 1986 paper.

The development team in Spiral-SDLC model starts with a small set of requirement and goes through each development phase for those set of requirements. The software engineering team adds functionality for the additional requirement in every-increasing spirals until the application is ready for the production phase.

Chapter 1

Software Development Life Cycle



Spiral Model Phases

Spiral Model Phases	Activities performed during phase
Planning	• It includes estimating the cost, schedule and resources for the iteration. It also involves understanding the system requirements for continuous communication between the system analyst and the customer
Risk Analysis	• Identification of potential risk is done while risk mitigation strategy is planned and finalized

Er. Deepak Ranabhatt

Chapter 1

Software Development Life Cycle

Engineering	• It includes testing, coding and deploying software at the customer site
Evaluation	• Evaluation of software by the customer. Also, includes identifying and monitoring risks such as schedule slippage and cost overrun

When to use Spiral Methodology?

- When project is large
- When releases are required to be frequent
- When creation of a prototype is applicable
- When risk and costs evaluation is important
- For medium to high-risk projects
- When requirements are unclear and complex
- When changes may require at any time
- When long term project commitment is not feasible due to changes in economic priorities

Advantages and Disadvantages of Spiral Model

Advantages	Disadvantages
• Additional functionality or changes can be done at a later stage	• Risk of not meeting the schedule or budget
• Cost estimation becomes easy as the prototype building is done in small fragments	• It works best for large projects only also demands risk assessment expertise

Er. Deepak Ranabhatt

• Continuous or repeated development helps in risk management	• For its smooth operation spiral model protocol needs to be followed strictly
• Development is fast and features are added in a systematic way	• Documentation is more as it has intermediate phases
• There is always a space for customer feedback	• It is not advisable for smaller project, it might cost them a lot

4. Rapid Application Development

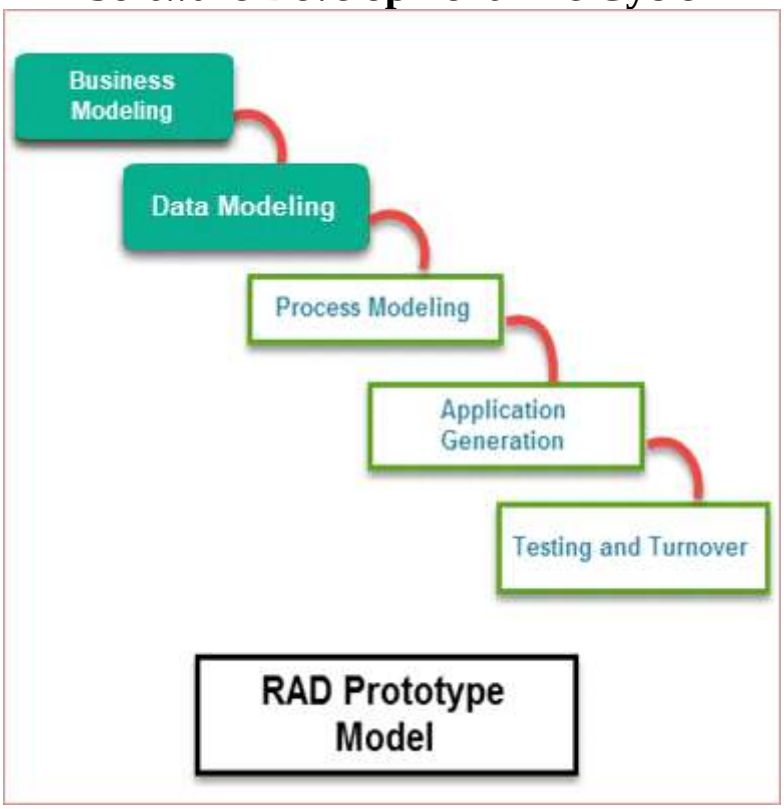
RAD or Rapid Application Development process is an adoption of the waterfall model; it targets at developing software in a short span of time. RAD follow the iterative

SDLC RAD model has following phases

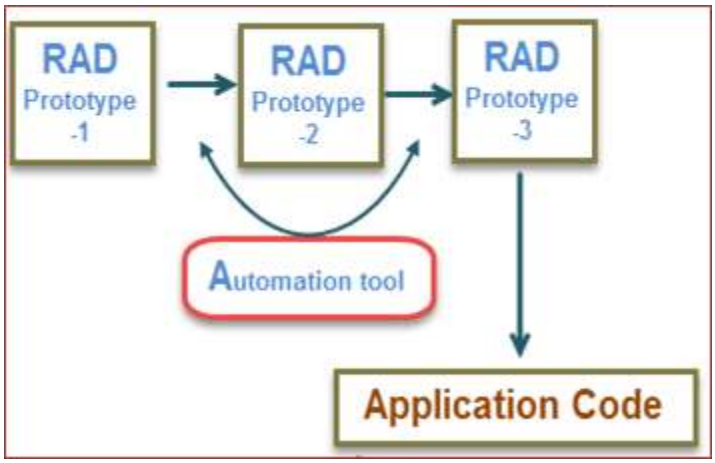
- Business Modeling
- Data Modeling
- Process Modeling
- Application Generation
- Testing and Turnover

Chapter 1

Software Development Life Cycle



It focuses on input-output source and destination of the information. It emphasizes on delivering projects in small pieces; the larger projects are divided into a series of smaller projects. The main features of RAD model are that it focuses on the reuse of templates, tools, processes, and code.



RAD Model in Software Engineering

Er. Deepak Ranabhatt

Chapter 1

Software Development Life Cycle

Different phases of RAD model includes

Phases of RAD model	Activities performed in RAD Model
Business Modeling	• On basis of the flow of information and distribution between various business channels, the product is designed
Data Modeling	• The information collected from business modeling is refined into a set of data objects that are significant for the business
Process Modeling	• The data object that is declared in the data modeling phase is transformed to achieve the information flow necessary to implement a business function
Application Generation	• Automated tools are used for the construction of the software, to convert process and data models into prototypes
Testing and Turnover	• As prototypes are individually tested during every iteration, the overall testing time is reduced in RAD.

When to use RAD Methodology?

- When a system needs to be produced in a short span of time (2-3 months)
- When the requirements are known
- When the user will be involved all through the life cycle
- When technical risk is less
- When there is a necessity to create a system that can be modularized in 2-3 months of time

Er. Deepak Ranabhatt

- When a budget is high enough to afford designers for modeling along with the cost of automated tools for code generation

Advantages and Disadvantages of SDLC RAD Model

Advantages	Disadvantages
• Flexible and adaptable to changes	• It can't be used for smaller projects
• It is useful when you have to reduce the overall project risk	• Not all application is compatible with RAD
• It is adaptable and flexible to changes	• When technical risk is high, it is not suitable
• It is easier to transfer deliverables as scripts, high-level abstractions and intermediate codes are used	• If developers are not committed to delivering software on time, RAD projects can fail
• Due to code generators and code reuse, there is a reduction of manual coding	• Reduced features due to time boxing, where features are pushed to a later version to finish a release in short period
• Due to prototyping in nature, there is a possibility of lesser defects	• Reduced scalability occurs because a RAD developed application begins as a prototype

	and evolves into a finished application
• Each phase in RAD delivers highest priority functionality to client	• Progress and problems accustomed are hard to track as such there is no documentation to demonstrate what has been done
• With less people, productivity can be increased in short time	• Requires highly skilled designers or developers

5. Prototype Model

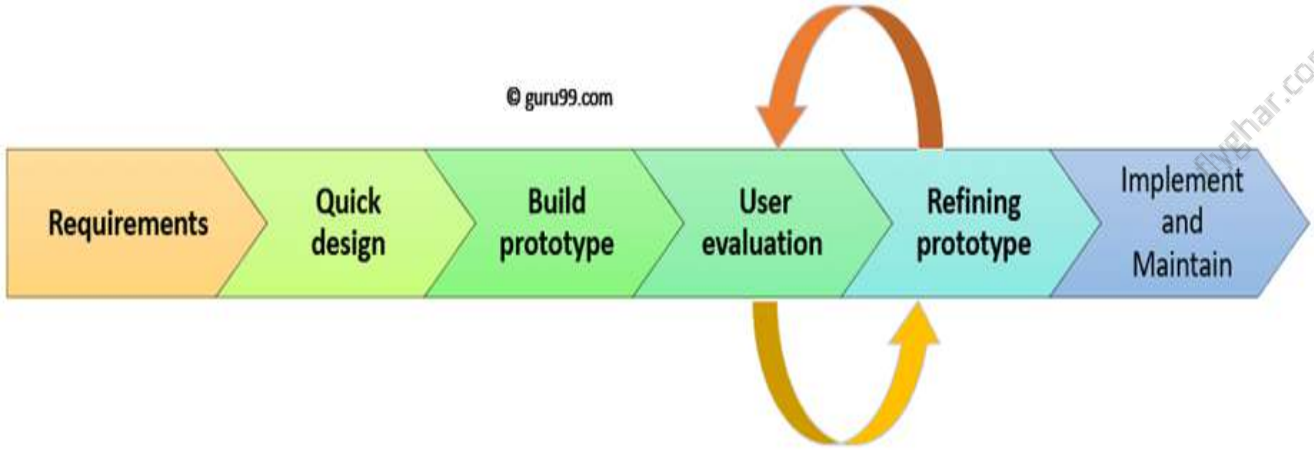
Prototype methodology is defined as a Software Development model in which a prototype is built, test, and then reworked when needed until an acceptable prototype is achieved. It also creates a base to produce the final system.

Software prototyping model works best in scenarios where the project's requirement are not known. It is an iterative, trial, and error method which take place between the developer and the client.

Chapter 1

Software Development Life Cycle

Prototyping Model Phases



Prototyping Model has following six SDLC phases as follow:

Step 1: Requirements gathering and analysis

A prototyping model starts with requirement analysis. In this phase, the requirements of the system are defined in detail. During the process, the users of the system are interviewed to know what is their expectation from the system.

Step 2: Quick design

The second phase is a preliminary design or a quick design. In this stage, a simple design of the system is created. However, it is not a complete design. It gives a brief idea of the system to the user. The quick design helps in developing the prototype.

Step 3: Build a Prototype

In this phase, an actual prototype is designed based on the information gathered from quick design. It is a small working model of the required system.

Step 4: Initial user evaluation

In this stage, the proposed system is presented to the client for an initial evaluation. It helps to find out the strength and weakness of the working model. Comment and suggestion are collected from the customer and provided to the developer.

Er. Deepak Ranabhatt

Chapter 1
Software Development Life Cycle

Step 5: Refining prototype

If the user is not happy with the current prototype, you need to refine the prototype according to the user's feedback and suggestions.

This phase will not over until all the requirements specified by the user are met. Once the user is satisfied with the developed prototype, a final system is developed based on the approved final prototype.

Step 6: Implement Product and Maintain

Once the final system is developed based on the final prototype, it is thoroughly tested and deployed to production. The system undergoes routine maintenance for minimizing downtime and prevent large-scale failures.

Types of Prototyping Models

Four types of Prototyping models are:

1. Rapid Throwaway prototypes
2. Evolutionary prototype
3. Incremental prototype
4. Extreme prototype

Advantages of the Prototyping Model

Here, are important pros/benefits of using Prototyping models:

- Users are actively involved in development. Therefore, errors can be detected in the initial stage of the software development process.
- Missing functionality can be identified, which helps to reduce the risk of failure as Prototyping is also considered as a risk reduction activity.
- Helps team member to communicate effectively
- Customer satisfaction exists because the customer can feel the product at a very early stage.
- There will be hardly any chance of software rejection.

- Quicker user feedback helps you to achieve better software development solutions.

Disadvantages of the Prototyping Model

Here, are important cons/drawbacks of prototyping model:

- Prototyping is a slow and time taking process.
- The cost of developing a prototype is a total waste as the prototype is ultimately thrown away.
- Prototyping may encourage excessive change requests.
- Some times customers may not be willing to participate in the iteration cycle for the longer time duration.
- There may be far too many variations in software requirements when each time the prototype is evaluated by the customer.
- Poor documentation because the requirements of the customers are changing.
- It is very difficult for software developers to accommodate all the changes demanded by the clients.
- After seeing an early prototype model, the customers may think that the actual product will be delivered to him soon.

Agile Methodology

Agile software development methodology involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing. Compressing these five sequences of the conventional software development methods, for one three-week cycle (iterations). At the end of each iteration a working product is available.

Principles of Agile Methodology

- To satisfy the customer through early and continuous delivery of valuable software is the highest priority of Agile Software.

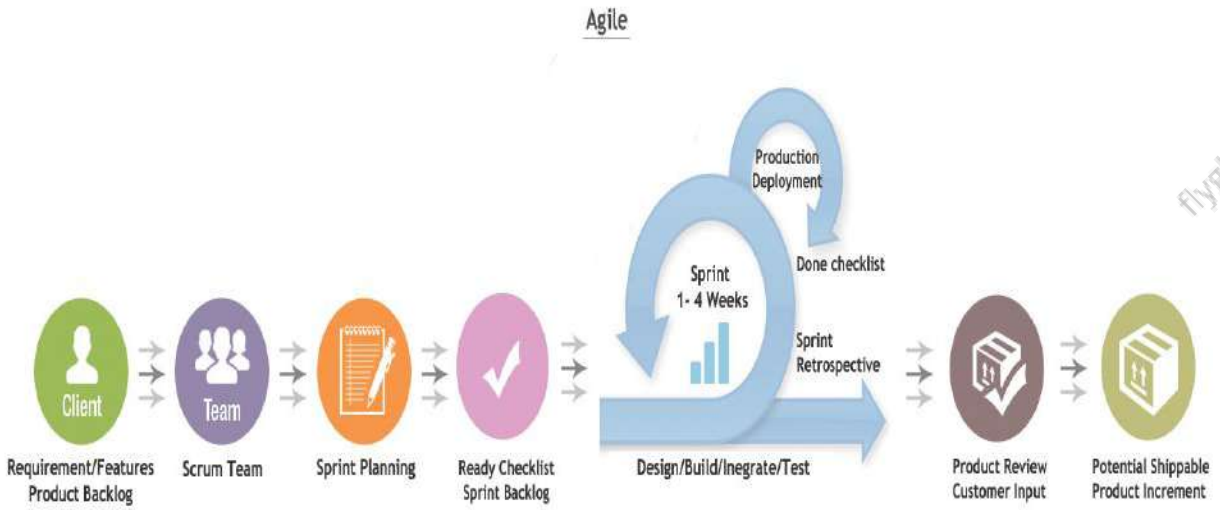
Chapter 1

Software Development Life Cycle

- With a preference to the shorter timescale, from a couple of weeks to a couple of months, delivering working software frequently.
- For the customer's competitive advantage, Agile processes harness change, welcoming changing requirements, even late in development.
- Working together of developers and business people, daily throughout the project.
- Trust the motivated individuals to get the job done, giving them the environment and support they need, by building projects around motivated individuals.
- Face to face conversation within a development team by the most effective and efficient method of conveying information.
- The primary measure of progress in Working Software.
- To be able to maintain a constant pace indefinitely by the developers, users and sponsors sustainable development is promoted by Agile processes.
- Good Design and Technical excellence enhance agility by continuous attention.
- The self-organizing teams give the best architectures, requirements, and designs.
- The team adjusts its behavior and tunes accordingly at regular intervals, reflecting on how to become more effective.

Er. Deepak Ranabhatt

Chapter 1
Software Development Life Cycle



Advantages of Agile model:

1. Continuous Improvement: To improve the future iteration, throughout the whole project, agile encourages feedback from users and team members.

2. Change is Embraced: As the planning cycles are shorter, it is easy to accept changes and accommodate at any point of time, throughout the project.

3. End-Goal can be Unknown: For that kind of projects where end-goal is not defined, Agile is very beneficial. The goals will come to light as the project processes.

4. Faster, High-Quality Delivery: The team focuses on high-quality development, collaboration, and testing, by breaking down the project into manageable units. The bugs get identified and solve more quickly by conducting testing during each iteration.

Er. Deepak Ranabhatt

5. Strong Team Interaction: To take responsibility and own parts of the project, Agile highlights the importance of team working together with frequent communication, and face-to-face interaction.

6. Customers are Heard: By working very closely with the project team, customers can gain a sense of ownership and have a real impact on the end product by getting many opportunities to see the work being delivered and share their input.

Disadvantages of Agile model:

- In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.
- There is lack of emphasis on necessary designing and documentation.
- The project can easily get taken off track if the customer representative is not clear what final outcome that they want.
- Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources.

When to use Agile model:

- When new changes need to be implemented. The freedom agile gives to change is very important. New changes can be implemented at very little cost because of the frequency of new increments that are produced.
- To implement a new feature the developers need to lose only the work of a few days, or even only hours, to roll back and implement it.
- Unlike the waterfall model in agile model very limited planning is required to get started with the project. Agile assumes that the end users' needs are ever changing in a dynamic business and IT world. Changes can be discussed and features can be newly effected or removed based on feedback. This effectively gives the customer the finished system they want or need.
- Both system developers and stakeholders alike, find they also get more freedom of time and options than if the software was developed in a more rigid sequential way. Having options gives them the ability to leave important

decisions until more or better data or even entire hosting programs are available; meaning the project can continue to move forward without fear of reaching a sudden standstill.

CASE Tools

CASE stands for **C**omputer **A**ided **S**oftware **E**ngineering. It means, development and maintenance of software projects with help of various automated software tools. CASE tools are set of software application programs, which are used to automate SDLC activities. CASE tools are used by software project managers, analysts and engineers to develop software system.

There are number of CASE tools available to simplify various stages of Software Development Life Cycle such as Analysis tools, Design tools, Project management tools, Database Management tools, Documentation tools are to name a few.

Use of CASE tools accelerates the development of project to produce desired result and helps to uncover flaws before moving ahead with next stage in software development.

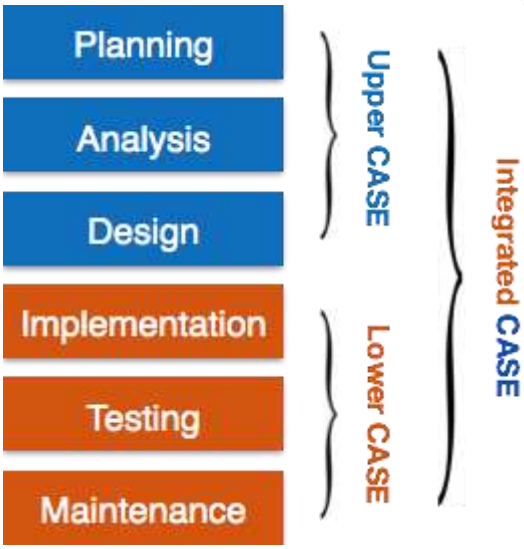
Components of CASE Tools

CASE tools can be broadly divided into the following parts based on their use at a particular SDLC stage:

- **Central Repository** - CASE tools require a central repository, which can serve as a source of common, integrated and consistent information. Central repository is a central place of storage where product specifications, requirement documents, related reports and diagrams, other useful information regarding management is stored. Central repository also serves as data dictionary.

Chapter 1

Software Development Life Cycle



- **Upper Case Tools** - Upper CASE tools are used in planning, analysis and design stages of SDLC.
- **Lower Case Tools** - Lower CASE tools are used in implementation, testing and maintenance.
- **Integrated Case Tools** - Integrated CASE tools are helpful in all the stages of SDLC, from Requirement gathering to Testing and documentation.

CASE tools can be grouped together if they have similar functionality, process activities and capability of getting integrated with other tools.

Scope of Case Tools

The scope of CASE tools goes throughout the SDLC.

Case Tools Types

Diagram tools

Process Modeling Tools

Project Management Tools

Documentation Tools

Configuration Management Tools

Er. Deepak Ranabhatt

Chapter 1

Software Development Life Cycle

Reasons for using CASE tools

· Savings in resources required for software development — with less.

· Quick development phase

· Reduction of generation of errors.

· Easier recognition of bugs during development.

· Savings in maintenance resources required.

· To increase productivity

· To help produce enhanced quality software at lower cost

Benefits of CASE Tools

Several benefits increase from the use of a CASE environment or even isolated CASE tools. Some of those pros are:

• An imperative benefit of a CASE environment is cost saving through all development stages.

• Use of CASE tools leads to considerable improvement to quality.

• Different phases of Software Development and the chances of human error are significantly reduced.

Er. Deepak Ranabhatt

Chapter 1

Software Development Life Cycle

• CASE tools help to produce high quality and consistent documents

• CASE tools take out most of the work in a software engineer's work.

Chapter 5
Implementation and Maintenance

What is Software Testing?

Software testing is a process, to evaluate the functionality of a software application with an intent to find whether the developed software met the specified requirements or not and to identify the defects to ensure that the product is defect free in order to produce the quality product. A process of analyzing a software item to detect the differences between existing and required conditions (i.e., defects) and to evaluate the features of the software item.

Some of the reasons why software testing becomes very significant and integral part in the field of information technology are as follows.

1. Cost effectiveness
2. Customer Satisfaction
3. Security
4. Product Quality

1. Cost effectiveness

As a matter of fact design defects can never be completely ruled out for any complex system. It is not because developers are careless but because the complexity of a system is intractable. If the design issues go undetected, then it will become more difficult to trace back defects and rectify it. It will become more expensive to fix it. Sometimes, while fixing one bug we may introduce another one in some other module unknowingly. If the bugs can be identified in early stages of development then it costs much less to fix them. That is why it is important to find defects in the early stages of the software development life cycle. One of the benefits of testing is cost effectiveness.

It is better to start testing earlier and introduce it in every phase of software development life cycle and regular testing is needed to ensure that the application is developed as per the requirement.

2. Customer Satisfaction

In any business, the ultimate goal is to give the best customer satisfaction. Yes, customer satisfaction is very important. Software testing improves the user

Chapter 5
Implementation and Maintenance

experience of an application and gives satisfaction to the customers. Happy customers means more revenue for a business. One of the reasons why software testing is necessary is to provide the best user experience.

3. Security

This is probably the most sensitive and vulnerable part of software testing. Testing (penetration testing & security testing) helps in product security. Hackers gain unauthorized access to data. These hackers steal user information and use it for their benefit. If your product is not secured, users won't prefer your product. Users always look for trusted products. Testing helps in removing vulnerabilities in the product.

4. Product Quality

Software Testing is an art which helps in strengthening the market reputation of a company by delivering the quality product to the client as mentioned in the requirement specification documents.

Due to these reasons software testing becomes very significant and integral part of Software Development process.

Principles of Software Testing:

Testing of software consist of some principles that play a vital role while testing the project.

The Principles of Software Testing are as follows:

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

Chapter 5
Implementation and Maintenance

Software Testing Types:

1. Manual Testing:

Manual testing is the process of testing software by hand to learn more about it, to find what is and isn't working. This usually includes verifying all the features specified in requirements documents, but often also includes the testers trying the software with the perspective of their end user's in mind. Manual test plans vary from fully scripted test cases, giving testers detailed steps and expected results, through to high-level guides that steer exploratory testing sessions. There are lots of sophisticated tools on the market to help with manual testing, but if you want a simple and flexible place to start, take a look at Testpad.

2. Automation Testing:

Automation testing is the process of testing the software using an automation tool to find the defects. In this process, testers execute the test scripts and generate the test results automatically by using automation tools. Some of the famous automation testing tools for functional testing are QTP/UFT and Selenium.

Software Testing Methods:

1. Static Testing:

It is also known as Verification in Software Testing. Verification is a static method of checking documents and files. Verification is the process, to ensure that whether we are building the product right i.e., to verify the requirements which we have and to verify whether we are developing the product accordingly or not.

Activities involved here are Inspections, Reviews, Walkthroughs

2. Dynamic Testing:

It is also known as Validation in Software Testing. Validation is a dynamic process of testing the real product. Validation is the process, whether we are building the right product i.e., to validate the product which we have developed is right or not.

Activities involved in this is Testing the software application

Chapter 5
Implementation and Maintenance

Testing Approaches:

There are three types of software testing approaches.

1. White Box Testing
2. Black Box Testing
3. Grey Box Testing

1. White Box Testing:

It is also called as Glass Box, Clear Box, and Structural Testing. White Box Testing is based on applications internal code structure. In white-box testing, an internal perspective of the system, as well as programming skills, are used to design test cases. This testing is usually done at the unit level.

Advantages:

1. White box testing is very thorough as the entire code and structures are tested.
2. It results in the optimization of code removing error and helps in removing extra lines of code.
3. It can start at an earlier stage as it doesn't require any interface as in case of black box testing.
4. Easy to automate.

Disadvantages:

1. Main disadvantage is that it is very expensive.
2. Redesign of code and rewriting code needs test cases to be written again.
3. Testers are required to have in-depth knowledge of the code and programming language as opposed to black box testing.
4. Missing functionalities cannot be detected as the code that exists is tested.
5. Very complex and at times not realistic.

Chapter 5
Implementation and Maintenance

2. Black Box Testing:

It is also called as Behavioral/Specification-Based/Input-Output Testing. Black Box Testing is a software testing method in which testers evaluate the functionality of the software under test without looking at the internal code structure.

Types of Black Box Testing:

1. Functionality Testing
2. Non-functionality Testing

Functional testing:

In simple words, what the system actually does is functional testing. To verify that each function of the software application behaves as specified in the requirement document. Testing all the functionalities by providing appropriate input to verify whether the actual output is matching the expected output or not. It falls within the scope of black box testing and the testers need not concern about the source code of the application.

Non-functional testing:

In simple words, how well the system performs is non-functionality testing. Non-functional testing refers to various aspects of the software such as performance, load, stress, scalability, security, compatibility etc., Main focus is to improve the user experience on how fast the system responds to a request.

Grey Box Testing:

Grey box is the combination of both White Box and Black Box Testing. The tester who works on this type of testing needs to have access to design documents. This helps to create better test cases in this process.

Chapter 5
Implementation and Maintenance

BLACK BOX TESTING	WHITE BOX TESTING
It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.	It is a way of testing the software in which the tester has knowledge about the internal structure r the code or the program of the software.
It is mostly done by software testers.	It is mostly done by software developers.
No knowledge of implementation is needed.	Knowledge of implementation is required.
It can be referred as outer or external software testing.	It is the inner or the internal software testing.
It is functional test of the software.	It is structural test of the software.
This testing can be initiated on the basis of requirement specifications document.	This type of testing of software is started after detail design document.
No knowledge of programming is required.	It is mandatory to have knowledge of programming.
It is the behavior testing of the software.	It is the logic testing of the software.
It is applicable to the higher levels of testing of software.	It is generally applicable to the lower levels of software testing.

Chapter 5
Implementation and Maintenance

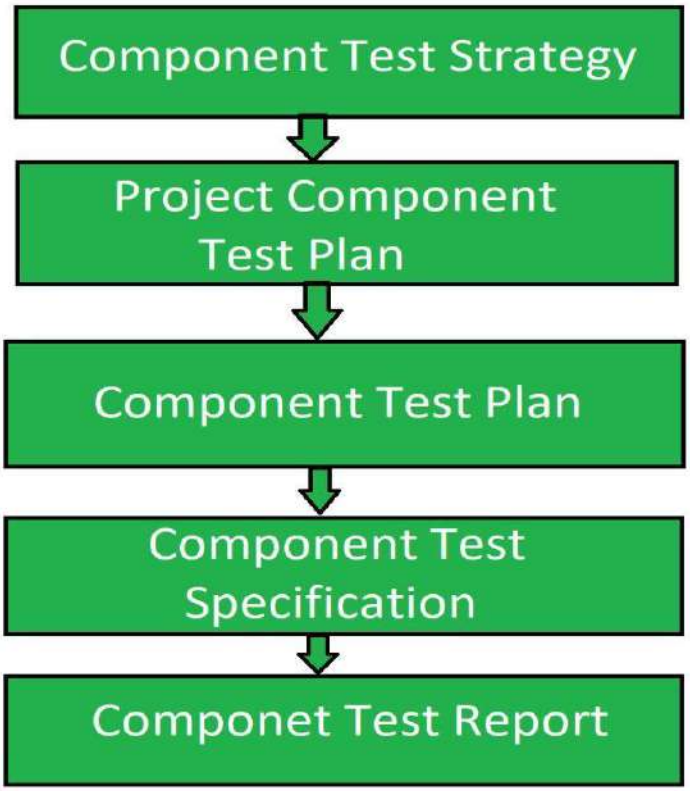
BLACK BOX TESTING	WHITE BOX TESTING
It is also called closed testing.	It is also called as clear box testing.
It is least time consuming.	It is most time consuming.
It is not suitable or preferred for algorithm testing.	It is suitable for algorithm testing.
Can be done by trial and error ways and methods.	Data domains along with inner or internal boundaries can be better tested.
Example: search something on google by using keywords	**Example:** by input to check and verify loops
Types of Black Box Testing: • A. Functional Testing • B. Non-functional testing • C. Regression Testing	**Types of White Box Testing:** • A. Path Testing • B. Loop Testing • C. Condition testing

Chapter 5
Implementation and Maintenance

Types of Testing:

Component Testing:

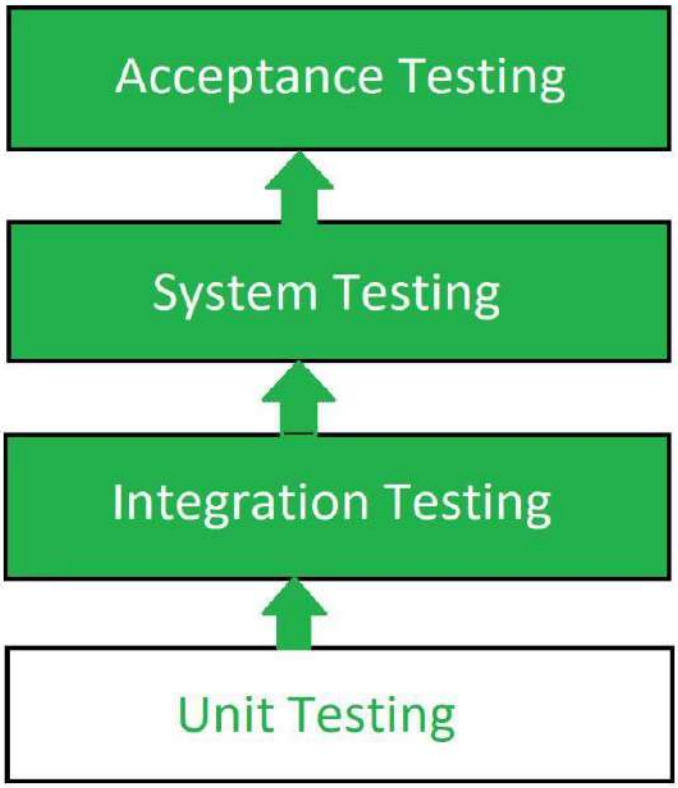
Component Testing is a type of software testing in which usability of each individual component is tested. Along with the usability test, behavioral evaluation is also done for each individual component. To perform this type of testing, each component needs to be in independent state and also should be in controllable state. Each component of the software should be user comprehensible.



Unit Testing:

Unit Testing is a type of software testing in which individual units of software i.e. group of computer program modules, usage procedures and operating procedures are tested to determine whether they are suitable for use or not. It is a testing method

using which every independent modules are tested to determine if there are any issue by the developer himself. It is correlated with functional correctness of the independent modules.



Difference between Component and Unit Testing:

COMPONENT TESTING	UNIT TESTING
Component Testing involves testing of each object or parts of the software separately.	Unit Testing involves testing of individual programs or modules for program execution.
It is performed by the testing team.	It is performed by the development team.

Chapter 5
Implementation and Maintenance

COMPONENT TESTING	UNIT TESTING
Component testing is a black box testing.	Unit testing is a white box testing.
Tester doesn't know the internal architecture of the software.	Tester knows the internal design of the software.
Component testing is performed once the unit testing is performed.	Unit testing is performed before the component testing.
Detection of defects is little difficult as compared to unit testing.	Detection of defects is easy in unit testing.
Once the whole software is developed then only component testing is performed.	Unit testing is performed after every development step.
It validates test requirements.	It validates design documents.

System Testing

The System Testing verifies the behavior of the system entirely. It performs various tests in progression. However, these tests have the distinct intent and check whether all the system components are working correctly in an integrated way or not.

Chapter 5
Implementation and Maintenance

In this testing, the test cases are designed according to the requirement specification, and its code is said to be similar to the black box. The developers having the broad knowledge and visibility about the structure of the system usually performs the system testing.

Types of system testing:

There are several forms of system testing among which few of them are describes below.

- **Recovery Testing**: This type of test make the software to decline in several ways for checking the proper recovery process.
- **Security Testing**: The security testing verifies the security mechanism and prevents the system from the penetration.
- **Stress Testing**: It is a testing technique in which the abnormal conditions of resources like quantity, frequency and volume are required.
- **Performance Testing**: Performance testing focuses on the software's runtime performance in relevance to the whole system.

Alpha Testing

It is a type of software testing performed to identify bugs before releasing the product to real users or to the public. Alpha Testing is one of the user acceptance testing.

Beta Testing

It is performed by real users of the software application in a real environment. Beta testing is one of the type of User Acceptance Testing.

ALPHA TESTING	BETA TESTING
Alpha testing involves both the white box and black box testing.	Beta testing commonly uses black box testing.

Chapter 5
Implementation and Maintenance

ALPHA TESTING	BETA TESTING
Alpha testing is performed by testers who are usually internal employees of the organization.	Beta testing is performed by clients who are not part of the organization.
Alpha testing is performed at developer's site.	Beta testing is performed at end-user of the product.
Reliability and security testing are not checked in alpha testing.	Reliability, security and robustness are checked during beta testing.
Alpha testing ensures the quality of the product before forwarding to beta testing.	Beta testing also concentrates on the quality of the product but collects users input on the product and ensures that the product is ready for real time users.
Alpha testing requires a testing environment or a lab.	Beta testing doesn't require a testing environment or lab.
Alpha testing may require long execution cycle.	Beta testing requires only a few weeks of execution.
Developers can immediately address the	Most of the issues or feedback collected from beta testing will

Chapter 5
Implementation and Maintenance

ALPHA TESTING	BETA TESTING
critical issues or fixes in alpha testing.	be implemented in future versions of the product.

System Installation

Implementation is a process of ensuring that the information system is operational. It involves −

- Constructing a new system from scratch
- Constructing a new system from the existing one.

Implementation allows the users to take over its operation for use and evaluation. It involves training the users to handle the system and plan for a smooth conversion.

Training

The personnel in the system must know in detail what their roles will be, how they can use the system, and what the system will or will not do. The success or failure of welldesigned and technically elegant systems can depend on the way they are operated and used.

1. Training Systems Operators

Systems operators must be trained properly such that they can handle all possible operations, both routine and extraordinary. The operators should be trained in what common malfunctions may occur, how to recognize them, and what steps to take when they come.

Training involves creating troubleshooting lists to identify possible problems and remedies for them, as well as the names and telephone numbers of individuals to contact when unexpected or unusual problems arise.

Training also involves familiarization with run procedures, which involves working through the sequence of activities needed to use a new system.

2. User Training

Chapter 5
Implementation and Maintenance

- End-user training is an important part of the computer-based information system development, which must be provided to employees to enable them to do their own problem solving.
- User training involves how to operate the equipment, troubleshooting the system problem, determining whether a problem that arose is caused by the equipment or software.
- Most user training deals with the operation of the system itself. The training courses must be designed to help the user with fast mobilization for the organization.

3. Training Guidelines

- Establishing measurable objectives
- Using appropriate training methods
- Selecting suitable training sites
- Employing understandable training materials

Training Methods

1. Instructor-led training

It involves both trainers and trainees, who have to meet at the same time, but not necessarily at the same place. The training session could be one-on-one or collaborative. It is of two types −

2. Virtual Classroom

In this training, trainers must meet the trainees at the same time, but are not required to be at the same place. The primary tools used here are: video conferencing, text based Internet relay chat tools, or virtual reality packages, etc.

3. Normal Classroom

The trainers must meet the trainees at the same time and at the same place. They primary tools used here are blackboard, overhead projectors, LCD projector, etc.

Chapter 5
Implementation and Maintenance

4. Self-Paced Training

It involves both trainers and trainees, who do not need to meet at the same place or at the same time. The trainees learn the skills themselves by accessing the courses at their own convenience. It is of two types −

5. Multimedia Training

In this training, courses are presented in multimedia format and stored on CD-ROM. It minimizes the cost in developing an in-house training course without assistance from external programmers.

6. Web-based Training

In this training, courses are often presented in hyper media format and developed to support internet and intranet. It provides just–in-time training for end users and allow organization to tailor training requirements.

Conversion

It is a process of migrating from the old system to the new one. It provides understandable and structured approach to improve the communication between management and project team.

Conversion Plan

It contains description of all the activities that must occur during implementation of the new system and put it into operation. It anticipates possible problems and solutions to deal with them.

It includes the following activities −

- Name all files for conversions.
- Identifying the data requirements to develop new files during conversion.
- Listing all the new documents and procedures that are required.
- Identifying the controls to be used in each activity.
- Identifying the responsibility of person for each activity.

Chapter 5
Implementation and Maintenance

- Verifying conversion schedules.

Conversion Methods

The four methods of conversion are −

- Parallel Conversion
- Direct Cutover Conversion
- Pilot Approach
- Phase-In Method

Method	Description	Advantages	Disadvantages
Parallel Conversion	Old and new systems are used simultaneously.	Provides fallback when new system fails. Offers greatest security and ultimately testing of new system.	Causes cost overruns. New system may not get fair trail.
Direct Cutover Conversion	New system is implemented and old system is replaced completely.	Forces users to make new system work Immediate benefit from new methods and control.	No fall back if problems arise with new system Requires most careful planning

Chapter 5
Implementation and Maintenance

Pilot Approach	Supports phased approach that gradually implement system across all users	Allows training and installation without unnecessary use of resources. Avoid large contingencies from risk management.	A long term phasein causes a problem of whether conversion goes well or not.
Phase-In Method	Working version of system implemented in one part of organization based on feedback, it is installed throughout the organization all alone or stage by stage.	Provides experience and line test before implementation When preferred new system involves new technology or drastic changes in performance.	Gives impression that old system is erroneous and it is not reliable.

1. Direct Conversion:

Direct conversion is the implementation of the new system and the immediate discontinuance of the old system.

This conversion is possible when:

(a) The system is not replacing any other system.

(b) The old system is judged absolutely without value.

(c) The new system is either very small or simple and

Chapter 5
Implementation and Maintenance

(d) The design of Ike new system is completely different from that of the old system.

2. Parallel conversion:

It is an approach wherein both the old and the new system operate simultaneously for some time. The outputs from both the systems are compared and difference is reconciled. The advantage of this conversion is that it gives a high degree of protection to the organization from the failure in the new system and has gained a wide spread popularity.

The disadvantage will be the costs associated with duplicating facilities and the personnel to maintain the dual systems. This conversion is opposite of direct conversion.

In parallel conversion a target data should be set to indicate when this conversion can be withdrawn and the new system will operate on its own. If the differences occur between the old and new systems, it should be verified with the same inputs to make sure of the transaction.

3. Modular Conversion:

The implementation of the new system takes place section by section. For example, it first can be installed in one sales region and if proved successful, the same can be installed in the second sales region and so on.

The advantages of this conversion are:

(a) The risk of system's failure can be localized.

(b) The problems occurred in the system can be rectified before further implementation.

(c) Other operating personnel can be trained before the implementation at their location.

The disadvantage is that this conversion is that the period of the conversion in very lengthy and is not feasible for every system or organization.

4. Phase-In Conversion:

Chapter 5
Implementation and Maintenance

Phase-In conversion is similar to the Modular Approach but here the system itself is segmented and not the organization as in Modular Approach. The new data collection activities are implemented and an interface mechanism with the old system is developed. This interface allows the old system to operate with the new input data. Like this, separate segments are installed until entire system in implemented.

The advantage of this is that the rate of change in a given organization can be minimized and the data processing resources can be acquired gradually over an extended period of time. The disadvantage is that the costs are incurred to develop temporary interfaces with old systems.

System Maintenance

Software maintenance is widely accepted part of SDLC now a days. It stands for all the modifications and updations done after the delivery of software product. There are number of reasons, why modifications are required, some of them are briefly mentioned below:

- Market Conditions - Policies, which changes over the time, such as taxation and newly introduced constraints like, how to maintain bookkeeping, may trigger need for modification.
- Client Requirements - Over the time, customer may ask for new features or functions in the software.
- Host Modifications - If any of the hardware and/or platform (such as operating system) of the target host changes, software changes are needed to keep adaptability.
- Organization Changes - If there is any business level change at client end, such as reduction of organization strength, acquiring another company, organization venturing into new business, need to modify in the original software may arise.

Chapter 5
Implementation and Maintenance

Types of maintenance

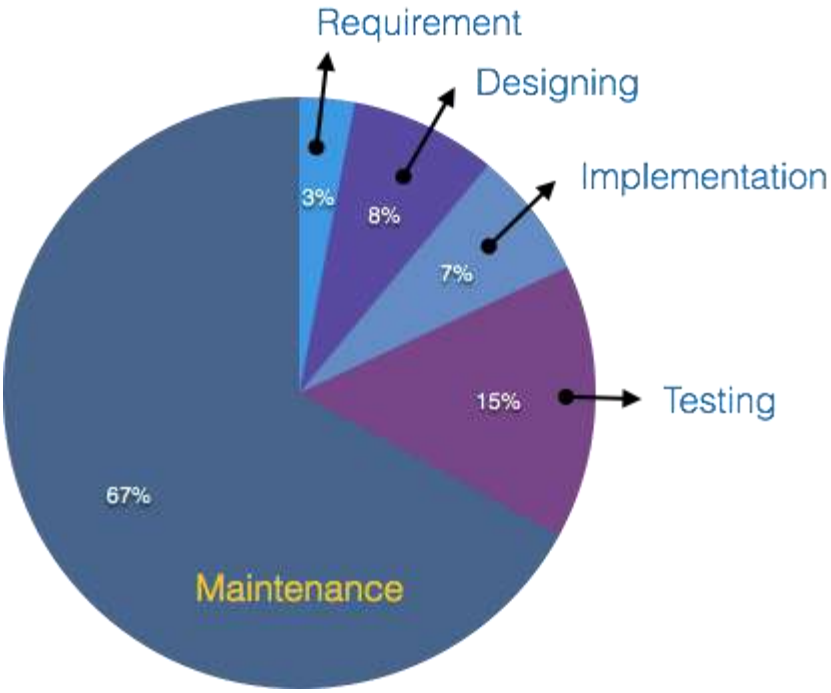
In a software lifetime, type of maintenance may vary based on its nature. It may be just a routine maintenance tasks as some bug discovered by some user or it may be a large event in itself based on maintenance size or nature. Following are some types of maintenance based on their characteristics:

- Corrective Maintenance - This includes modifications and updations done in order to correct or fix problems, which are either discovered by user or concluded by user error reports.
- Adaptive Maintenance - This includes modifications and updations applied to keep the software product up-to date and tuned to the ever changing world of technology and business environment.
- Perfective Maintenance - This includes modifications and updates done in order to keep the software usable over long period of time. It includes new features, new user requirements for refining the software and improve its reliability and performance.
- Preventive Maintenance - This includes modifications and updations to prevent future problems of the software. It aims to attend problems, which are not significant at this moment but may cause serious issues in future.

Cost of Maintenance

Reports suggest that the cost of maintenance is high. A study on estimating software maintenance found that the cost of maintenance is as high as 67% of the cost of entire software process cycle.

Chapter 5
Implementation and Maintenance



On an average, the cost of software maintenance is more than 50% of all SDLC phases. There are various factors, which trigger maintenance cost go high, such as:

Real-world factors affecting Maintenance Cost

- The standard age of any software is considered up to 10 to 15 years.
- Older softwares, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming enhanced softwares on modern hardware.
- As technology advances, it becomes costly to maintain old software.
- Most maintenance engineers are newbie and use trial and error method to rectify problem.
- Often, changes made can easily hurt the original structure of the software, making it hard for any subsequent changes.
- Changes are often left undocumented which may cause more conflicts in future.

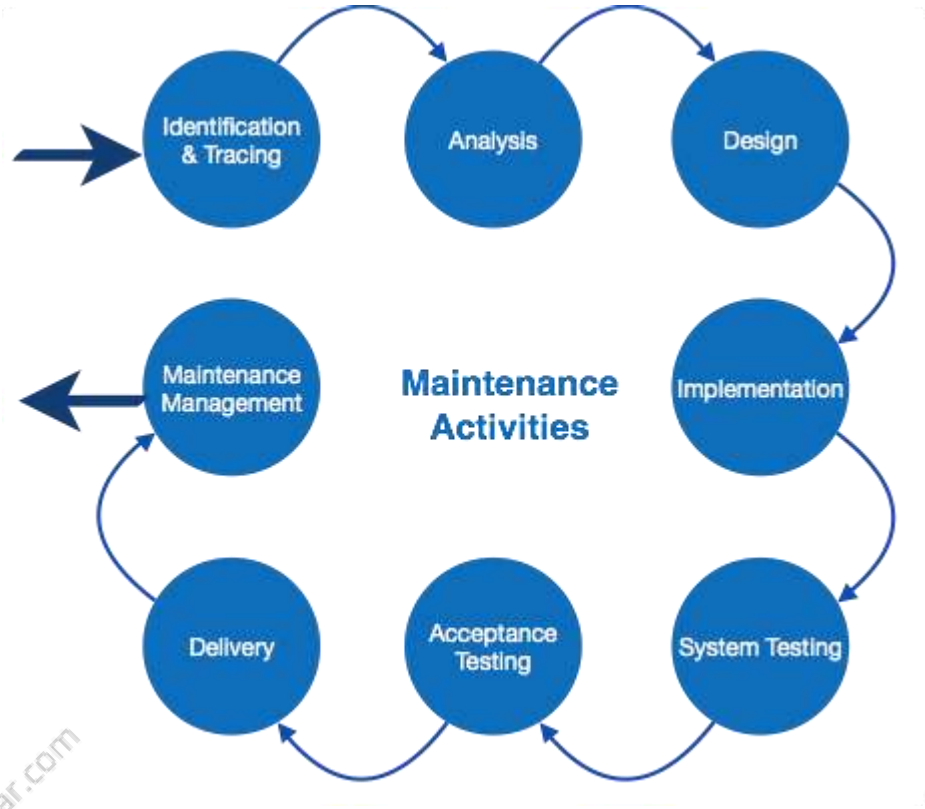
Software-end factors affecting Maintenance Cost

Chapter 5
Implementation and Maintenance

- Structure of Software Program
- Programming Language
- Dependence on external environment
- Staff reliability and availability

Maintenance Activities

IEEE provides a framework for sequential maintenance process activities. It can be used in iterative manner and can be extended so that customized items and processes can be included.



These activities go hand-in-hand with each of the following phase:

- **Identification & Tracing** - It involves activities pertaining to identification of requirement of modification or maintenance. It is generated by user or

system may itself report via logs or error messages.Here, the maintenance type is classified also.

- **Analysis** - The modification is analyzed for its impact on the system including safety and security implications. If probable impact is severe, alternative solution is looked for. A set of required modifications is then materialized into requirement specifications. The cost of modification/maintenance is analyzed and estimation is concluded.
- **Design** - New modules, which need to be replaced or modified, are designed against requirement specifications set in the previous stage. Test cases are created for validation and verification.
- **Implementation** - The new modules are coded with the help of structured design created in the design step.Every programmer is expected to do unit testing in parallel.
- **System Testing** - Integration testing is done among newly created modules. Integration testing is also carried out between new modules and the system. Finally the system is tested as a whole, following regressive testing procedures.
- **Acceptance Testing** - After testing the system internally, it is tested for acceptance with the help of users. If at this state, user complaints some issues they are addressed or noted to address in next iteration.
- **Delivery** - After acceptance test, the system is deployed all over the organization either by small update package or fresh installation of the system. The final testing takes place at client end after the software is delivered.

Training facility is provided if required, in addition to the hard copy of user manual.

- **Maintenance management** - Configuration management is an essential part of system maintenance. It is aided with version control tools to control versions, semi-version or patch management.

Software Re-engineering

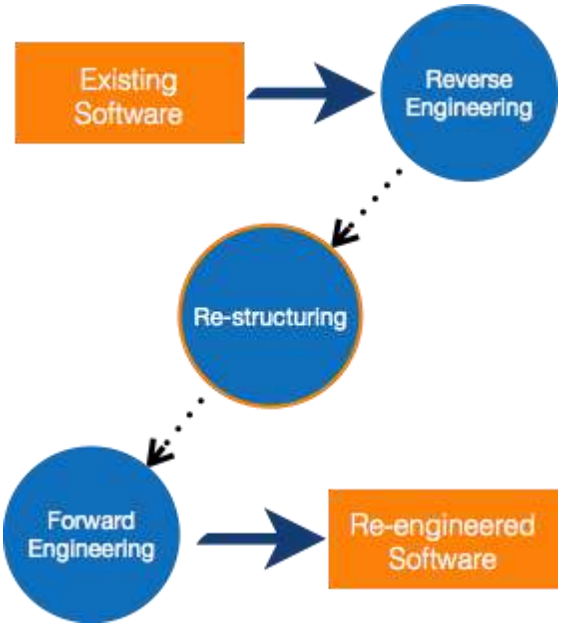
Chapter 5
Implementation and Maintenance

When we need to update the software to keep it to the current market, without impacting its functionality, it is called software re-engineering. It is a thorough process where the design of software is changed and programs are re-written.

Legacy software cannot keep tuning with the latest technology available in the market. As the hardware become obsolete, updating of software becomes a headache. Even if software grows old with time, its functionality does not.

For example, initially Unix was developed in assembly language. When language C came into existence, Unix was re-engineered in C, because working in assembly language was difficult.

Other than this, sometimes programmers notice that few parts of software need more maintenance than others and they also need re-engineering.



Re-Engineering Process

- Decide what to re-engineer. Is it whole software or a part of it?
- Perform Reverse Engineering, in order to obtain specifications of existing software.

Chapter 5
Implementation and Maintenance

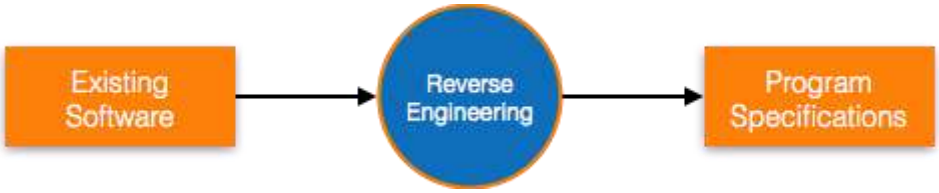
- Restructure Program if required. For example, changing function-oriented programs into object-oriented programs.
- Re-structure data as required.
- Apply Forward engineering concepts in order to get re-engineered software.

There are few important terms used in Software re-engineering

Reverse Engineering

It is a process to achieve system specification by thoroughly analyzing, understanding the existing system. This process can be seen as reverse SDLC model, i.e. we try to get higher abstraction level by analyzing lower abstraction levels.

An existing system is previously implemented design, about which we know nothing. Designers then do reverse engineering by looking at the code and try to get the design. With design in hand, they try to conclude the specifications. Thus, going in reverse from code to system specification.



Program Restructuring

It is a process to re-structure and re-construct the existing software. It is all about re-arranging the source code, either in same programming language or from one programming language to a different one. Restructuring can have either source code-restructuring and data-restructuring or both.

Re-structuring does not impact the functionality of the software but enhance reliability and maintainability. Program components, which cause errors very frequently can be changed, or updated with re-structuring.

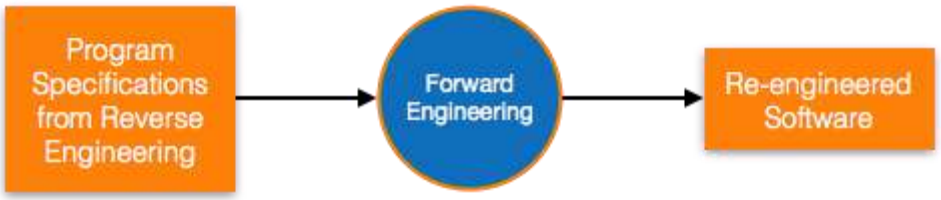
The dependability of software on obsolete hardware platform can be removed via re-structuring.

Chapter 5
Implementation and Maintenance

Forward Engineering

Forward engineering is a process of obtaining desired software from the specifications in hand which were brought down by means of reverse engineering. It assumes that there was some software engineering already done in the past.

Forward engineering is same as software engineering process with only one difference – it is carried out always after reverse engineering.



Component reusability

A component is a part of software program code, which executes an independent task in the system. It can be a small module or sub-system itself.

Example

The login procedures used on the web can be considered as components, printing system in software can be seen as a component of the software.

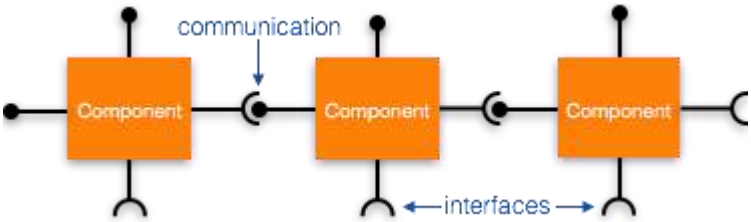
Components have high cohesion of functionality and lower rate of coupling, i.e. they work independently and can perform tasks without depending on other modules.

In OOP, the objects are designed are very specific to their concern and have fewer chances to be used in some other software.

In modular programming, the modules are coded to perform specific tasks which can be used across number of other software programs.

There is a whole new vertical, which is based on re-use of software component, and is known as Component Based Software Engineering (CBSE).

Chapter 5
Implementation and Maintenance



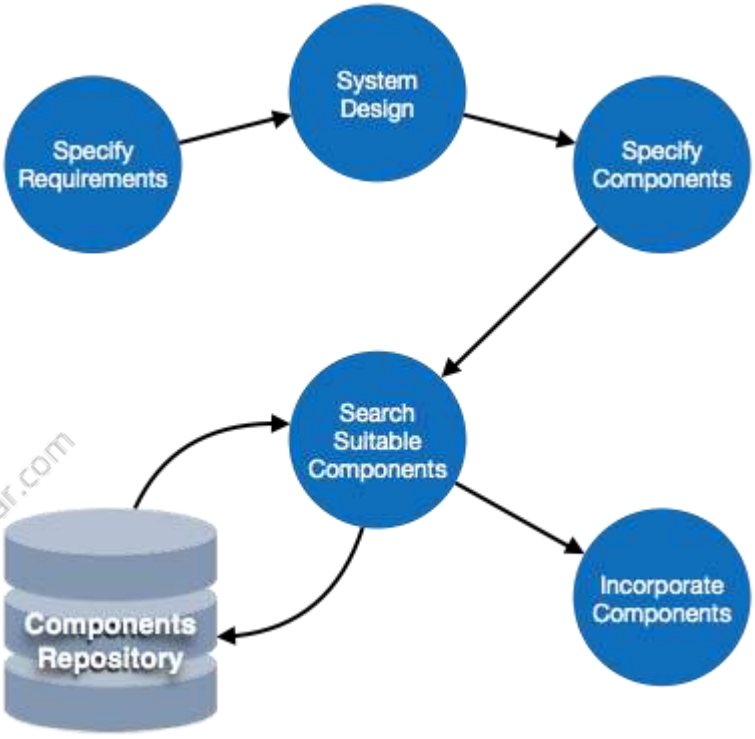
Re-use can be done at various levels

- **Application level** - Where an entire application is used as sub-system of new software.
- **Component level** - Where sub-system of an application is used.
- **Modules level** - Where functional modules are re-used.

Software components provide interfaces, which can be used to establish communication among different components.

Reuse Process

Two kinds of method can be adopted: either by keeping requirements same and adjusting components or by keeping components same and modifying requirements.



Chapter 5
Implementation and Maintenance

- **Requirement Specification** - The functional and non-functional requirements are specified, which a software product must comply to, with the help of existing system, user input or both.
- **Design** - This is also a standard SDLC process step, where requirements are defined in terms of software parlance. Basic architecture of system as a whole and its sub-systems are created.
- **Specify Components** - By studying the software design, the designers segregate the entire system into smaller components or sub-systems. One complete software design turns into a collection of a huge set of components working together.
- **Search Suitable Components** - The software component repository is referred by designers to search for the matching component, on the basis of functionality and intended software requirements..
- **Incorporate Components** - All matched components are packed together to shape them as complete software.