SOFTWARE ENGINEERING MANUAL

BY MS. MARTHA WERE

IT INSTRUCTOR

@ 2020

INTRODUCTION TO SOFTWARE ENGINEERING:

What is software?

- Computer programs and associated documentation such as requirements, design models and user manuals.
- Software products may be developed for a particular customer or may be developed for a general market.
- Software products may be
   i. Generic - developed to be sold to a range of different customers e.g. PC software such as Excel or Word.
   ii. Bespoke (custom) - developed for a single customer according to their specification.
- New software can be created by developing new programs, configuring generic software systems or reusing existing software.

What is software engineering?

- Software engineering is an engineering discipline that is concerned with all aspects of software production.
- Software engineers should adopt a systematic and organised approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available.

Why is software engineering important?

- Software must be reliable, secure, usable and maintainable. Software engineering explicitly focuses on delivering software with these attributes and, unlike programming, is not just concerned with the functionality or features of a system.
- Software engineering is particularly important for systems which people and businesses depend on and which are used for many years.

What is a software process?

- A set of activities whose goal is the development or evolution of software.
- Generic activities in all software processes are:

- Specification - what the system should do and its development constraints
- Development - production of the software system
- Validation - checking that the software is what the customer wants
- Evolution - changing the software in response to changing demands.

What is the difference between software engineering and computer science?

Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
Computer science theories are still insufficient to act as a complete underpinning for software engineering (unlike e.g. physics and electrical engineering).

What is the difference between software engineering and system engineering?

System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this process concerned with developing the software infrastructure, control, applications and databases in the system.

System engineers are involved in system specification, architectural design, integration and deployment.

What is a software process model?

•A simplified representation of a software process, presented from a specific perspective.

•Examples of process perspectives are

- Workflow perspective - sequence of activities;
- Data-flow perspective - information flow;
- Role/action perspective - who does what.

•Generic process models are:

- Waterfall;
- Iterative development;
- Component-based software engineering.

What is CASE (Computer-Aided Software Engineering)

Software systems that are intended to provide automated support for software process activities.

CASE systems are often used for method support.

Upper-CASE

Tools to support the early process activities of requirements and design; specification requirement

Lower-CASE

Tools to support later activities such as programming, debugging and testing.

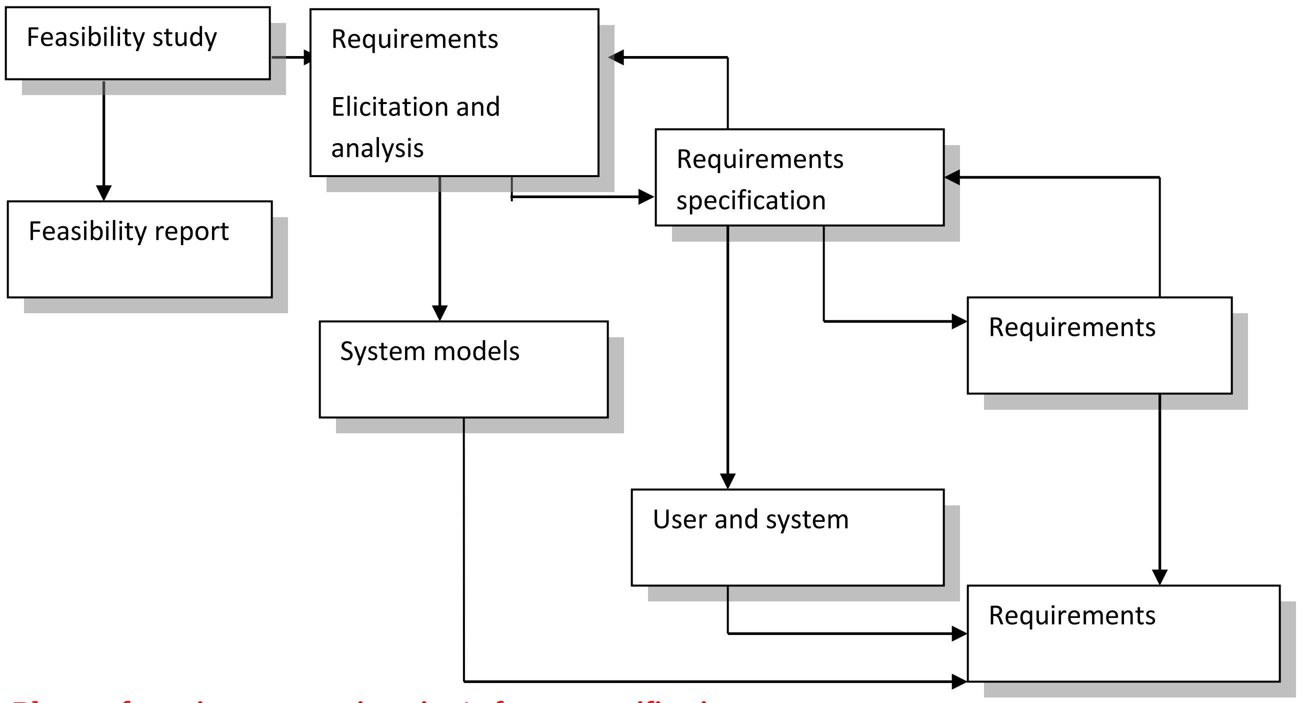
SOFTWARE PROCESS

Software Process Activities

- Software specification/requirement Engineering process
- Software design and implementation
- Software validation
- Software evolution

1. SOFTWARE SPECIFICATION/ REQUIREMENTS ENGINEERING PROCESS

- The process of establishing what services are required and the constraints on the system's operation and development
- Leads to the production of **Requirements document** that is the specification for the system.



Phases of requirements engineering/software specification

- Feasibility study;
- Requirements elicitation and analysis;
- Requirements specification;
- Requirements validation.

1.1 Feasibility study

- Technical, operational, economic, schedule feasibility of the system/project should be considered.
- Feasibility study report is produced

1.2 .Requirements elicitation and analysis

Analysts work with end users to establish functional and non functional requirements.

Fact finding techniques and prototyping may be used.

A requirement is a description of a system. It may describe the function (service) of the system, a desirable feature or characteristic of the system as well as constraints that limit the boundaries of the proposed system. (Constraint on the product and the process)

A requirement It may range from a high-level abstract statement of a service or of a system constraint (User requirements) to a detailed mathematical functional specification.(System requirements).

Throw-away prototype is used to explore requirements and design options.

Types of requirements

1. Functional
2. Nonfunctional
3. Domain
4. Usability

1. Functional requirements

A function is a set of ongoing business activities with a start to the end. Functions are named with nouns e.g Planning, Scheduling

A function consists of processes that support specific activities. Logical processes can be:

- ❖ Calculate Grade
- ❖ Generate report
- ❖ Create a record etc.

An event is a logical unit of work that must be completed as a whole (A transaction).

Events have triggers (input) and responses (defined output).

A functional requirement is a description of the function or service of the system. When writing them, use action verb e.g:

- i. Process a cheque
- ii. Calculate grade
- iii. Generate report

Example of functional requirement for a library system

- i. The system should generate weekly report of books borrowed – output
- ii. The system should calculate overdue charges automatically. – Processing
- iii. The system should capture student’s Adm No, names, Book ID, Current date etc, when a student borrows a book.

In other words, with functional requirements, we answer the question of what the system should be able to do in terms of input, output and processing. The question of how it should do it is left to designers.

Statements of services the system should provide how the system should react to particular inputs and how the system should behave in particular situations.

2. Non-functional requirements

Describe system properties or desirable attributes, constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.

We can have three types of such requirements namely: Product, Organizational or External requirements

Examples

i. Product (Software)
   - The user interface of the proposed system should be easy to use
   - The user interface should be implemented as simple html

Product requirements may touch on reliability, portability, efficiency and usability (although I have covered usability separately for the sake of clarity), efficiency etc- of the software product.

ii. Organizational(The organization developing or procuring the software- its policies)
   - The system development process and deliverables shall conform to the process and deliverables defined (by specific standards) IEEE standards

This requirement may focus on deliver time of the product, implementations techniques and standards to be adhered to.

iii. External
   - The system shall not disclose personal information of users.

Ethical and legislative issues are paramount here.

Note:

A software constraint is something, event or situation that limits the flexibility in defining a software solution to your objectives. A constraint cannot be changed (no wonder we state them with "must" word). Deadlines and budgets, among others constraint. It may be a constraint on the software process itself or a constraint on the software product to be developed and delivered into use. The environment where the product will be operational may also be constraint.

For example you may state a constraint as: "Any system developed must be compatible with the existing Windows XP OS." Or "There will be no increase in workforce"

3. Domain requirements

Requirements that come from the application domain of the system and that reflect characteristics of that domain. Domain requirements be new functional requirements, constraints on existing requirements or define specific computations.

Example: Library system domain requirements:

- There shall be a standard user interface to all databases which shall be based on the Z39.50 standard.
- Because of copyright restrictions, some documents must be deleted immediately on arrival. Depending on the user's requirements, these documents will either be printed locally on the system server for manually forwarding to the user or routed to a network printer.

Note:

- System requirements are more detailed specifications of system functions, services and constraints than user requirements. They are intended to be a basis for designing the system. They may be incorporated into the system contract. System requirements may be defined or illustrated using system models
- User requirements are high-level statements of what the system should do.
- Functional user requirements may be high-level statements of what the system should do but functional system requirements should describe the system services in detail.
- User requirements should be written using natural language, tables and diagrams.
- User requirements are defined using natural language, tables and diagrams as these can be understood by all users.

4. **Usability**

The issue of **usability** has grown more important as grate number of technically complicated products have become available to a wider population. While manufacturers have concentrated upon increasing the functionality of their products, users have grown steadily more confused that they cannot operate the machinery that they have bought.

Within the IT industry, this problem has been even more serious. Many software products have had to be abandoned, not because they did not work but because they could not or would not use them.

Usability is not determined by just one or two constituents but is influenced by a number of factors which interact with one another in sometimes complex ways. Eason (1984) has suggested a series of concepts that explain what these variables might be.

Techniques used in Requirements engineering/software specification

- ❖ Throw-away prototype is used to explore requirements and design options. (a small part of the system is developed and then given to the end user to try out and evaluate. The user provides feedback which can quickly be incorporated into the development of the main system. The prototype is then discarded or thrown away)
- ❖ Fact finding techniques may also be used to gather data from uses. (sampling of existing documentation, forms, and databases, research and site visits, observation, questionnaires, interviews, prototyping etc)
- ❖ Modeling

Logical system models depict what the system is or what the system does- not how it's physically implemented to express essential requirements of the system. They focus on the logical design of the system and not physical design as do physical models. Models should be highly cohesive. That is each module should accomplish one and only one function so that they can be reusable. Models should also be loosely coupled i.e should minimally depend on each other. This minimizes the effect that future changes on one module will have on others:

1.3Requirements elicitation and analysis

Analysts work with end users to establish functional and non functional requirements.

Fact finding techniques and prototyping may be used.

A requirement is a description of a system. It may describe the function (service) of the system, a desirable feature or characteristic of the system as well as constraints that limit the boundaries of the proposed system. (Constraint on the product and the process)

1.4 Requirements specification

Is the activity of translating the information gathered during the analysis activity into a document that is a set of requirements called Software Requirement Specification Document (SRS)-Used as a reference object

SRS Documents contents

- Introduction
- Requirements definition
- Requirements specification
- System models
- Hardware specification
- Software and database specification

1.5 Requirements validation

- Checks requirements for realism, consistency and completeness.
- SRS document must be modified to correct errors if discovered.

Requirements imprecision

Misunderstanding of user requirements is fatal! Always avoid ambiguous requirements statements and requirement assumptions.

When requirements are wrong (due to ambiguities, assumptions and misunderstanding):

i. The system may cost more than projected.
ii. The system may be delivered later than promised.
iii. The system may not meet user expectations, who when dissatisfied may not use it.
iv. Cost of maintenance and enhancement may be excessively high.
v. The system may be unreliable and prone to errors and downtime.
vi. Reputation of IT staff or the development team may be tarnished.

System requirements should meet the following criteria:

i.	Consistent:	Requirements should not be conflicting or ambiguous.
ii.	Complete:	Requirements should describe all possible system inputs and responses.
iii.	Feasible:	Requirements should be satisfied within available resources and constraints.
iv.	Required:	The requirements should be truly needed.
v.	Accurate:	The requirements should be stated correctly

vi. Traceable: The requirements should directly map to the functions and features of the system.

vii. Verifiable: The requirements should be testable.

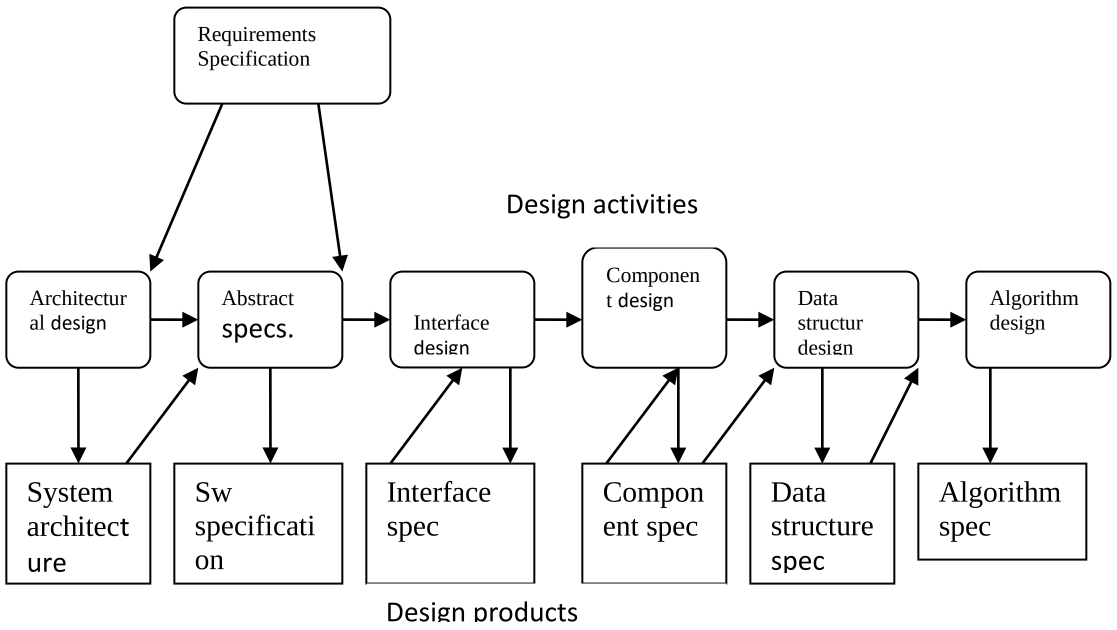
- In principle, requirements should be both complete and consistent.
- In practice, it is impossible to produce a complete and consistent requirements document.

SOFTWARE DESIGN

- A software design is a description of the structure of the software to be implemented, the data, interfaces between components and algorithms used
- The design process may involve developing several models of the system at different levels of abstraction:

Design phase activities are.

i. Architectural design
ii. Abstract specification
iii. Interface Design
iv. Component design
v. Data structure design
vi. Program /algorithm design



The above diagram shows that stages of the design process are sequential and may be interleaved. Feedback from one stage and consequent design rework is inevitable in all design processes.

A specification for the next stage is the output of each design activity. They start as abstract specifications (or formal specification) but as design continues, the specification becomes more detailed. The final result of the process are precise specifications of the algorithms and data structures to be implemented.

The design process activities are:

1. *Architectural design*

The subsystems making up the system and their relationships are identified and documented

2. *Abstract specification*

For each subsystem, an abstract specification of its services and constraints under which it must operate is produced.

3. *Interface design*

For each subsystem, its interface with other subsystems is designed and documented.

4. *Components design*

- Services are allocated to different components and the interfaces of these components are designed.

5. *Data structure design*

The data structures used in the system implementation are designed in detail and specified. Databases and files are also designed.

6. *Program/Algorithm design*

Algorithms used to provide services are designed in detail and specific.

The program's processing logic may be modeled with decision tables, decision trees, structured English statements, and pseudo-codes etc.

This is a very general model of the design process and real, practical processes may adapt it in different ways. For example, the last two stages, data structure and algorithm design, may be part of the design, or may constrain the architecture of the system and the interfaces of the system modules. An exploratory approach to the design may be used and the system interfaces may be designed after the data structures have been specified.

Software Analysis & Design Tools

Software analysis and design includes all activities, which help the transformation of requirement specification into implementation.

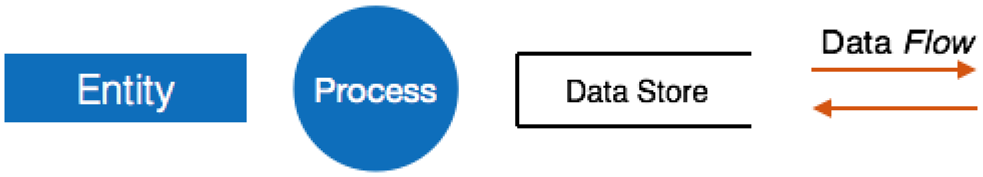
Software analysis and design is the intermediate stage, which helps human-readable requirements to be transformed into actual code.

1. Data Flow Diagram

Data flow diagram is graphical representation of flow of data in an information system.

DFD Components

DFD can represent Source, destination, storage and flow of data using the following set of components -



- **Entities -** Entities are source and destination of information data. Entities are represented by a rectangles with their respective names.
- **Process -** Activities and action taken on the data are represented by Circle or Round-edged rectangles.
- **Data Storage -** There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.
- **Data Flow -** Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

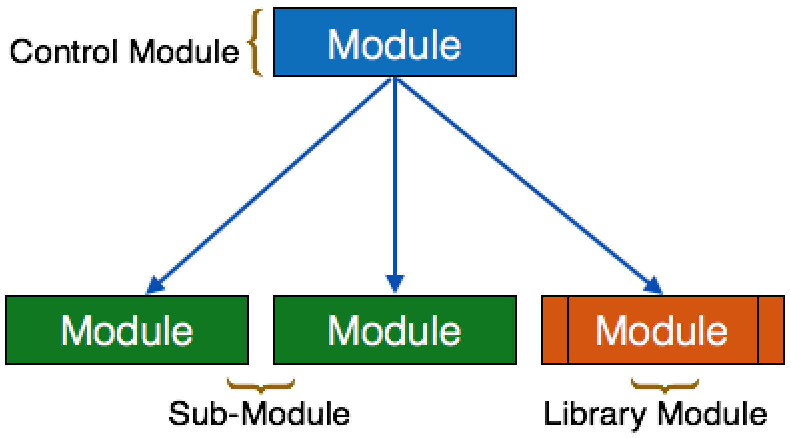
2. Structure Charts

Structure chart is a chart derived from Data Flow Diagram. It represents the system in more detail than DFD. It breaks down the entire system into lowest functional modules, describes functions and sub-functions of each module of the system to a greater detail than DFD.

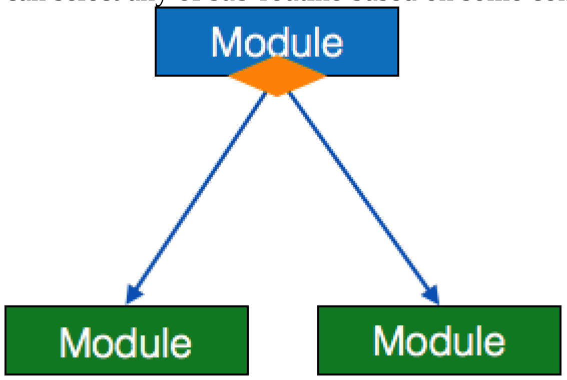
Structure chart represents hierarchical structure of modules. At each layer a specific task is performed.

Here are the symbols used in construction of structure charts -

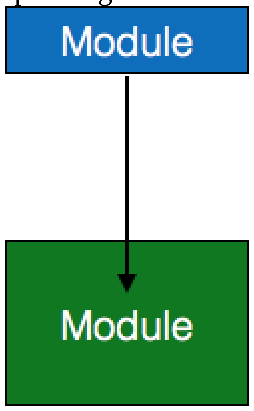
- **Module** - It represents process or subroutine or task. A control module branches to more than one sub-module. Library Modules are re-usable and invokable from any module.



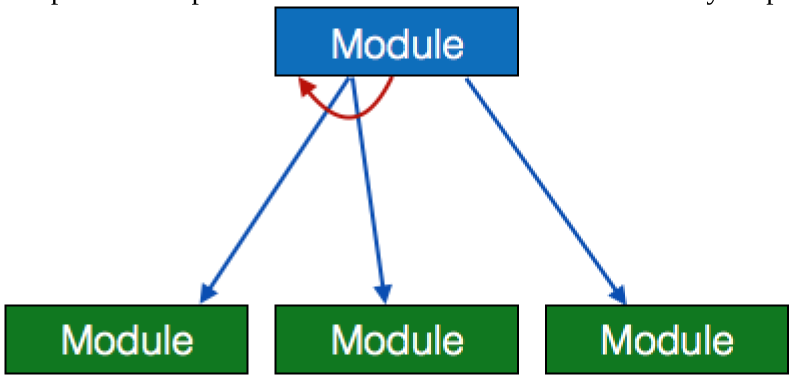
- **Condition** - It is represented by small diamond at the base of module. It depicts that control module can select any of sub-routine based on some condition.



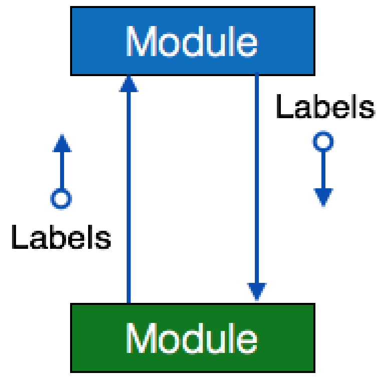
- **Jump** - An arrow is shown pointing inside the module to depict that the control will jump in the middle of the sub-module.



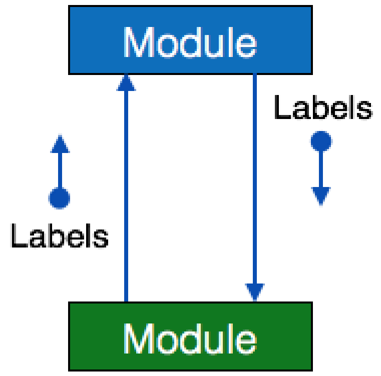
- **Loop** - A curved arrow represents loop in the module. All sub-modules covered by loop repeat execution of module.



- **Data flow -** A directed arrow with empty circle at the end represents data flow.



- **Control flow -** A directed arrow with filled circle at the end represents control flow.



3. Pseudo-Code

Pseudo code is written more close to programming language. It may be considered as augmented programming language, full of comments and descriptions.

Pseudo code avoids variable declaration but they are written using some actual programming language's constructs, like C, Fortran, Pascal etc.

Pseudo code contains more programming details than Structured English. It provides a method to perform the task, as if a computer is executing the code.

Example

Program to print Fibonacci up to n numbers.

```
void function Fibonacci
Get value of n;
Set value of a to 1;
Set value of b to 1;
Initialize I to 0
for (i=0; i< n; i++)
{
  if a greater than b
  {
    Increase b by a;
    Print b;
  }
  else if b greater than a
  {
      increase a by b;
      print a;
   }
}
```

4. Decision Tables

A Decision table represents conditions and the respective actions to be taken to address them, in a structured tabular format.

It is a powerful tool to debug and prevent errors. It helps group similar information into a single table and then by combining tables it delivers easy and convenient decision-making.

Creating Decision Table

To create the decision table, the developer must follow basic four steps:

- Identify all possible conditions to be addressed
- Determine actions for all identified conditions
- Create Maximum possible rules
- Define action for each rule

Decision Tables should be verified by end-users and can lately be simplified by eliminating duplicate rules and actions.

Example

Let us take a simple example of day-to-day problem with our Internet connectivity. We begin by identifying all problems that can arise while starting the internet and their respective possible solutions.

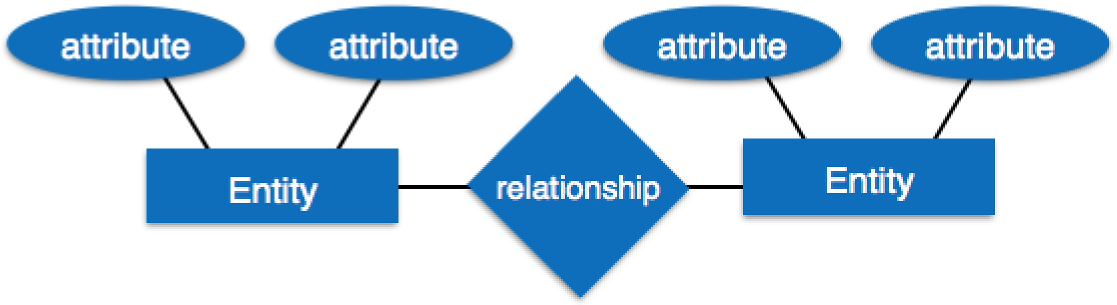
We list all possible problems under column conditions and the prospective actions under column Actions.

	Conditions/Actions	**Rules**							
	Shows Connected	N	N	N	N	Y	Y	Y	Y
Conditions	Ping is Working	N	N	Y	Y	N	N	Y	Y
	Opens Website	Y	N	Y	N	Y	N	Y	N
	Check network cable	X							
	Check internet router	X				X	X	X	
Actions	Restart Web Browser							X	
	Contact Service provider		X	X	X	X	X	X	
	Do no action								

5. Entity-Relationship Model

Entity-Relationship model is a type of database model based on the notion of real world entities and relationship among them. We can map real world scenario onto ER database model. ER Model creates a set of entities with their attributes, a set of constraints and relation among them.

ER Model is best used for the conceptual design of database. ER Model can be represented as follows :



- **Entity -** An entity in ER Model is a real world being, which has some properties called ***attributes***. Every attribute is defined by its corresponding set of values, called ***domain***.

  For example, Consider a school database. Here, a student is an entity. Student has various attributes like name, id, age and class etc.

- **Relationship -** The logical association among entities is called ***relationship***. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of associations between two entities.

  Mapping cardinalities:

  - one to one
  - one to many
  - many to one
  - many to many

HUMAN-COMPUTER DIALOGUE STYLES

i. *Command Language*

Command language is one of the oldest and most commonly used dialogue styles. In Command language, the user types in commands to the computer system and the system then carries out these commands. The user my type

Delete wambuicvfile

And the system may respond with either a prompt to indicate that the command has been carried out or with a message stating why this command could not be executed.

Command language leaves the user in control of the interaction; the system simply implements the commands that the user issues. The difficulty is that the user has to remember many commands and the syntax of these commands.

ii. *Menus*

In a menu approach, the user simply chooses the command from a list (menu) of possible commands. Where there are many possible commands and displaying them all might prove difficult, then menus are sometimes organised hierarchically in a tree-likestructure.

Like command language, menus also leave the user in control of the interaction. It should however be noted that the user never has complete freedom within the dialogue because the total assemblage of possible command alternatives is dictated in advance by the design of the system.

iii. *Forms fill-ins*

The user is presented with a form where the various portions must be filled-in, leaving the user with few alternatives.

Forms fill-in leaves the user very little control over the dialogue. However an advantage is that the user rarely needs to remember commands or their syntax.

iv. *Direct manipulation*

This is an interaction style in which the objects of interest in the UI are visible and can be acted upon via physical, reversible, incremental actions that receive immediate feedback.

Example: moving a file from the source folder to a destination via dragging instead of copying and pasting

Some of the advantages of direct manipulation interfaces that Shneiderman (1982) has listed are:

- ✓ Novices can learn basic functionality quickly, usually through a demonstration by a more experienced user.
- ✓ Experts can work extremely rapidly to carry out a wide range of tasks.

- ✓ Error messages are rarely needed.
- ✓ Users can see immediately if their actions are furthering their goals and if not, they can

v. *Mixed initiative style*

Most dialogue styles tend to reside towards one end or the other of the *control continuum*.

Mixed initiatives are those where the user may control the dialogue up to a point. The system may then take control, possibly asking questions of the user, before handing control back. This idea of mixed initiative dialogue or a system where dialogue control moves back and forth during the process of communication is presently the subject of research. Mixed initiative systems are part of a future perspective where systems are no longer tools but collaborate intelligently with their users.

User Interface Dialogue design guidelines

There ha been a great deal of research into the structure, content and style of HC dialogue. So far there is no unified theory or explanatory framework has been produced. Nevertheless, general design guidelines have emerged.

Some of the proposals of McMillan and Moran (1985)

- ✓ A system should recognise any reasonable approximation of a command
- ✓ After a command is entered by the user, the system should provide an acknowledgment of some sort which includes some reference as to how the system interpreted the command.
- ✓ The syntax of commands should be kept simple and natural
- ✓ The number of commands in a system should be limited and in a limited format.
- ✓ The facility to undo the last command should be made available to the user.

Shneiderman (1987) suggests eight rules for HC dialogue design

- ✓ Dialogue should be consistent
- ✓ The system should allow users shortcuts through some parts of familiar dialogue
- ✓ Dialogues should offer informative feedback
- ✓ Sequence of dialogues should be organised into logical groups
- ✓ Systems should offer simple error handling
- ✓ Systems should allow actions to be reversed
- ✓ Systems should allow experienced users to feel as though they are in control rather than the system
- ✓ Systems should aim to reduce short-term memory load – users should not be expected to remember too much

Another usability professional enumerates the following usability principles. I want you to compare with Shneiderman's and make a generalization.

Principle	Description
User familiarity	The interface should use terms and concepts which are drawn from the experience of the people who will make most use of the system
Consistency	The interface should be consistent in that, whether possible, comparable operations should be activated in the same way.
Minimal surprise	User should never be surprised by the behavior of a system
Recoverability	The interface should include mechanisms to allow users to recover from errors such as confirmation for destructive actions, provision of undo facility
User guidance	The interface should provide meaningful feedback when errors occur and provide context-sensitive user help
User	The interface should provide appropriate interaction facilities for different types of system users

Shneiderman (1998) has classified these different forms of interaction into five primary styles:

Interactive style	**Main advantages**	**Main disadvantages**	**Application domain**
Direct manipulation	Fast and intuitive interaction Easy to learn	May be hard to implement Only suitable where there is a visual metaphor for tasks and objects	Video games CAD systems
Menu selection	Avoids user errors Little typing required	Slow for experienced user Can become complex if many menu options	Most general-purpose systems
Form fill-in	Simple data entry Easy to learn	Take up a lot of screen space	Stock control Personal loan processing

Command language	Powerful and flexible	Hard to learn Poor error management	OS Library information retrieval systems
Natural language	Accessible to casual users Easily extended	Requires more typing Natural language understanding systems are unreliable	Timetable systems WWW information Retrieval systems

These interaction styles may be mixed and several different styles are used in the same application. For example, MS Windows supports direct manipulation of the iconic representation of files and directories, menu based command selection and, for some commands such as configuration commands, the user must fill in a special-purpose form that is presented to them.

User interfaces of the WWW are based on the support provided by HTML along with languages such as Java that can associate programs with components on a page.

GRAPHICAL USER INTERFACE- GUIs

GUI Characteristics	**Description**
Windows	Multiple windows allow different information to be displayed simultaneously on the user's screen
Icons	Represent different types of information-files, programs etc.
Menus	Commands are selected from a menu rather than typed in a command language
Pointing	
Graphics	Graphical elements can be mixed with text on the screen display

Fig: Characteristics of GUIs

Advantages of GUIs

- They are relatively easy to learn and use. Users with no computing experience can learn to use the interface after a training session
- The user has multiple screens (windows) for system interaction. Switching from one task to another is possible without losing sight of information generated during the fist tak.
- Fast full-screen interaction is possible with immediate access to anywhere on the screen.

PROGRAMMING AND DEBUGGING

- Involves translating a design into a program (programming) and removing errors from that program (debugging).
- Programming is a personal activity - there is no generic programming process.
- Programmers carry out some program testing to discover faults in the program and remove these faults in the debugging process.

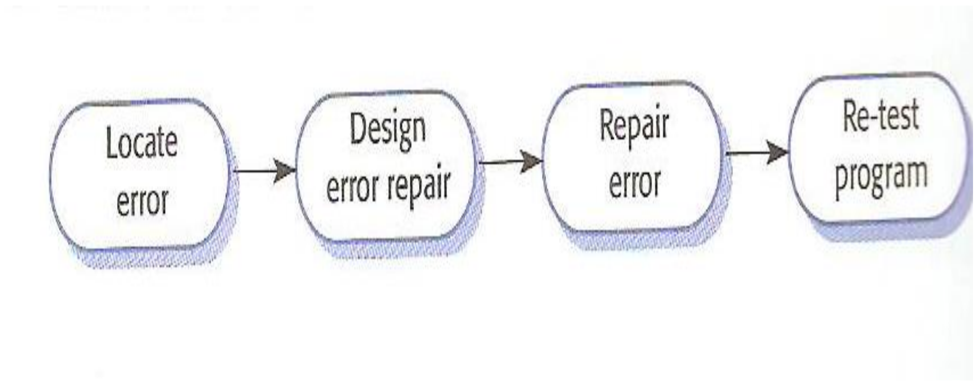


Diagram: The debugging process

SOFTWARE VERIFICATION AND VALIDATION

Software validation, or more generally, verification and validation is intended to show that a system conforms to its specification and that the system meets the expectations of the customer buying the system.

- Validation is intended to ensure that the software meets the requirements of the system users customer work- validation

  Validation: Are we building the right product?
- Verification is intended to show that a system conforms to its specification. Involves checking and review processes and system testing.

  Verification": Are we building the product right?

Verification and validation techniques

- Software inspections
- Testing

1. Software inspections

- Analyses and checks system representations such as the requirements documentations, design diagrams and program source code. They may be applied at all stages of the process unlike testing which can only be used when a prototype or an executable program is available.
- Inspection techniques include program inspection, automated source code analysis, or analysis of associated documents and formal verification.
- Software inspections and automated analyses are static techniques as they do not require the software to be executed can only check the correspondence between a program and its specification (verification) .They cannot demonstrate that the software is operationally useful; nor can they check non-functional characteristics of the software such as its performance and reliability.

Reviews and Inspections have proved to be an effective technique of error detection in components and subsystems: Error can be found more cheaply through inspection than by extensive program testing. Reasons are:

i. Many different defects may be discovered in a single inspection session. The problem with testing is that it can only detect one error per test because defects can cause the program to crash or interfere with the symptoms of other program defects.

ii. They reuse domain and programming language knowledge. In essence, the reviewers are likely to have seen the types of error that commonly occur in particular programming languages and in particular types of applications. They can therefore focus on these error types during analysis.

However inspections:

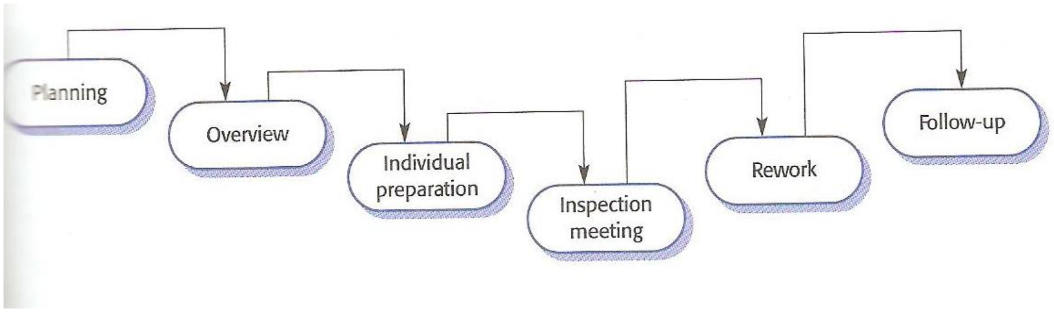
i. Cannot validate dynamic behavior of the system

ii. It's often impractical to inspect a complete system that is integrated from a number of different sub-systems. Testing is the only possible V & V technique at the system level.

The process of inspection is a formal one carried out by a team of at least 4 people. Inspection Team members systematically analyses the code and point out the possible defects. Proposal for team members' roles may be:

i. Author or owner- The programmer or designer responsible for producing the program or document. Also responsible for fixing defects discovered during the inspection process.
ii. Reader- Reads the code aloud to the inspection team
iii. Tester – Inspects the code from a testing perspective
iv. Inspector- Finds errors, omissions and inconsistencies in program and documents.
v. Moderator or chairperson - organizes the inspection process and facilitates the process.
vi. Chief moderator – Responsible for inspection process improvements, checklist updating, and standards development and so on.

A very general inspection process is shown below



The inspection process

- The moderator is responsible for inspection planning. This involves selecting an inspection team, organizing a meeting room and ensuring that the materials to be inspected and its specifications are complete.
- The program to be inspected is presented to the inspection team during overview stage where the author of the code describes what the program is intended to do.
- This is followed by a period of individual preparation where each inspection team member studies the specification and the program and looks for defects in the code.
- The inspection is done and should be relatively short (<= 2 hours) and should be exclusively concerned with identifying defects, anomalies and non-compliance standards. The inspection team should not suggest how these defects should be corrected nor recommend changes to these components.
- Following inspection, the program is modified by its author to correct the identified problems.

- In the follow-up stage the moderator must decide whether a re-inspection of the code is required. If it is not required, then the document is approved by the moderator for release.

2. **Testing**

- Testing is the process of examining a software product to find errors. This is necessary not just for code but for all life-cycle products and all documents in support of the software such as user manuals.
- The basic unit of testing is the test case. A test case consists of a test case type, which is the aspect of the system that the test case is supposed to exercise; test conditions, which consist of the input values for the test; the environmental state of the system to be used in the test; and the expected behavior of the system given the inputs and environmental factors.
- It involves executing an implementation of the software with test data and examining the outputs of the software and its operational behavior to check if it is performing as required. Testing is a dynamic technique of verification and validation because it works with an executable representation of the system

Software inspection can be used at all stages of the software process. However, testing can only be used when a prototype or an executable program is available. Testing may be carried out during the implementation phase to verify that the software behaves as intended by the designer and after the implementation is complete. Testing is always required to validate software.

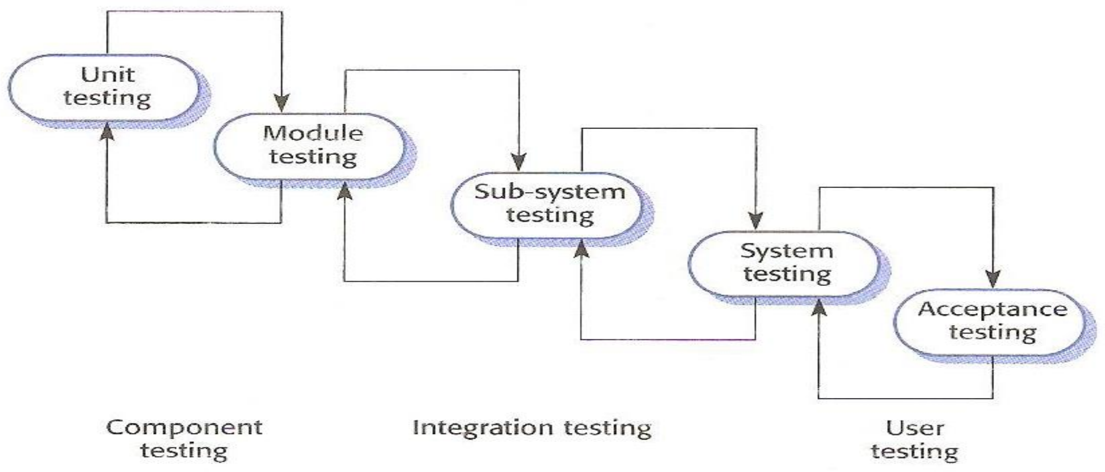
The ultimate goal of the verification and validation process is to establish confidence that the software system is “fit for purpose”. This does not mean that the product must be completely free of defects. Rather, it means that the system must be good enough for its intended purpose.

Why testing than inspection:

i. Testing is the only possible V & V technique at the system level.
ii. Testing is necessary for reliability assessment, performance anaysis, user interface validation and to check that the software requirements are what the user really wants.

The testing process

There is a five-stage testing process where system components are tested, the integrated system tested, and finally the system is tested with customers. Ideally component defects are discovered early in the process and interface problems when system is integrated.



- Unit testing: Individual components are tested to ensure they operate correctly. Each component that is tested independently, without other system components. E.g unit registration
- Module testing: A module is a collection of dependent components such as class objects, an abstract data type or some collection of procedures and functions. The module is independently tested. E.g academic details
- Sub-system testing: Involves Collection of modules which have been integrated into subsystem. The subsystem test process should concentrate on the detection of module interface errors rigorously exercising these interfaces e.g portal
- System testing: The subsystems are integrated to make up the system. Eg ERP

  This process is concerned with finding errors that result from anticipated interaction between subsystems interface problems.

  It is also concerned about validating that the system meets its functional and non-functional requirements and testing the emergent system properties.
- Acceptance testing: This testing may reveal errors and omissions in the system requirements definitions because the real data exercise the system in different ways from the test data. It's the final testing process before the system is accepted for operational use. The system is tested with data supplied by the system customer rather than simulated test data. Acceptance testing may also reveal requirements problems where the system's facilities do not really meet the user's needs or the system performance is unacceptable.

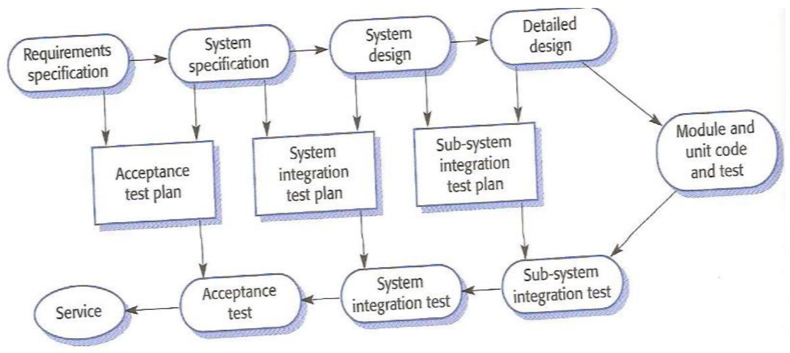

Requirements specification
System specification
System design
Detailed design
Acceptance test plan
System integration test plan
Sub-system integration test plan
Module and unit code and test
Service
Acceptance test
System integration test
Sub-system integration test

Diagram : Testing phases in SE

Before shipping the final version of software, *alpha* and *beta* testing are often done additionally:

- *Alpha testing* is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing, before the software goes to beta testing. Its also called verification testing
- *Beta testing* comes after alpha testing. Versions of the software, known as beta versions, are released to a limited audience outside of the programming team. The software is released to groups of people so that further testing can ensure the product has few faults or bugs. Sometimes, beta versions are made available to the open public to increase the feedback field to a maximal number of future users. It is alos called validation testing which requires the system to perform correctly using a given acceptance test cases.

  Finally, acceptance testing can be conducted by the end-user, customer, or client to validate whether or not to accept the product. Acceptance testing may be performed as part of the hand-off process between any two phases of development.

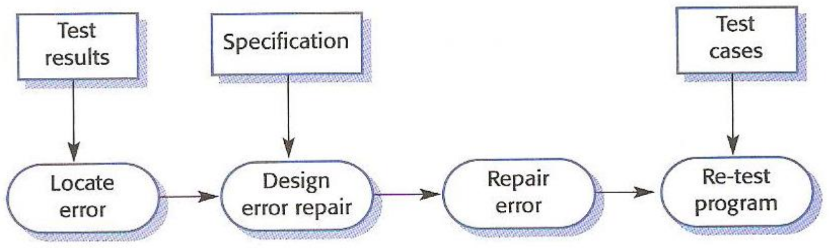
Approaches to testing:

a. Defect testing
b. Integration testing

DEBUGGING PROCESS

Defect testing and debugging are different processes. The debugging process is concerned with locating and correcting defects while testing establishes the existence of defects

Debugging is the process that locates and corrects these defects as shown below.



After each statement has been executed, the values of the variables can be examined by use of debugging tools.

After a defect in the program has been discovered, it must be corrected and the system should be revalidated. This may involve reinspecting the program or repeating the previous test runs (regression testing)

. Regression testing is used to that the check that the changes made to a program have not introduced new faults into the system

QUALITY MANAGEMENT

Software Quality Management (SQM) is concerned with ensuring that the required level of quality is achieved in a software product. It involves defining appropriate quality standards and procedures and ensuring that these are followed. SQM should aim to develop a 'quality culture' where quality is seen as everyone's responsibility. What is quality? Quality, simplistically, means that a product should meet its specification. This is problematical for software systems

- There is a tension between customer quality requirements (efficiency, reliability, etc.) and developer quality requirements (maintainability, reusability, etc.);
- Some quality requirements are difficult to specify in an unambiguous way;
- Software specifications are usually incomplete and often inconsistent.

  Note:

  - We must put quality management procedures into place to improve quality in spite of imperfect specification.

Scope of quality management

- Quality management is particularly important for large, complex systems. The quality documentation is a record of progress and supports continuity of development as the development team changes.
- For smaller systems, quality management needs less documentation and should focus on establishing a quality culture.

Quality management activities

- Quality planning
  - Select applicable procedures and standards for a particular project from the framework and the adoption of them for a specific software project
  - Testers create goals and objectives for your software as well as a strategic plan that will help you to successfully meet those objectives

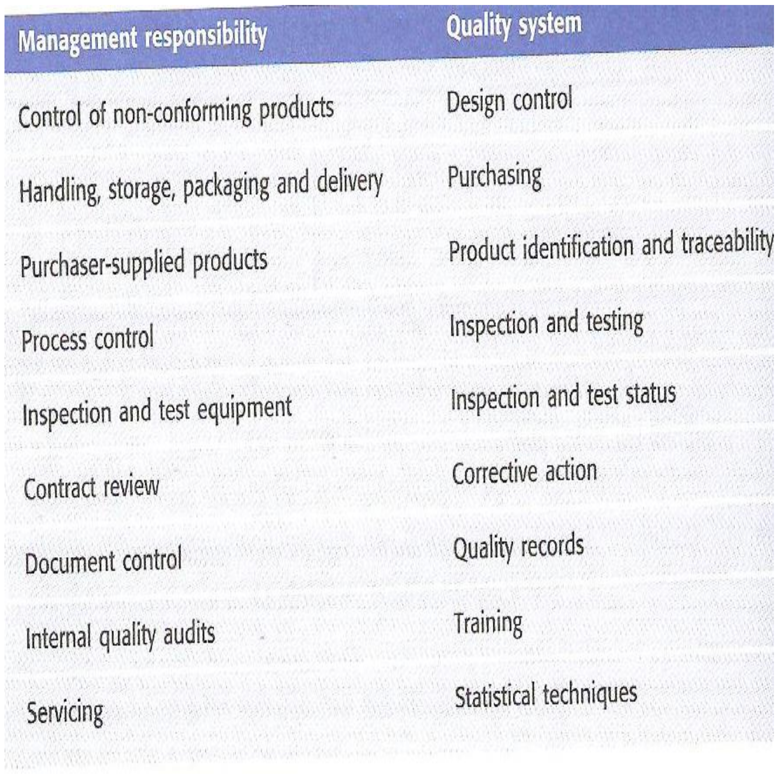
- Quality assurance

    - Establish organizational procedures and standards for quality which is a framework that leads to high quality software.
    - Involves the actual building of the software
- Quality control
    - The definition and enactment of processes which ensure that the project quality procedures and standards are followed by the software development team.
    - The team verifies the product's compliance with the functional requirements. Testing is done here.

Quality management should be separate from project management to ensure independence.

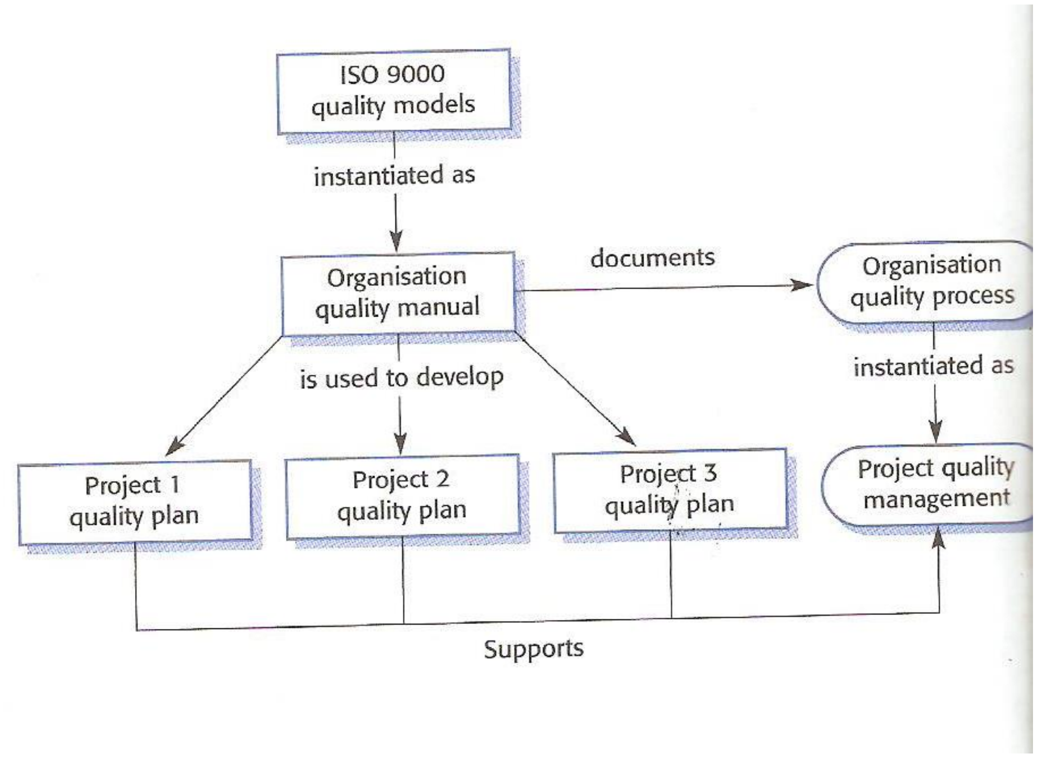
Standards development-ISO 9000

- An international set of standards for quality management.
- Applicable to a range of organisations from manufacturing to service industries.
- ISO 9001 applicable to organisations which design, develop and maintain products.
- ISO 9001 is a generic model of the quality process that must be instantiated for each organisation using the standard.



Management responsibility	Quality system
Control of non-conforming products	Design control
Handling, storage, packaging and delivery	Purchasing
Purchaser-supplied products	Product identification and traceability
Process control	Inspection and testing
Inspection and test equipment	Inspection and test status
Contract review	Corrective action
Document control	Quality records
Internal quality audits	Training
Servicing	Statistical techniques

The above figure shows areas which are covered in ISO 19001. The quality assurance procedure in an organization is documented in a quality manual which defines the quality manual which defines the quality process. The relationship between ISO 9000, the quality manual and individual project quality plans is shown below.



ISO 9000 certification

- Quality standards and procedures should be documented in an organizational quality manual.
- An external body may certify that an organization's quality manual conforms to ISO 9000 standards.
- Some customers require suppliers to be ISO 9000 certified although the need for flexibility here is increasingly recognized.

Quality assurance and standards

There are two types of standards:

1. *Product standards*: These are standards that apply to the software product being developed. They include documentation standards and coding standards
2. *Process standards* define the processes which should be followed during software development. They may include definitions for specs, design and validation processes, how reviews should be conducted, configuration and a description of documents which must be generated in the course of these processes. Process standards monitor the development process to ensure that standards are being followed. Don't use inappropriate practices simply because standards have been established.

Standards are the key to effective quality management. They may be international, national, and organizational or project standards. Product standards define characteristics that all components should exhibit e.g. a common programming style. Process standards define how the software process should be enacted.

Importance of standards

- Encapsulation of best practice- avoids repetition of past mistakes.
- They are a framework for quality assurance processes
- They provide continuity - new staff can understand the organization by understanding the standards that are used.

Problems with standards

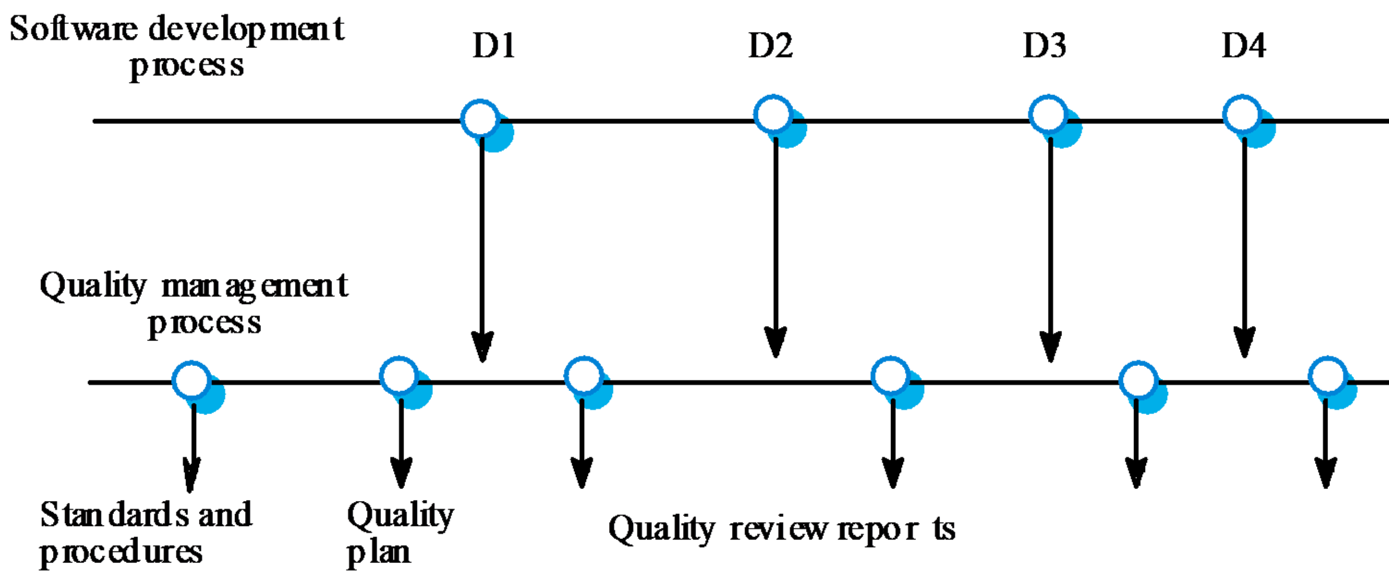
- They may not be seen as relevant and up-to-date by software engineers.
- They often involve too much bureaucratic form filling.
- If they are unsupported by software tools, tedious manual work is often involved to maintain the documentation associated with the standards.

There is a straightforward link between process and product qualities in manufactured goods. The quality of a developed product is influenced by the quality of the production process. This is important in software development as some product quality attributes are hard to assess. However, there is a very complex and poorly understood relationship between software processes and product quality.

Process and product quality is more complex for software because:

- The application of individual skills and experience is particularly important in software development;
- External factors such as the novelty of an application or the need for an accelerated development schedule may impair product quality.

Care must be taken not to impose inappropriate process standards - these could reduce rather than improve the product quality.



Documentation standards

Documentation standards in a software project are particularly important as documents are the only tangible way of representing the software and the software process/ standardized documents have a consistent appearance, structure and quality and should therefore be easier to read and understand.

There are 3 types of documentation standards:

- *Documentation process standards* - They define the process which should be followed for document production
- *Document standards* – They govern the structure and presentation of documents.
- *Documents interchange standards*- standards that ensure that all electronic copies of the documents are compatible.

Quality Planning

- A quality plan sets out the desired product qualities and how these are assessed and defines the most significant quality attributes.
- It should set out which organisational standards should be applied and, where necessary, define new standards to be used.

Quality plan structure

- Product introduction;
- Product plans;
- Process descriptions;
- Quality goals;
- Risks and risk management.

- Quality plans should be short, succinct(brief, concise)documents

If they are too long, no-one will read them

Figure showing Software Quality attributes

Safety	Understandability	Portability
Security	Testability	Usability
Reliability	Adaptability	Reusability
Resilience	Modularity	Efficiency
Robustness	Complexity	Learnability

Quality control

This involves checking the software development process to ensure that procedures and standards are being followed.

There are two approaches to quality control:

i. Quality reviews
ii. Automated software assessment and software measurement-

1. *Quality reviews* – The software, its documentation and the processes used to produce that software are reviewed by a group of people who are responsible for checking that the project standards have been followed and the software and documents conform to these standards. Deviations from these standards are noted and brought to the attention of project team. This is the principal method of validating the quality of a process or of a product There are different types of review with different objectives
   - Inspections for defect removal (product);
   - Reviews for progress assessment (product and process);
   - Quality reviews (product and standards).

Types of review

Review type	Principal purpose
Design or program inspections	To detect detailed errors in the requirements, design or code. A checklist of possible errors should drive the review.
Progress reviews	To provide information for management about the overall progress of the project. This is both a process and a product review and is concerned with costs, plans and schedules.
Quality reviews	To carry out a technical analysis of product components or documentation to find mismatches between the specification and the component design, code or documentation and to ensure that defined quality standards have been followed.

Review functions

- Quality function - they are part of the general quality management process.
- Project management function - they provide information for project managers.
- Training and communication function - product knowledge is passed between development team members.

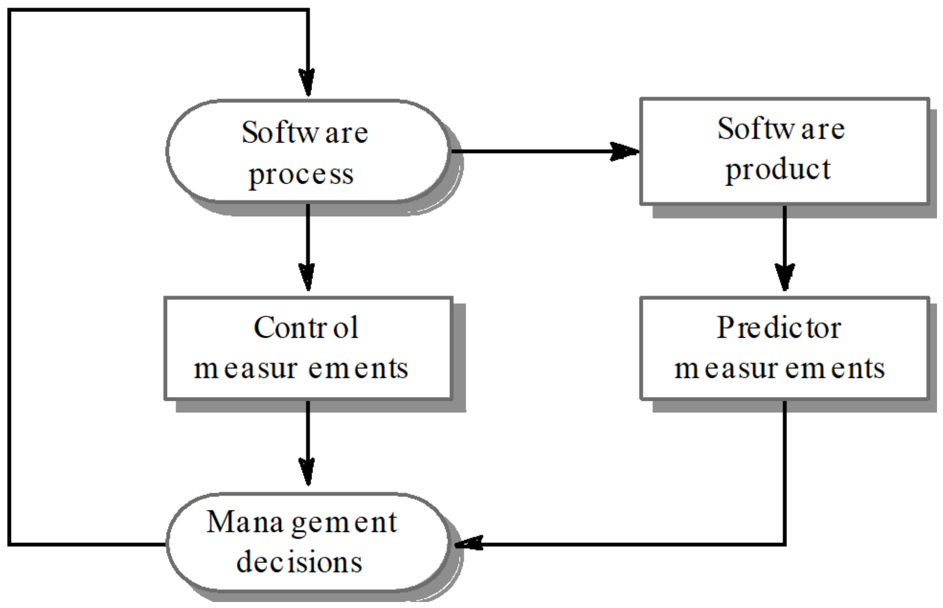
2. *Automated software assessment and software measurement*- The software and documents which are produced are processed by some program and compared to the standards which apply to that particular development project. This automated assessment may involve a quantitative measurement of some software attributes

Software measurement and metrics

Software measurement is concerned with deriving a numeric value for an attribute of a software product or process. This allows for objective comparisons between techniques and processes.

Software metric

Predictor and control metrics



Software metric is any type of measurement which relates to a software system, process or related documentation. Examples are measures of the size of product in lines of code, the number of reported faults in delivered software product and the number of person-days required to develop a system component.

Metrics may be either control metrics or predictor metrics.

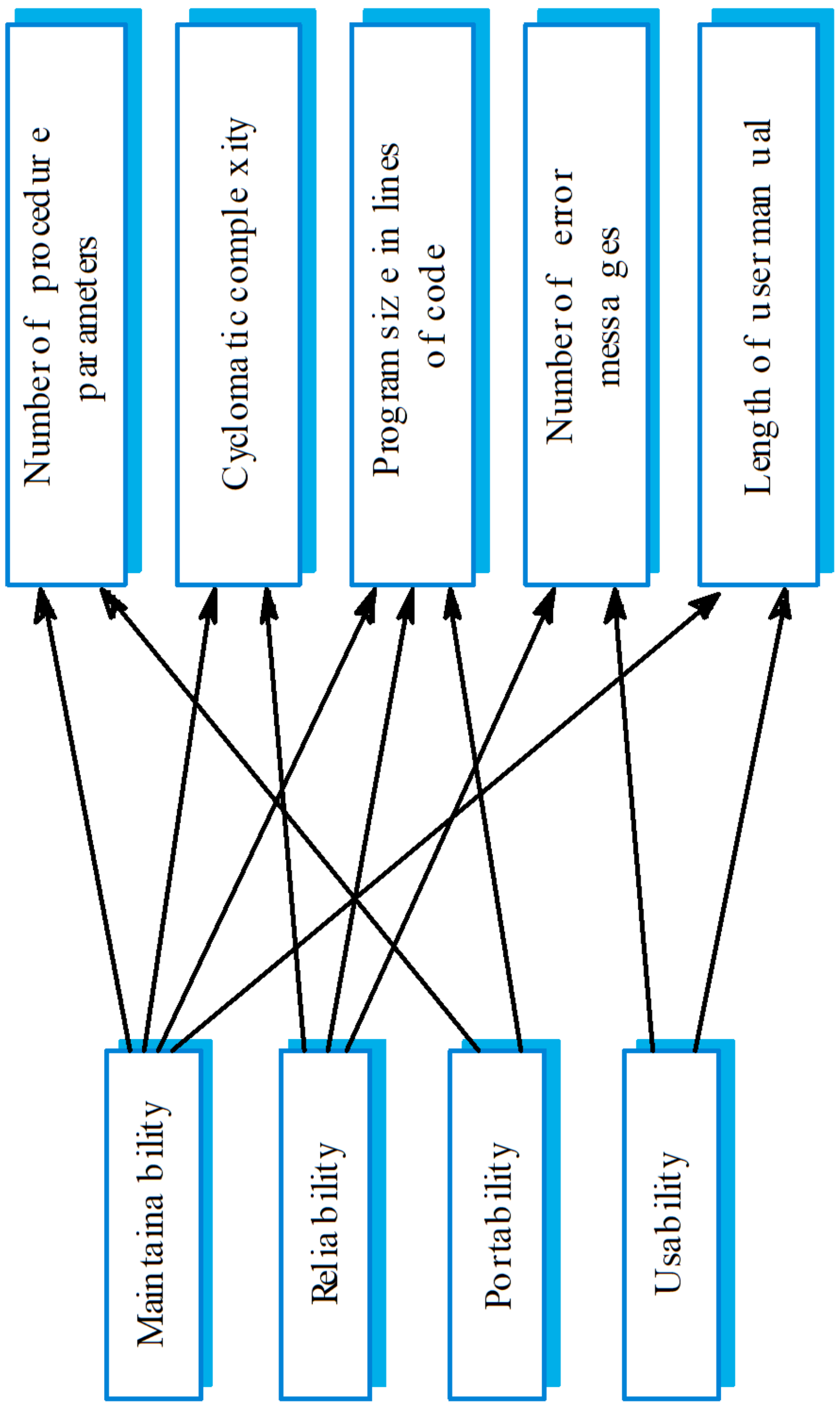
- *Control metrics* usually are associated with the software process e.g the average effort and time required to repair reported defects.
- *Predictor metrics* are associated with the software product. An example is the average length of identifiers in a program (the longer the identifiers, the more likely they are to be meaningful and hence the more understandable the program.), length of code(usually the larger the size of the code of a program component, the more complex and error-prone that component is likely to be.) and the number of attributes and operations associated with objects in a design. Both types of metrics as shown in the diagram may influence management decision making.

Two classes of product metrics are:

- Dynamic metrics are closely related to software quality attributes
  - It is relatively easy to measure the response time of a system (performance attribute) or the number of failures (reliability attribute).
- Static metrics have an indirect relationship with quality attributes
  - You need to try and derive a relationship between these metrics and properties such as complexity, understandability and maintainability.

Internal and external attributes

Some external quality attributes which might be of interest and internal attributes which can be measured and which might be related to the external attribute.



z

Importance of quality.

If a system is of such poor quality that it doesn't meet the user's requirements and contains errors then the users will be dissatisfied with the system leading to:

i. If the user is not satisfied, he can insist on rework free of charge (depending on contractual management or on the method of finding maintenance). This could cause disastrous financial implications to the developers- human efforts, time and other resources are diverted from doing productive work of building new systems to reworking the system to improve it.

ii. Because of the above stated problem of concentrating on maintaining old systems instead of building new ones, efficienciency of business will deteriorate.

iii. Incase maintenance is regarded as an overhead to the user, then there will be ill will and lack of confidence by the user on the developers. This will create problems for any subsequent projects on the hand. Demand for systems from that developer(s) will go down- due to decreased user satisfaction.

iv. Equally, the customer incurs more maintenance costs incase maintenance is regarded as an overhead to the user

Implementing the steps necessary to ensure quality systems could incur greater development, cost and delayed delivery dates. However long-term benefits of this must be considered such as *reduced maintenance costs and increased user satisfaction*.

SOFTWARE IMPLEMENTATION AND MAINTENANCE

Software implementation

In this chapter, we will study about programming methods, documentation and challenges in software implementation.

Structured Programming

In the process of coding, the lines of code keep multiplying, thus, size of the software increases. Gradually, it becomes next to impossible to remember the flow of program. If one forgets how software and its underlying programs, files, procedures are constructed it then becomes very difficult to share, debug and modify the program. The solution to this is structured programming. It encourages the developer to use subroutines and loops instead of using simple jumps in the code, thereby bringing clarity in the code and improving its efficiency Structured programming also helps programmer to reduce coding time and organize code properly.

Functional Programming

Functional programming is style of programming language, which uses the concepts of mathematical functions. A function in mathematics should always produce the same result on receiving the same argument. In procedural languages, the flow of the program runs through procedures, i.e. the control of program is transferred to the called procedure. While control flow is transferring from one procedure to another, the program changes its state.

In procedural programming, it is possible for a procedure to produce different results when it is called with the same argument, as the program itself can be in different state while calling it. This is a property as well as a drawback of procedural programming, in which the sequence or timing of the procedure execution becomes important.

Functional programming provides means of computation as mathematical functions, which produces results irrespective of program state. This makes it possible to predict the behavior of the program.

Programming style

Programming style is set of coding rules followed by all the programmers to write the code. When multiple programmers work on the same software project, they frequently need to work with the program code written by some other developer. This becomes tedious or at times impossible, if all developers do not follow some standard programming style to code the program.

An appropriate programming style includes using function and variable names relevant to the intended task, using well-placed indentation, commenting code for the convenience of reader and overall presentation of code. This makes the program code readable and understandable by all, which in turn makes debugging and error solving easier. Also, proper coding style helps ease the documentation and updation.

Coding Guidelines

Practice of coding style varies with organizations, operating systems and language of coding itself.

The following coding elements may be defined under coding guidelines of an organization:

- **Naming conventions** - This section defines how to name functions, variables, constants and global variables.
- **Indenting** - This is the space left at the beginning of line, usually 2-8 whitespace or single tab.
- **Whitespace** - It is generally omitted at the end of line.
- **Operators** - Defines the rules of writing mathematical, assignment and logical operators. For example, assignment operator '=' should have space before and after it, as in "x = 2".
- **Control Structures** - The rules of writing if-then-else, case-switch, while-until and for control flow statements solely and in nested fashion.
- **Line length and wrapping** - Defines how many characters should be there in one line, mostly a line is 80 characters long. Wrapping defines how a line should be wrapped, if is too long.
- **Functions** - This defines how functions should be declared and invoked, with and without parameters.
- **Variables** - This mentions how variables of different data types are declared and defined.
- **Comments** - This is one of the important coding components, as the comments included in the code describe what the code actually does and all other associated descriptions. This section also helps creating help documentations for other developers.

Software Documentation

Software documentation is an important part of software process. A well written document provides a great tool and means of information repository necessary to know about software process. Software documentation also provides information about how to use the product.

A well-maintained documentation should involve the following documents:

- **Requirement documentation** - This documentation works as key tool for software designer, developer and the test team to carry out their respective tasks. This document contains all the functional, non-functional and behavioral description of the intended software.

  Source of this document can be previously stored data about the software, already running software at the client's end, client's interview, questionnaires and research. Generally it is stored in the form of spreadsheet or word processing document with the high-end software management team.

  This documentation works as foundation for the software to be developed and is majorly used in verification and validation phases. Most test-cases are built directly from requirement documentation.

- **Software Design documentation** - These documentations contain all the necessary information, which are needed to build the software. It contains:

  (a) High-level software architecture,
  (b) Software design details,
  (c) Data flow diagrams,
  (d) Database design

  These documents work as repository for developers to implement the software. Though these documents do not give any details on how to code the program, they give all necessary information that is required for coding and implementation.

- **Technical documentation** - These documentations are maintained by the developers and actual coders. These documents, as a whole, represent information about the code. While writing the code, the programmers also mention objective of the code, who wrote it, where will it be required, what it does and how it does, what other resources the code uses, etc.

  The technical documentation increases the understanding between various programmers working on the same code. It enhances re-use capability of the code. It makes debugging easy and traceable.

  There are various automated tools available and some comes with the programming language itself. For example java comes JavaDoc tool to generate technical documentation of code.

- **User documentation** - This documentation is different from all the above explained. All previous documentations are maintained to provide information about the software and its development process. But user documentation explains how the software product should work and how it should be used to get the desired results.

  These documentations may include, software installation procedures, how-to guides, user-guides, uninstallation method and special references to get more information like license updation etc.

Software Implementation Challenges

There are some challenges faced by the development team while implementing the software. Some of them are mentioned below:

- **Code-reuse** - Programming interfaces of present-day languages are very sophisticated and are equipped huge library functions. Still, to bring the cost down of end product, the organization management prefers to re-use the code, which was created earlier for some other software. There are huge issues faced by programmers for compatibility checks and deciding how much code to re-use.
- **Version Management** - Every time a new software is issued to the customer, developers have to maintain version and configuration related documentation. This documentation needs to be highly accurate and available on time.

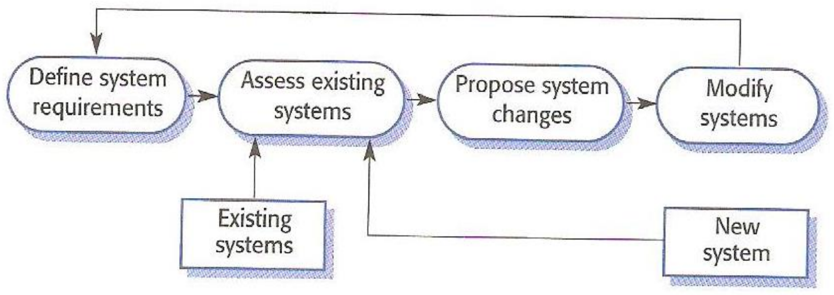
- **Target-Host** - The software program, which is being developed in the organization, `needs to be designed for host machines at the customers end. But at times, it is impossible to design a software that works on the target machines.

Software Maintenance Overview

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.

Software is inherently flexible and can change. As requirements change through changing business circumstances, the software that supports the business must also evolve and change. Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new. The figure below shows system evolution process.



SYSTEM BUILDING

Is the process of compiling and linking software components into an executable system.

Different systems are built from different combinations of components. This process is now always supported by automated tools that are driven by ‘build scripts’. These may be separate tools or part of the programming language environment (as in Java).

System building problems

- Do the build instructions include all required components?
- When there are many hundreds of components making up a system, it is easy to miss one out. This should normally be detected by the linker.
- Is the appropriate component version specified?
- A more significant problem. A system built with the wrong version may work initially but fail after delivery.
- Are all data files available?
- The build should not rely on 'standard' data files. Standards vary from place to place.

CASE tools are available to support all configuration management activities. These include tools such as RCS to manage versions, tools to support change management and system building tools. CASE tools for CM may be stand-alone tools supporting change management, version management and system building or may be integrated systems which provide a single interface to all CM support.

CHANGE MANAGEMENT

Software systems are subject to continual change requests:

i. From users;
ii. From developers;
iii. From market forces.

Change management is concerned with keeping track of these changes and ensuring that they are implemented in the most cost-effective way.

Change management requirements

- Some means for users and developers to suggest required system changes
- A process to decide if changes should be included in a system
- Software to keep track of suggested changes and their status

•Software support for managing changing system configurations and building new systems.

Software change is very important because organizations are now completely dependent on their software systems and have invested millions of money in these systems. These systems are critical business assets and they must invest in system change to maintain the value of these assets.

Changes should be reviewed by an external group who decide whether or not they are cost-effective from a strategic and organizational viewpoint rather than a technical viewpoint. It should be independent of project team responsible for system. The group is sometimes called a change control board.

1.1: <u>The demand for a configuration or engineering change can be generated either from within the company or externally from the customer or a supplier in order to:</u>

- Correct a drawing or engineering document error
- Correct a usability, reliability or safety problem
- Fix a bug or product defect
- Improve performance and/or functionality
- Improve productivity
- Lower cost
- Incorporate new customer requirements
- Specify a new supplier or supplier part/material
- ‘Enhance installation, service, or maintenance
- Respond to regulatory requirements

When a change is being evaluated, the following must be considered:

i. Inventory status of the new and old item. How many of the old item are in inventory? Must they be scrapped or can they be used on other products or reworked? What is the cost to rework or scrap? Is the new item in inventory?
ii. Production status of the new and old item. How many of the old items are in work-in-process? Can they be reworked to the new configuration considering their current stage of completion, completed and used up before the change is effective, or must they be scrapped? Has production of the new items begun? What is the leadtime and cost for production of the new item? What is the additional leadtime for building tooling, fixtures and test equipment?
iii. Procurement status of the new and old item. Is the old item on order? Can it be cancelled or reworked? At what cost? What is the leadtime for procuring the new item? Are new suppliers required?
iv. The impact on the distributors, dealers, customers and field service organizations. What notification is required? How long will the process take? What documentation, manuals and catalogs need to be updated? What are the implications on spare parts requirements?
v. Testing and regulatory requirements. Are the changes significant enough to require retesting? What testing needs to be performed? Does the product need to be recertified? What regulatory approvals are required?

There are a number of different strategies for software change:

i. *Software maintenance*

ii. *Architectural transformation*

iii. *Software reengineering*

i. ***Software maintenance***

Is the general process of changing a system after it has been delivered. The changes may be simple changes to correct coding errors, more extensive changes to correct design errors or significant enhancement to correct specification errors or accommodate new requirements.

Types of maintenance are

- Corrective maintenance- maintenance to repair software defects/ faults.
- Adaptive maintenance- maintenance to adapt the software to a different operating environment- such as hardware, the platform OS or other support software systems.
- Perfective maintenance – To add or modify a system's functionality especially when requirements change in response to organizational or business change.
- Preventive maintenance –To ensure the system can withstand stress.

In practice, there isn't a clear-cut distinction between these different types of maintenance.

The costs of system maintenance represent a large proportion of the budget of most organizations that use software systems. It is usually cost-effective to invest effort when designing and implementing a system to reduce maintenance costs. Good software engineering techniques all contribute to maintenance cost reduction

ii. ***Architectural transformation***

It involves making significant changes to the architecture of the software system. Most commonly systems evolve from a centralized, data-centric architecture to client-server architecture.

iii. ***Software reengineering***

The system is modified without adding any new functionality to make it easier to understand and change. It may involve some structural modifications but does not usually involve major architectural change.

Software reengineering is concerned with re-implementing legacy systems to make them more maintainable. It may involve redocumenting the system, organizing and restructuring the system and translating the system to more modern programming language and modifying and updating the structure and values of the system's data. The software's functionality is not changed and normally the system's architecture remains the same.

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.
- **Restructure Program** if required. For example, changing function-oriented programs into object-oriented programs.
- **Re-structure data** as required.

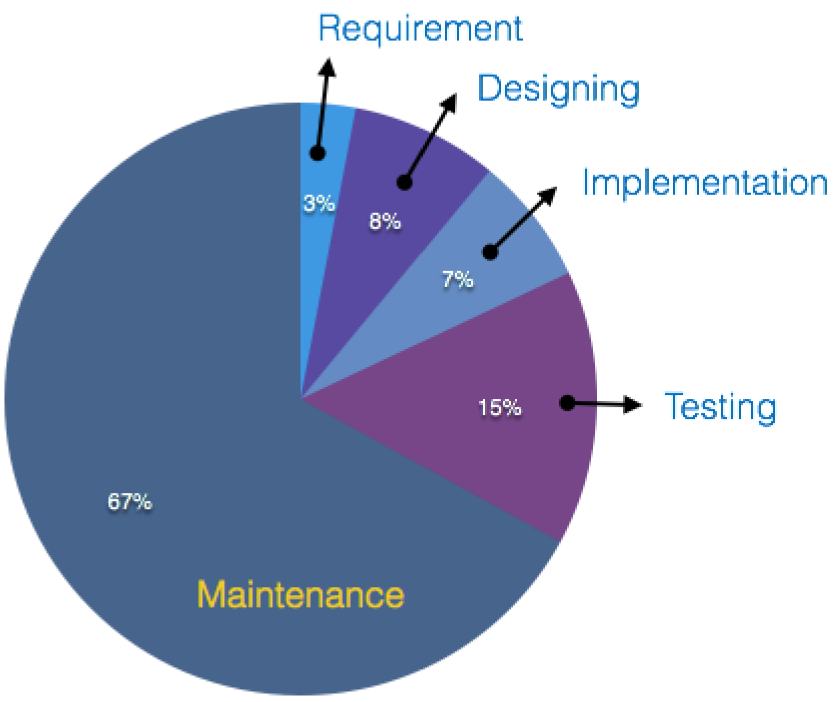
- **Apply Forward engineering** concepts in order to get re-engineered software.

Reengineering a software system has key advantages over radical approaches to system evolution:

- Reduced risk: There is a high risk in redeveloping software that is essential for an organization. Errors may be made in the system specification, there may be development problems etc.
- Reduced cost: The cost of re-engineering is significantly less than the cost of developing new software.

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.



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

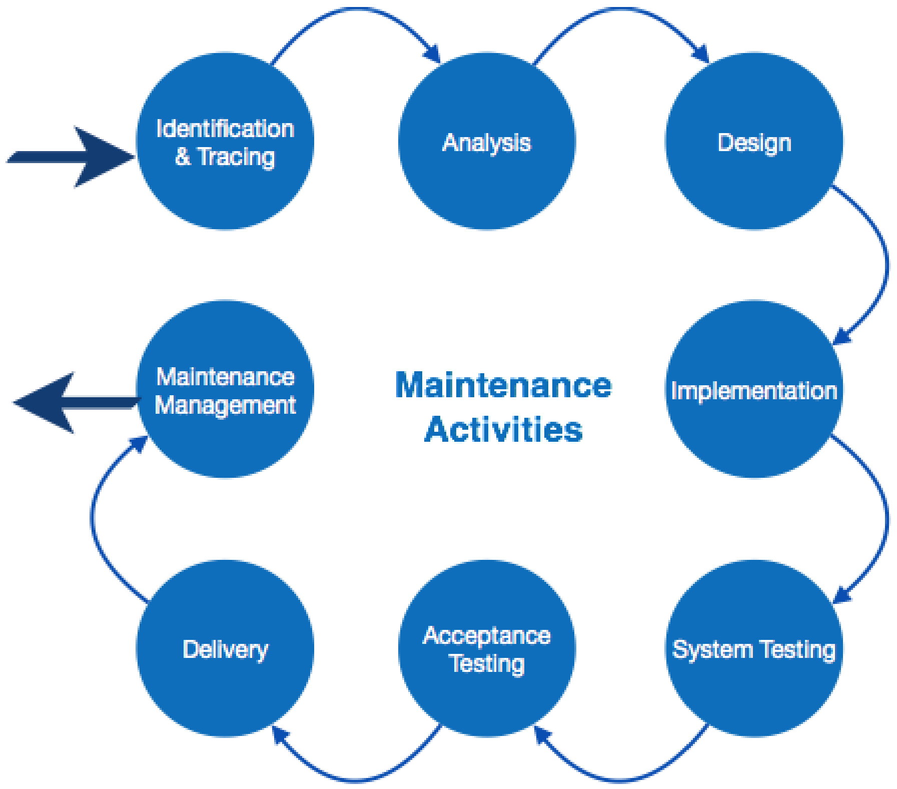
- The standard age of any software is considered up to 10 to 15 years.
- Older softwares, which were meant to work on slow machines with less memory and storage capacity cannot keep themselves challenging against newly coming enhanced softwares on modern hardware.
- As technology advances, it becomes 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

- 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 COST ESTIMATION

Factors affecting Software pricing

Factor	**Description**
Market opportunity	A development org. may quote low price because it wishes to move into a new segment of software market. Accepting a low profit on one project may give the opportunity of more profits later.
Cost estimates uncertainty	If an org is unsure of its cost estimate; it may increase its price by some contingency over and above its normal profit.
Contractual terms	A customer may be willing to allow the developer to retain ownership of the source code and reuse it in other projects. The price charged may then be less than if the software source code is handed over to the customer
Requirements volatility	If the requirements are likely to change, an organization may lower its price to win a contract. After the contract is awarded, high prices maybe charged for changes to the requirements.

Financial health	Developers in financial difficulty may lower their prices to gain a contract

There are three parameters involved in computing the total cost of software cost of a software development project:

- Hardware and software costs, including maintenance.
- Travel and training costs;
- Efforts costs (the cost of paying software engineers.)

Software costing should be carried out objectively with the aim of accurately predicting the cost to the contractor of developing the software.

Software price is not just price plus profit. Factors shown in the above table must be considered. Hence, software pricing must consider broader organizational, economic, political and business considerations.

Productivity

Productivity in a manufacturing system can be measured by counting the number of units which are produced and dividing this by the number of persons-hours required to produce them. However for any software problem, there are many different solutions which have different attributes..

Factors that affect productivity include individuals' aptitude, domain experience, the development process, the size of the project, tool support and the working environment.

There are 2 types of measures that may be used:

Size related measures that are related to the size of some output from an activity i.e. lines of delivered source code. And number of delivered object code instructions or the number of pages of system documentation.

Function-related measures that are related to the overall functionality of the delivered software. Examples are function units and object points

Cost estimation techniques

Techniques	**Description**
Algorithmic modelling	A model is developed using historical cost information that relates some software metric(usually its size) to the project cost. An estimate is made of that metric and the model predicts the efforts required
Expert judgement	Several experts on the proposed software development techniques and the application domain are consulted. They each estimated the project cost. These estimates are compared and discussed. The estimation process iterates until

	an agreed estimates is reached.
Estimation by analogy	The technique is applicable when other projects in the same application domain have been completed. The cost of a new project is estimated by analogy with these completed projects
Parkinson’s law	The law states that work expands to fill the time available. The cost is determined by available resources rather than by objective assessment. If software has to deliver in 12 months and 5 people are available, the effort required is estimated to be 90 person-months.
Pricing to win	The software cost is estimated to be whatever the customer has available to spend on the project.

Algorithmic cost modelling

The most systematic, although not necessarily the most accurate, approach to software estimation is algorithmic cost estimation. An algorithmic cost model can be built by analysing the costs based on estimates of project size, number of programmers and other process and product factors.

In most general form, an algorithmic cost estimation cost estimate fro software cost can be expressed as:

Effort = A *SizeB * M

A is a constant factor that depends on local organizational practices and the type of software developed.

Size may be either an assessment of the code size of the software or functionality estimates expressed in function or object points.

The value of the exponent *B* reflects the disproportionate effort required for large projects and usually lies between 1 and 1.5.

M is a multiplier made up by combining different process, product and development attributes.

Difficulties of algorithmic models are.

It is often difficult to estimate Size at an early stage in the project where only a speciation is available Function point and object point estimates are easier to produce than estimates of code size but may still be inaccurate.

The estimates of factors contributing to B and M are subjective. Estimates vary from one person to another depending on their background and experience.

The COCOMO model

There are a number of algorithmic models that have been proposed as a basis for estimating the effort, schedule and costs of a software project. The COCOMO model is an empirical model. It was derived by collecting data from a large number of software projects, then analysing that data to discover formulae that were the best-fit to the observations. COCOMO:

- Is well documented, in the public domain and is supported by public domain and commercial tools.
- It has widely used and evaluated.

The first version of the COCOMO model (known as COCOMO 81) was a three-level model where the levels reflected the detail of the analysis of the cost estimate.
The first level-basic provided an initial, rough estimate.
The second level modified this using a number of project and process multiplier.
The most detailed level produced estimates for different phases of the project
The figure below shows the basic COCOMO formula for different types of project.
COCOMO 81 assumed that the software would be developed according to a water-fall process and that the vast majority of the software would be developed from scratch.

Project complexity	**Formula**	**Description**
Simple	PM= 2.4(KDSI)1.05*M	Well understood applications developed by small teams.
Moderate	PM= 3.0(KDSI)1.12*M	More complex projects where team members may have limited experience of related systems.
Embedded	PM= 3.6(KDSI)1.20*M	Complex projects where the software is part of a strongly coupled complex of hardware, software, regulations and operational procedures

COCOMO 2 model recognizes different approaches to software development such as prototyping, development by component composition, use of 4GLs etc.

The model levels do not simply reflect increasingly complex and detailed estimates. Levels are associated with activities in the software process so that initial estimates may be made early in the process with more detailed estimating carried out after the system architecture has been defined.

PROJECT MANAGEMENT

A Project is a sequence of unique, complex activities having one goal and that must be completed by specific time, within budget and according to specifications.

Project management is the process of defining, planning, monitoring, staffing, estimating, scheduling, organizing, directing, scooping, and controlling the development of an acceptable system at a minimum cost within a specified time frame.

For any system development project, project management is necessary to ensure that the project meets the deadline, is developed within an acceptable budget and fulfill expectations and specifications. Project management is a cross life activity that’s its tasks overlap all system development.

Project planning is an attempt to develop accurate estimates for the time, costs and expected benefits and to adjust the estimates if they prove inaccurate.

Project control is the process of monitoring the actual project to see whether it is meeting the objectives detailed in the project plan.

It is taking of corrective actions when activities fall behind schedule or more resources are needed to complete the project.

A Project can be characterized as:

- Every project may has a unique and distinct goal.
- Project is not routine activity or day-to-day operations.
- Project comes with a start time and end time.
- Project ends when its goal is achieved hence it is a temporary phase in the lifetime of an organization.
- Project needs adequate resources in terms of time, manpower, finance, material and knowledge-bank.

The following factors if not well addressed could cause expensive disaster and eventual **failure of the project.**

i. **Inadequate user involvement:**

Perhaps users are not allowed to educate analyst on policies, requirements and applications.

ii. **User changing their requirements**: Users change their requirements resulting in costly changes to the systems. This is due to the fact that users needs were not known early enough.
iii. **Lack of user training:** User training is essential for the success and acceptance of the new system.
iv. **Lack of communication:** Improper communication among users, management and information specialists.
v. **Continuation of projects that should have been cancelled.** These are unviable projects.
vi. **Politics:** A successful outcome is unlikely in a project environment characterized by political maneuvering and hidden motives.
vii. **Lack of management support**: lack of this support makes low level employees to lose confidence in and enthusiasm for the project and whole work may fail.
viii. **Technological incompetence:** Perhaps the skills of the people working on the project are not up to the challenge.
ix. **Lack of technical resources** e.g. hardware or networks required for the successful implementation and operation of the system
x. **Costs inflates** i.e costs exceed the budget. Sometimes the budget may be underestimated.
xi. **Time frame** exceeds the projected project schedule.
xii. **Change of objectives**

Information makes it easier to provide better service to major customers.

Factors of success during IS implementation is:

i. Organizational alignment.
ii. Management support and support from other stakeholders.
iii. Process change management.
iv. Sufficient interaction between users and developers.
v. Motivated and trained users.

Measures of project success include:

i. The resulting information system is acceptable to the clients/users.
ii. The system is delivered on time.
iii. The system is delivered within budget.
iv. Development process had minimum impact on ongoing business operations.
v. The system complied with the user requirements.

Why Software project management is different from other disciplines:

- The product is intangible.
- The product is uniquely flexible.
- Software engineering is not recognized as an engineering discipline with the sane status as mechanical, electrical engineering, etc.
- The software development process is not standardised.
- Many software projects are 'one-off' projects.

Software Management Activities

Software project management comprises of a number of activities, which contains planning of project, deciding scope of software product, estimation of cost in various terms, scheduling of tasks and events, and resource management. Project management activities may include:

- **Project Planning**
- **Scope Management**
- **Project Estimation**

Project Planning

Software project planning is task, which is performed before the production of software actually starts. It is there for the software production but involves no concrete activity that has any direction connection with software production; rather it is a set of multiple processes, which facilitates software production. Project planning may include the following: xxxxxxxxxx

Scope Management

It defines the scope of project; this includes all the activities, process need to be done in order to make a deliverable software product. Scope management is essential because it creates boundaries of the project by clearly defining what would be done in the project and what would not be done. This makes project to contain limited and quantifiable tasks, which can easily be documented and in turn avoids cost and time overrun.

During Project Scope management, it is necessary to -

- Define the scope

- Decide its verification and control
- Divide the project into various smaller parts for ease of management.
- Verify the scope
- Control the scope by incorporating changes to the scope

Project Estimation

For an effective management accurate estimation of various measures is a must. With correct estimation managers can manage and control the project more efficiently and effectively.

Project estimation may involve the following:

- **Software size estimation**

  Software size may be estimated either in terms of KLOC (Kilo Line of Code) or by calculating number of function points in the software. Lines of code depend upon coding practices and Function points vary according to the user or software requirement.

- **Effort estimation**

  The managers estimate efforts in terms of personnel requirement and man-hour required to produce the software. For effort estimation software size should be known. This can either be derived by managers' experience, organization's historical data or software size can be converted into efforts by using some standard formulae.

- **Time estimation**

  Once size and efforts are estimated, the time required to produce the software can be estimated. Efforts required is segregated into sub categories as per the requirement specifications and interdependency of various components of software. Software tasks are divided into smaller tasks, activities or events by Work Breakthrough Structure (WBS). The tasks are scheduled on day-to-day basis or in calendar months.

  The sum of time required to complete all tasks in hours or days is the total time invested to complete the project.

- **Cost estimation**

  This might be considered as the most difficult of all because it depends on more elements than any of the previous ones. For estimating project cost, it is required to consider -

  - Size of software
  - Software quality
  - Hardware
  - Additional software or tools, licenses etc.
  - Skilled personnel with task-specific skills
  - Travel involved
  - Communication
  - Training and support

Project Estimation Techniques

We discussed various parameters involving project estimation such as size, effort, time and cost.

Project manager can estimate the listed factors using two broadly recognized techniques –

Decomposition Technique

This technique assumes the software as a product of various compositions.

There are two main models -

- **Line of Code** Estimation is done on behalf of number of line of codes in the software product.
- **Function Points** Estimation is done on behalf of number of function points in the software product.

Empirical Estimation Technique

This technique uses empirically derived formulae to make estimation.These formulae are based on LOC or FPs.

- **Putnam Model**

  This model is made by Lawrence H. Putnam, which is based on Norden's frequency distribution (Rayleigh curve). Putnam model maps time and efforts required with software size.

- **COCOMO**

  COCOMO stands for COnstructive COst MOdel, developed by Barry W. Boehm. It divides the software product into three categories of software: organic, semi-detached and embedded.

Project Scheduling

Project Scheduling in a project refers to roadmap of all activities to be done with specified order and within time slot allotted to each activity. Project managers tend to define various tasks, and project milestones and them arrange them keeping various factors in mind. They look for tasks lie in critical path in the schedule, which are necessary to complete in specific manner (because of task interdependency) and strictly within the time allocated. Arrangement of tasks which lies out of critical path are less likely to impact over all schedule of the project.

For scheduling a project, it is necessary to -

- Break down the project tasks into smaller, manageable form
- Find out various tasks and correlate them
- Estimate time frame required for each task
- Divide time into work-units
- Assign adequate number of work-units for each task

- Calculate total time required for the project from start to finish

Resource management

All elements used to develop a software product may be assumed as resource for that project. This may include human resource, productive tools and software libraries.

The resources are available in limited quantity and stay in the organization as a pool of assets. The shortage of resources hampers the development of project and it can lag behind the schedule. Allocating extra resources increases development cost in the end. It is therefore necessary to estimate and allocate adequate resources for the project.

Resource management includes -

- Defining proper organization project by creating a project team and allocating responsibilities to each team member
- Determining resources required at a particular stage and their availability
- Manage Resources by generating resource request when they are required and de-allocating them when they are no more needed.

Project Risk Management

Risk management involves all activities pertaining to identification, analyzing and making provision for predictable and non-predictable risks in the project. Risk may include the following:

- Experienced staff leaving the project and new staff coming in.
- Change in organizational management.
- Requirement change or misinterpreting requirement.
- Under-estimation of required time and resources.
- Technological changes, environmental changes, business competition.

Risk Management Process

There are following activities involved in risk management process:

- **Identification -** Make note of all possible risks, which may occur in the project.
- **Categorize -** Categorize known risks into high, medium and low risk intensity as per their possible impact on the project.
- **Manage -** Analyze the probability of occurrence of risks at various phases. Make plan to avoid or face risks. Attempt to minimize their side-effects.
- **Monitor -** Closely monitor the potential risks and their early symptoms. Also monitor the effects of steps taken to mitigate or avoid them.

Project Execution & Monitoring

In this phase, the tasks described in project plans are executed according to their schedules.

Execution needs monitoring in order to check whether everything is going according to the plan. Monitoring is observing to check the probability of risk and taking measures to address the risk or report the status of various tasks.

These measures include -

- **Activity Monitoring -** All activities scheduled within some task can be monitored on day-to-day basis. When all activities in a task are completed, it is considered as complete.
- **Status Reports -** The reports contain status of activities and tasks completed within a given time frame, generally a week. Status can be marked as finished, pending or work-in-progress etc.
- **Milestones Checklist -** Every project is divided into multiple phases where major tasks are performed (milestones) based on the phases of SDLC. This milestone checklist is prepared once every few weeks and reports the status of milestones.

Project Communication Management

Effective communication plays vital role in the success of a project. It bridges gaps between client and the organization, among the team members as well as other stake holders in the project such as hardware suppliers.

Communication can be oral or written. Communication management process may have the following steps:

- **Planning** - This step includes the identifications of all the stakeholders in the project and the mode of communication among them. It also considers if any additional communication facilities are required.
- **Sharing** - After determining various aspects of planning, manager focuses on sharing correct information with the correct person on correct time. This keeps everyone involved the project up to date with project progress and its status.
- **Feedback** - Project managers use various measures and feedback mechanism and create status and performance reports. This mechanism ensures that input from various stakeholders is coming to the project manager as their feedback.
- **Closure** - At the end of each major event, end of a phase of SDLC or end of the project itself, administrative closure is formally announced to update every stakeholder by sending email, by distributing a hardcopy of document or by other mean of effective communication.

After closure, the team moves to next phase or project.

Configuration Management

Configuration management is a process of tracking and controlling the changes in software in terms of the requirements, design, functions and development of the product.

IEEE defines it as “the process of identifying and defining the items in the system, controlling the change of these items throughout their life cycle, recording and reporting the status of items and change requests, and verifying the completeness and correctness of items”.

Generally, once the SRS is finalized there is less chance of requirement of changes from user. If they occur, the changes are addressed only with prior approval of higher management, as there is a possibility of cost and time overrun.

Baseline

A phase of SDLC is assumed over if it baselined, i.e. baseline is a measurement that defines completeness of a phase. A phase is baselined when all activities pertaining to it are finished and well documented. If it was not the final phase, its output would be used in next immediate phase.

Configuration management is a discipline of organization administration, which takes care of occurrence of any change (process, requirement, technological, strategical etc.) after a phase is baselined. CM keeps check on any changes done in software.

Change Control

Change control is function of configuration management, which ensures that all changes made to software system are consistent and made as per organizational rules and regulations.

A change in the configuration of product goes through following steps -

- **Identification** - A change request arrives from either internal or external source. When change request is identified formally, it is properly documented.
- **Validation** - Validity of the change request is checked and its handling procedure is confirmed.
- **Analysis** - The impact of change request is analyzed in terms of schedule, cost and required efforts. Overall impact of the prospective change on system is analyzed.
- **Control** - If the prospective change either impacts too many entities in the system or it is unavoidable, it is mandatory to take approval of high authorities before change is incorporated into the system. It is decided if the change is worth incorporation or not. If it is not, change request is refused formally.
- **Execution** - If the previous phase determines to execute the change request, this phase take appropriate actions to execute the change, does a thorough revision if necessary.
- **Close request** - The change is verified for correct implementation and merging with the rest of the system. This newly incorporated change in the software is documented properly and the request is formally is closed.

Project Management Tools

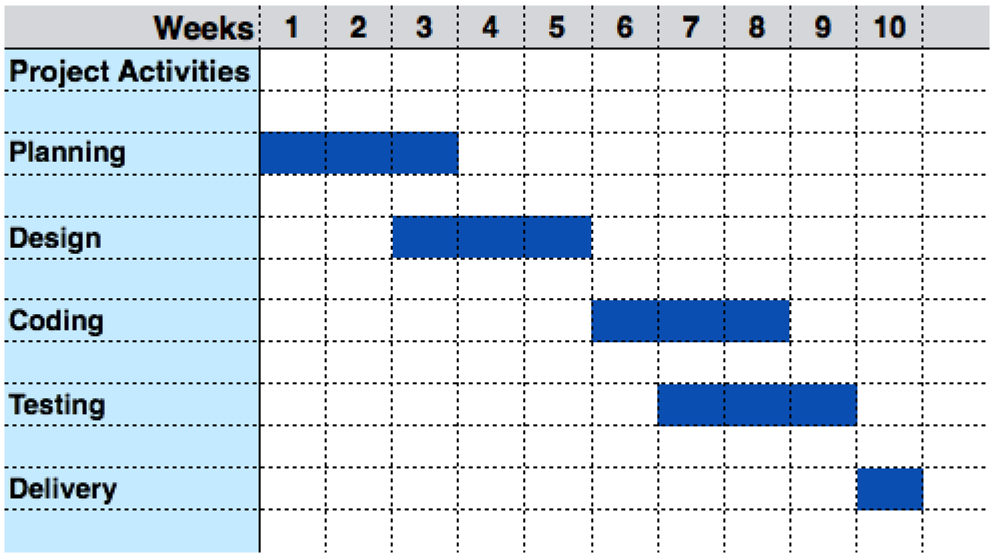
The risk and uncertainty rises multifold with respect to the size of the project, even when the project is developed according to set methodologies.

There are tools available, which aid for effective project management. A few are described –

- Gantt chart
- PERT(Program Evaluation & Review Technique)
- Resource histogram
- Critical path analysis

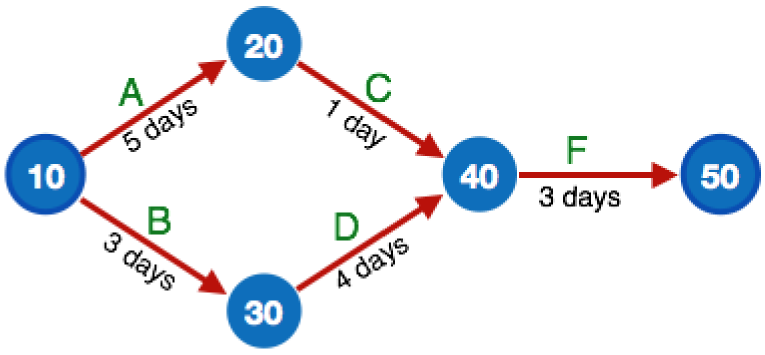
Gantt Chart

Gantt charts was devised by Henry Gantt (1917). It represents project schedule with respect to time periods. It is a horizontal bar chart with bars representing activities and time scheduled for the project activities.



PERT Chart

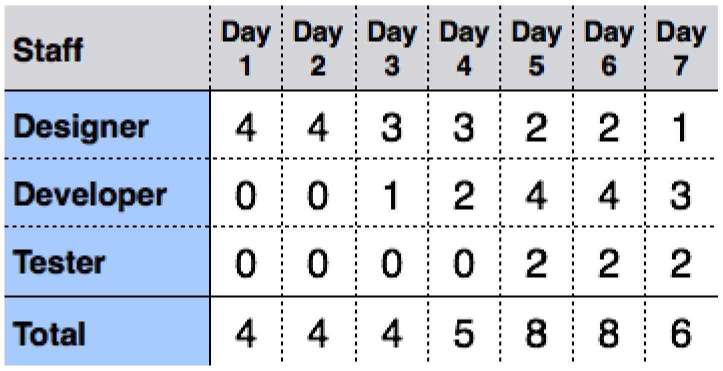
PERT (Program Evaluation & Review Technique) chart is a tool that depicts project as network diagram. It is capable of graphically representing main events of project in both parallel and consecutive way. Events, which occur one after another, show dependency of the later event over the previous one.



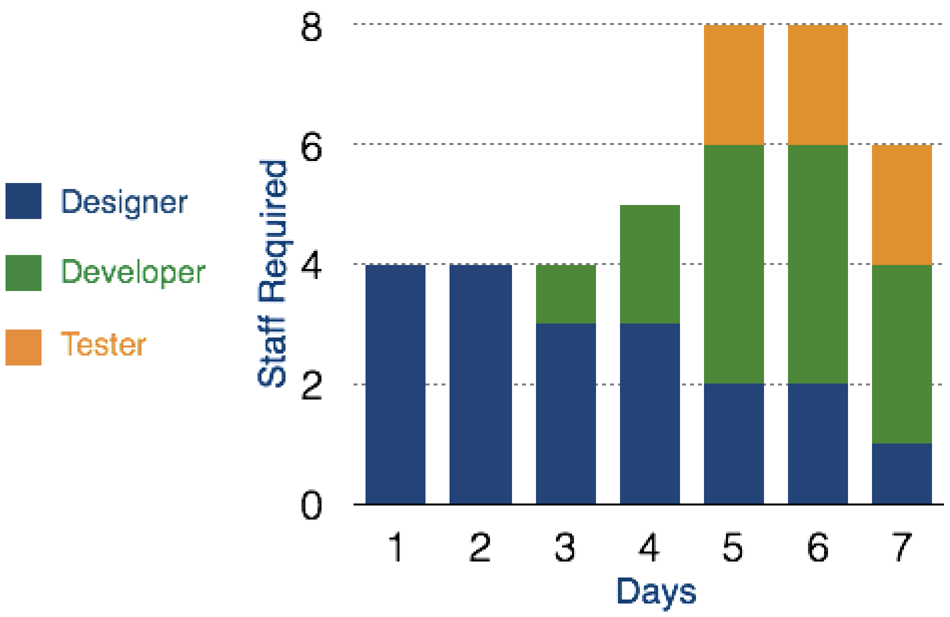
Events are shown as numbered nodes. They are connected by labeled arrows depicting sequence of tasks in the project.

Resource Histogram

This is a graphical tool that contains bar or chart representing number of resources (usually skilled staff) required over time for a project event (or phase). Resource Histogram is an effective tool for staff planning and coordination.



Staff	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
Designer	4	4	3	3	2	2	1
Developer	0	0	1	2	4	4	3
Tester	0	0	0	0	2	2	2
Total	4	4	4	5	8	8	6



Critical Path Analysis

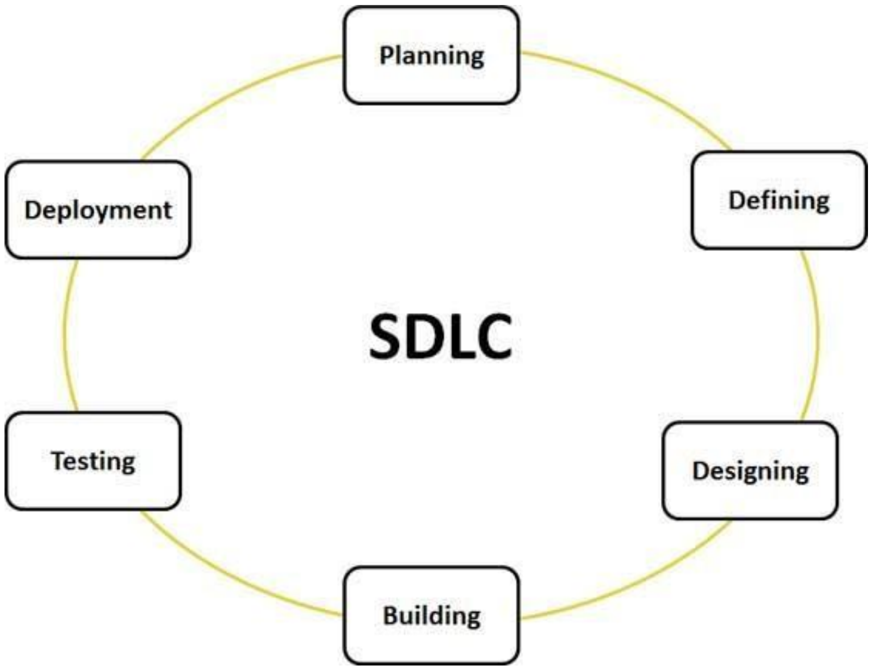
This tools is useful in recognizing interdependent tasks in the project. It also helps to find out the shortest path or critical path to complete the project successfully. Like PERT diagram, each event is allotted a specific time frame. This tool shows dependency of event assuming an event can proceed to next only if the previous one is completed.

The events are arranged according to their earliest possible start time. Path between start and end node is critical path which cannot be further reduced and all events require to be executed in same order.

SYSTEM DEVELOPMENT LIFE CYCLE

SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

The following figure is a graphical representation of the various stages of a typical SDLC.



A typical Software Development life cycle consists of the following stages:

Stage 1: *Planning* and Requirement Analysis : Requirement analysis is the most important and fundamental stage in SDLC. It is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational, and technical areas.

Planning for the quality assurance requirements and identification of the risks associated with the project is also done in the planning stage. The outcome of the technical feasibility study is to define the various technical approaches that can be followed to implement the project successfully with minimum risks.

Stage 2: *Defining* Requirements : Once the requirement analysis is done the next step is to clearly define and document the product requirements and get them approved from the customer or the market analysts. This is done through 'SRS' – Software Requirement

Specification document which consists of all the product requirements to be designed and developed during the project life cycle.

Stage 3: *Designing* the product architecture : SRS is the reference for product architects to come out with the best architecture for the product to be developed. Based on the requirements specified in SRS, usually more than one design approach for the product architecture is proposed and documented in a DDS - Design Document Specification. This DDS is reviewed by all the important stakeholders and based on various parameters as risk assessment, product robustness, design modularity , budget and time constraints , the best design approach is selected for the product.

A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS.

Stage 4: *Building* or Developing the Product : In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.

Developers have to follow the coding guidelines defined by their organization and programming tools like compilers, interpreters, debuggers etc are used to generate the code. Different high level programming languages such as C, C++, Pascal, **Java**, and PHP are used for coding. The programming language is chosen with respect to the type of software being developed.

Stage 5: *Testing* the Product : This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities are mostly involved in all the stages of SDLC. However this stage refers to the testing only stage of the product where products defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS.

Stage 6: *Deployment* in the Market and Maintenance : Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometime product deployment happens in stages as per the organizations' business strategy. The product may first be released in a limited segment and tested in the real business environment (UAT- User acceptance testing).

Then based on the feedback, the product may be released as it is or with suggested enhancements in the targeting market segment. After the product is released in the market, its maintenance is done for the existing customer base.

SDLC MODELS

There are various software development life cycle models defined and designed which are followed during software development process. These models are also referred as "Software Development Process Models". Each process model follows a Series of steps unique to its type, in order to ensure success in process of software development. Following are the most important and popular SDLC models followed in the industry:

- Waterfall Model
- Iterative Model
- Spiral Model

The other related methodologies are Agile Model, RAD Model – Rapid Application Development and Prototyping Models.

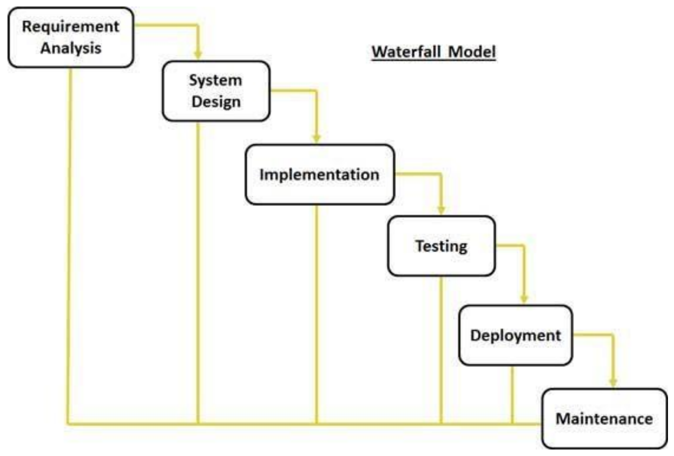
WATERFALL MODEL

Waterfall model is the earliest SDLC approach that was used for software development .The waterfall Model illustrates the software development process in a linear sequential flow; hence it is also referred to as a linear-sequential life cycle model. This means that any phase in the development process begins only if the previous phase is complete. In waterfall model phases do not overlap..

Waterfall Model design

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

Following is a diagrammatic representation of different phases of waterfall model.



The sequential phases in Waterfall model are:

- **Requirement Gathering and analysis** All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification doc.
- **System Design:** The requirement specifications from first phase are studied in this phase and system design is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture.
- **Implementation:** With inputs from system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality which is referred to as Unit Testing.
- **Integration and Testing:** All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of system:** Once the functional and non functional testing is done, the product is deployed in the customer environment or released into the market.
- **Maintenance:** There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

All these phases are cascaded to each other in which progress is seen as flowing steadily downwards (like a waterfall) through the phases. The next phase is started only after the defined set of goals are achieved for previous phase and it is signed off, so the name "Waterfall Model". In this model phases do not overlap.

Waterfall Model Application

Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are:

- Requirements are very well documented, clear and fixed
- Product definition is stable
- Technology is understood and is not dynamic
- There are no ambiguous requirements
- Ample resources with required expertise are available to support the product
- The project is short

Waterfall Model Pros & Cons

The advantage of waterfall development is that it allows for departmentalization and control. A schedule can be set with deadlines for each stage of development and a product can proceed through the development process model phases one by one. Development moves from concept, through design, implementation, testing, installation, troubleshooting, and ends up at operation and maintenance. Each phase of development proceeds in strict order. The disadvantage of waterfall development is that it does not allow for much reflection or revision. Once an application is in the testing stage, it is very difficult to go back and change something that was not well-documented or thought upon in the concept stage.

The following table lists out the pros and cons of Waterfall model:

Pros	**Cons**

▪ Simple and easy to understand and use. ▪ Easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process. ▪ Phases are processed and completed one at a time. ▪ Works well for smaller projects where requirements are very well understood. ▪ Clearly defined stages. ▪ Well understood milestones. ▪ Easy to arrange tasks. ▪ Process and results are well documented.	▪ No working software is produced until late during the life cycle. ▪ High amounts of risk and uncertainty. ▪ Not a good model for complex and object-oriented projects. ▪ Poor model for long and ongoing projects. ▪ Not suitable for the projects where requirements are at a moderate to high risk of changing. So risk and uncertainty is high with this process model. ▪ It is difficult to measure progress within stages. ▪ Cannot accommodate changing requirements. ▪ No working software is produced until late in the life cycle. ▪ Adjusting scope during the life cycle can end a project

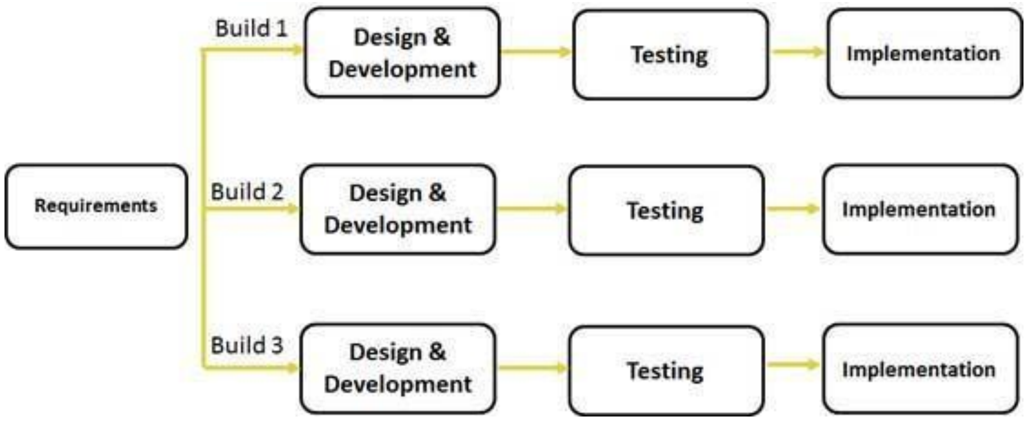
ITERATIVE MODEL

An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which is then reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software at the end of each iteration of the model.

Iterative Model design

Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented. At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).

Following is the pictorial representation of Iterative and Incremental model:



Iterative and Incremental development is a combination of both iterative design or iterative method and incremental build model for development. "During software development, more than one iteration of the software development cycle may be in progress at the same time." and "This process may be described as an "evolutionary acquisition" or "incremental build" approach."

In incremental model the whole requirement is divided into various builds. During each iteration, the development module goes through the requirements, design, implementation and testing phases. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is ready as per the requirement.

The key to successful use of an iterative software development lifecycle is rigorous validation of requirements, and verification & testing of each version of the software against those requirements within each cycle of the model. As the software evolves through successive cycles, tests have to be repeated and extended to verify each version of the software.

Iterative Model Application

Like other SDLC models, Iterative and incremental development has some specific applications in the software industry. This model is most often used in the following scenarios:

- Requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some functionalities or requested

enhancements may evolve with time.

- There is a time to the market constraint.
- A new technology is being used and is being learnt by the development team while working on the project.
- Resources with needed skill set are not available and are planned to be used on contract basis for specific iterations.
- There are some high risk features and goals which may change in the future.

Iterative Model Pros and Cons

The advantage of this model is that there is a working model of the system at a very early stage of development which makes it easier to find functional or design flaws. Finding issues at an early stage of development enables to take corrective measures in a limited budget.

The disadvantage with this SDLC model is that it is applicable only to large and bulky software development projects. This is because it is hard to break a small software system into further small serviceable increments/modules.

The following table lists out the pros and cons of Iterative and Incremental SDLC Model:

Pros	**Cons**

▪ Some working functionality can be developed quickly and early in the life cycle. ▪ Results are obtained early and periodically. ▪ Parallel development can be planned. ▪ Progress can be measured. ▪ Less costly to change the scope/requirements. ▪ Testing and debugging during smaller iteration is easy. ▪ Risks are identified and resolved during iteration; and each iteration is an easily managed milestone. ▪ Easier to manage risk - High risk part is done first. ▪ With every increment operational product is delivered. ▪ Issues, challenges & risks identified from each increment can be utilized/applied to the next increment. ▪ Risk analysis is better. ▪ It supports changing requirements. ▪ Initial Operating time is less. ▪ Better suited for large and mission- critical projects. ▪ During life cycle software is produced early which facilitates customer evaluation and feedback.	▪ More resources may be required. ▪ Although cost of change is lesser but it is not very suitable for changing requirements. ▪ More management attention is required. ▪ System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle. ▪ Defining increments may require definition of the complete system. ▪ Not suitable for smaller projects. ▪ Management complexity is more. ▪ End of project may not be known which is a risk. ▪ Highly skilled resources are required for risk analysis. ▪ Project's progress is highly dependent upon the risk analysis phase.

SPIRAL MODEL

Spiral model is a combination of iterative development process model and sequential linear development model i.e. waterfall model with very high emphasis on risk analysis. It allows for incremental releases of the product, or incremental refinement through each iteration around the spiral.

Spiral Model design

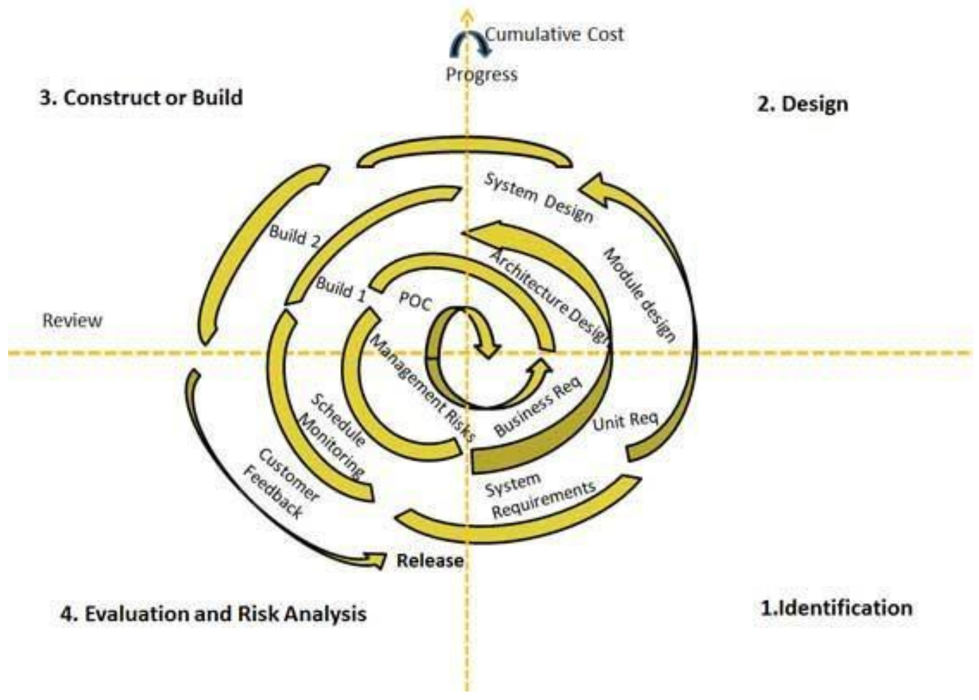
The spiral model has four phases. A software project repeatedly passes through these phases in iterations called Spirals.

- Identification

This phase starts with gathering the business requirements in the baseline spiral. In the subsequent spirals as the product matures, identification of system requirements, subsystem requirements and unit requirements are all done in this phase.

This also includes understanding the system requirements by continuous communication between the customer and the system analyst. At the end of the spiral the product is deployed in the identified market.

<u>Following is a diagrammatic representation of spiral model listing the activities in each phase</u>



- Design

Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and final design in the subsequent spirals.

- **Construct or Build**

Construct phase refers to production of the actual software product at every spiral. In the baseline spiral when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.

Then in the subsequent spirals with higher clarity on requirements and design details a working model of the software called build is produced with a version number. These builds are sent to customer for feedback.

- **Evaluation and Risk Analysis**

Risk Analysis includes identifying, estimating, and monitoring technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

Based on the customer evaluation, software development process enters into the next iteration and subsequently follows the linear approach to implement the feedback suggested by the customer. The process of iterations along the spiral continues throughout the life of the software.

Spiral Model Application

Spiral Model is very widely used in the software industry as it is in synch with the natural development process of any product i.e. learning with maturity and also involves minimum risk for the customer as well as the development firms. Following are the typical uses of Spiral model:

- When costs there is a budget constraint and risk evaluation is important
- For medium to high-risk projects
- Long-term project commitment because of potential changes to economic priorities as the requirements change with time

- Customer is not sure of their requirements which is usually the case
- Requirements are complex and need evaluation to get clarity
- New product line which should be released in phases to get enough customer feedback
- Significant changes are expected in the product during the development cycle

Spiral Model Pros and Cons

The advantage of spiral lifecycle model is that it allows for elements of the product to be added in when they become available or known. This assures that there is no conflict with previous requirements and design. This method is consistent with approaches that have multiple software builds and releases and allows for making an orderly transition to a maintenance activity. Another positive aspect is that the spiral model forces early user involvement in the system development effort.

On the other side, it takes very strict management to complete such products and there is a risk of running the spiral in indefinite loop. So the discipline of change and the extent of taking change requests is very important to develop and deploy the product successfully.

The following table lists out the pros and cons of Spiral SDLC Model:

Pros	Cons
▪ Changing requirements can be accommodated. ▪ Allows for extensive use of prototypes ▪ Requirements can be captured more accurately. ▪ Users see the system early. ▪ Development can be divided into smaller parts and more risky parts can be developed earlier which helps better risk management.	▪ Management is more complex. ▪ End of project may not be known early. ▪ Not suitable for small or low risk projects and could be expensive for small projects. ▪ Process is complex ▪ Spiral may go indefinitely. ▪ Large number of intermediate stages requires excessive documentation.

RAPID APPLICATION DEVELOPMENT

Rapid application development (RAD) is a software development methodology that uses minimal planning in favor of rapid prototyping. A prototype is a working model that is functionally equivalent to a component of the product. In RAD model the functional modules are developed in parallel as prototypes and are integrated to make the complete product for faster product delivery.

Since there is no detailed preplanning, it makes it easier to incorporate the changes within the development process. RAD projects follow iterative and incremental model and have small teams comprising of developers, domain experts, customer representatives and other IT resources working progressively on their component or prototype. The most important aspect for this model to be successful is to make sure that the prototypes developed are reusable.

RAD Model Design

RAD model distributes the analysis, design, build, and test phases into a series of short, iterative development cycles. Following are the phases of RAD Model:

- **Business Modeling:**

  The business model for the product under development is designed in terms of flow of information and the distribution of information between various business channels. A complete business analysis is performed to find the vital information for business, how it can be obtained, how and when is the information processed and what are the factors driving successful flow of information.

- **Data Modeling:**

The information gathered in the Business Modeling phase is reviewed and analyzed to form sets of data objects vital for the business. The attributes of all data sets is identified and defined. The relation between these data objects are established and defined in detail in relevance to the business model.

- **Process Modeling:**

The data object sets defined in the Data Modeling phase are converted to establish the business information flow needed to achieve specific business objectives as per the business model. The process model for any changes or enhancements to the data object sets is defined in this phase. Process descriptions for adding , deleting, retrieving or modifying a data object are given.

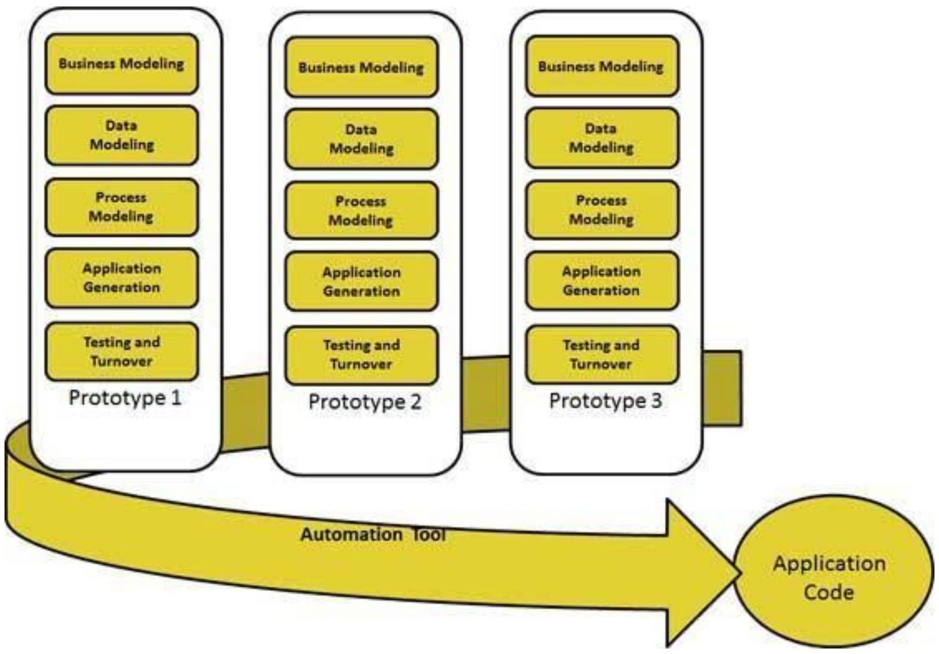
- **Application Generation:**

The actual system is built and coding is done by using automation tools to convert process and data models into actual prototypes.

- **Testing and Turnover:**

The overall testing time is reduced in RAD model as the prototypes are independently tested during every iteration. However the data flow and the interfaces between all the components need to be thoroughly tested with complete test coverage. Since most of the programming components have already been tested, it reduces the risk of any major issues.

Following image illustrates the RAD Model:



RAD Model Vs Traditional SDLC

The traditional SDLC follows a rigid process models with high emphasis on requirement analysis and gathering before the coding starts. It puts a pressure on the customer to sign off the requirements before the project starts and the customer doesn't get the feel of the product as there is no working build available for a long time.

The customer may need some changes after he actually gets to see the software, however the change process is quite rigid and it may not be feasible to incorporate major changes in the product in traditional SDLC.

RAD model focuses on iterative and incremental delivery of working models to the customer. This results in rapid delivery to the customer and customer involvement during the complete development cycle of product reducing the risk of non conformance with the actual user requirements.

RAD Model Application

RAD model can be applied successfully to the projects in which clear modularization is possible. If the project cannot be broken into modules, RAD may fail. Following are the typical scenarios where RAD can be used:

- RAD should be used only when a system can be modularized to be delivered in incremental manner.
- It should be used if there's high availability of designers for modeling
- It should be used only if the budget permits use of automated code generating tools.
- RAD SDLC model should be chosen only if domain experts are available with relevant business knowledge
- Should be used where the requirements change during the course of the project and working prototypes are to be presented to customer in small iterations of 2- 3 months.

RAD Model Pros and Cons

RAD model enables rapid delivery as it reduces the overall development time due to reusability of the components and parallel development.

RAD works well only if high skilled engineers are available and the customer is also committed to achieve the targeted prototype in the given time frame. If there is commitment lacking on either side the model may fail.

Following table lists out the pros and cons of RAD Model

Pros	Cons
▪ Changing requirements can be accommodated. ▪ Progress can be measured. ▪ Iteration time can be short with use of powerful RAD tools. ▪ Productivity with fewer people in short time. ▪ Reduced development time. ▪ Increases reusability of components	▪ Dependency on technically strong team members for identifying business requirements. ▪ Only system that can be modularized can be built using RAD ▪ Requires highly skilled developers/designers. ▪ High dependency on modeling skills

SOFTWARE PROTOTYPING

Prototype is a working model of software with some limited functionality. The prototype does not always hold the exact logic used in the actual software application and is an extra effort to be considered under effort estimation. Prototyping is used to allow the users evaluate developer proposals and try them out before implementation. It also helps understand the requirements which are user specific and may not have been considered by the developer during product design.

Following is the stepwise approach to design a software prototype:

- **Basic Requirement Identification**

  This step involves understanding the very basics product requirements especially in terms of user interface. The more intricate details of the internal design and external aspects like performance and security can be ignored at this stage.

- **Developing the initial Prototype**

  The initial Prototype is developed in this stage, where the very basic requirements are showcased and user interfaces are provided. These features may not exactly work in the same manner internally in the actual software developed and the workarounds are used to give the same look and feel to the customer in the prototype developed.

- **Review of the Prototype**

  The prototype developed is then presented to the customer and the other important stakeholders in the project. The feedback is collected in an organized manner and used for further enhancements in the product under development.

- **Revise and enhance the Prototype**

  The feedback and the review comments are discussed during this stage and some negotiations happen with the customer based on factors like , time and budget constraints and technical feasibility of actual implementation. The changes accepted are again incorporated in the new Prototype developed and the cycle repeats until customer expectations are met.

Prototypes can have horizontal or vertical dimensions. Horizontal prototype displays the user interface for the product and gives a broader view of the entire system, without concentrating on internal functions. A vertical prototype on the other side is a detailed elaboration of a specific function or a sub system in the product.

The purpose of both horizontal and vertical prototype is different. Horizontal prototypes are used to get more information on the user interface level and the business requirements. It can even be presented in the sales demos to get business in the market. Vertical prototypes are technical in nature and are used to get details of the exact functioning of the sub systems. For example, database requirements, interaction and data processing loads in a given sub system.

Software Prototyping Types

There are different types of software prototypes used in the industry. Following are the major software prototyping types used widely:

- **Throwaway/Rapid Prototyping**

Throwaway prototyping is also called as rapid or close ended prototyping. This type of prototyping uses very little efforts with minimum requirement analysis to build a prototype. Once the actual requirements are understood, the prototype is discarded and the actual system is developed with a much clear understanding of user requirements.

- **Evolutionary Prototyping**

Evolutionary prototyping also called as breadboard prototyping is based on building actual functional prototypes with minimal functionality in the beginning. The prototype developed forms the heart of the future prototypes on top of which the entire system is built. Using evolutionary prototyping only well understood requirements are included in the prototype and the requirements are added as and when they are understood.

- **Incremental Prototyping**

Incremental prototyping refers to building multiple functional prototypes of the various sub systems and then integrating all the available prototypes to form a complete system.

- **Extreme Prototyping**

Extreme prototyping is used in the web development domain. It consists of three sequential phases. First, a basic prototype with all the existing pages is presented in the html format. Then the data processing is simulated using a prototype services layer. Finally the services are implemented and integrated to the final prototype. This process is called Extreme Prototyping used to draw attention to the second phase of the process, where a fully functional UI is developed with very little regard to the actual services.

Software Prototyping Application

Software Prototyping is most useful in development of systems having high level of user interactions such as online systems. Systems which need users to fill out forms or go through various screens before data is processed can use prototyping very effectively to give the exact look and feel even before the actual software is developed.

Software that involves too much of data processing and most of the functionality is internal with very little user interface does not usually benefit from prototyping. Prototype development could be an extra overhead in such projects and may need lot of extra efforts.

Software Prototyping Pros and Cons

Software prototyping is used in typical cases and the decision should be taken very carefully so that the efforts spent in building the prototype add considerable value to the final software developed. The model has its own pros and cons discussed as below.