Software Requirement Specification (SRS)

- A document that captures complete description about how the system is expected to perform. It is usually signed off at the end of requirements engineering phase.
- Software Requirement Specification is termed as SRS document. This document serves as a detailed illustration of functional and non-functional requirements needed that software should fulfill.

Features of SRS :

- This document bridges gap between user and developer.
- Documents board imaginations into a structural layout.
- Used for measuring initial costs and efforts.
- Works as an agreement between communicating parties. Example : Consider a software to monitor employee performance. This will require basic modules such as Login Module, Administrator Module, Employee Module, and Reporting Module. SRS document helps to manage these modules.

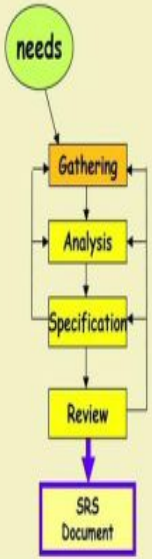
Qualities of SRS:

Correct, Unambiguous, Complete, Consistent, Ranked for importance and/or stability, Verifiable, Modifiable, Traceable

- After we have removed all inconsistency, anomalies, incompleteness from the document, we go about documenting the requirements. Now, let see how to write the SRS document.

How to Gather Requirements?

- Observe existing (manual) systems,
- Study existing procedures,
- Discuss with customer and end-users,
- Input and Output analysis
- Analyze what needs to be done



Software Requirements Specification (SRS) Document Example

1. Introduction → Why is this document being created? What problem or need does the software address? What functionalities will the software include?
- Purpose
- Scope
- Overview

2. General Description → What are the main functions or features of the software? What user objectives does it fulfill? Who are the intended users?
- Functions
- User Community

3. Functional Requirements → What specific actions or behaviors must the software perform?
- Possible Outcomes
- Ranked Order
- Input-Output Relationship

4. User Interface Requirements → How does the software interact with other systems or users?
- **Software Interfaces**
- **Examples**

5. Performance Requirements → What are the acceptable response times for various operations? How much work can the system handle in a given time?
- Response Time
- Throughput
- Scalability

6. Non-Functional Attributes → How user-friendly is the software? What access controls and data protection mechanisms are needed?
- **Usability**
- **Reliability**
- **Security**

7. Schedule and Budget
- Timeline
- Cost Estimate

8. Appendices
- Supplementary Information
- Glossary

Introduction:
Purpose of this Document – At first, main aim of why this document is necessary and what's purpose of document is explained and described.
Scope of this document – In this, overall working and main objective of document and what value it will provide to customer is described and explained. It also includes a description of development cost and time required.
Overview – In this, description of product is explained. It's simply summary or overall review of product.

General description:
In this, general functions of product which includes objective of user, a user characteristic, features, benefits, about why its importance is mentioned. It also describes features of user community.

Functional Requirements:
In this, possible outcome of software system which includes effects due to operation of program is fully explained. All functional requirements which may include calculations, data processing, etc. are placed in a ranked order. Functional requirements specify the expected behavior of the system-which outputs should be produced from the given inputs. They describe the relationship between the input and output of the system. For each functional requirement, detailed description all the data inputs and their source, the units of measure, and the range of valid inputs must be specified.

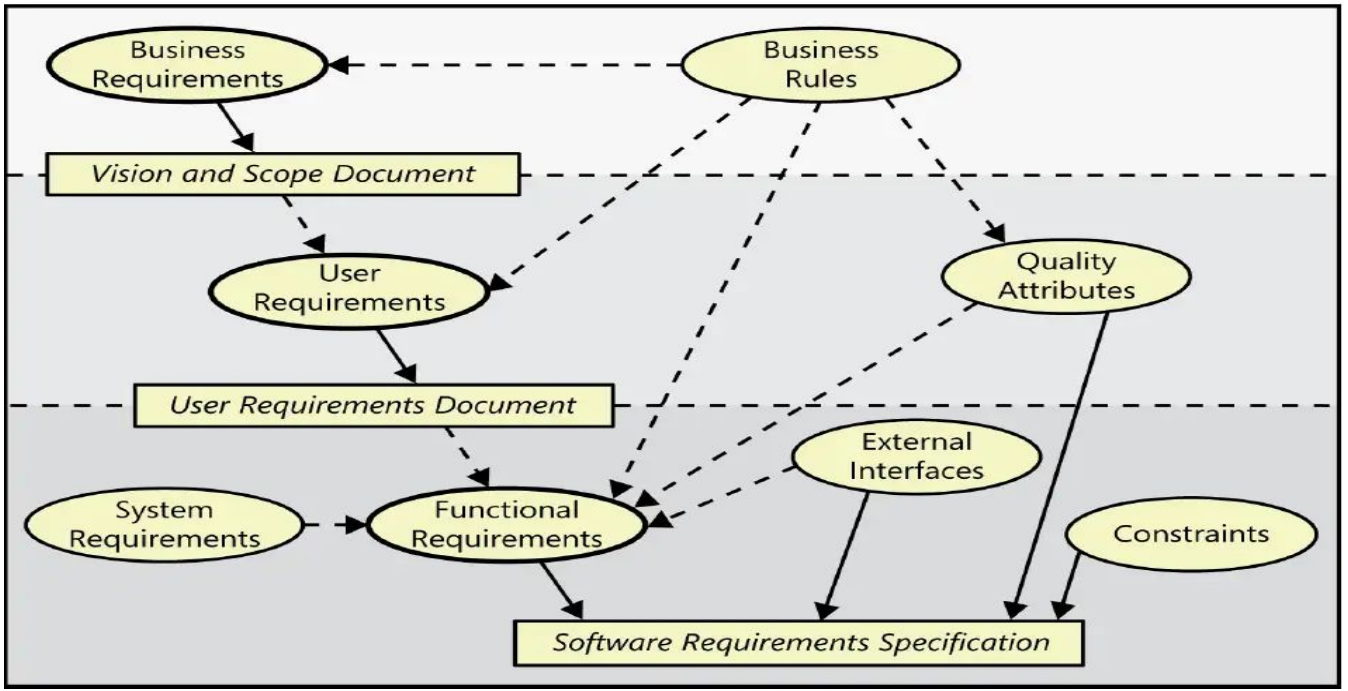
Interface Requirements:
In this, software interfaces which mean how software program communicates with each other or users either in form of any language, code, or message are fully described and explained. Examples can be shared memory, data streams, etc.

Performance Requirements:
In this, how a software system performs desired functions under specific condition is explained. It also explains required time, required memory, maximum error rate, etc. The performance requirements part of an SRS specifies the performance constraints on the software system. All the requirements relating to the performance characteristics of the system must be clearly specified. There are two types of performance requirements: static and dynamic. Static requirements are those that do not impose constraint on the execution characteristics of the system. Dynamic requirements specify constraints on the execution behaviour of the system.

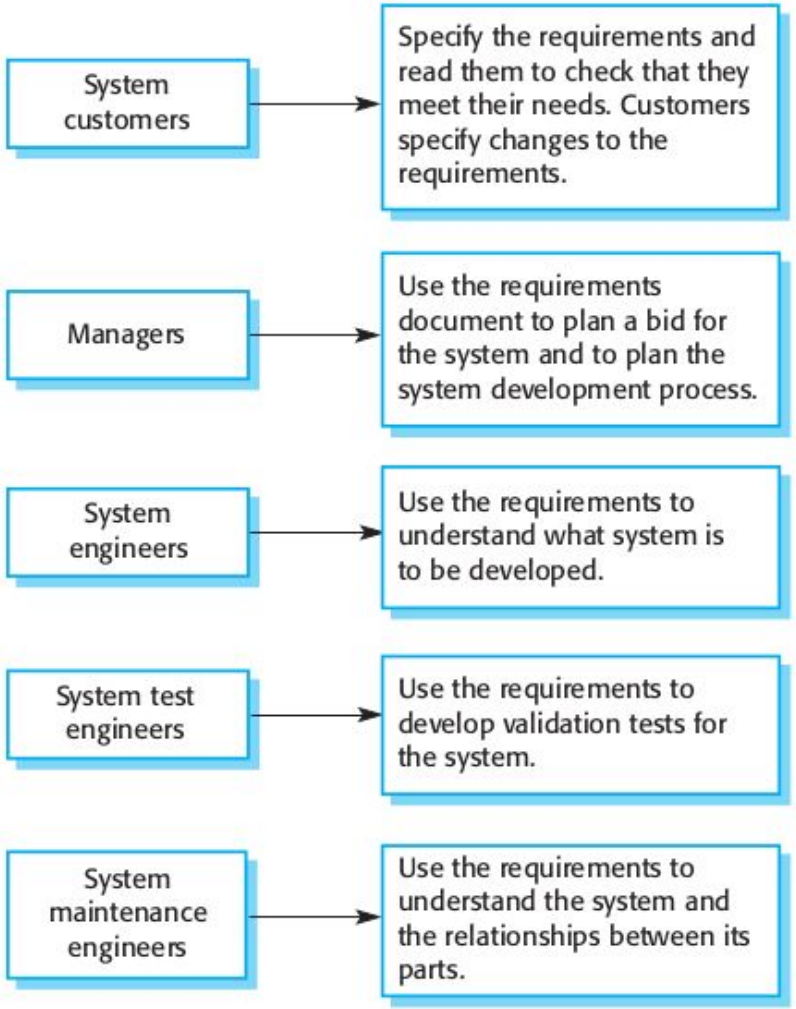
Design Constraints:
In this, constraints which simply means limitation or restriction are specified and explained for design team. Examples may include use of a particular algorithm, hardware and software limitations, etc. There are a number of factors in the client's environment that may restrict the choices of a designer leading to design constraints such factors include standards that must be followed resource limits, operating environment, reliability and security requirements and policies that may have an impact on the design of the system. An SRS should identify and specify all such constraints.

Non-Functional Attributes:
In this, non-functional attributes are explained that are required by software system for better performance. An example may include Security, Portability, Reliability, Reusability, Application compatibility, Data integrity, Scalability capacity, etc.

Preliminary Schedule and Budget:
In this, initial version and budget of project plan are explained which include overall time duration required and overall cost required for development of project.
Appendices:
In this, additional information like references from where information is gathered, definitions of some specific terms, acronyms, abbreviations, etc. are given and explained.



Users of a requirements document



Chapter	Description
Preface	This should define the expected readership of the document and describe its version history, including a rationale for the creation of a new version and a summary of the changes made in each version.
Introduction	This should describe the need for the system. It should briefly describe its functions and explain how it will work with other systems. It should describe how the system fits into the overall business or strategic objectives of the organisation commissioning the software.
Glossary	This should define the technical terms used in the document. You should not make assumptions about the experience or expertise of the reader.
User requirements definition	The services provided for the user and the non-functional system requirements should be described in this section. This description may use natural language, diagrams or other notations that are understandable by customers. Product and process standards which must be followed should be specified.
System architecture	This chapter should present a high-level overview of the anticipated system architecture showing the distribution of functions across system modules. Architectural components that are reused should be highlighted.
System requirements specification	This should describe the functional and non-functional requirements in more detail. If necessary, further detail may also be added to the non-functional requirements, e.g. interfaces to other systems may be defined.
System models	This should set out one or more system models showing the relationships between the system components and the system and its environment. These might be object models, data-flow models and semantic data models.
System evolution	This should describe the fundamental assumptions on which the system is based and anticipated changes due to hardware evolution, changing user needs, etc.
Appendices	These should provide detailed, specific information which is related to the application which is being developed. Examples of appendices that may be included are hardware and database descriptions. Hardware requirements define the minimal and optimal configurations for the system. Database requirements define the logical organisation of the data used by the system and the relationships between data.
Index	Several indexes to the document may be included. As well as a normal alphabetic index, there may be an index of diagrams, an index of functions, etc.

Structure of a requirements document (SRS)

Nonfunctional Requirements

- Characteristics of the system which can not be expressed as functions:
  - **Maintainability,**
  - **Portability,**
  - **Usability,**
  - **Security,**
  - **Safety, etc.**

Functional Requirement Documentation

- **Processing**
  - validation of input data
  - exact sequence of operations
  - responses to abnormal situations
  - any methods (eg. equations, algorithms) to be used to transform inputs to outputs

Nonfunctional Requirements

- Reliability issues
- Performance issues:
  - **Example:** How fast can the system produce results?
    - At a rate that does not overload another system to which it supplies data, etc.
    - Response time should be less than 1sec 90% of the time
    - Needs to be measurable (verifiability)

Constraints

- Hardware to be used,
- Operating system
  - or DBMS to be used
- Capabilities of I/O devices
- Standards compliance
- Data representations by the interfaced system

- User interfaces
- Hardware interfaces
- Software interfaces
- Communications interfaces with other systems
- File export formats

External Interface Requirements

Goals of Implementation

- Goals describe things that are desirable of the system:
  - But, would not be checked for compliance.
  - For example,
    - Reusability issues
    - Functionalities to be developed in future

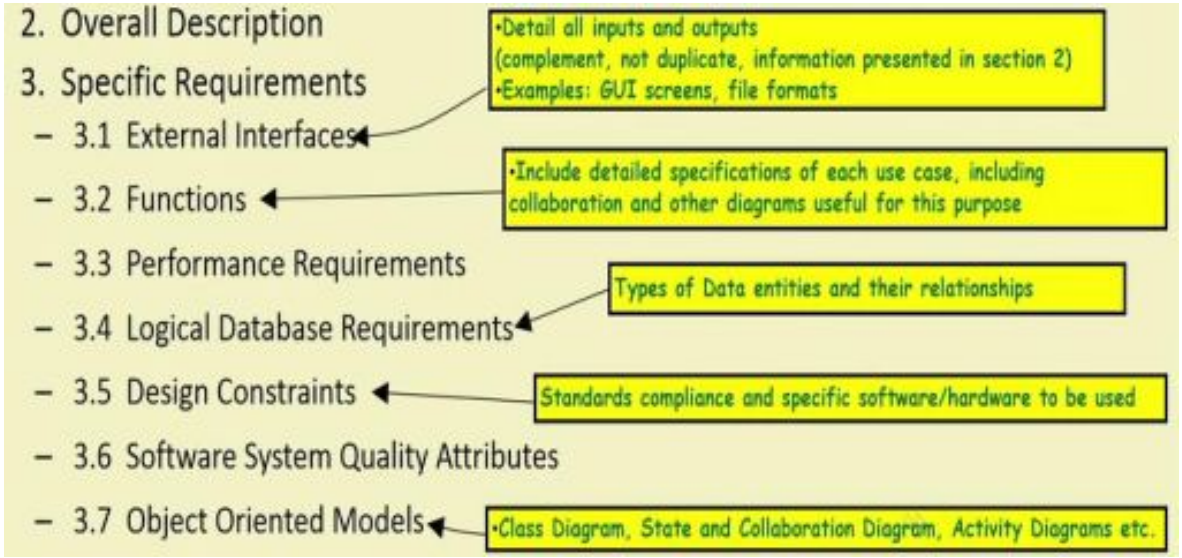
IEEE 830-1998 Standard for SRS

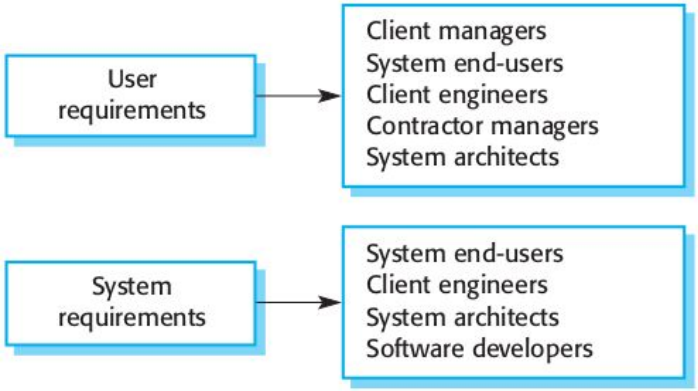
- Title
- Table of Contents
- 1. Introduction
  - 1.1 Purpose
  - 1.2 Scope
  - 1.3 Definitions. Acronyms, and Abbreviations
  - 1.4 References
  - 1.5 Overview
- 2. Overall Description
- 3. Specific Requirements

- Describe purpose of the system
- Describe intended audience
- What the system will and will not do
- Define the vocabulary of the SRS (may also be in appendix)
- List all referenced documents and their sources SRS (may also be in appendix)
- Describe how the SRS is organized

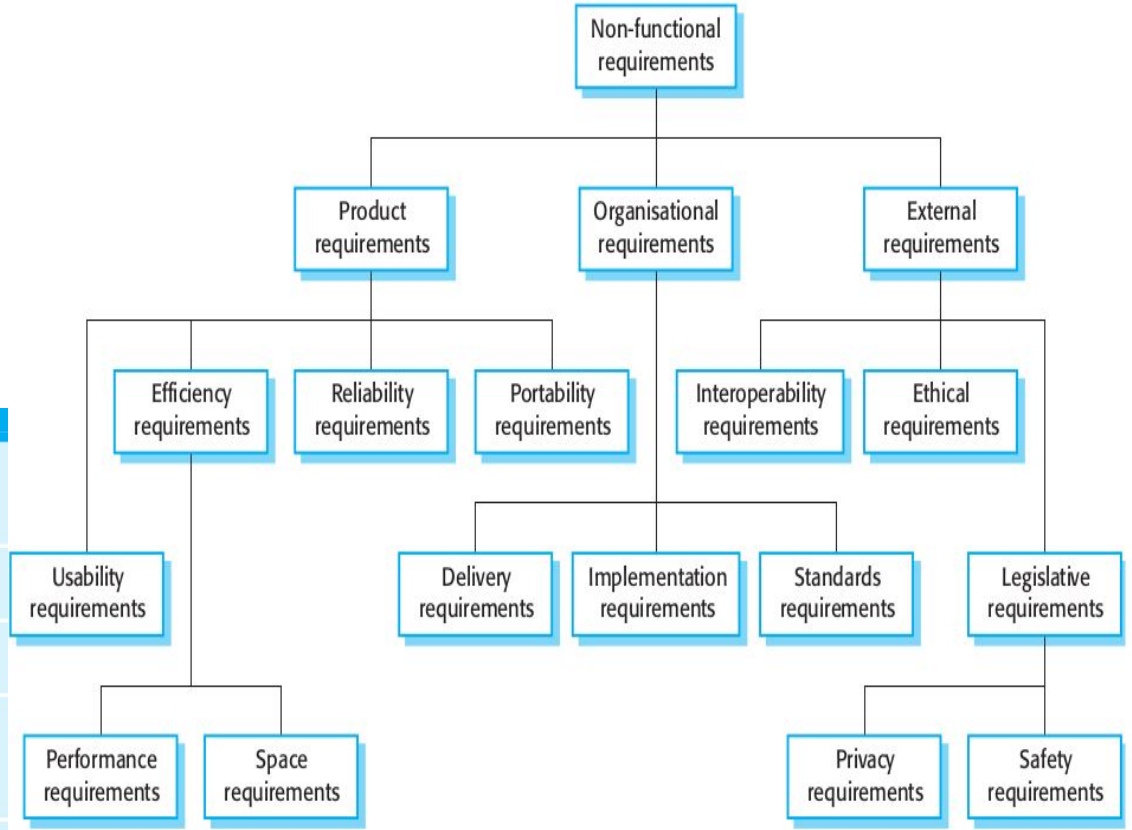
- Table of Contents
- 1. Introduction
- 2. Overall Description
  - 2.1 Product Perspective
  - 2.2 Product Functions
  - 2.3 User Characteristics
  - 2.4 Constraints
  - 2.5 Assumptions and Dependencies
- 3. Specific Requirements
- 4. Appendices

- Present the business case and operational concept of the system
- Describe external interfaces: system, user, hardware, software, communication
- Describe constraints: memory, operational, site adaptation
- Summarize the major functional capabilities
- Describe technical skills of each user class
- Describe other constraints that will limit developer's options; e.g., regulatory policies; target platform, database, network, development standards requirements


2. Overall Description
3. Specific Requirements
– 3.1 External Interfaces
– 3.2 Functions
– 3.3 Performance Requirements
– 3.4 Logical Database Requirements
– 3.5 Design Constraints
– 3.6 Software System Quality Attributes
– 3.7 Object Oriented Models
•Detail all inputs and outputs
(complement, not duplicate, information presented in section 2)
•Examples: GUI screens, file formats
•Include detailed specifications of each use case, including collaboration and other diagrams useful for this purpose
Types of Data entities and their relationships
Standards compliance and specific software/hardware to be used
•Class Diagram, State and Collaboration Diagram, Activity Diagrams etc.



Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	K bytes Number of RAM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target-dependent statements Number of target systems

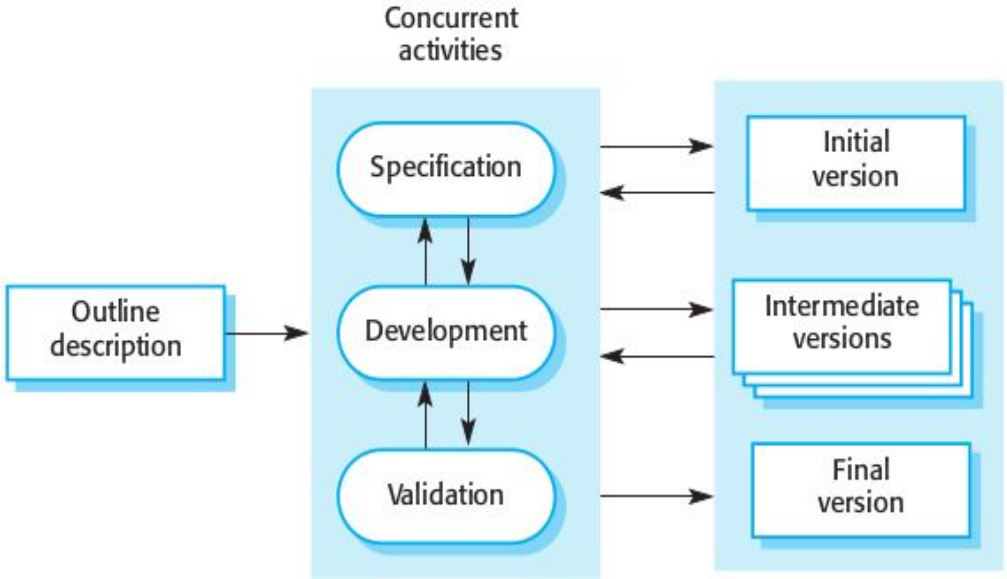


Evolutionary Model

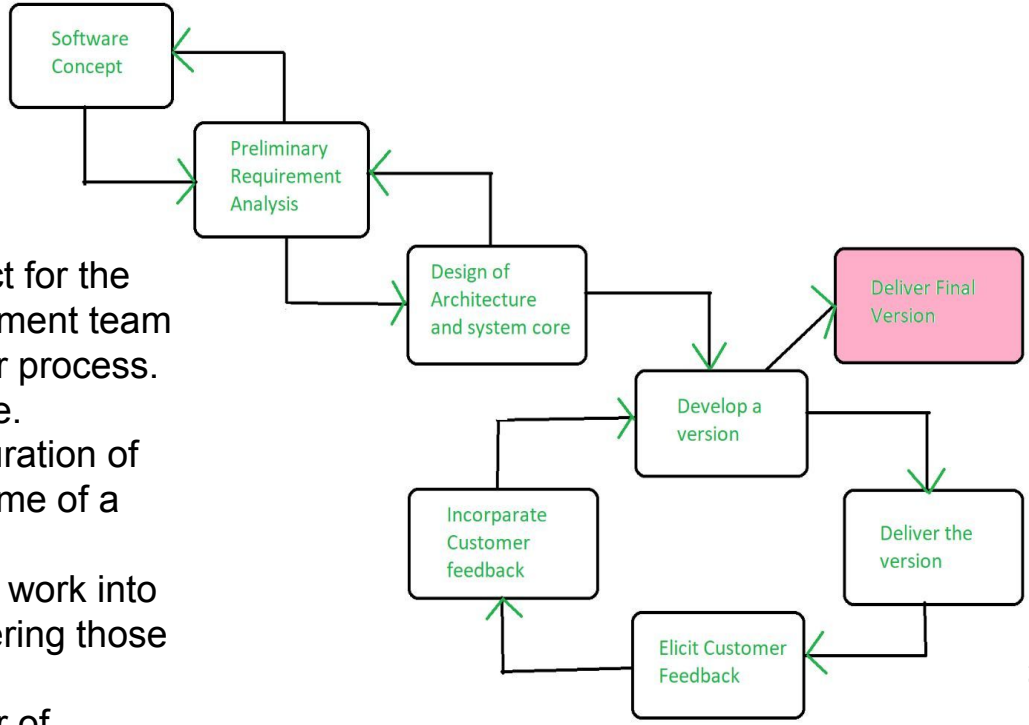
- The evolutionary model is a combination of the Iterative and Incremental models of the software development life cycle. Delivering your system in a big bang release, delivering it in incremental process over time is the action done in this model. Some initial requirements and architecture envisioning need to be done. It is better for software products that have their feature sets redefined during development because of user feedback and other factors

- This approach interleaves the activities of specification, development and validation. An initial system is rapidly developed from abstract specifications. This is then refined with customer input to produce a system that satisfies the customer's needs.

- developing an initial implementation, exposing this to user comment and refining it through many versions until an adequate system has been developed . Specification, development and validation activities are interleaved rather than separate, with rapid feedback across activitie.



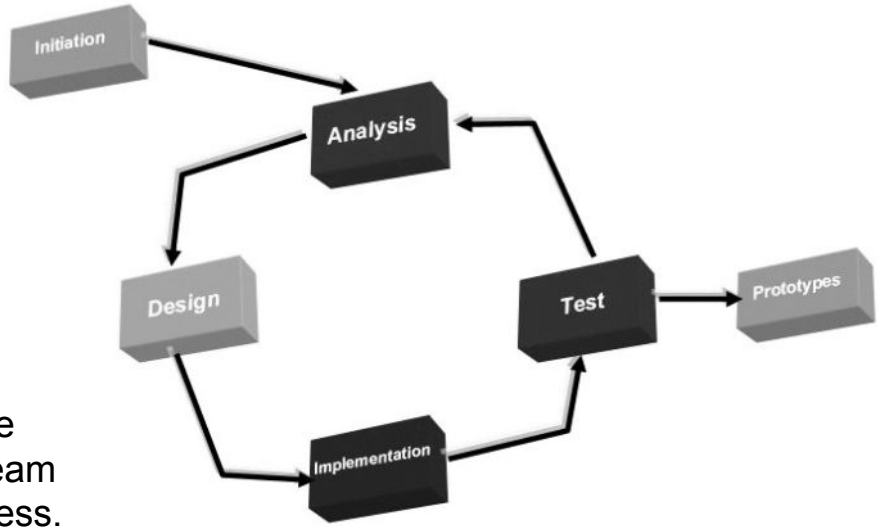
- Divides the development cycle into smaller, incremental waterfall models in which users can get access to the product at the end of each cycle



- Feedback is provided by the users on the product for the planning stage of the next cycle and the development team responds, often by changing the product, plan, or process.
- Therefore, the software product evolves with time.
- All the models have the disadvantage that the duration of time from the start of the project to the delivery time of a solution is very high.
- The evolutionary model suggests breaking down work into smaller chunks, prioritizing them, and then delivering those chunks to the customer one by one.
- The number of chunks is huge and is the number of deliveries made to the customer.

- The model allows for changing requirements as well as all work is broken down into maintainable work chunks.
- customer's confidence increases as he constantly gets quantifiable goods or services from the beginning of the project to verify and validate his requirements.

- Evolutionary models are iterative. They are characterized in a manner that enables software engineers to develop increasingly more complete versions of the software.
- The Evolutionary development model divides the development cycle into smaller, incremental waterfall models in which users are able to get access to the product at the end of each cycle.



- Each of the activities are handled in sequential order. After each of the iterations an evolutionary prototype is produced.
- Feedback is provided by the users on the product for the planning stage of the next cycle and the development team responds, often by changing the product, plans, or process.

Benefits:

- Benefit not only business results but marketing and internal operations as well.
- Use of EVO brings significant reduction in risk for software projects.
- EVO can reduce costs by providing a structured, disciplined avenue for experimentation.
- EVO allows the marketing department access to early deliveries, facilitating development of documentation and demonstrations.
- Short, frequent EVO cycles have some distinct advantages for internal processes and people considerations.
- The cooperation and flexibility required by EVO of each developer results in greater teamwork.
- Better fit the product to user needs and market requirements.
- Manage project risk with definition of early cycle content.
- Uncover key issues early and focus attention appropriately.
- Increase the opportunity to hit market windows.
- Accelerate sales cycles with early customer exposure.
- Increase management visibility of project progress.
- Increase product team productivity and motivation.
- Suitable for small and medium-sized systems (up to 500,000 lines of code)

Problems:

1.The process is not visible Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system.

2.Systems are often poorly structured Continual change tends to corrupt the software structure. Incorporating software changes becomes increasingly difficult and costly.

3. Communication Difficulties, Dependence on an Expert Group, Increasing Management Complexity, Greater Initial Expenditure:

Software Project Management (SPM):

- SPM is a proper way of planning and leading software projects. It is a part of project management in which software projects are planned, implemented, monitored, and controlled. It is necessary to incorporate user requirements along with budget and time constraints.
- **Types of Management in SPM:**

- Conflict management is the process to restrict the negative features of conflict while increasing the positive features of conflict.
- Risk management is the analysis and identification of risks that is followed by synchronized and economical implementation of resource
- Requirement Management is the process of analyzing, prioritizing, tracking, and documenting requirements and then supervising change and communicating to pertinent stakeholders.
- Change management is a systematic approach to dealing with the transition or transformation of an organization's goals, processes, or technologies.
- Software configuration management is the process of controlling and tracking changes in the software, part of the larger cross-disciplinary field of configuration management.
- Release Management is the task of planning, controlling, and scheduling the built-in deploying releases.

- **Aspects of Software Project Management:**

- Planning: The software project manager lays out the complete project's blueprint.
- Leading: A software project manager brings together and leads a team of engineers, strategists, programmers, designers, and data scientists.
- Execution: SPM comes to the rescue here also as the person in charge of software projects will ensure that each stage of the project is completed successfully.

- Time Management: Abiding by a timeline is crucial to completing deliverables successfully.
- Budget: Software project managers are responsible for generating a project budget and adhering to it as closely as feasible, regulating spending, and reassigning funds as needed.
- Maintenance: Software project management emphasizes continuous product testing to find and repair defects early, tailor the end product to the needs of the client, and keep the project on track.

Software Project management has several drawbacks, including resource loss, scheduling difficulty, data protection concerns, and interpersonal conflicts between Developers/Engineers/Stakeholders.

- Costs are High: Consider spending money on various kinds of project management tools, software, & services if ones engage in Software Project Management strategies.
- Complexity will be increased: Software Project management is a multi-stage, complex process.
- Overhead in Communication: Recruits enter your organization when we hire software project management personnel.
- Lack of Originality: Software Project managers can sometimes provide little or no space for creativity.

Project Size Estimation Techniques

- Help in predicting the required resources, time, and cost, thus ensuring the project's feasibility and efficiency from the onset.
- Project size estimation is determining the scope and resources required for the project.
- It involves assessing the various aspects of the project to estimate the effort, time, cost, and resources needed to complete the project.
- Accurate project size estimation is important for effective and efficient project planning, management, and execution.

why project size estimation is critical in project management:

- Financial Planning: Project size estimation helps in planning the financial aspects of the project, thus helping to avoid financial shortfalls.
- Resource Planning: It ensures the necessary resources are identified and allocated accordingly.
- Timeline Creation: It facilitates the development of realistic timelines and milestones for the project.
- Identifying Risks: It helps to identify potential risks associated with overall project execution.
- Detailed Planning: It helps to create a detailed plan for the project execution, ensuring all the aspects of the project are considered.
- Planning Quality Assurance: It helps in planning quality assurance activities and ensuring that the project outcomes meet the required standards.

Methods of Project Estimation:

Expert Judgment: In this technique, a group of experts in the relevant field estimates the project size based on their experience and expertise. This technique is often used when there is limited information available about the project.

Analogous Estimation: This technique involves estimating the project size based on the similarities between the current project and previously completed projects. This technique is useful when historical data is available for similar projects.

Bottom-up Estimation: In this technique, the project is divided into smaller modules or tasks, and each task is estimated separately. The estimates are then aggregated to arrive at the overall project estimate.

Three-point Estimation: This technique involves estimating the project size using three values: optimistic, pessimistic, and most likely. These values are then used to calculate the expected project size using a formula such as the PERT formula.

Function Points: This technique involves estimating the project size based on the functionality provided by the software. Function points consider factors such as inputs, outputs, inquiries, and files to arrive at the project size estimate.

Use Case Points: This technique involves estimating the project size based on the number of use cases that the software must support. Use case points consider factors such as the complexity of each use case, the number of actors involved, and the number of use cases.

Parametric Estimation: For precise size estimation, mathematical models founded on project parameters and historical data are used.

COCOMO (Constructive Cost Model): It is an algorithmic model that estimates effort, time, and cost in software development projects by taking into account several different elements.

Wideband Delphi: Consensus-based estimating method for balanced size estimations that combines expert estimates from anonymous experts with cooperative conversations.
Monte Carlo Simulation: This technique, which works especially well for complicated and unpredictable projects, estimates project size and analyses hazards using statistical methods and random sampling.

Estimating the Size of the Software:
Estimation of the size of the software is an essential part of Software Project Management. It helps the project manager to further predict the effort and time that will be needed to build the project. Here are some of the measures that are used in project size estimation:

1. Lines of Code (LOC):
As the name suggests, LOC counts the total number of lines of source code in a project. The units of LOC are:

KLOC: Thousand lines of code
NLOC: Non-comment lines of code
KDSI: Thousands of delivered source instruction

- The size is estimated by comparing it with the existing systems of the same kind. The experts use it to predict the required size of various components of software and then add them to get the total size.
- It’s tough to estimate LOC by analyzing the problem definition. Only after the whole code has been developed can accurate LOC be estimated. This statistic is of little utility to project managers because project planning must be completed before development activity can begin.

- Two separate source files having a similar number of lines may not require the same effort. A file with complicated logic would take longer to create than one with simple logic. Proper estimation may not be attainable based on LOC.
- The length of time it takes to solve an issue is measured in LOC. This statistic will differ greatly from one programmer to the next. A seasoned programmer can write the same logic in fewer lines than a newbie coder.

2. Number of Entities in ER Diagram:

ER model provides a static view of the project. It describes the entities and their relationships. The number of entities in the ER model can be used to measure the estimation of the size of the project. The number of entities depends on the size of the project. This is because more entities needed more classes/structures thus leading to more coding.

3. Total Number of Processes in DFD:

Data Flow Diagram(DFD) represents the functional view of software. The model depicts the main processes/functions involved in software and the flow of data between them. Utilization of the number of functions in DFD to predict software size. Already existing processes of similar type are studied and used to estimate the size of the process. The sum of the estimated size of each process gives the final estimated size.

4. Function Point Analysis

In this method, the number and type of functions supported by the software are utilized to find FPC(function point count). The steps in function point analysis are:

- **Count the number of functions of each proposed type:** Find the number of functions belonging to the following types:
  - ❖ External Inputs: Functions related to data entering the system.
  - ❖ External outputs: Functions related to data exiting the system.
  - ❖ External Inquiries: They lead to data retrieval from the system but don't change the system.
  - ❖ Internal Files: Logical files maintained within the system. Log files are not included here.
  - ❖ External interface Files: These are logical files for other applications which are used by our system.
- **Compute the Unadjusted Function Points(UFP):** Categorize each of the five function types as simple, average, or complex based on their complexity. Multiply the count of each function type with its weighting factor and find the weighted sum.

Function type	Simple	Average	Complex
External Inputs	3	4	6
External Output	4	5	7
External Inquiries	3	4	6
Internal Logical Files	7	10	15
External Interface Files	5	7	10

- Find the Total Degree of Influence(TDI):
  Use the ’14 general characteristics of a system to find the degree of influence of each of them. The sum of all 14 degrees of influence will give the TDI. The range of TDI is 0 to 70. The 14 general characteristics are: Data Communications, Distributed Data Processing, Performance, Heavily Used Configuration, Transaction Rate, On-Line Data Entry, End-user Efficiency, Online Update, Complex Processing Reusability, Installation Ease, Operational Ease, Multiple Sites and Facilitate Change.
  Each of the above characteristics is evaluated on a scale of 0-5.

- Compute Value Adjustment Factor(VAF): $VAF = (TDI * 0.01) + 0.65$

- Find the Function Point Count(FPC). $FPC = UFP * VAF$

System configuration management:

- An arrangement of exercises that controls change by recognizing the items for change, setting up connections between those things, making/characterizing instruments for overseeing diverse variants, controlling the changes being executed in the current framework, inspecting and revealing/reporting on the changes made.
- Effective Bug Tracking, Continuous Deployment and Integration, Risk management, Support for Big Projects, Reproducibility, Parallel Development.

Objectives of SCM:

- Control the evolution of software systems, Enable collaboration and coordination,Provide version control, Facilitate replication and distribution.

Constructive Cost Model (COCOMO):

- COCOMO Model is a procedural cost estimate model for software projects and is often used as a process of reliably predicting the various parameters associated with making a project such as size, effort, cost, time, and quality. It was proposed by Barry Boehm in 1981 and is based on the study of 63 projects, which makes it one of the best-documented models.

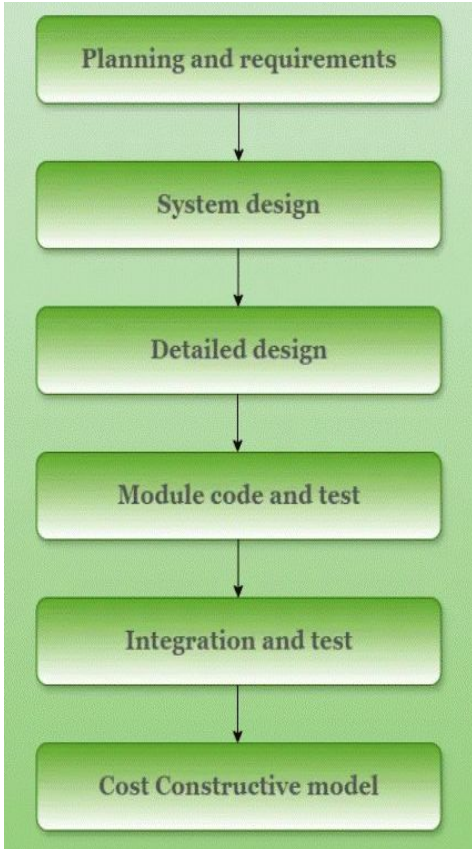
- The key parameters that define the quality of any software product, which are also an outcome of COCOMO, are primarily effort and schedule:

- **Effort**: Amount of labor that will be required to complete a task. It is measured in person-months units.
- **Schedule**: This simply means the amount of time required for the completion of the job, which is, of course, proportional to the effort put in. It is measured in the units of time such as weeks, and months.

Types of Projects:

Organic: A software project is said to be an organic type if the team size required is adequately small, the problem is well understood and has been solved in the past and also the team members have a nominal experience regarding the problem.
Semi-detached: A software project is said to be a Semi-detached type if the vital characteristics such as team size, experience, and knowledge of the various programming environments lie in between organic and embedded. The projects classified as Semi-Detached are comparatively less familiar and difficult to develop compared to the organic ones and require more experience better guidance and creativity. Eg: Compilers or different Embedded Systems can be considered Semi-Detached types.
Embedded: A software project requiring the highest level of complexity, creativity, and experience requirement falls under this category. Such software requires a larger team size than the other two models and also the developers need to be sufficiently experienced and creative to develop such complex models.

Phases of COCOMO



Aspects	Organic	Semidetached	Embedded
Project Size	2 to 50 KLOC	50-300 KLOC	300 and above KLOC
Complexity	Low	Medium	High
Team Experience	Highly experienced	Some experienced as well as inexperienced staff	Mixed experience, includes experts
Environment	Flexible, fewer constraints	Somewhat flexible, moderate constraints	Highly rigorous, strict requirements
Effort Equation	$E = 2.4(400)^{1.05}$	$E = 3.0(400)^{1.12}$	$E = 3.6(400)^{1.20}$
Example	Simple payroll system	New system interfacing with existing systems	Flight control software

- Planning and requirements: This initial phase involves defining the scope, objectives, and constraints of the project. It includes developing a project plan that outlines the schedule, resources, and milestones
- System design: : In this phase, the high-level architecture of the software system is created. This includes defining the system's overall structure, including major components, their interactions, and the data flow between them.
- Detailed design: This phase involves creating detailed specifications for each component of the system. It breaks down the system design into detailed descriptions of each module, including data structures, algorithms, and interfaces.
- Module code and test: This involves writing the actual source code for each module or component as defined in the detailed design. It includes coding the functionalities, implementing algorithms, and developing interfaces.
- Integration and test: This phase involves combining individual modules into a complete system and ensuring that they work together as intended.
- Cost Constructive model: The Constructive Cost Model (COCOMO) is a widely used method for estimating the cost and effort required for software development projects.

Types of COCOMO Model: There are three types of COCOMO Model:

1. **Basic COCOMO Model**
2. **Intermediate COCOMO Model**
3. **Detailed COCOMO Model**

- The **Basic COCOMO model i**s a straightforward way to estimate the effort needed for a software development project. It uses a simple mathematical formula to predict how many person-months of work are required based on the size of the project, measured in thousands of lines of code (KLOC).

$E = a*(KLOC)^b\ PM$

$Tdev = c*(E)^d$

Person required = Effort/ Time

Where,

E is effort applied in Person-Months

KLOC is the estimated size of the software product indicate in Kilo Lines of Code

Tdev is the development time in months

a, b, c are constants determined by the category of software project given in below table.

The above formula is used for the cost estimation of the basic COCOMO model and also is used in the subsequent models.

- The effort is measured in Person-Months and as evident from the formula is dependent on Kilo-Lines of code. The development time is measured in months.
- These formulas are used as such in the Basic Model calculations, as not much consideration of different factors such as reliability, and expertise is taken into account, henceforth the estimate is rough.

The constant values a, b, c, and d for the Basic Model for the different categories of the software projects are:

Software Projects	a	b	c	d
Organic	2.4	1.05	2.5	0.38
Semi-Detached	3.0	1.12	2.5	0.35
Embedded	3.6	1.20	2.5	0.32

Example of Basic COCOMO Model

Suppose that a Basic project was estimated to be 400 KLOC (kilo lines of code). Calculate effort and time for each of the three modes of development. All the constants value provided in the following table:

Solution

From the above table we take the value of constant a,b,c and d.

1. For organic mode,
   - effort = $2.4 \times (400)^{1.05} \approx 1295$ person-month.
   - dev. time = $2.5 \times (1295)^{0.38} \approx 38$ months.

2. For semi-detach mode,
   - effort = $3 \times (400)^{1.12} \approx 2462$ person-month.
   - dev. time = $2.5 \times (2462)^{0.35} \approx 38$ months.

3. For Embedded mode,
   - effort = $3.6 \times (400)^{1.20} \approx 4772$ person-month.
   - dev. time = $2.5 \times (4772)^{0.32} \approx 38$ months.

Intermediate COCOMO Model

other factors such as reliability, experience, and Capability. These factors are known as Cost Drivers (multipliers) and the Intermediate Model utilizes 15 such drivers for cost estimation.

Classification of Cost Drivers and their Attributes:
The cost drivers are divided into four categories

Product attributes:

- Required software reliability extent
- Size of the application database
- The complexity of the product
- Hardware attributes

Run-time performance constraints:

- Memory constraints
- The volatility of the virtual machine environment
- Required turnabout time
- Personal attributes

Analyst capability:

- Software engineering capability
- Application experience
- Virtual machine experience
- Programming language experience

Project attributes:

- Use of software tools
- Application of software engineering methods
- Required development schedule

Each of the 15 such attributes can be rated on a six-point scale ranging from “very low” to “extra high” in their relative order of importance. Each attribute has an effort multiplier fixed as per the rating.

The **Effort Adjustment Factor (EAF)** is determined by multiplying the effort multipliers associated with each of the 15 attributes.

$E = a*(KLOC)^b * EAF\ PM$

$Tdev = c*(E)^d$

Where,

- *E is effort applied in Person-Months*
- *KLOC is the estimated size of the software product indicate in Kilo Lines of Code*
- *EAF is the Effort Adjustment Factor (EAF) is a multiplier used to refine the effort estimate obtained from the basic COCOMO model.*
- *Tdev is the development time in months*

Software Projects	a	b	c	d
Organic	3.2	1.05	2.5	0.38
Semi-Detached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

Detailed COCOMO Model:

team experience, development practices, and software complexity. By analyzing these factors in more detail, Detailed COCOMO provides a highly accurate estimation of effort, time, and cost for software projects.

Advantages :

- **Systematic cost estimation:** Provides a systematic way to estimate the cost and effort of a software project.
- **Helps to estimate cost and effort:** This can be used to estimate the cost and effort of a software project at different stages of the development process.
- **Helps in high-impact factors:** Helps in identifying the factors that have the greatest impact on the cost and effort of a software project.
- **Helps to evaluate the feasibility of a project:** This can be used to evaluate the feasibility of a software project by estimating the cost and effort required to complete it.

Disadvantages:

- **Assumes project size as the main factor:** Assumes that the size of the software is the main factor that determines the cost and effort of a software project, which may not always be the case.
- **Does not count development team-specific characteristics**: Does not take into account the specific characteristics of the development team, which can have a significant impact on the cost and effort of a software project.
- **Not enough precise cost and effort estimate:** This does not provide a precise estimate of the cost and effort of a software project, as it is based on assumptions and averages.

1. A company needs to develop digital signal processing software for one of its newest inventions. The software is expected to have 20000 lines of code. The company needs to determine the effort in person-months needed to develop this software using the basic COCOMO model. The multiplicative factor for this model is given as 2.2 for the software development on embedded systems, while the exponentiation factor is given as 1.50. What is the estimated effort in person-months?

(A) 196.77 (B) 206.56 (C) 199.56 (D) 210.68 **Solution:** The correct Answer is **(A)**.

2. Estimation of software development effort for organic software in basic COCOMO is

(A) $E = 2.0(KLOC)^{1.05}PM$

(B) $E = 3.4(KLOC)^{1.06}PM$

(C) $E = 2.4(KLOC)^{1.05}PM$

(D) $E = 2.4(KLOC)^{1.07}PM$

Solution: The correct Answer is **(C)**.

Risk Management

- Risk is an expectation of loss, a potential problem that may or may not occur in the future.
- Risk concerns future happenings. It involves change in mind, opinion, actions, places, etc.
- Something that you'd prefer not to have happen. Risks may threaten the project, the software that is being developed or the organisation.

Two **characteristics** of risk

- **Uncertainty** – the risk may or may not happen, that is, there are no 100% risks (those, instead, are called constraints)
- **Loss** – the risk becomes a reality and unwanted consequences or losses occur

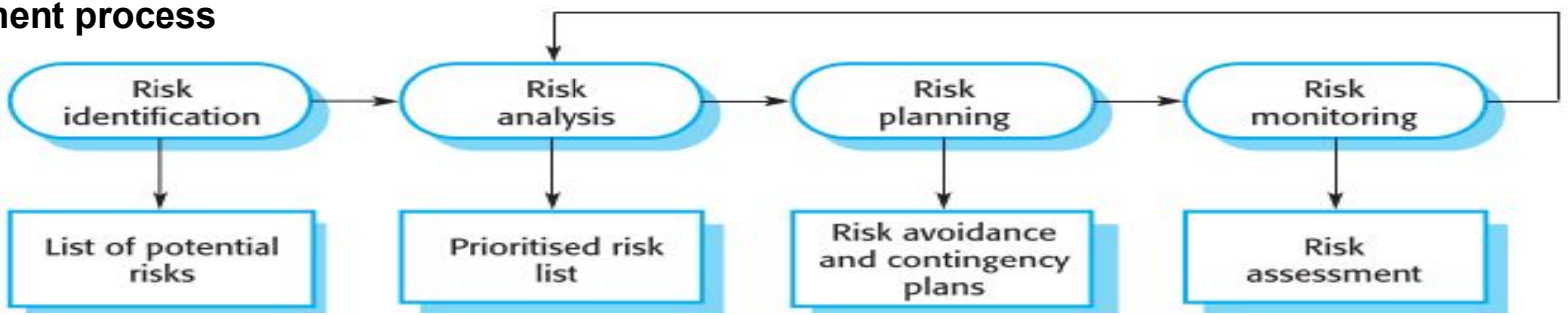
Categories of risk:

1. **Project risks** are risks that affect the project schedule or resources. An example might be the loss of an experienced designer.
2. **Product risks** are risks that affect the quality or performance of the software being developed. An example might be the failure of a purchased component to perform as expected.
3. **Business risks** are risks that affect the organisation developing or procuring the software. For example, a competitor introducing a new product is a business risk.

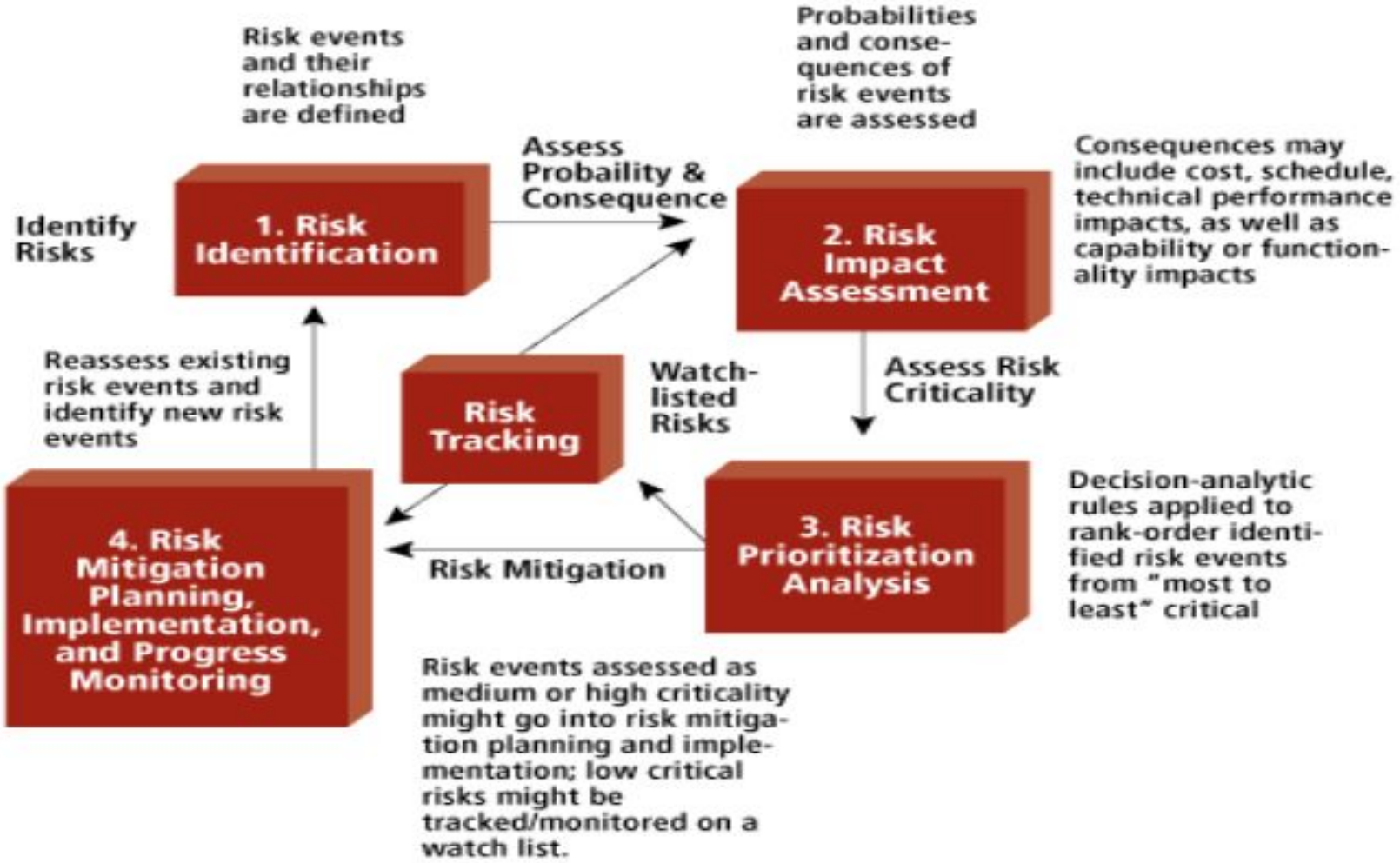
Most common risks

Risk	Risk type	Description
Staff turnover	Project	Experienced staff will leave the project before it is finished.
Management change	Project	There will be a change of organisational management with different priorities.
Hardware unavailability	Project	Hardware which is essential for the project will not be delivered on schedule.
Requirements change	Project and product	There will be a larger number of changes to the requirements than anticipated.
Specification delays	Project and product	Specifications of essential interfaces are not available on schedule.
Size underestimate	Project and product	The size of the system has been underestimated.
CASE tool under-performance	Product	CASE tools which support the project do not perform as anticipated.
Technology change	Business	The underlying technology on which the system is built is superseded by new technology.
Product competition	Business	A competitive product is marketed before the system is completed.

Risk management process



Risk Management: Fundamental Steps



1.Risk **identification** Possible project, product and business risks are identified.
2.Risk **analysis** The likelihood and consequences of these risks are assessed.
3.Risk **planning** Plans to address the risk either by avoiding it or minimising its effects on the project are drawn up.
4.Risk **monitoring** The risk is constantly assessed and plans for risk mitigation are revised as more information about the risk becomes available.

1. Risk identification:
   - It is concerned with discovering possible risks to the project.

Risk type	Possible risks
Technology	The database used in the system cannot process as many transactions per second as expected. Software components which should be reused contain defects which limit their functionality.
People	It is impossible to recruit staff with the skills required. Key staff are ill and unavailable at critical times. Required training for staff is not available.
Organisational	The organisation is restructured so that different management are responsible for the project. Organisational financial problems force reductions in the project budget.
Tools	The code generated by CASE tools is inefficient. CASE tools cannot be integrated.
Requirements	Changes to requirements which require major design rework are proposed. Customers fail to understand the impact of requirements changes.
Estimation	The time required to develop the software is underestimated. The rate of defect repair is underestimated. The size of the software is underestimated.

2. Risk analysis:

- During the risk analysis process, you have to consider each identified risk and make a judgement about the probability and the seriousness of it.
- The probability of the risk might be assessed as very low (<10%), low (10–25%), moderate (25-50%), high (50–75%) or very high (>75%).
- The effects of the risk might be assessed as catastrophic, serious, tolerable or insignificant.

Risk type	Possible risks
Technology	The database used in the system cannot process as many transactions per second as expected. Software components which should be reused contain defects which limit their functionality.
People	It is impossible to recruit staff with the skills required. Key staff are ill and unavailable at critical times. Required training for staff is not available.
Organisational	The organisation is restructured so that different management are responsible for the project. Organisational financial problems force reductions in the project budget.
Tools	The code generated by CASE tools is inefficient. CASE tools cannot be integrated.
Requirements	Changes to requirements which require major design rework are proposed. Customers fail to understand the impact of requirements changes.
Estimation	The time required to develop the software is underestimated. The rate of defect repair is underestimated. The size of the software is underestimated.

Risk analysis

Risk	Probability	Effects
Organisational financial problems force reductions in the project budget.	Low	Catastrophic
It is impossible to recruit staff with the skills required for the project.	High	Catastrophic
Key staff are ill at critical times in the project.	Moderate	Serious
Software components which should be reused contain defects which limit their functionality.	Moderate	Serious
Changes to requirements which require major design rework are proposed.	Moderate	Serious
The organisation is restructured so that different management are responsible for the project.	High	Serious
The database used in the system cannot process as many transactions per second as expected.	Moderate	Serious
The time required to develop the software is underestimated.	High	Serious
CASE tools cannot be integrated.	High	Tolerable
Customers fail to understand the impact of requirements changes.	Moderate	Tolerable
Required training for staff is not available.	Moderate	Tolerable
The rate of defect repair is underestimated.	Moderate	Tolerable
The size of the software is underestimated.	High	Tolerable
The code generated by CASE tools is inefficient.	Moderate	Insignificant

3. Risk planning:

- The risk planning process considers each of the key risks that have been identified and identifies strategies to manage the risk. strategies fall into three categories:
  - Avoidance strategies Following these strategies means that the probability that the risk will arise will be reduced.
  - Minimisation strategies Following these strategies means that the impact of the risk will be reduced.
  - Contingency plans Following these strategies means that you are prepared for the worst and have a strategy in place to deal with it.

Risk	Strategy
Organisational financial problems	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Recruitment problems	Alert customer of potential difficulties and the possibility of delays, investigate buying-in components.
Staff illness	Reorganise team so that there is more overlap of work and people therefore understand each other's jobs.
Defective components	Replace potentially defective components with bought-in components of known reliability.
Requirements changes	Derive traceability information to assess requirements change impact, maximise information hiding in the design.
Organisational restructuring	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Database performance	Investigate the possibility of buying a higher-performance database.
Underestimated development time	Investigate buying-in components, investigate the use of a program generator.

Risk table: A risk table provides a project manager with a simple technique for risk projection. It consists of five columns

1. Risk Summary – short description of the risk
2. Risk Category – one of seven risk categories (slide 12)
3. Probability – estimation of risk occurrence based on group input
4. Impact – (1) catastrophic (2) critical (3) marginal (4) negligible
5. RMMM – Pointer to a paragraph in the Risk Mitigation, Monitoring, and Management Plan

Risk Summary	Risk Category	Probability	Impact (1-4)	RMMM

Risk Mitigation, Monitoring, and Management (RMMM)

- RMMM plan may be a part of the software development plan or may be a separate document. Once RMMM has been documented and the project has begun, the risk mitigation, and monitoring steps begin.
  – Risk mitigation is a problem avoidance activity
  – Risk monitoring is a project tracking activity

Risk analysis support the project team in constructing a strategy to deal with risks.
There are three important issues considered in developing an effective strategy:

▪ **Risk avoidance or mitigation (Risk control)** - It is the primary strategy which is fulfilled through a plan.
▪ **Risk monitoring** - The project manager monitors the factors and gives an indication whether the risk is becoming more or less.
▪ **Risk management and planning** - It assumes that the mitigation effort failed and the risk is a reality.
▪ **Risk mitigation (avoidance)** is the primary strategy and is achieved through a plan.

• **Assume/Accept**: Acknowledge the existence of a particular risk, and make a deliberate decision to accept it without engaging in special efforts to control it. Approval of project or program leaders is required.
• **Avoid**: Adjust program requirements or constraints to eliminate or reduce the risk. This adjustment could be accommodated by a change in funding, schedule, or technical requirements.
• **Control**: Implement actions to minimize the impact or likelihood of the risk.
• Transfer: Reassign organizational accountability, responsibility, and authority to another stakeholder willing to accept the risk.
• **Watch/Monitor**: Monitor the environment for changes that affect the nature and/or the impact of the risk.

Assessing Risk Impact

- Three factors affect the consequences that are likely if a risk does occur
  - Its nature – This indicates the problems that are likely if the risk occurs
  - Its scope – This combines the severity of the risk (how serious was it) with its overall distribution (how much was affected)
  - Its timing – This considers when and for how long the impact will be felt
- The overall risk exposure formula is RE = P x C
  - P = the probability of occurrence for a risk
  - C = the cost to the project should the risk actually occur
- Example
  - P = 80% probability that 18 of 60 software components will have to be developed
  - C = Total cost of developing 18 components is $25,000
  - RE = .80 x $25,000 = $20,000

4. Risk monitoring:

- Risk monitoring involves regularly assessing each of the identified risks to decide whether or not that risk is becoming more or less probable and whether the effects of the risk have changed. It is a continuous process, and, at every management progress review, you should consider and discuss each of the key risks separately.

Risk type	Potential indicators
Technology	Late delivery of hardware or support software, many reported technology problems
People	Poor staff morale, poor relationships amongst team members, job availability
Organisational	Organisational gossip, lack of action by senior management
Tools	Reluctance by team members to use tools, complaints about CASE tools, demands for higher-powered workstations
Requirements	Many requirements change requests, customer complaints
Estimation	Failure to meet agreed schedule, failure to clear reported defects

Risk monitoring has three objectives

- To assess whether predicted risks do, in fact, occur
- To ensure that risk aversion steps defined for the risk are being properly applied
- To collect information that can be used for future risk analysis

Principles of Risk Management

Maintain a global perspective	View software risks within the context of a system and the business problem that is intended to solve
Take a forward-looking view	Think about risks that may arise in the future; establish contingency plans
Encourage open communication	Encourage all stakeholders and users to point out risks at any time
Integrate risk management	Integrate the consideration of risk into the software process
Emphasize a continuous process of risk management	Modify identified risks as more becomes known and add new risks as better insight is achieved
Develop a shared product vision	A shared vision by all stakeholders facilitates better risk identification and assessment
Encourage teamwork when managing risk	Pool the skills and experience of all stakeholders when conducting risk management activities

Risk Analysis in SDLC:

- Risk Analysis is simply identifying risks in applications and prioritizing them for testing purpose. How risk assessment and management are incorporated into these steps to ensure less risk in the software being developed.

1. Preliminary Analysis: In this step, you need to find out the organization's objective

- Nature and scope of problem under study
- Propose alternative solutions and proposals after having a deep understanding of the problem and what competitors are doing
- Describe costs and benefits.

Support from Risk Management Activities: Below mentioned is the support from the activities of Risk Management.

- Establish a process and responsibilities for risk management
- Document Initial known risks
- The Project Manager should prioritize the risks

2. System Analysis and Requirement Definition

This step is very important for a clear understanding of customer expectations and requirements. Thus it is very important to conduct this phase with utmost care and given due time as any possible error will cause the failure of the entire process. Following are the series of steps that are conducted during this stage.

- End-user requirements are obtained through documentation, client interviews, observation, and questionnaires
- Pros and cons of the current system are identified so as to avoid the cons and carry forward the pros in the new system.
- Any Specific user proposals are used to prepare the specifications and solutions for the shortcomings discovered in step two are found.
- Identify assets that need to be protected and assign their criticality in terms of confidentiality, integrity, and availability.
- Identify threats and resulting risks to those assets.
- Determine existing security controls to reduce that risks.

3. Feasibility Study: This is the first and most important phase. Often this phase is conducted as a standalone phase in big projects not as a sub-phase under the requirement definition phase. This phase allows the team to get an estimate of major risk factors cost and time for a given project. You might be wondering why this is so important. A feasibility study helps us to get an idea of whether it is worth constructing the system or not. It helps to identify the main risk factors.

Risk Factors: Following is the list of risk factors for the feasibility study phase.

- Project managers often make a mistake in estimating the cost, time, resources, and scope of the project. Unrealistic budget, time, inadequate resources, and unclear scope often lead to project failure.
- Unrealistic Budget: As discussed above inaccurate estimation of the budget may lead to the project running out of funds early in the SDLC. An accurate estimation budget is directly related to correct knowledge of time, effort, and resources.
- Unrealistic Schedule: Incorrect time estimation lead to a burden on developers by project managers to deliver the project on time. Thus compromising the overall quality of the project and thus making the system less secure and more vulnerable.
- Insufficient resources: In some cases, the technology, and tools available are not up-to-date to meet project requirements, or resources(people, tools, technology) available are not enough to complete the project. In any case, it is the project will get delayed, or in the worst case it may lead to project failure.
- Unclear project scope: Clear understanding of what the project is supposed to do, which functionalities are important, which functionalities are mandatory, and which functionalities can be considered as extra is very important for project managers. Insufficient knowledge of the system may lead to project failure.

4. Requirement Elicitation: It starts with an analysis of the application domain. This phase requires the participation of different stakeholders to ensure efficient, correct, and complete gathering of system services, their performance, and constraints. This data set is then reviewed and articulated to make it ready for the next phase.

Risk Factors: Following is the list of risk factors for the Requirement Elicitation phase.

- Incomplete requirements: In 60% of the cases users are unable to state all requirements in the beginning. Therefore requirements have the most dynamic nature in the complete SDLC (Software Development Life Cycle) Process. If any of the user needs, constraints, or other functional/non-functional requirements are not covered then the requirement set is said to be incomplete.
- Inaccurate requirements: If the requirement set does not reflect real user needs then in that case requirements are said to be inaccurate.
- Unclear requirements: Often in the process of SDLC there exists a communication gap between users and developers. This ultimately affects the requirement set. If the requirements stated by users are not understandable by analysts and developers then these requirements are said to be unclear.
- Ignoring nonfunctional requirements: Sometimes developers and analysts ignore the fact that nonfunctional requirements hold equal importance as functional requirements. In this confusion, they focus on delivering what the system should do rather than how the system should be like scalability, maintainability, testability, etc.
- Conflicting user requirements: Multiple users in a system may have different requirements. If not listed and analyzed carefully, this may lead to inconsistency in the requirements.
- Gold plating: It is very important to list out all requirements in the beginning. Adding requirements later during development may lead to threats in the system. Gold plating is nothing but adding extra functionality to the system that was not considered earlier. Thus inviting threats and making the system vulnerable.

5. Requirement Analysis Activity: In this step requirements that are gathered by interviewing users or brainstorming or by another means will be: first analyzed and then classified and organized such as functional and nonfunctional requirements groups and then these are prioritized to get a better knowledge of which requirements are of high priority and need to be definitely present in the system. After all these steps requirements are negotiated.

Risk Factors: Risk management in this Requirement Analysis Activity step has the following task to do.

- Nonverifiable requirements: If a finite cost-effective process like testing, inspection, etc is not available to check whether the software meets the requirement or not then that requirement is said to be nonverifiable.
- Infeasible requirement: if sufficient resources are not available to successfully implement the requirement then it is said to be an infeasible requirement.
- Inconsistent requirement: If a requirement contradicts any other requirement then the requirement is said to be inconsistent.
- Nontraceable requirement: It is very important for every requirement to have an origin source. During documentation, it is necessary to write the origin source of each requirement so that it can be traced back in the future when required.
- Unrealistic requirement: If a requirement meets all the above criteria like it is complete, accurate, consistent, traceable, verifiable, etc then that requirement is realistic enough to be documented and implemented.

6. Requirement Validation Activity: This involves validating the requirements that are gathered and analyzed till now to check whether they actually define what users want from the system.

Risk Factors: Following is the list of risk factors for the Requirement Validation Activity phase.

- Misunderstood domain-specific terminology: Developers and Application specialists often use domain-specific terminology or we can say technical terms which are not understandable for the majority of end users. Thus creating a misunderstanding between end users and developers.
- Using natural language to express requirements: Natural language is not always the best way to express requirements as different users may have different signs and conventions. Thus it is advisable to use formal language for expressing and documenting.

7. Requirement Documentation Activity: This step involves creating a Requirement Document (RD) by writing down all the agreed-upon requirements using formal language. RD serves as a means of communication between different stakeholders.

Risk Factors: Following is the list of risk factors for the Requirement Documentation Activity phase.

- Inconsistent requirements data and RD: Sometimes it might happen, due to glitches in the gathering and documentation process, actual requirements may differ from the documented ones.
- Nonmodifiable RD: If during RD preparation, structuring of RD with maintainability is not considered then it will become difficult to edit the document in the course of change without rewriting it.

- In SDLC, there are four main risk response strategies- Avoidance, Mitigation, Transfer, Acceptance
- Traditional Risk Management focuses on individual risks, while Integrated Risk Management also focuses on interactions between different risks.
- challenges that are faced while implementing Integrated Risk Management in SDLC
  - resistance to change, difficulty in obtaining full support from all stakeholders, complex risk interdependencies, data integration issues, etc.

- Stakeholders has crucial role in Risk Management during SDLC. They help in risk identification by providing insights into potential risks using different perspectives.

- Risk Monitoring and Control happens during SDLC by continuously tracking and reviewing the project during its development phase.

1. Which SDLC Model is Best for Risk Management?

Answer: The Spiral Model is a systems development lifecycle (SDLC) that is the best method for risk management.

2. What is Risk Analysis in SDLC?

Answer: Risk Analysis is simply identifying risks in applications and prioritizing them for testing purpose.

3. How Risk is Managed in the Waterfall Model?

Answer: Risks in Waterfall Model are managed with the help of Charts. After the detection of Risks, Risk Chart begins.

Risk management in System Design and Development phase:

system architecture must be established and all requirements that are documented needs to be addressed. The system (operations and features) is described in detail using screen layouts, pseudocode, business rules, process diagrams, etc.

System Design undergoes a list of seven activities that are listed as follows.

1.Examine the Requirement Document: It is quite important for the developers to be a part of examining the requirement document to ensure the understandability of the requirements listed in the requirement document.

Risk Factors – RD is not clear for developers: I

2. Choosing the Architectural Design Method Activity: It is the method to decompose the system into components. Thus it is a way to define software system components. There exist many methods for architectural design including structured object-oriented, Jackson system development, and formal methods. But there is no standard architectural design method.

Risk Factors – Improper Architectural Design Method

3. Choosing the Programming Language Activity: Choosing a programming language should be done side by side with the architectural method. As programming language should be compatible with the architectural method chosen.

Risk Factors – Improper choice of programming language: Incorrect choice of programming language may not support chosen architectural method. This may reduce the maintainability and portability of the system.

4. Constructing Physical Model Activity: The physical model consisting of symbols is a simplified description of a hierarchically organized system.

Risk Factors – Complex System, Complicated Design, Large-Size Components, Unavailability of Expertise for Reusabilit, Less Reusable Components:

5. Verifying Design Activity: Verifying design means ensuring that the design is the correct solution for the system under construction and it meets all user requirements.

Risk Factors – Difficulties in Verifying Design to Requirements, Many Feasible Solutions,Incorrect Design.

6. Specifying Design Activity: This activity involves the following main tasks. It involves the components and defines the data flow between them and for each identified component state its function, data input, data output, and utilization of resources.

Risk Factors – Difficulty in allocating functions to components, Extensive specification, Omitting Data Processing Functions

7. Documenting Design Activity: In this phase design document(DD) is prepared. This will help to control and coordinate the project during implementation and other phases.

Risk Factors – Incomplete DD, Inconsistent DD, Unclear DD, Large DD

Development

- This stage involves the actual coding of the software as per the agreed-upon requirements between the developer and the client.

Support from Risk Management Activities: All designed controls are implemented during this stage. This phase involves two steps: Coding and Unit Testing.

1. Coding Activity: This step involves writing the source code for the system to be developed. User interfaces are developed in this step. Each developed module is then tested in the unit testing step.

Risk Factors – Unclear design document, Lack of independent working environment, Wrong user interface and user functions developed, Programming language incompatible with architectural design, Repetitive code, Modules developed by different programmers

2. Unit Testing Activity: Each module is tested individually to check whether it meets the specified requirements or not and performs the functions it is intended to do.

Risk Factors – Lack of fully automated testing tools, Code not understandable by reviewers, Coding drivers and stubs, Poor documentation of test cases, The testing team is not experienced, Poor regression testing.

5. Integration and System Testing

- all modules are independently checked for errors, bugs. Then they are related to their dependents and dependency is checked for errors finally all modules are integrated into one complete software and checked as a whole for bugs.

Support from Risk Management Activities

In this phase, designed controls are tested to see whether they work accurately in an integrated environment. This phase includes three activities: Integration Activity, Integration Testing Activity, and System Testing Activity. We will be discussing these activities in a bit more detail along with the risk factors in each activity.

1. Integration Activity: In this phase, individual units are combined into one working system.

Risk Factors – Difficulty in combining components,Integrate wrong versions of components,Omissions

2. Integration Testing Activity: After integrating the components next step is to test whether the components interface correctly and to evaluate their integration. This process is known as integration testing.

Risk Factors – Bugs during integration,Data loss through the interface,Desired functionality not achieved, Difficulty in locating and repairing errors

3. System Testing Activity: In this step integrated system is tested to ensure that it meets all the system requirements gathered from the users.

Risk Factors: Unqualified testing team,Limited testing resources,Not possible to test in a real environment,Testing cannot cope with requirements change,The system being tested is not testable enough.

6. Installation, Operation, and Acceptance Testing:
This is the last and longest phase in SDLC. This system is delivered, installed, deployed, and tested for user acceptance.

Support from Risk Management Activities
The system owner will want to ensure that the prescribed controls, including any physical or procedural controls, are in place prior to the system going live. Decisions regarding risks identified must be made prior to system operation. This phase involves three activities: Installation, Operation, and Acceptance Testing.

1. Installation Activity: The software system is delivered and installed at the customer site.
Risk Factors:Problems in installation, Change in the environment

2. Operation Activity: Here end users are given training on how to use software systems and their services.
Risk Factors: New requirements emerge, Difficulty in using the system

3. Acceptance Testing Activity: The delivered system is put into acceptance testing to check whether it meets all user requirements or not.
Risk Factors: User resistance to change, Too many software faults, Insufficient data handling, Missing requirements.

7. Maintenance
In this stage, the system is assessed to ensure it does not become obsolete. This phase also involves continuous evaluation of the system in terms of performance and changes are made from time to time to initial software to make it up-to-date. Errors, and faults discovered during acceptance testing are fixed in this phase. This step involves making improvements to the system, fixing errors, enhancing services, and upgrading software.

Support from Risk Management Activities
Any change to a system has the potential to reduce the effectiveness of existing controls or to otherwise have some impact on the confidentiality, availability, or integrity of the system. The solution is to ensure that a risk assessment step is included in evaluating system changes.

Risk Factors: Budget overrun, Problems in upgrading

8. Disposal

In this phase, plans are developed for discarding system information, hardware, and software to make the transition to a new system. The purpose is to prevent any possibility of unauthorized disclosure of sensitive data due to improper disposal of information. All of this should be done in accordance with the organization's security requirements.

Support from Risk Management Activities

The Risk Management plan developed must also include threats to the confidentiality of residual data, proper procedures, and controls to reduce the risk of data theft due to improper disposal. However, by identifying the risk early in the project, the controls could be documented in advance ensuring proper disposition.

Risk Factors: Lack of knowledge for proper disposal, Lack of proper procedures