Software Engineering

STDC, CDAC, Kochi

Sunitha C S
Joint Director
STDC, Kochi

Software

A **software** is a computer programs along with the associated documents and the configuration data that make these programs operate correctly.

A **program** is a set of instructions (written in form of human-readable code) that performs a specific task.

Types of Software Systems

There are many different types of software systems from simple to complex systems. These systems may be developed for a particular customer, like systems to support a particular business process, or developed for a general purpose, like any software for our computers such as word processors.

Software Engineering

Software engineering is an **engineering discipline** that's applied to the development of software in a ***systematic*** approach (called a software process).

It's not only concerned with the technical process of building a software, it also includes activities to manage the project, develop tools, methods and theories that support the software production.

Computer Science vs Software Engineering

Computer science focuses on the **theory and fundamentals**, like algorithms, programming languages, theories of computing, artificial intelligence, and hardware design, while software engineering is concerned with the **activities of developing and managing a software.**

Software Engineers

The job of a software engineer is difficult. It has to balance between different people involved, such as:

- **Dealing with users**: User don't know what to expect exactly from the software. The concern is always about the ease of use and response time.
- **Dealing with technical people**: Developers are more technically inclined people so they think more of database terms, functionality, etc.
- **Dealing with management**: They are concerned with return on their investment, and meeting the schedule.

For success in large software developments, it is important to follow an engineering approach, consisting of a well defined process.

Software Process and Software Process Models

A software process (also knows as **software methodology**) is a set of related activities that leads to the production of the software. These activities may involve the **development of the software from the scratch, or, modifying an existing system**.

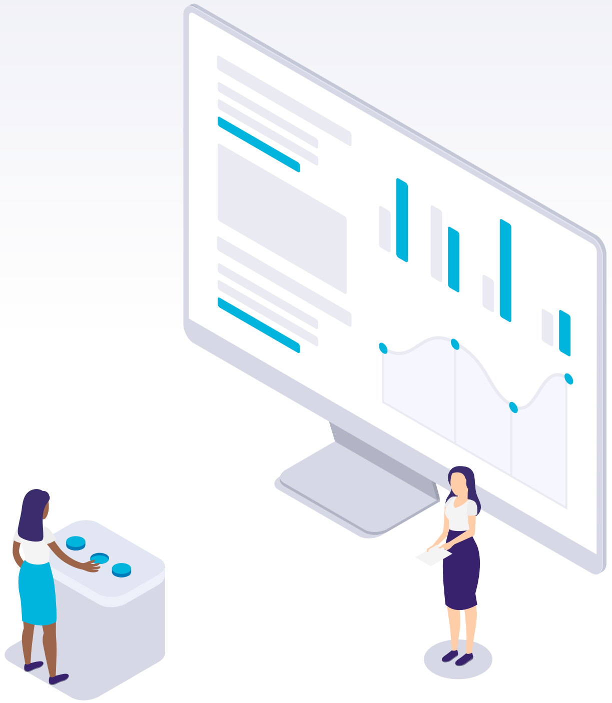
Any software process must include the following four activities:

- **Software specification** (or requirements engineering): Define the main functionalities of the software and the constrains around them.
- **Software design and implementation**: The software is to be designed and programmed.
- **Software verification and validation**: The software must conforms to it's specification and meets the customer needs.
- **Software evolution** (software maintenance): The software is being modified to meet customer and market requirements changes.

There are also **supporting activities** such as **configuration and change management, quality assurance, project management, user experience**.

1

Software Process Models



Software Process Models

A software process model is a simplified representation of a software process. Each model represents a process from a specific perspective.

Some methodologies are sometimes known as **software development life cycle (SDLC)** methodologies, though this term could also be used more generally to refer to any methodology.

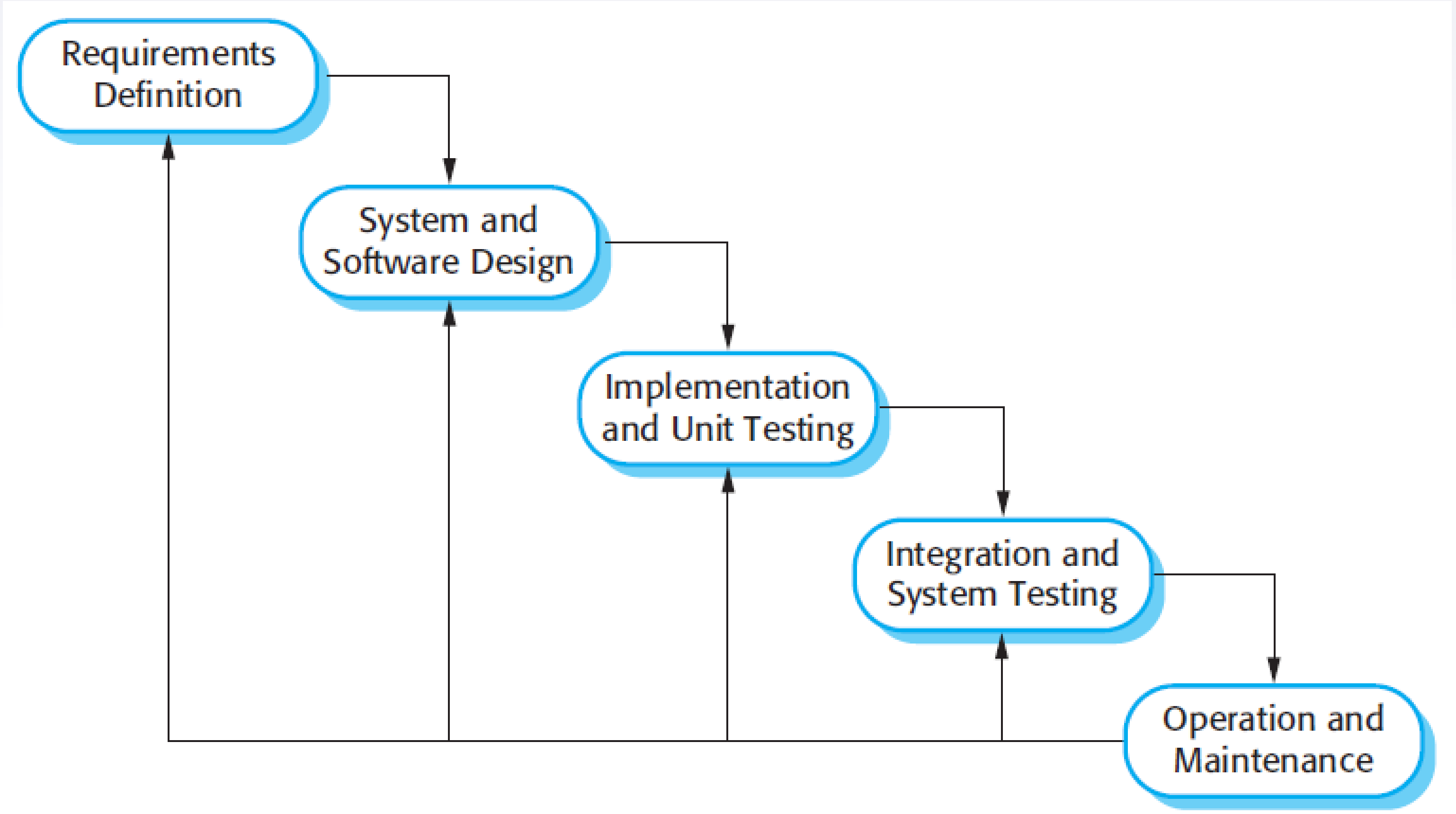
1. Waterfall Model

- The waterfall model is a sequential approach, where each fundamental activity of a process represented as a separate phase, arranged in linear order.
- In the waterfall model, you must plan and schedule all of the activities before starting working on them (plan-driven process).

Plan-driven process is a process where all the activities are planned first, and the progress is measured against the plan. While the agile process, planning is incremental and it's easier to change the process to reflect requirement changes.

- The phases of the waterfall model are: **Requirements, Design, Implementation, Testing, and Maintenance.**

Waterfall Model


Requirements Definition
System and Software Design
Implementation and Unit Testing
Integration and System Testing
Operation and Maintenance

The Nature of Waterfall Phases

- In principle, the result of each phase is **one or more documents** that should be approved and the **next phase shouldn't be started until the previous phase has completely been finished.**
- In practice, however, **these phases overlap** and feed information to each other. For example, during design, problems with requirements can be identified, and during coding, some of the design problems can be found, etc.

When To Use?

- In principle, the waterfall model should only be applied **when requirements are well understood and unlikely to change radically during development** as this model has a relatively **rigid structure** which makes it relatively hard to accommodate change when the process in underway.

2. Prototyping

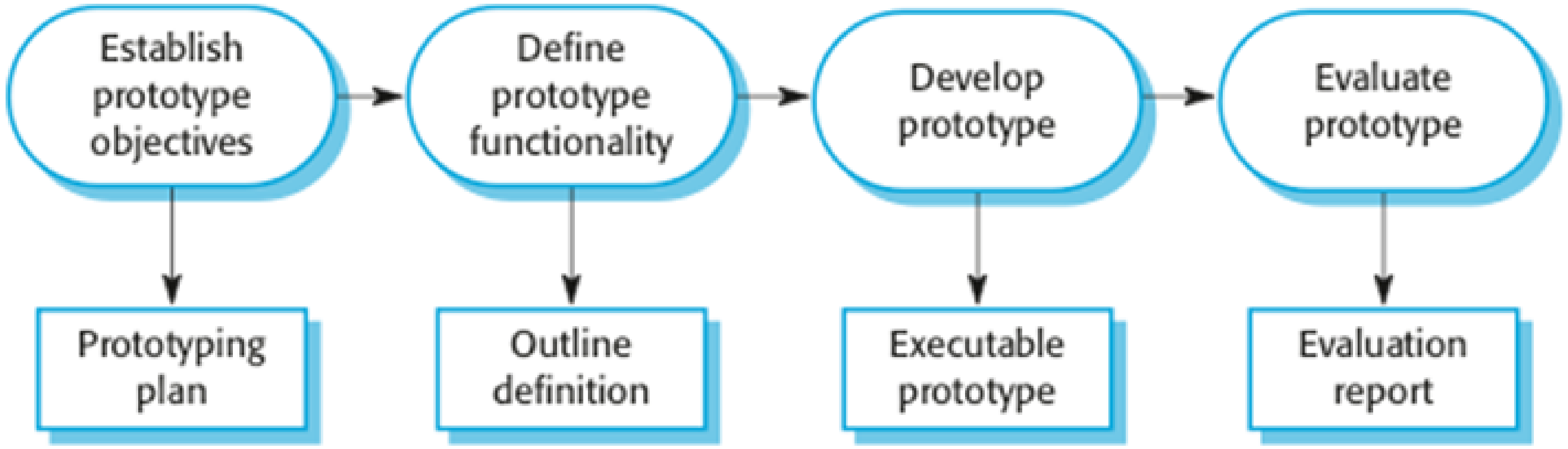
A prototype is a **version of a system or part of the system that's developed quickly** to check the customer's requirements or **feasibility of some design decisions.**

So, a prototype is **useful when a customer or developer is not sure of the requirements**, or of algorithms, efficiency, business rules, response time, etc.

In prototyping, the client is involved throughout the development process, which increases the likelihood of client acceptance of the final implementation.

A software prototype can be used:

- In the **requirements engineering**, a prototype can help with the elicitation and validation of system requirements. It allows the users to experiment with the system, and so, refine the requirements. They may get new ideas for requirements, errors and find areas of strength and weakness in the software.
- In the **system design**, a prototype can help to carry out design experiments to check the feasibility of a proposed design. For example, a database design may be prototyped and tested to check it supports efficient data access for the most common user queries.

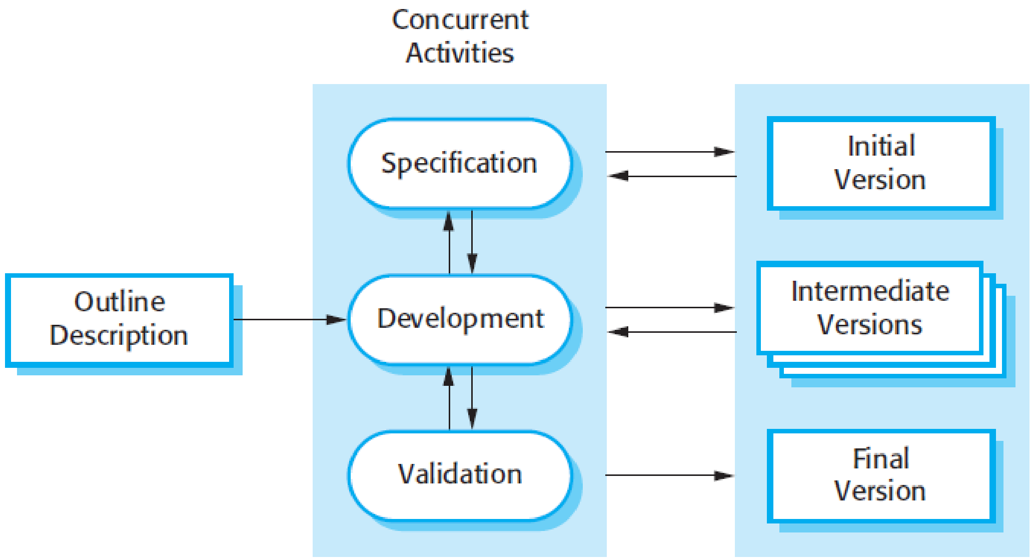

Establish prototype objectives
Define prototype functionality
Develop prototype
Evaluate prototype
Prototyping plan
Outline definition
Executable prototype
Evaluation report

The phases of a prototype

- **Establish objectives**: The objectives of the prototype should be made explicit from the start of the process. Is it to validate system requirements, or demonstrate feasibility, etc.
- **Define prototype functionality**: Decide what are the inputs and the expected output from a prototype. To reduce the prototyping costs and accelerate the delivery schedule, you may ignore some functionality, such as response time and memory utilization unless they are relevant to the objective of the prototype.
- **Develop the prototype**: The initial prototype is developed that includes only user interfaces.
- **Evaluate the prototype**: Once the users are trained to use the prototype, they then discover requirements errors. Using the feedback both the specifications and the prototype can be improved. If changes are introduced, then a repeat of steps 3 and 4 may be needed.

3. Incremental Development

- Incremental development is based on the idea of developing an initial implementation, exposing this to user feedback, and evolving it through several versions until an acceptable system has been developed.
- The activities of a process are not separated but interleaved with feedback involved across those activities.



Incremental Development

- Each system increment reflects a piece of the functionality that is needed by the customer. Generally, the early increments of the system should include the most important or most urgently required functionality.
- This means that the customer can evaluate the system at early stage in the development to see if it delivers what's required.

Incremental Vs Waterfall Model

Compared to the waterfall model, incremental development has three important benefits:

- The **cost of accommodating changing** customer requirements is reduced.
- It's easier to get **customer feedback** on the work done during development than when the system is fully developed, tested, and delivered.
- More **rapid delivery** of *useful* software is possible even if all the functionality hasn't been included. Customers are able to use and gain value from the software earlier than it's possible with the waterfall model.

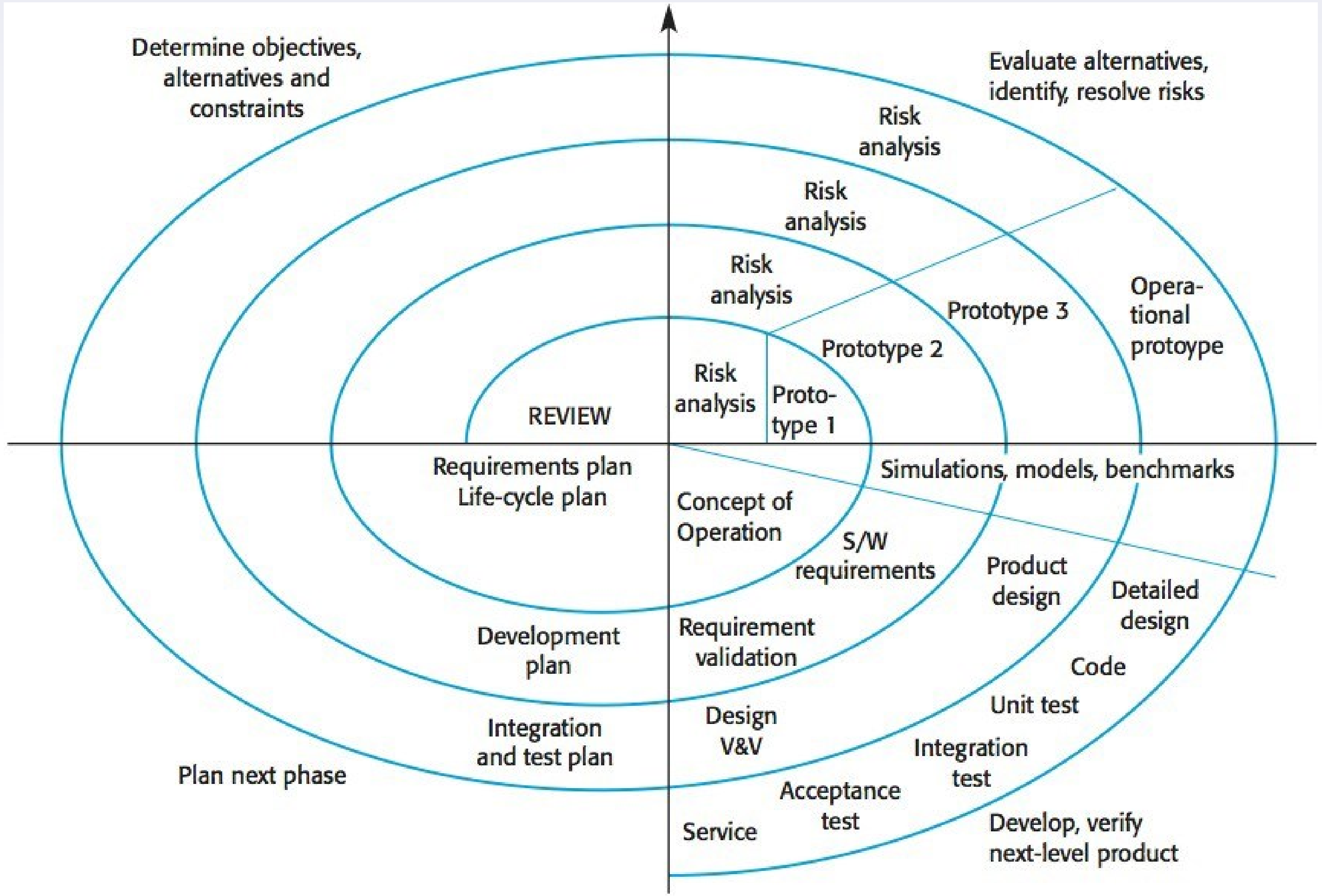
Incremental Development

It can be a plan-driven or agile, or both

- Incremental development is one of the most common approaches. This approach can be either a plan-driven or agile, or both.
- In a plan-driven approach, the system increments are identified in advance, but, in the agile approach, only the early increments are identified and the development of later increments depends on the progress and customer priorities.

4. Spiral Model

- The spiral model is a risk-driven where the process is represented as spiral rather than a sequence of activities.
- It was designed to include the best features from the waterfall and prototyping models, and introduces a new component; risk-assessment.
- Each loop (from *review* till *service* — see figure below) in the spiral represents a phase. Thus the first loop might be concerned with system feasibility, the next loop might be concerned with the requirements definition, the next loop with system design, and so on.


Determine objectives, alternatives and constraints
Evaluate alternatives, identify, resolve risks
Risk analysis
Risk analysis
Risk analysis
Risk analysis
Prototype 3
Opera-tional protoype
Prototype 2
Proto-type 1
REVIEW
Requirements plan Life-cycle plan
Simulations, models, benchmarks
Concept of Operation
S/W requirements
Product design
Detailed design
Development plan
Requirement validation
Code
Unit test
Integration and test plan
Design V&V
Integration test
Plan next phase
Acceptance test
Service
Develop, verify next-level product

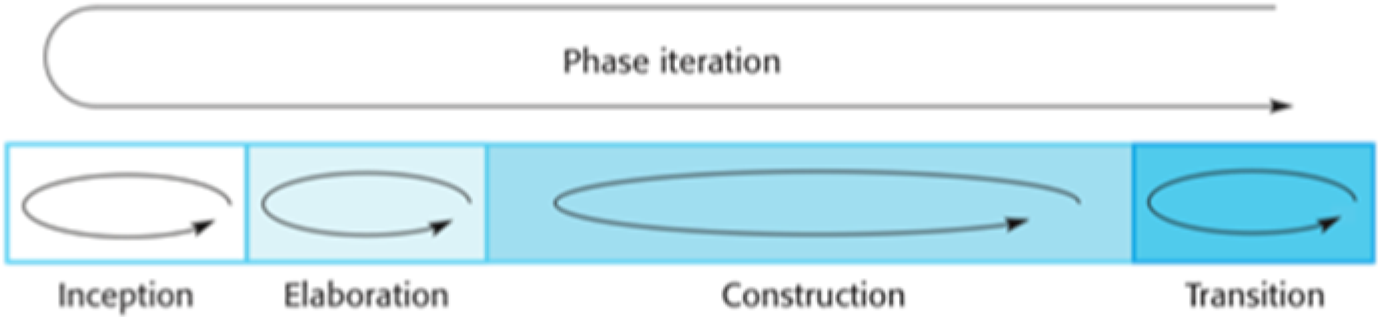
Spiral Model

Each loop in the spiral is split into four sectors:

- **Objective setting**: The objectives and risks for that phase of the project are defined.
- **Risk assessment and reduction:** For each of the identified project risks, a detailed analysis is conducted, and steps are taken to reduce the risk. For example, if there's a risk that the requirements are inappropriate, a prototype may be developed.
- **Development and validation:** After risk evaluation, a process model for the system is chosen. So if the risk is expected in the user interface then we must prototype the user interface. If the risk is in the development process itself then use the waterfall model.
- **Planning:** The project is reviewed and a decision is made whether to continue with a further loop or not.

5. Iterative Development

- Iterative development model aims to develop a system through building small portions of all the features, across all components.
- We build a product which meets the initial scope and release it quickly for customer feedback. An early version with limited features important to establish market and get customer feedback.
- In each increment, a slice of system features is delivered, passing through the requirements till the deployment.



The phases of iterative development are:

- **Inception:** The goal is to establish a business case for the system. We should identify all the external entities that will interact with the system, and define these interactions. Then, uses this information to assess the contribution that the system makes to the business. If the contribution is minor, then the project may be cancelled.
- **Elaboration:** We develop an understanding of the problem domain and architecture framework, develop the project plan, and identify risks.
- **Construction:** *Incrementally* fills-in the architecture with production-ready code produced from analysis, design, implementation, and testing of the requirements. The components of the system are dependent on each other and they're developed in parallel and integrated during this phase. On the completion of this phase, you should have a complete working software.
- **Transition:** We deliver the system into the production operating environment.

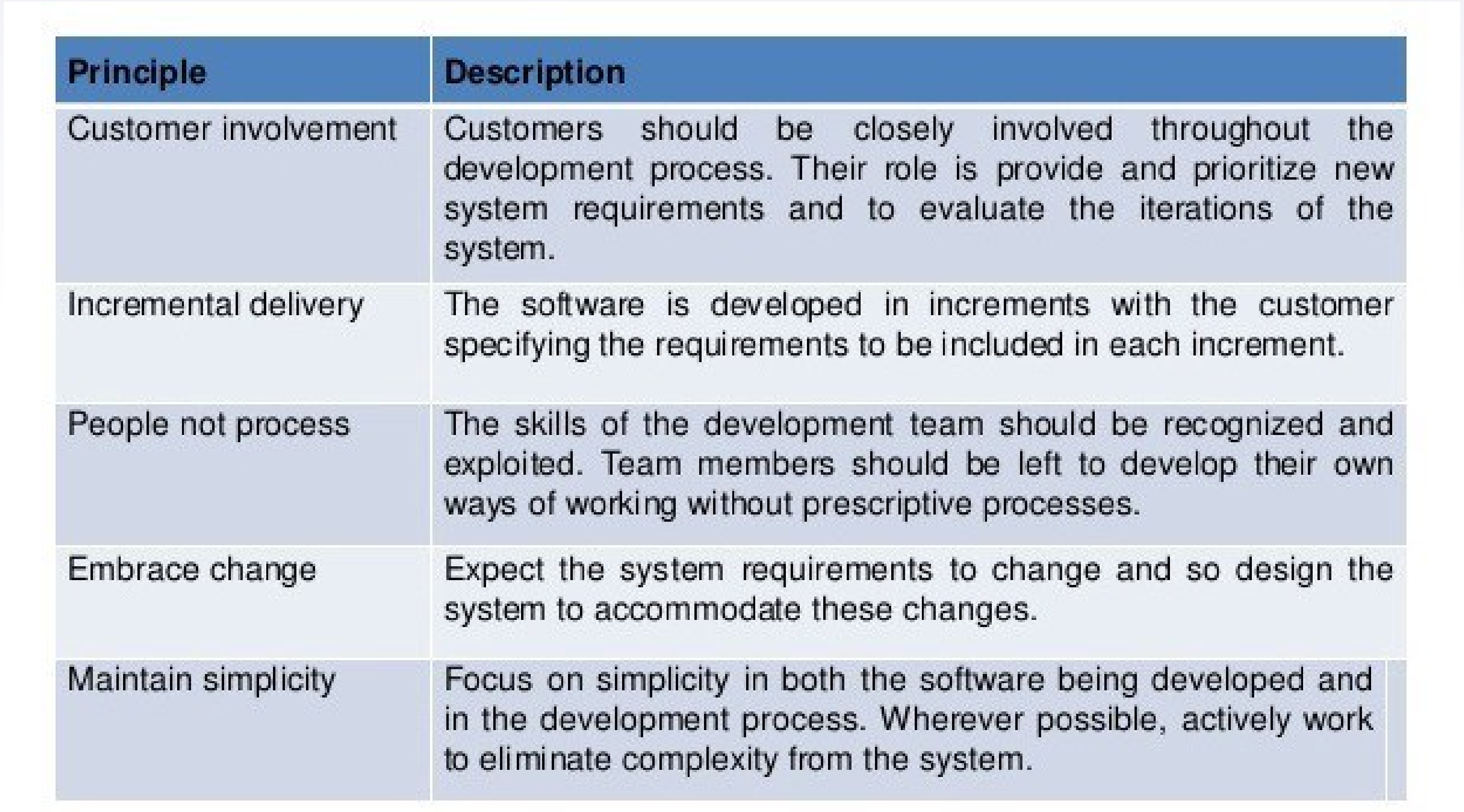
All the phases will be done once, while the construction phase will be incrementally visited for each increment; for each slice of system features.

6. Agile

Agility is flexibility, it is a state of dynamic, adapted to the specific circumstances.

- The agile methods refers to a group of software development models based on the incremental and iterative approach, in which the increments are small and typically, new releases of the system are created and made available to customers every few weeks.
- They are best suited for application where the requirements change rapidly during the development process.
- There are a number of different agile methods available such as: Scrum, Crystal, Agile Modeling (AM), Extreme Programming (XP), etc.

Principles of Agile



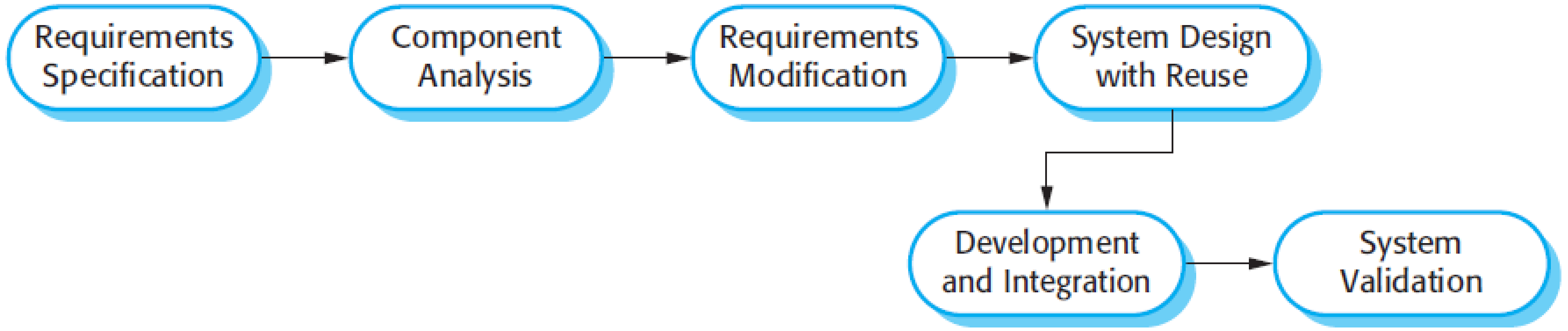
Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

Increment vs Iterative vs Agile

- Each increment in the incremental approach builds a complete feature of the software, while in iterative, it builds small portions of all the features.
- An agile approach combines the incremental and iterative approach by building a small portion of each feature, one by one, and then both gradually adding features and increasing their completeness.

7. Reuse-oriented Software Engineering

- It's attempting to reuse an existing design or code (probably also tested) that's similar to what's required. It's then modified, and incorporated to the new system.


Requirements Specification
Component Analysis
Requirements Modification
System Design with Reuse
Development and Integration
System Validation

Although the initial "requirements specification" phase and the "validation " phase are comparable with other software processes, the intermediate phases in a reuse-oriented process are different. These phases are:

- **Component analysis:** A search is made for the components to implement the given requirements specification. Usually, there's no exact match, and components may be only provide some of the functionality required.
- **Requirements modification**: During this phase, the requirements are analyzed using information about the components that have been discovered. They are then modified to reflect the available components. If the modifications are impossible, the *component analysis* activity may be re-entered to search for alternative solutions.
- **System design with reuse:** During this phase, the framework of the system is designed or an existing framework is reused. The designers take into account the components that are reused and they will organize the framework accordingly. Some new software has to be designed if some reusable components are not available.
- **Development and integration:** The components are integrated to create the new system. System integration, in this model, may be part of the development process rather than a separate activity.

There are basically three types of software components that can be used in a reuse-oriented process:

- **Web services** that are developed according to well-known service standards and which will become available for remote invocation.
- **Collections of objects** that are developed as a **package** to be integrated with a component framework such as .NET or Java EE.
- **Standalone software systems** that are configured for use in a particular environment.

It's has an obvious advantage, But!

- This method has an obvious advantage of reducing the amount of software to be developed and therefore reduced cost and risks, and usually leads to faster delivery.
- But, requirements compromises can't be avoided, which may lead to a system that does not meet the real needs of users.

Summary

Waterfall

- It's useful when the **requirements are clear**, or following a very structured process as in critical systems which needs a detailed, precise, and accurate documents describes the system to be produced.
- Not good when **requirements are ambiguous**, and doesn't support frequent interaction with the customers for feedback and proposing changes. It's not suitable for large projects that might take long time to be developed and delivered.

Prototype

- This is very useful when **requirements aren't clear**, and the interactions with the customer and experimenting an initial version of the software results in high satisfaction and a clearance of what to be implemented.
- It's downsides are, good tools need to be acquired for quick development (like coding) in order to complete a prototype. In addition, the costs for training the development team on prototyping may be high.

Summary

Incremental & Iterative

- They're suited **for large projects**, less expensive to the change of requirements as they support customer interactions with each increment.
- Initial versions of the software are produced early, which facilitates customer evaluation and feedback.
- They don't fit into **small projects**, or projects that waterfall are best suited for; A structured process with a detailed, and accurate description of the system.

Spiral

- It's good for high risky or large projects where the requirements are ambiguous. The risks might be due to cost, schedule, performance, user interfaces, etc.
- Risk analysis requires highly specific expertise, and project's success is highly dependent on the risk analysis phase. It doesn't work well for smaller projects.

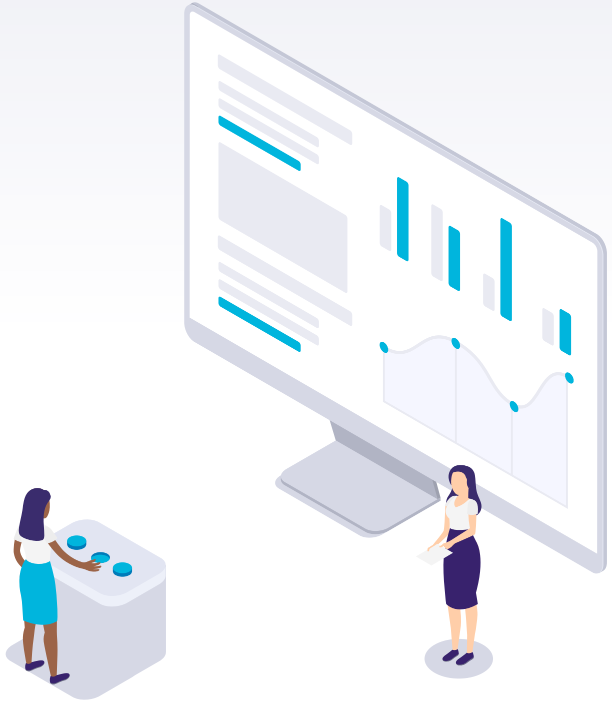
Summary

Agile

- It suits small-medium size project, with rapidly changes in the requirements as customer is involved during each phase.
- Very limited planning is required to get started with the project. It helps the company in saving time and money (as result of customer physical interaction in each phase). The daily meetings make it possible to measure productivity.
- Difficult to scale up to large projects where documentation is essential. A highly skilled team is also needed.
- If team members aren't committed, the project will either never complete or fail. And there's always a limitation in time, like in increments, meetings, etc.

2

Process Activities



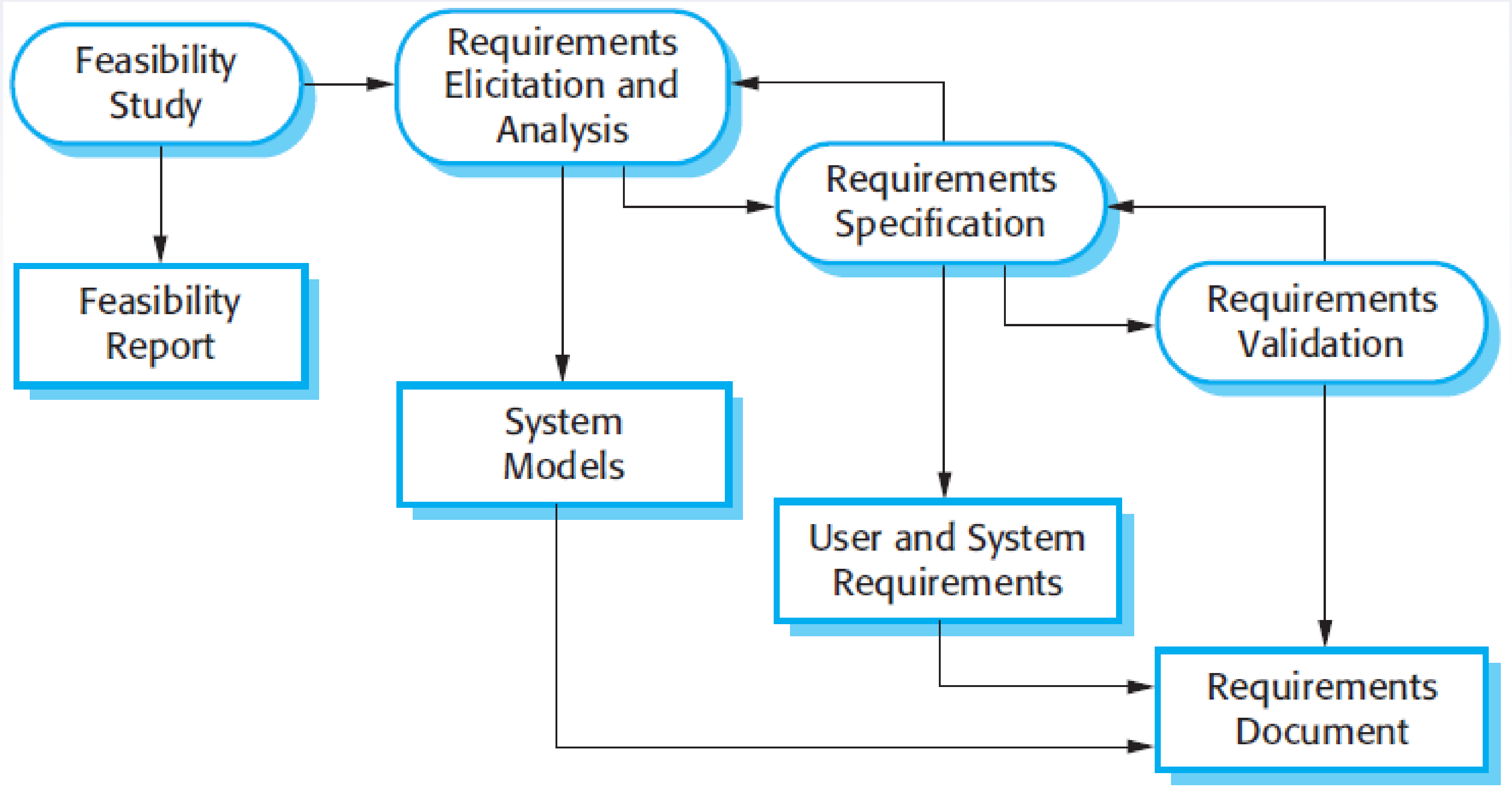
Process Activities

The four basic process activities are:-

- **Requirements Engineering**
- **Software Design And Implementation**
- **Software Verification And Validation**
- **Software Maintenance**

Requirements Engineering

- Software specification or requirements engineering is the process of understanding and defining what services are required and identifying the constraints on these services.
- Requirements engineering processes ensures your software will meet the user expectations, and ending up with a high quality software.
- It’s a critical stage of the software process as errors at this stage will reflect later on the next stages, which definitely will cause you a higher costs.
- At the end of this stage, a requirements document that specifies the requirements will be produced and validated with the stockholders.
- There are four main activities (or sub-activities) of requirements engineering:

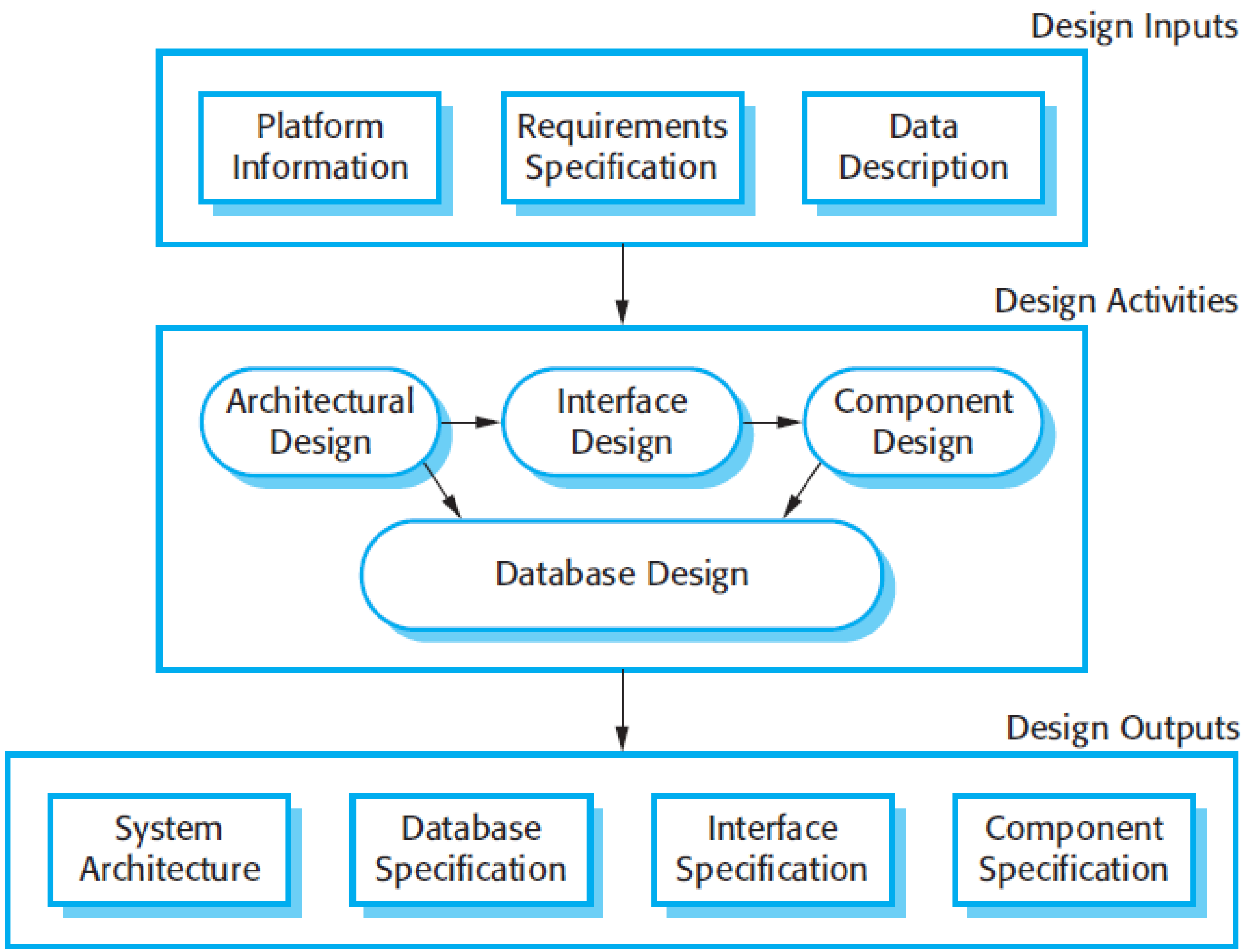

Feasibility Study
Requirements Elicitation and Analysis
Requirements Specification
Requirements Validation
Feasibility Report
System Models
User and System Requirements
Requirements Document

- **Feasibility study:** An estimate is made of whether the identified can be achieved using the current software and hardware technologies, under the current budget, etc. The feasibility study should be cheap and quick; it should inform the decision of whether or not to go ahead with the project.
- **Requirements elicitation and analysis:** This is the process of deriving the system requirements through observation of existing systems, discussions with stakeholders, etc. This may involve the development of one or more system models and prototypes that can help us understanding the system to be specified.
- **Requirements specification:** It's the activity writing down the information gathered during the elicitation and analysis activity into a document that defines a set of requirements. Two types of requirements may be included in this document; user and system requirements.
- **Requirements validation:** It's the process of checking the requirements for realism, consistency and completeness. During this process, our goal is to discover errors in the requirements document. When errors are found, it must be modified to correct these problems.

In agile methods, requirements are developed incrementally according to user priorities and the elicitation of requirements comes from users who are part of the development team.

Software Design And Implementation

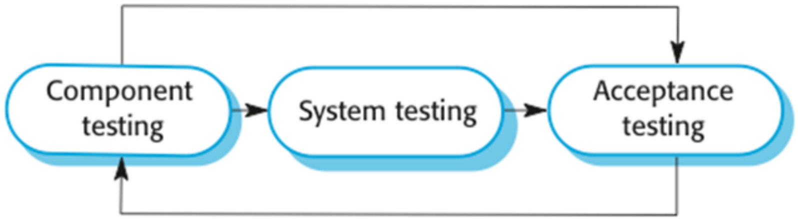
- The implementation phase is the process of converting a system specification into an executable system. If an incremental approach is used, it may also involve refinement of the software specification.
- A software design is a description of the structure of the software to be implemented, data models, interfaces between system components, and maybe the algorithms used.
- The software designers develop the software design iteratively; they add formality and detail and correct the design as they develop their design.
- Here’s an abstract model of the design process showing the inputs, activities, and the documents to be produced as output.


Design Inputs
Platform Information
Requirements Specification
Data Description
Design Activities
Architectural Design
Interface Design
Component Design
Database Design
Design Outputs
System Architecture
Database Specification
Interface Specification
Component Specification

- The diagram suggests that the stages of the design process are sequential. In fact, they are interleaved. A feedback from one stage to another and rework can't be avoided in any design process.
- These activities can vary depending on the type of the system needs to be developed. We've showed four main activities that may be part of the design process for information systems, and they are:
- **Architectural design:** It defines the overall structure of the system, the main components, their relationships.
- **Interface design:** It defines the interfaces between these components. The interface specification must be clear. Therefore, a
- component can be used without having know it's implemented. Once the interface specification are agreed, the components can be designed and developed concurrently.
- **Component design:** Take each component and design how it will operate, with the specific design left to the programmer, or a list of changes to be made to a *reusable* component.
- **Database design:** The system data structures are designed and their representation in a database is defined. This depends on whether an existing database is to be reused or a new database to be created.

Software Verification And Validation

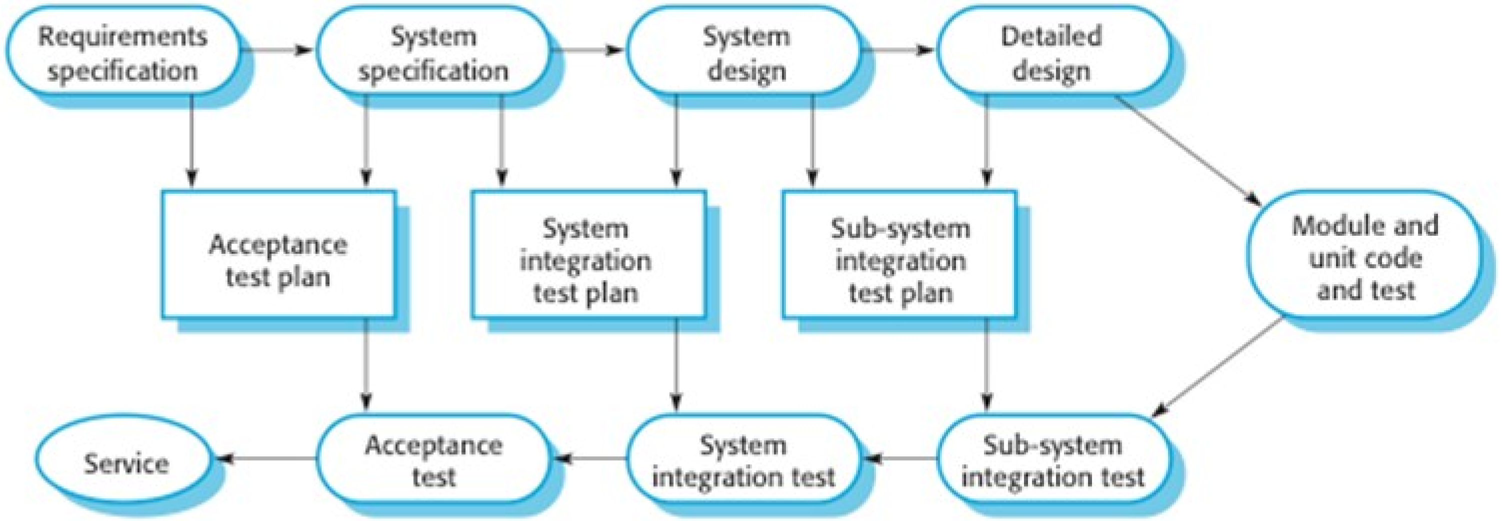
- Software validation or, more generally, verification and validation (V&V) is intended to show that a system both conforms to its specification and that it meets the expectations of the customer.
- Validation may also involve checking processes, such as inspections or reviews at each stage of the software process, from defining the requirements till the software development.
- Testing, where the system is executed using simulated test data, is an important validation technique.



Testing has three main stages:

- **Development (or component) testing:** The components making up the system are tested by the people developing the system. Each component is tested independently, without other system components.
- **System testing:** System components are integrated to create a complete system. This process is concerned with finding errors that result from interactions between components. It is also concerned with showing that the system meets its functional and non-functional requirements.
- **Acceptance testing:** This is the final stage in the testing process before the system is accepted for operational use. The system is tested with data supplied by the system customer rather than using simulated test data. It may reveal errors in the system requirements definition.

Components may be simple entities such as functions or object classes, or may be coherent groupings of these entities. Test automation tools, such as JUnit are commonly used to run component tests.


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

Testing phases in a plan-driven software process

Software Maintenance

- Requirements are always changing, even after the system has been put into its operating environment. The flexibility of software systems is one of the main reasons why software is being used in large, complex systems.
- Historically, there has always been a split between the process of software development and the process of software evolution (software maintenance).
- However, this distinction is increasingly irrelevant, and it makes much more sense to see development and maintenance as a continuum.
- Rather than two separate processes, it is more realistic to think of software engineering as an evolutionary process where software is continually changed over its lifetime in response to changing requirements and customer needs.

3

Software Requirements Specification (SRS) Document

How to Write a Software Requirements Specification (SRS Document)

At a glance, this is how to write a requirements document:

- **Define the purpose of your product.**
- **Describe what you're building.**
- **Detail the requirements.**
- **Get it approved.**

A typical SRS includes:

- **A purpose**
- **An overall description**
- **Specific requirements**

Software Requirements Specification vs. **System Requirements Specification**

- A **software requirements specification (SRS)** includes in-depth descriptions of the software that will be developed.
- A **system requirements specification (SyRS)** collects information on the requirements for a system.

How to Write an SRS Document

1. Create an Outline (Or Use an SRS Template)

This may be something you create yourself. Or you may use an existing SRS template.

2. Start With a Purpose

The introduction to your SRS is very important. It sets the expectation for the product you're building.

3. Give an Overview of What You'll Build

Your next step is to give a description of what you're going to build. Is it an update to an existing product? Is it a new product? Is it an add-on to a product you've already created?

4. Detail Your Specific Requirements

The next section is key for your development team. This is where you detail the specific requirements for building your product.

- *System Features* – User, Hardware, Software, Communications
- *Other Nonfunctional Requirements* - Performance, Safety, Security, Quality

5. Get Approval for the SRS

Once you've completed the SRS, you'll need to get it approved by key stakeholders. And everyone should be reviewing the latest version of the document.

1. Introduction
1.1 Purpose
1.2 Intended Audience
1.3 Intended Use
1.4 Scope
1.5 Definitions and Acronyms
2. Overall Description
2.1 User Needs
2.2 Assumptions and Dependencies
3. System Features and Requirements
    3.1 Functional Requirements
    3.2 External Interface Requirements
    3.3 System Features
    3.4 Nonfunctional Requirements

Prepare software requirement specification for web application