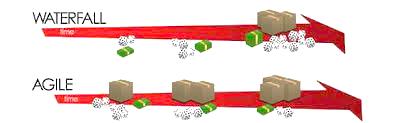
Chapter 2: Software Processes

**- What all processes have in common:**

**Specification:** Define what the system should do

**Design & Implementation:** Define the organization & Implement the system itself

**Validation:** Check that it does what the customer wants

 **Evolution:** Changing the system in response to changing customer needs

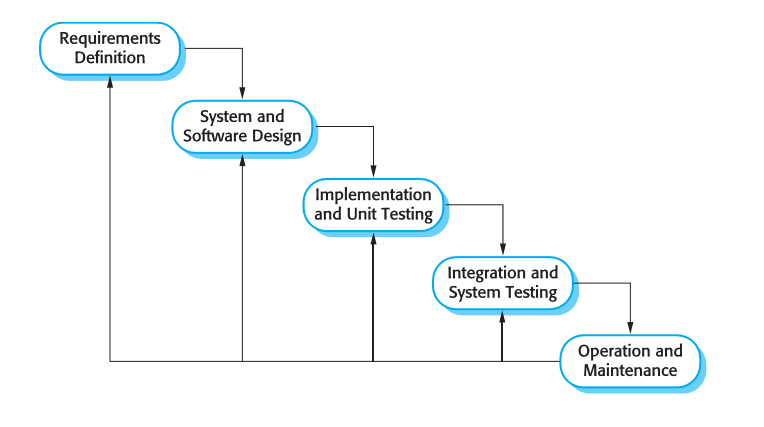
**-Planning:**

**Plan-Driven:** All activities are planned in advance

**Agile:** Planning is incremental => changes are easier to make

**-Software Process Models:**

**Waterfall: Plan-driven** model, separate and distinct phases of specification and development

**Phases:**

- **Requirements** analysis and definition

- System and software **design**

- **Implementation** and unit **testing**

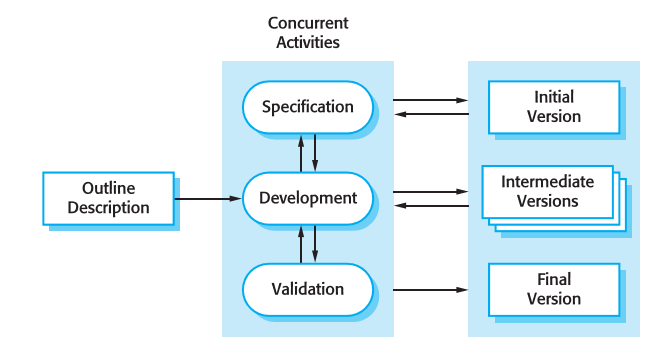
- **Integration** and system **testing**

- Operation and **maintenance**

**Drawbacks:** Changes are not easy to do due to inflexible partitioning. A phase has to be complete before moving onto the next phase

**Use for large systems developed over several sites**

**Incremental Development: Can be Plan-driven or Agile,** specification, development and validation are interleaved

**Benefits:**

- **Reduced** cost of changing customer requirements (less documentation has to be redone)

- Easier to get **feedback** on work already done

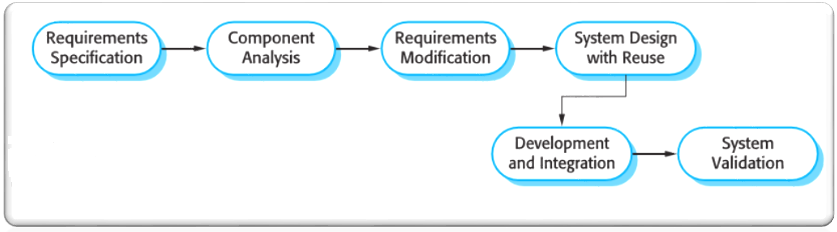
- **Rapid delivery** of useful software to the customer which he can **use and gain** from before the project is finished

- Operation and **maintenance**

**Drawbacks:** Process isn’t visible, customer needs regular deliverables to measure it

-Structure tends to degrade per new incrimination

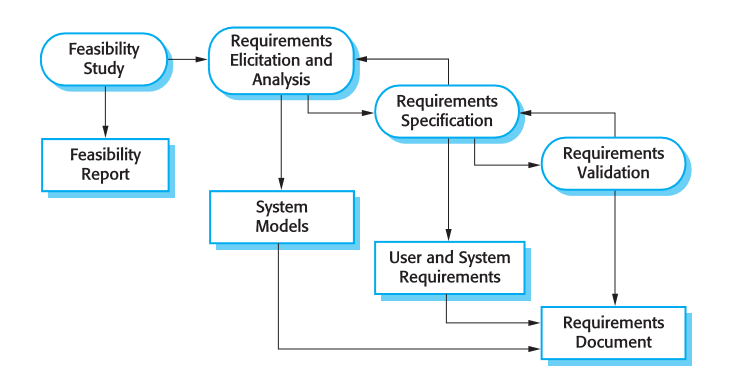
**Reuse-oriented: Can be Plan-driven or Agile,** the system is assembled from existing components

- ****Based on systematic reuse where systems are integrated from **existing components** or COTS (Commercial-off-the-shelf) systems.

-Reuse is now the standard approach for building many types of business systems

**Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.**

**The four basic process activities of specification, development, validation and evolution are organized differently in different development processes**

**-Software specification:** establishing what services are required and constraints on the system’s operation and dev.

**Requirements engineering** process:

**Feasibility study**: Is it technically and financially feasible to

build the system?

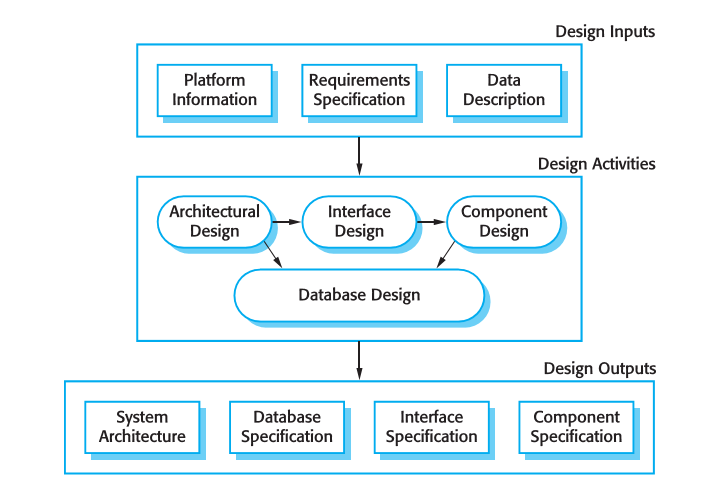
**Requirements elicitation and analysis**: What does the

customer require or expect from the system?

**Requirements specification**: Defining the requirements in detail

**Requirements validation:** Checking the validity of the requirements

**-Design & Implementation:** converting the system specification into an executable system

**Software Design:** Design a software structure that realises the specification

**Implementation:** Translate this structure into an executable program

Design & implementation are closely related and may be inter-leaved

**Design Activities:**

***Architectural design:*** identify the overall structure of the system, the

principal component, their relationships and how they are distributed

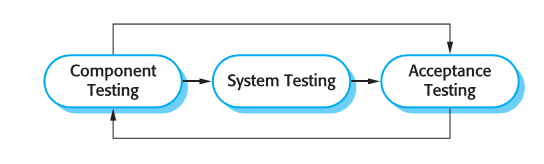
***Interface design****:* define the interfaces between system components

***Component design****:* design how each component will operate

***Database design****:* design the system data structures and how

these are to be represented in a database.

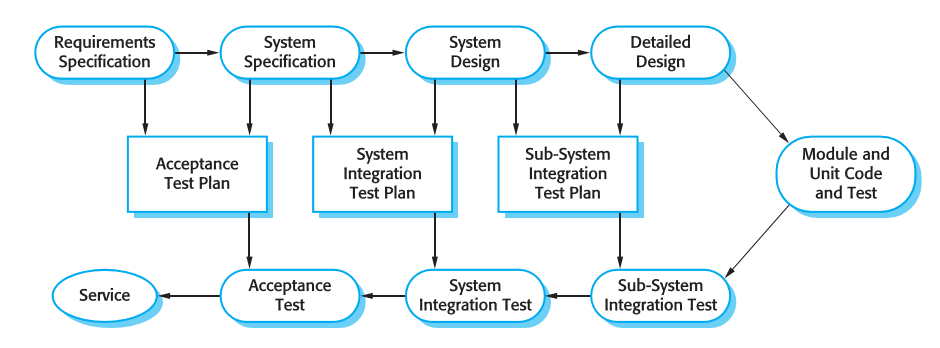
**-Verification & Validation:** show that a system **conforms** to its specifications & meets the customer’s requirements

**Testing** is the most commonly used V & V activity

**Testing Stages:**

**Development/component testing:** Individual components are tested

independently, components may be functions or objects or coherent

groupings of these entities.

**System testing:** Testing of the system as a whole. Testing of emergent properties is particularly important.

**Acceptance testing:** Testing with customer data to check that the system meets the customer’s needs.

**-Software Evolution:** software should be inherently flexible and can change

As requirements change through changing business circumstances, the software that supports the business must also evolve and change to remain useful.

**- Coping with Change**

**Caused by:** Business changes (requirements), new technologies (possibility for improvement), and changing platforms

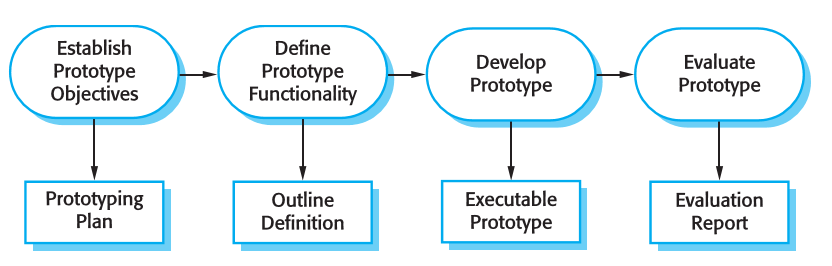
**Leads to** rework so the costs of change include both rework (e.g. re-analyzing requirements) as well as the costs of implementing new functionality

**- Cost Reduction**

**Avoid Change** by including activities that **anticipate** possible changes before the rework is required (Prototype systems)

**Change Tolerance** is achieved by following processes designed to **accommodate** changes at low cost

**- Prototyping:** initial version of a system used to demonstrate concepts and try out design options

**Used in:**

**Requirements engineering** process to help with requirements elicitation and validation

**Design processes** to explore options and develop a UI design

**Benefits:**

- Improved system usability

- A closer match to users’ real needs May be based on rapid prototyping languages or tools

- Improved design quality May involve leaving out functionality such as error checking and recovery

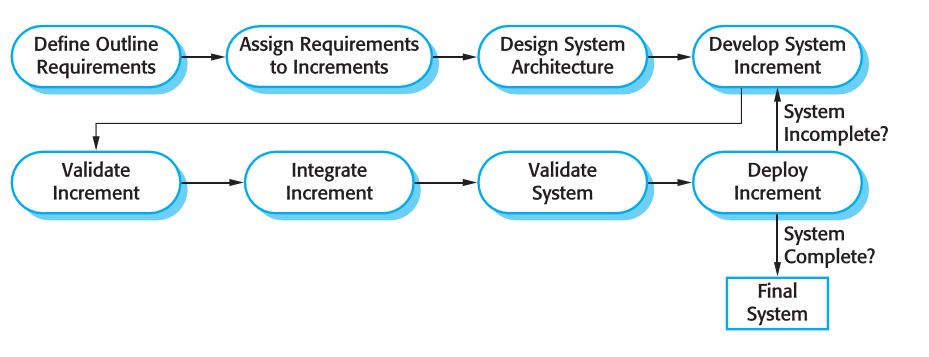
- Improved maintainability Must focus on areas that are **not well-understood**

- Reduced development effort Should be **discarded** and are normally undocumented (quality standards not met)

**- Incremental Delivery:** development is broken down into increments with each giving part of the requirements

**Prioritize** user requirements and deliver them early on

Once work is started on an increment, its specific requirements are **frozen** though later ones continue to evolve

- E**valuate** each increment before proceeding

to the development of the **next increment**

(usually used in **agile** methods).

-Evaluation is done by **user/customer proxy**

**-Deploy** an increment for use **by end-users** to

get more realistic evaluation about practical use of software

**Note: Difficult** to do for **replacement systems** as increments have less functionality than that being replaced

**Advantages**

- System functionality is available earlier

- Lower risk of overall project failure

- High priority services receive the most testing

**Disadvantages**

Systems require basics facilities from different parts thus it is hard to identify common ones needed by all increments

**- Boehm’s spiral model:** process is represented as a **spiral** rather than as a **sequence** of activities with backtracking

**Explicitly** assesses and resolves risks

**Sectors:**

-Objective setting

-Risk assessment and reduction

-Development and validation

-Planning

- Spiral model has been very influential in helping people think about iteration in software processes and introducing the risk-driven approach to development

- In practice, however, the model is rarely used as published for practical software development

