INTRODUCTION TO SOFTWARE ENGINEERING

VAISHNAVI

ASSISTANT PROFESSOR

Chapter 1 - INTRODUCTION

- Software engineering is essential for the functioning of governments, societies, and global businesses.
- Modern infrastructure, utilities, and electronics rely heavily on software systems.
- Industries like manufacturing, finance, and entertainment are fully software-driven.
- Over 75% of the global population uses software-controlled mobile phones, most internet-enabled.
- Software is abstract and not limited by physical laws, allowing for vast possibilities.
- There are many types of software systems, from embedded devices to large global systems.

- All software applications need engineering, but not the same tools or methods.
- There are still many reports of software projects failing.

- These failures happen mainly because of two reasons:

1. **Increasing system complexity** :
2. **Failure to use software engineering methods**:

History of software engineering :

- The notion of software engineering was first proposed in 1968 at a conference held to discuss what was then called the software crisis (Naur and Randell 1969).
- People realized that small programming methods didn’t work well for large, complex systems.
- These big systems were often unreliable, too expensive, and delivered late.
- In the 1970s and 1980s, new methods like structured programming and object-oriented programming were created.
- These became the foundation of today’s software engineering.

Chapter 1.1 - Professional software development

A professional system may have many programs, user guides, and setup instructions.

There are two kinds of software product:

1. **Generic products :** They are standalone systems created to be sold to anyone on the open market. Examples :mobile apps, word processors.
2. **Customized (or bespoke) software:** Customized software is made for a specific customer. Examples: control systems, business tools, and air traffic control systems.

Question	Answer
What is software?	Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good software?	Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production from initial conception to operation and maintenance.
What are the fundamental software engineering activities?	Software specification, software development, software validation and software evolution.
What is the difference between software engineering and computer science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What is the difference between software engineering and system engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.

What are the key challenges facing software engineering?	Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.
What are the costs of software engineering?	Roughly 60% of software costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
What are the best software engineering techniques and methods?	While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system. For example, games should always be developed using a series of prototypes whereas safety critical control systems require a complete and analyzable specification to be developed. There are no methods and techniques that are good for everything.
What differences has the Internet made to software engineering?	Not only has the Internet led to the development of massive, highly distributed, service-based systems, it has also supported the creation of an "app" industry for mobile devices which has changed the economics of software.

Figure 1.1 Frequently asked questions about software engineering

1.1.1 Software engineering

Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use.

In this definition, there are two key phrases:

1. **Engineering discipline**
2. **All aspects of software production**

Software engineering is important for two reasons:

- People need software that is fast, low-cost, and works well.
- Using good software methods saves money and avoids costly fixes later.

A **software process** is a sequence of activities that leads to the production of a software product.

4 fundamental activities are common to all software processes:

1.Software specification : Defining what the software should do and its constraints. It Involves requirement gathering, analysis, and documentation. Stakeholders, users, and engineers collaborate to specify **functional** (what the system should do) and **non-functional** (performance, security, reliability, etc.) requirements.

2. **Software development:** Converting the specification into an executable system. It Includes **system design** (architecture, data structures, interfaces) and **implementation (coding)** in a programming language. Focuses on transforming ideas and requirements into working software.

3. **Software validation:** Ensuring the software meets user needs and conforms to requirements. Involves activities like unit testing, integration testing, system testing, and acceptance testing. Goal: detect and fix errors before deployment.

4. **Software evolution:** Software is rarely “finished” after release. This activity involves adapting software to changing requirements, fixing issues, and improving performance. Includes **corrective, adaptive, perfective, and preventive maintenance**.

Together, these activities form the **software life cycle**, ensuring the product is **useful, reliable, and adaptable**.

Software engineering is related to both computer science and systems engineering.

1. **Computer science** is concerned with the theories and methods that underlie computers and software systems, whereas **software engineering** is concerned with the practical problems of producing software.
2. **System engineering** deals with building and improving complex systems that include software. It also covers hardware, rules, and setup.

There are four related issues that affect many different types of software:

1. **Heterogeneity** software should work on many devices like phones, tablets, and computers.

2. Business and social change
3. Security and trust
4. Scale

Software engineering diversity :

Software engineering is a structured way to build software, focusing on cost, time, quality, and user needs. **SEMAT**(Software Engineering Method and Theory) proposes a flexible basic process, still under development.

Different types of application :

1. **Stand-alone applications**:

   Example: Microsoft office.

2. **Interactive transaction-based applications :**

Ex: These include web applications such as e-commerce applications

3. **Embedded control systems :**

Examples: software in a mobile (cell) phone, software that controls antilock braking in a car, and software in a microwave oven to control the cooking process.

4. **Batch processing systems :**

Example: salary payment systems.

5. **Entertainment systems :**

These are for personal use and made to entertain people. Most are games that run on special-purpose console hardware.

6. **Systems for modeling and simulation :**

Eg.: Weather or Traffic.

7. **Data collection and analysis systems :**

Eg.: weather monitoring system

8. **Systems of systems :**

Eg.: ERP software

Internet software engineering :

- Software is now used online, often as a **service** (e.g., Google Apps).
- It runs on the **cloud** and can be paid for by usage or supported by ads.

  Examples: **Webmail, online storage, video streaming**.

How software development has changed:

- Software is made by **reusing existing parts**.
- Built and improved **step by step**.
- Uses **web services**.
- Tools like **AJAX** and **HTML5** help create better web apps.

1.2 Software engineering ethics

- Confidentiality
- Competence : You should not misrepresent your level of competence.
- Intellectual property rights : Patents and copyrights.
- Computer misuse : You should not use your technical skills to misuse other people’s computers.

Software Engineering Code of Ethics and Professional Practice

ACM/IEEE-CS Joint Task Force on Software Engineering Ethics and Professional Practices

PREAMBLE

The short version of the code summarizes aspirations at a high level of the abstraction; the clauses that are included in the full version give examples and details of how these aspirations change the way we act as software engineering professionals. Without the aspirations, the details can become legalistic and tedious; without the details, the aspirations can become high sounding but empty; together, the aspirations and the details form a cohesive code.

Software engineers shall commit themselves to making the analysis, specification, design, development, testing, and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety, and welfare of the public, software engineers shall adhere to the following Eight Principles:

1. PUBLIC – Software engineers shall act consistently with the public interest.
2. CLIENT AND EMPLOYER – Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.
3. PRODUCT – Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. JUDGMENT – Software engineers shall maintain integrity and independence in their professional judgment.
5. MANAGEMENT – Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION – Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. COLLEAGUES – Software engineers shall be fair to and supportive of their colleagues.
8. SELF – Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the

Figure 1.3 The A
IEEE Code of Ethi

1.3 - Case studies

The system types that used as case studies are:

- **An embedded system** :

  Example: **insulin pump** for diabetic patients.

- **An information system** :

  Example: **medical records system** used in hospitals.

- **A sensor-based data collection system** : It gathers and processes data from sensors. It must be **reliable** and easy to **maintain**, even in tough environments.

- **A support environment** :

  Example: **Eclipse**, **digital learning environment**

1.3.1 - An insulin pump control system

- An **insulin pump** is a medical device that acts like the **pancreas**.
- It uses **software and sensors** to check blood sugar and deliver insulin.
- People with **diabetes** use it because their pancreas cannot make enough **insulin**.
- **Insulin** helps control sugar (glucose) in the blood.
- Normally, diabetics check their sugar and inject insulin themselves.
- Too much insulin can cause **low blood sugar**, leading to **fainting or even death** and too little insulin causes **high blood sugar**, which can damage the **eyes, kidneys, and heart** over time.

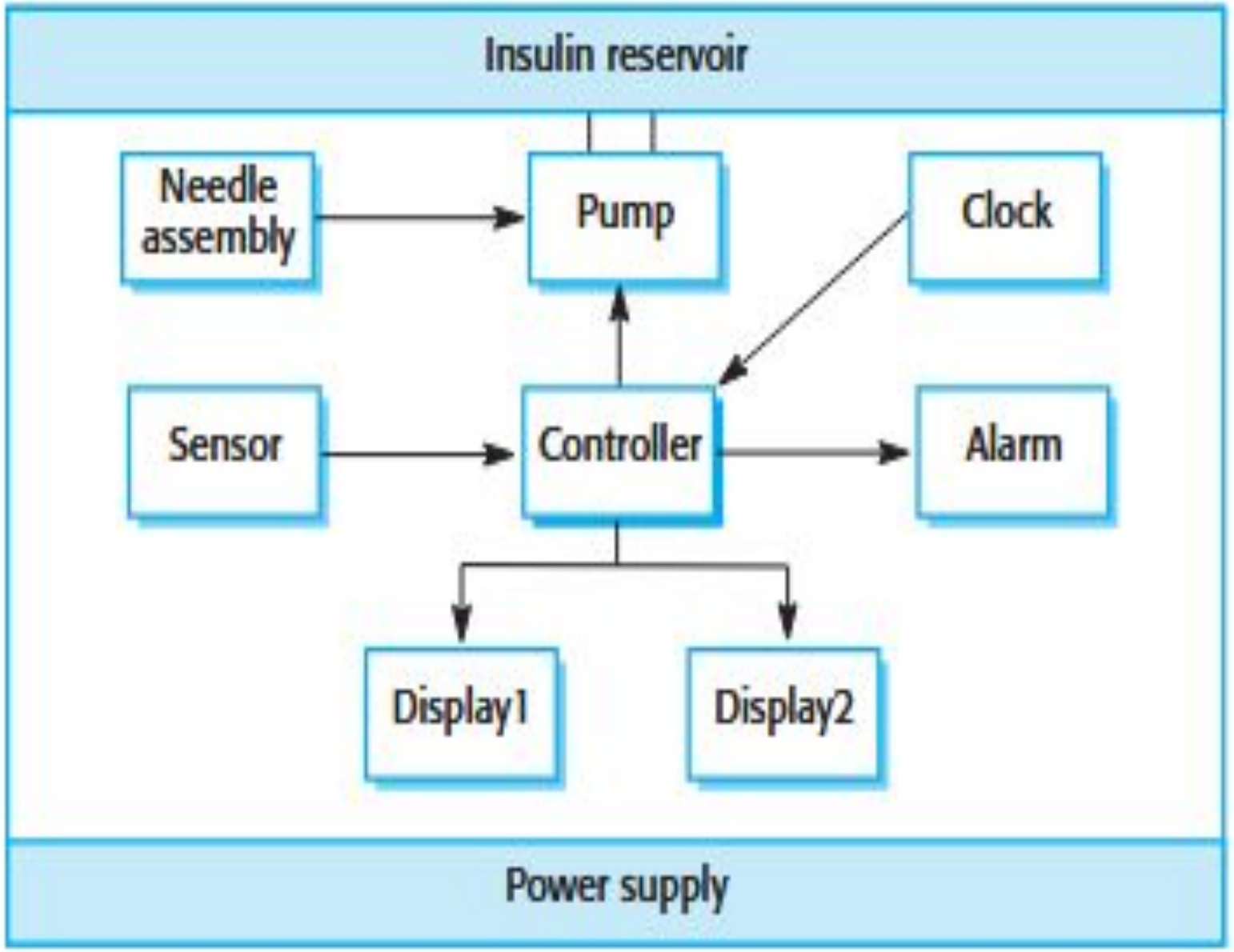


Figure 1.4 Insulin pump hardware architecture

1. **Insulin Reservoir**: Stores the insulin supply. It is connected to the pump to deliver insulin when required.
2. **Pump:** Mechanism that pushes insulin from the reservoir into the **needle assembly** for injection.
3. **Needle Assembly:** Delivers insulin directly into the patient's body.
4. **Sensor:** Continuously monitors the patient's blood glucose level. It sends glucose readings to the **controller**.
5. **Controller (Brain of the system):** Receives input from the **sensor** (blood glucose data) and **clock** (time-based insulin schedules).It makes decisions on **when and how much insulin to deliver**. It sends signals to the pump, displays, and alarm.
6. **Clock:** Provides timing information for scheduled insulin delivery (basal doses).
7. **Alarm:** Alerts the patient in case of abnormal conditions (e.g., low battery, blocked needle, dangerously high/low glucose level).
8. **Displays (Display1 & Display2):** Show information to the patient such as glucose level, insulin dose, battery status, and error messages.
9. **Power Supply:** Provides electrical energy to the entire system for functioning.

Working:

1. The **sensor** continuously monitors the patient's blood glucose level.
2. Sensor data is sent to the **controller**.
3. The **controller**, with input from the **clock**, decides if insulin is needed.
4. If insulin is required, the **controller** activates the **pump**.
5. The pump pushes insulin from the **reservoir** to the **needle assembly**, injecting it into the patient.
6. Information about dosage, glucose levels, and pump status is shown on **Display1 & Display2**.
7. The **alarm** is triggered in case of irregularities (e.g., missed dose, blockage, low insulin).
8. The cycle repeats continuously, ensuring proper blood glucose regulation.

- A **microsensor** checks blood sugar, and the **controller** calculates how much insulin is needed.
- The controller then sends pulses to a **mini pump**, which injects the insulin.
- For example, 10 pulses = 10 units of insulin.

So, the system must:

- Always be ready to give insulin when needed.
- Work properly and give the **right amount of insulin**.

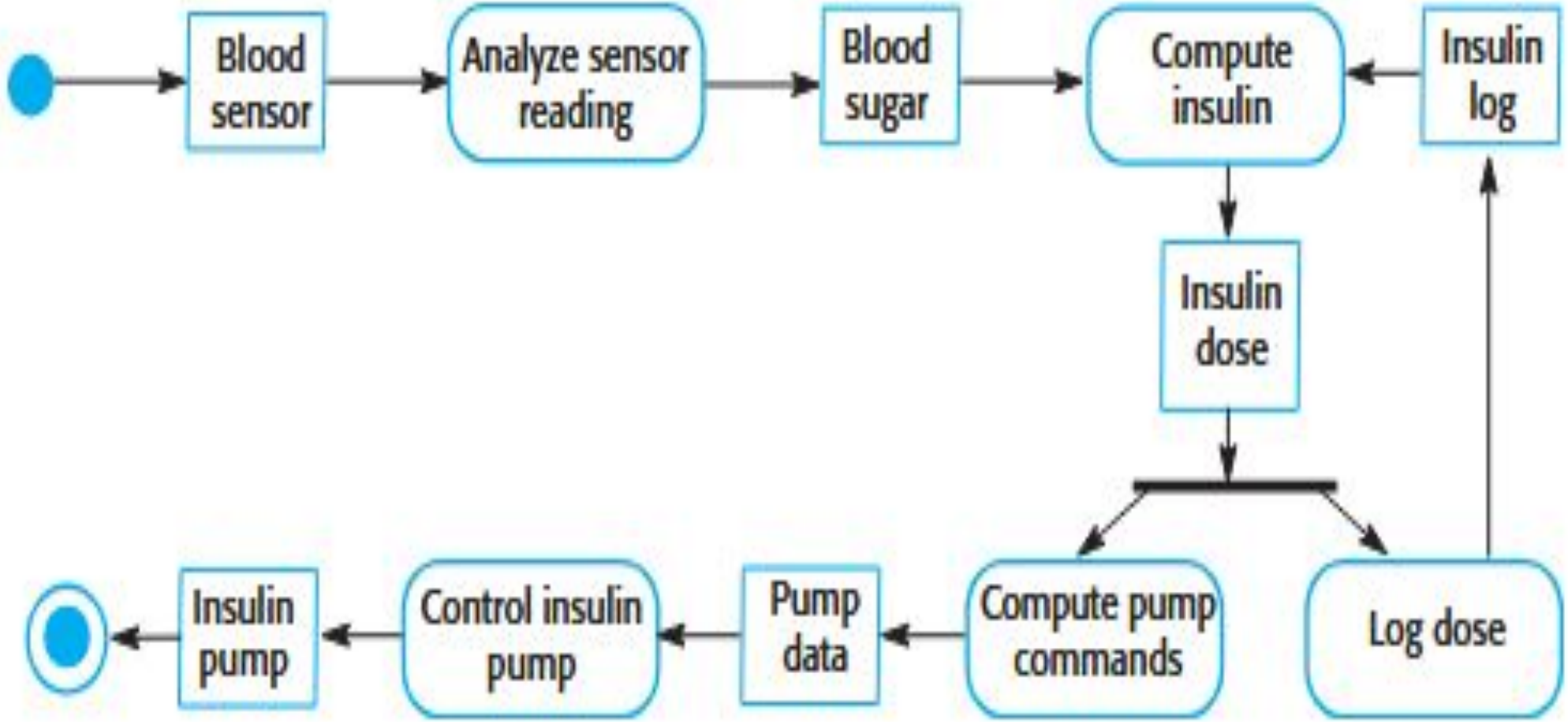


Figure 1.5 Activity model of the insulin pump

1.3.2 - A patient information system for mental health care

- Mentcare is a system for mental health care.
- It stores patient and treatment details.
- Most patients don’t stay in hospitals.
- They visit clinics to meet doctors
- Clinics are in hospitals, local medical centers, or community centers

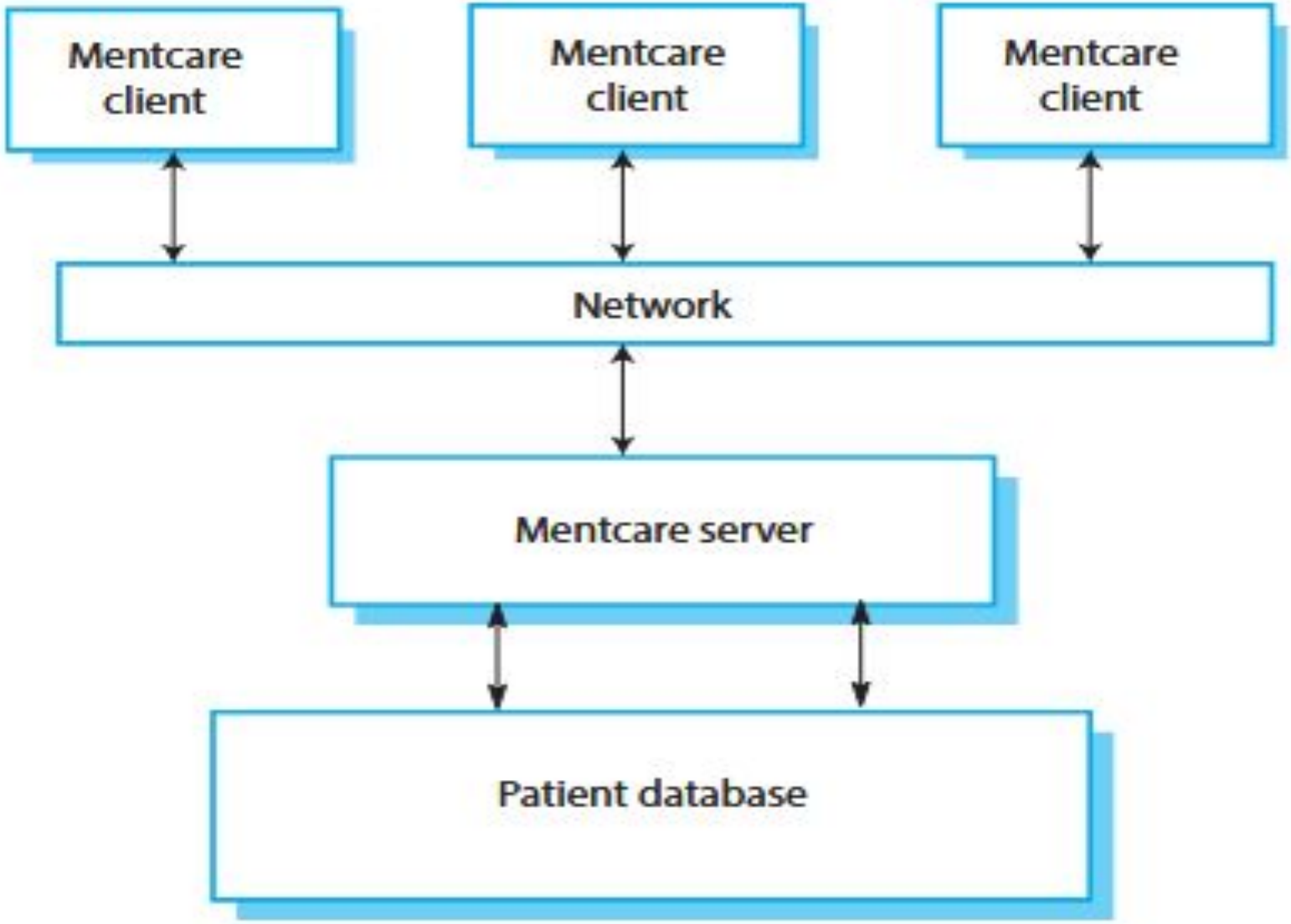


Figure 1.6 The organization of the Mentcare system

Mentcare Clients (Users):

- These are laptops or computers used by doctors, nurses, or staff.
- They are connected to the network to access patient data.

Network :

- Connects all the Mentcare clients to the main server.
- If the network is unavailable, the client can use local copies of patient data.

Mentcare Server :

- The main system that processes requests from the clients.
- It manages data access and updates between clients and the patient database.

Patient Database :

- Stores all patient information, consultations, treatments, and reports.
- The server communicates with the database to fetch or store data.

Working:

- A user (client) requests patient information through the Mentcare application.
- The request is sent via the network to the Mentcare server.
- The server retrieves or updates the information in the patient database.
- The data is then sent back to the client for use.
- If the client is offline, it can use a local copy of the patient records and later sync with the server when connected.

The system has two purposes:

1. To give managers information to check performance.
2. To give medical staff quick information to treat patients.

The key features of the system are:

- Individual care management
- Patient monitoring
- Administrative reporting

Two laws affect the system:

1. Data protection laws
2. Mental health laws

1.3.3 - A wilderness weather station

- The government wants to monitor climate change and improve weather forecasts.
- They will set up hundreds of weather stations in remote areas.
- These stations will collect data on:

Temperature and pressure, Sunshine, Rainfall, Wind speed and direction.

- Wilderness weather stations are part of a bigger **weather information system**.
- This system collects data from stations and shares it with other systems (Weather forecasting systems, Climate change monitoring systems, Disaster warning systems like floods or storms, Agriculture or farming support systems) for processing.

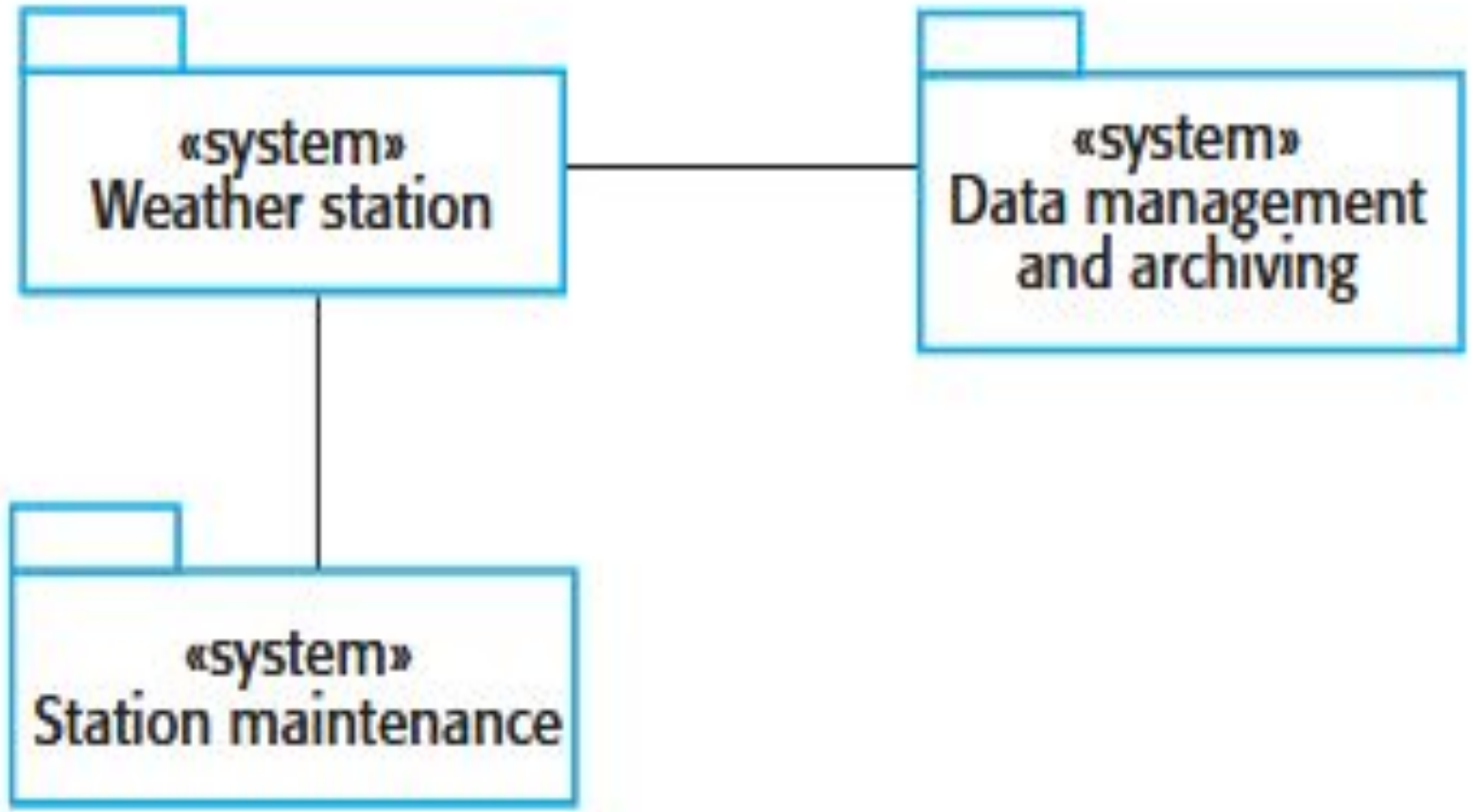


Figure 1.7 The weather station's environment

- **Weather station system** – Collects weather data, does basic processing, and sends it to the data system.
- **Data management system** – Gathers data from all stations, processes, analyzes, and stores it for other systems like forecasting.
- **Station maintenance system** – Uses satellite to monitor stations, fix problems, update software, and control stations remotely.

- Weather stations are battery-powered and fully self-contained.
- They use solar or wind power to charge batteries.
- Communication is via a slow satellite link.
- **Software tasks**: Monitor instruments, power, and communication hardware; report faults, Manage power, Support reconfiguration

1.3.4 - A digital learning environment for schools

- A **digital learning environment** is a framework with tools and applications for learners' needs.
- It provides services like login, communication, and storage.
- Teachers and students choose tools like spreadsheets, simulations, and specific learning content.

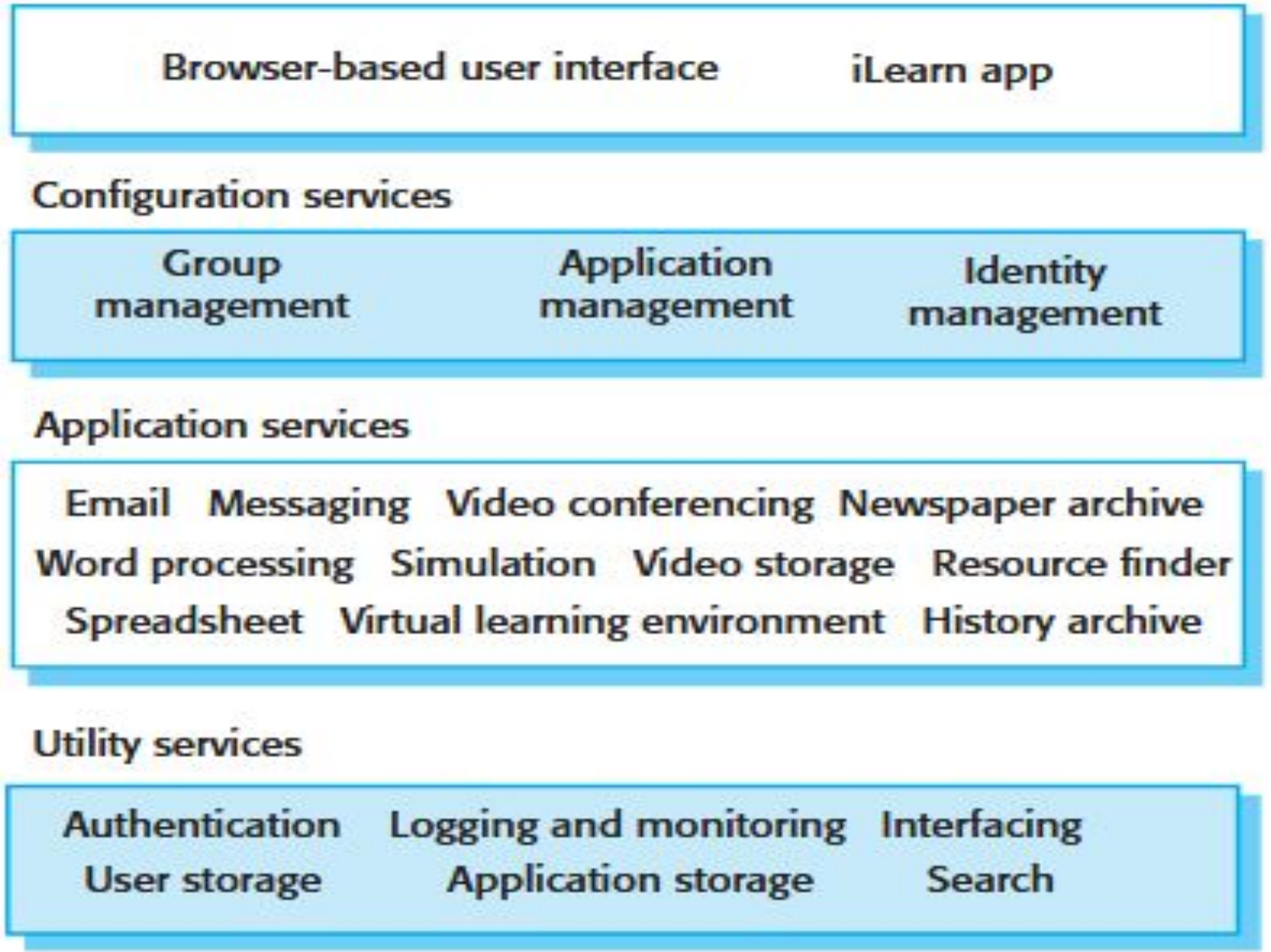


Figure 1.8 The architecture of a digital learning environment (iLearn)

- **iLearn** is a digital learning system for students aged 3–18.
- It is a distributed, service-oriented system where all parts can be accessed online and replaced if needed.

Three types of services:

1. **Utility services** – basic functions used by other services (built for the system).
2. **Application services** – apps like email, video, photo sharing, and educational content (bought or free online).
3. **Configuration services** – set up which apps are used and how they are shared by students, teachers, and parents.

Two types of service integration:

1. **Integrated services** – Have an API, can communicate directly with other services (e.g., authentication service). Users don't need to log in again.
2. **Independent services** – Accessed through a browser, work alone, and require separate logins. Data is shared manually (e.g., copy-paste).

If an independent service is widely used, it can later be made an integrated service.

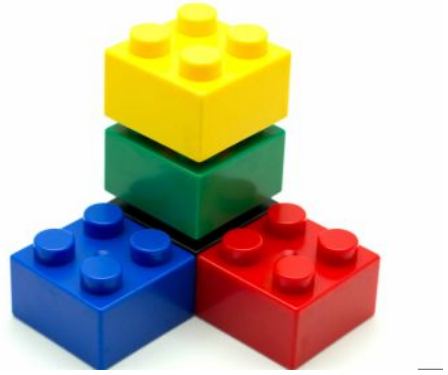
THANK YOU

Chapter 2
Software processes

Vaishnavi
Assistant Professor

A **software process** is a set of activities to build software.

- A **software process model** (or SDLC) is a simplified view of a software process.
- Each model shows only part of the process (e.g., activities and order, not roles).
- These models are high-level frameworks that explain different development approaches.
- They can be adapted to create specific software processes.



The general process models are:

- **Waterfall model** – Follows phases: requirements, design, coding, and testing in order.
- **Incremental development** – Builds the system in versions, adding features step by step.
- **Integration & configuration** – Uses existing components, configures, and integrates them into a new system.

The waterfall model

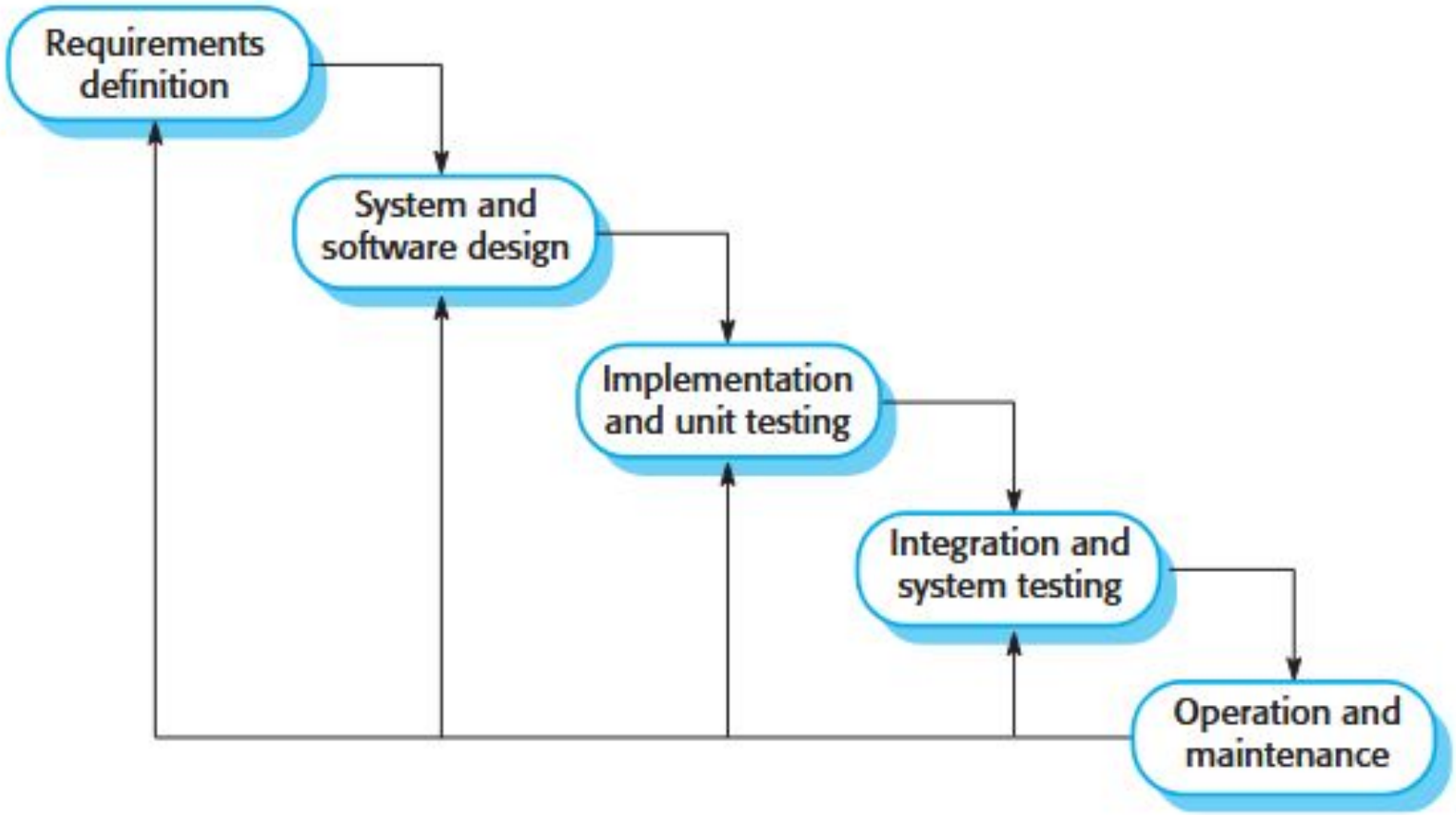


Figure 2.1 The waterfall model

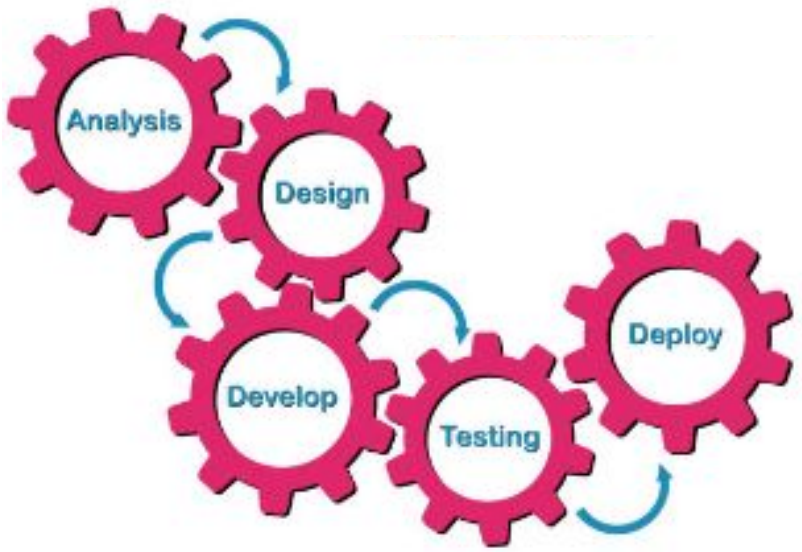
This diagram represents the **Waterfall Model** in software development. It is a **linear and sequential approach**, where each phase must be completed before moving to the next.

1. **Requirements Definition:** Gather and document all functional and non-functional requirements of the software. Its output will be a clear Software Requirement Specification (SRS) document.
2. **System and Software Design:** Based on the SRS, the system architecture and software design are created. High-level design (system architecture) and low-level design (module details) are defined. Its output is the Design documents that guide implementation.
3. **Implementation and Unit Testing:** Developers write code based on the design documents. Each module is tested individually (unit testing) to ensure it works as intended. Its output is Verified program modules.
4. **Integration and System Testing:** All modules are integrated into a complete system. System testing ensures the entire software meets requirements and works correctly. Here Defects are identified and fixed.
5. **Operation and Maintenance:** The software is deployed to users and enters operational use. Maintenance includes fixing bugs, updating features, and improving performance as needed.

- The **waterfall model** (Royce 1970) was the first published software development model.
- It has stages that flow one after another, like a cascade.
- It is a **plan-driven process** where all activities are planned before development starts.

The stages of the waterfall model directly reflect the fundamental software development activities:

- **Requirements analysis and definition**
- **System and software design**
- **Implementation and unit testing**
- **Integration and system testing**
- **Operation and maintenance**



- In the **waterfall model**, one phase must finish and be approved before the next starts.
- Changing requirements needs customer approval, which can slow the project.
- To save time, teams may **freeze requirements too early**, which can cause missing features or poor design.
- After the software is released (**operation and maintenance**), errors are fixed and new features are added. This may require repeating earlier steps.

The **waterfall model** is best for:

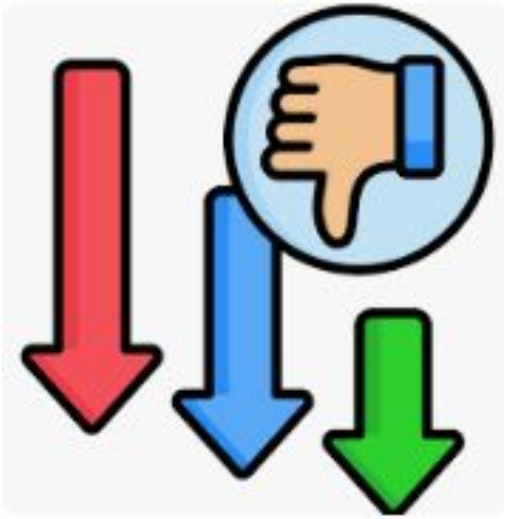
- **Embedded systems** – software must match fixed hardware.
- **Critical systems** – need complete documents for safety and security checks.
- **Big projects with many companies** – need detailed plans so each can work separately.

Advantages:

- Simple and easy to understand.
- Clear structure with well-defined stages.
- Works well for small projects with fixed requirements.

Disadvantages:

- Not flexible if requirements change.
- Hard to change things once a step is finished.
- Testing happens late, so problems are found late.
- Customers see the product only at the end.



Incremental development:

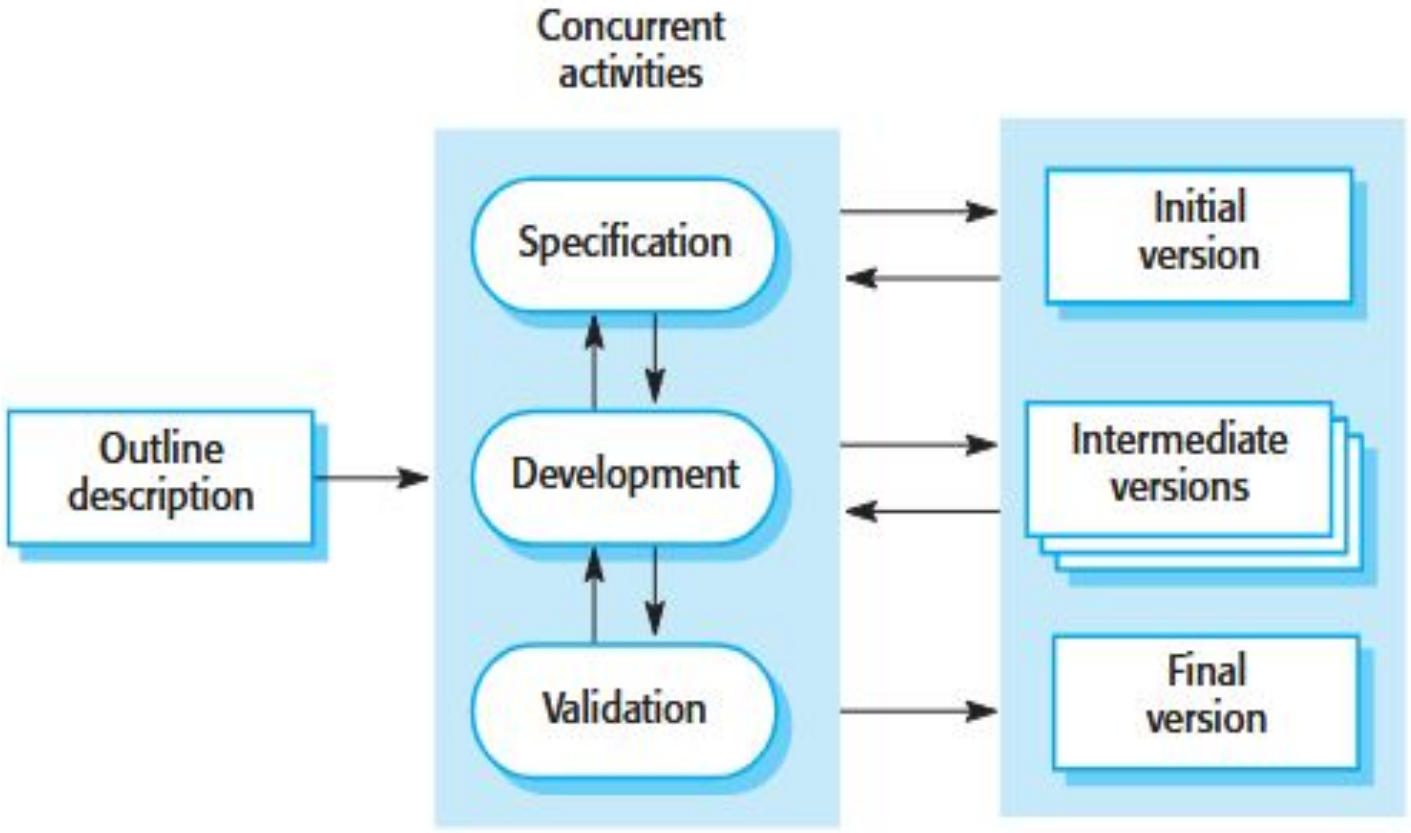


Figure 2.2 Incremental development

Working of Incremental Model:

1. **Outline Description:** This is the starting point, a high-level idea or concept of the system.
2. **Concurrent Activities:**

   These three activities run **in parallel** and interact with each other:

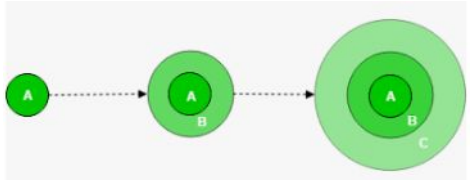
   - **Specification**: Define system requirements and features. It can be updated based on feedback from development and validation.
   - **Development:** Build the software based on the current specification. It may influence or be influenced by changes in specification and validation.
   - **Validation:** Test and verify the software to ensure it meets requirements.

Arrows between these activities show that: They **communicate and influence each other continuously**. Changes in one area can trigger updates in the others.

3. Versions of the System: As the concurrent activities progress, they produce different versions:

- **Initial Version**: First working version based on early specifications.
- **Intermediate Versions**: Updated versions as development and validation refine the system.
- **Final Version**: The completed and validated system.

- **Incremental development** builds software in small versions, getting feedback and improving it step by step.
- Specification, development, and testing happen together with quick feedback.
- Can be **plan-driven**, **agile**, or a mix.
- Better than waterfall model for systems where requirements may change.



Incremental development has three major advantages over the waterfall model:

1. Cheaper and easier to handle changes.
2. Customers can give feedback by using early versions, not just documents.
3. Useful software is delivered early, even without full functionality.

From a management perspective, the incremental approach has two problems:

1. Hard for managers to track progress because frequent documentation is costly.
2. Code can get messy as new features are added, making future changes harder.

- Incremental development can gather feedback without delivering each version to customers.
- **Incremental delivery** means deploying the software for real use, giving more realistic feedback.

Integration and configuration

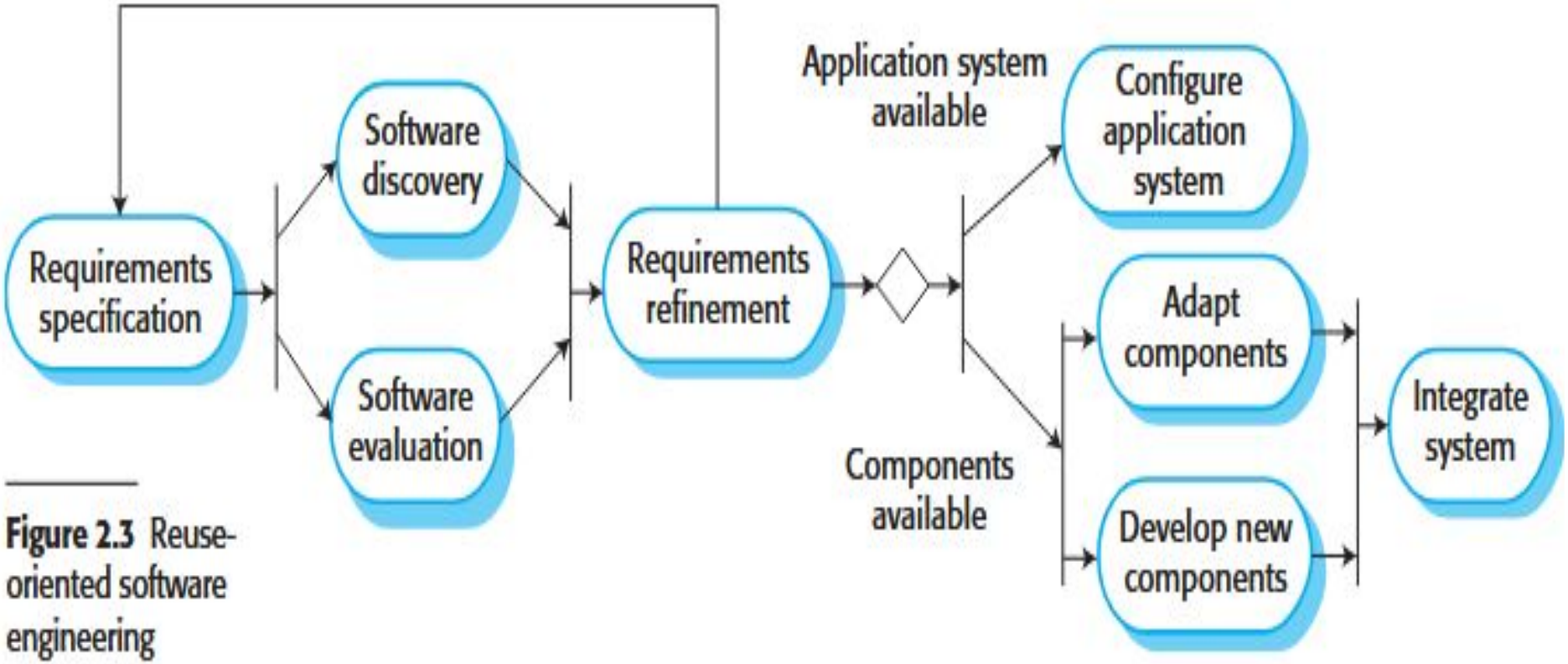


Figure 2.3 Reuse-oriented software engineering

Working of Reuse oriented software engineering:

Figure 2.3 shows a general process model for reuse-based development, based on integration and configuration. The stages in this process are:

1. **Requirements Specification**: define what the system needs to do.
2. **Software Discovery & Evaluation:**
   - **Software Discovery**: Search for existing components or systems that meet the specified requirements.
   - **Software Evaluation**: Assess discovered components for suitability, compatibility, and quality.
3. **Requirements Refinement:** Based on what components are available and suitable, refine the original requirements. This step leads to a decision point:

   - If an **Application System is available**, go to **Configure Application System.**

   - If only **Components are available**, follow the path to: Adapt Components, Develop New Components, Integrate System.

4. Two Possible Paths Forward:

Path A: Application System Available

- **Configure Application System**: Customize the existing system to meet refined requirements.

Path B: Components Available

- **Adapt Components**: Modify existing parts to fit.
- **Develop New Components**: Build new ones if needed.
- **Integrate System**: Combine all parts into the final system.

Final Outcome: Whether you configure an existing system or build one from components, the goal is to **assemble a working software system** using as much reuse as possible.

Integration and configuration:

- Most software projects reuse existing code, often informally by finding and modifying similar code.
- Since 2000, software is often built by reusing existing parts to save time and effort.

Three commonly reused software components:

1. **Standalone applications** – adapted for specific environments.
2. **Object/component packages** – integrated with frameworks (e.g., Java Spring).
3. **Web services** – accessed remotely over the Internet.


Thank you