Chapter 7: Design and Implementation

7.1 Introduction:

Software design and implementation: is the stage in the software engineering process at which an **executable software system** is developed.

- Software design and implementation activities ar **inter-leaved**.
  - **Software design** is a creative activity in which you identify software components and their relationships, based on a customer's requirements.
  - **Implementation** is the process of realizing the design as a program.

- **Build or buy:**

  - In many domains, it is now possible to **buy off-the-shelf systems** (**COTS**) that can be adapted and tailored to the users' requirements.

    - **Ex:** if you want to implement a medical records system, you can buy a package that is already used in hospitals.
    - It can be **cheaper** and **faster** to use this approach rather than developing a system in a conventional programming language.

  - When you develop an application in this way, the design process becomes concerned with **how to use the configuration features** of that system to deliver the system requirements.

7.2 An object-oriented design process:

- **Structured object-oriented design** processes involve developing a number of **different system models**.

- They require a **lot** of **effort** for development and maintenance of these models:
  - **For small systems**, this may not be cost-effective.
  - **For large systems** developed by different groups, **design models** are an important **communication mechanism**.

- **Process stages:**

  - There are a **variety** of different object-oriented design processes that **depend on** the organization using the process.

  - Common activities in these processes include:
    1. **Define the context and modes of use of the system.**
    2. **Design the system architecture.**
    3. **Identify the principal system objects.**
    4. **Develop design models.**
    5. **Specify object interfaces.**

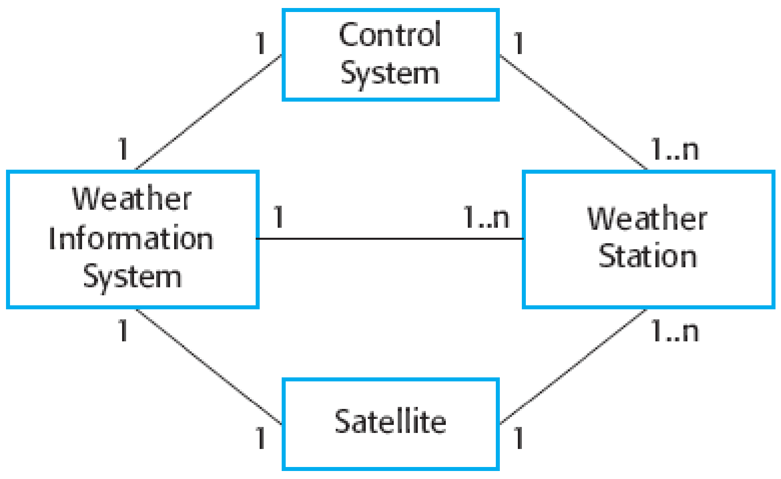
  - Process illustrated here using a design for a **wilderness weather station**.

1. System context and interactions

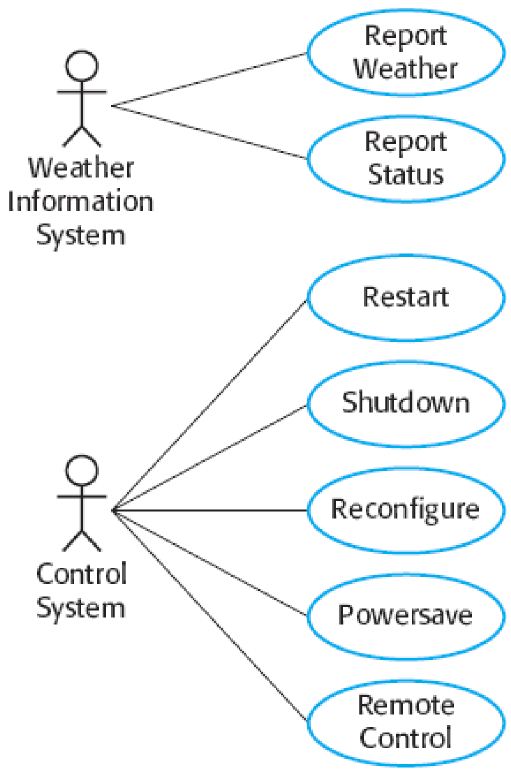
A system context model: is a structural model that **shows the other systems** in the environment of the system being developed.

An interaction model: is a dynamic model that **shows how the system interacts** with its environment as it is used.

- **Ex: System context for the weather station**.



- **Weather Control station use cases.**



- **Use case description: Report weather**

System	Weather station
Use case	Report weather
Actors	Weather information system, Weather station
Dat	The weather station sends a summary of the weather data that has been collected from the instruments in the collection period to the weather information system. The data sent are the maximum, minimum, and average ground and air temperatures; the maximum, minimum, and average air pressures; the maximum, minimum, and average wind speeds; the total rainfall; and the wind direction as sampled at five-minute intervals.
Stimulus	The weather information system establishes a satellite communication link with the weather station and requests transmission of the data.
Response	The summarized data are sent to the weather information system.
Comments	Weather stations are usually asked to report once per hour but this frequency may differ from one station to another and may be modified in the future.

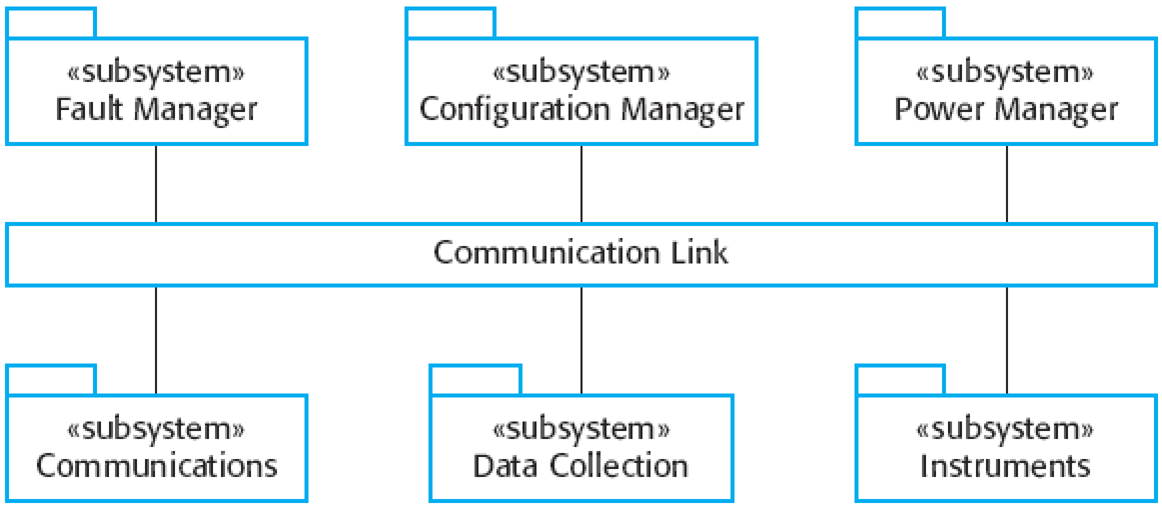
2. <u>Architectural design</u>:

- Once interactions between the system and its environment have been understood, you use this information for **designing the system architecture**.

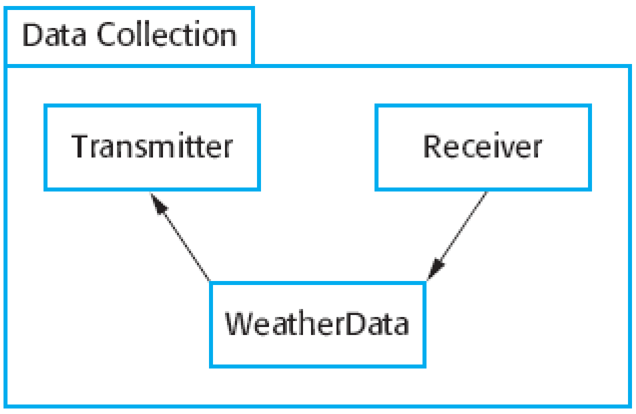
- You identify the **major components** that make up the system and their **interactions**, and **then** may organize the components using an **architectural pattern** such as a **layered** or **client-server model**.

- The **weather station** is composed of independent subsystems that communicate by **broadcasting messages** on a common infrastructure.

- **High-level architecture of the weather station:**


«subsystem» Fault Manager
«subsystem» Configuration Manager
«subsystem» Power Manager
Communication Link
«subsystem» Communications
«subsystem» Data Collection
«subsystem» Instruments

❖ **Architecture of data collection system**



3. Object class identification:

❖ Identifying object classes is often a **difficult part** of object oriented design.

❖ There is no **"magic formula"** for object identification. It **relies on the skill**, **experience** and **domain knowledge** of system designers.

❖ Object identification is an **iterative process**. You are unlikely to get it right first time.

❖ **Approaches to identification**:

1. Use a **grammatical approach** based on a **natural language description** of the system.
   - ✓ Objects and attributes are **nouns**; operations or services are **verbs**.

2. Base the identification on **tangible things** in the application domain.
   - ✓ manager , doctor.

3. Use a behavioural approach and **identify objects** based on **what** participates in **what** behaviour.

4. **Use a scenario-based analysis**. The objects, attributes and methods in each scenario are identified.

❖ **Weather station description:**

A **weather station** is a package of software controlled instruments which collects data, performs some data processing and transmits this data for further processing. The instruments include air and ground thermometers, an anemometer, a wind vane, a barometer and a rain gauge. Data is collected periodically.

When a command is issued to transmit the weather data, the weather station processes and summarises the collected data. The summarised data is transmitted to the mapping computer when a request is received.

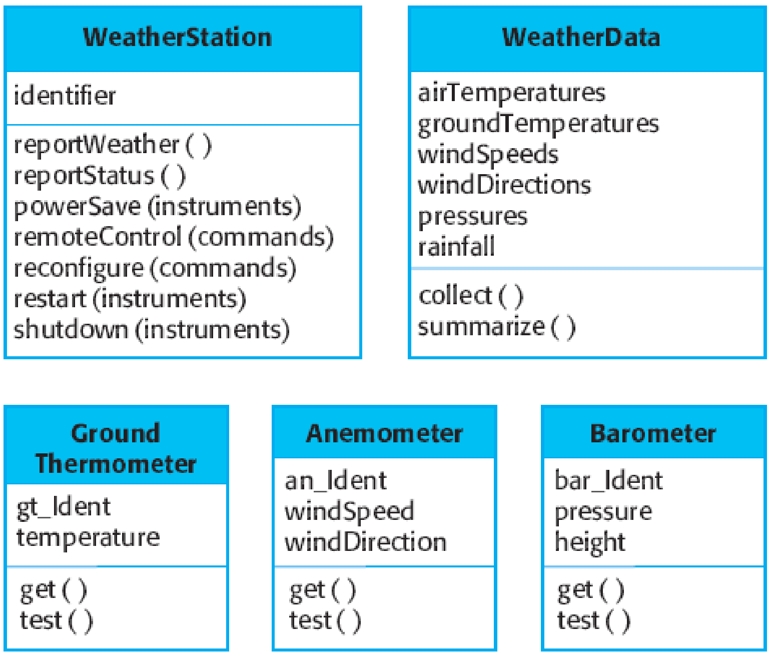
☒ **Weather station object classes:**

Object class identification in the weather station system may be based on the tangible hardware and data in the system:

❖ **Ground thermometer, Anemometer, Barometer**
- Application domain objects that are 'hardware' objects related to the instruments in the system.

❖ **Weather station**
- The basic interface of the weather station to its environment. It therefore reflects the interactions identified in the use-case model.

❖ **Weather data**
- Encapsulates the summarized data from the instruments.

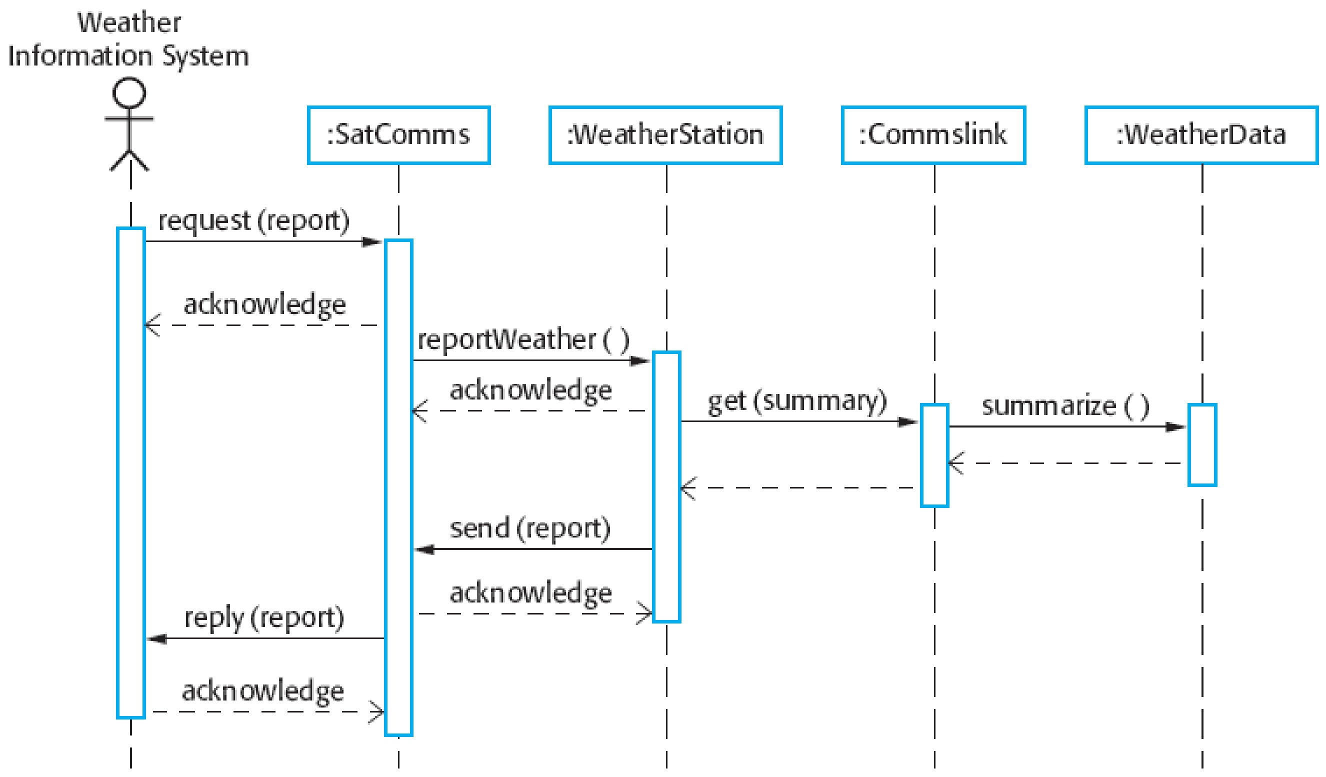


4. Design models:

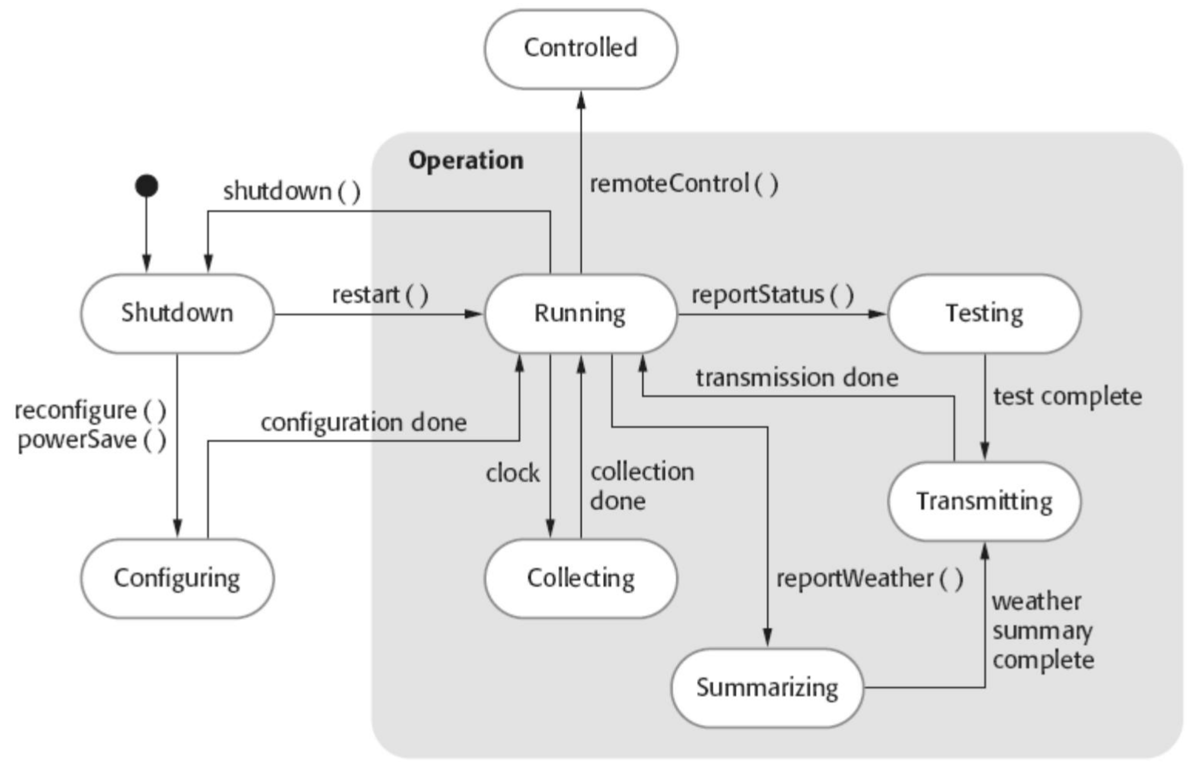
- **Design models** show the objects and object classes and relationships between these entities.
  - **Static models** describe the static structure of the system in terms of object classes and relationships.
  - **Dynamic models** describe the dynamic interactions between objects.

- **Examples of design models:**
  - **Subsystem models** that show logical groupings of objects into coherent subsystems.
  - **Sequence models** that show the sequence of object interactions.
  - **State machine models** that show how individual objects change their state in response to events.

- **Ex: Sequence diagram describing data collection:**



❖ **Ex: Weather station state diagram:**



7.3 <u>Implementation issues</u>:

☒ **Focus** here is **not** on **programming** but on other implementation issues that are often not covered in programming texts:

1. <u>Reuse</u>:

❖ Most modern software is constructed by reusing existing components or systems.

❖ **<u>Reuse levels</u>:**

- **The abstraction level:** you don't reuse software directly but use knowledge of successful abstractions in the design of your software.
- **The object level:** you directly reuse objects from a library rather than writing the code yourself.
- **The component level:** Components are collections of objects and object classes that you reuse in application systems.
- **The system level:** At this level, you reuse entire application systems.

- **Reuse costs:**
  - **The costs of the time** spent in looking for software to reuse.
  - Where applicable, **the costs of buying** the reusable software.
  - **The costs of adapting and configuring** the reusable software components.
  - **The costs of integrating** reusable software elements with each other and with the new code that you have developed.

- **Host-target development:**
  - Most software is **developed** on one computer (**the host**), but **runs** on a separate machine (**the target**).
  - **Development platform** and **execution platform**.
    - A platform is more than just hardware.
    - It includes the installed operating system plus other supporting software such as a database management system.
  - Development platform usually has **different installed software** than execution platform.

❖ **Development platform tools**:

1. An **integrated compiler** and **syntax-directed editing system** that allows you to create, edit and compile code.
2. A **language debugging system**.
3. **Graphical editing tools**, such as tools to edit UML models.
4. **Testing tools**, such as Junit that can automatically run a set of tests on a new version of a program.
5. **Project support tools** that help you organize the code for different development projects.

❖ **Integrated development environments (IDEs)**:

- Software development tools are often grouped to create an **integrated development environment (IDE)**.

An **IDE** is a set of software tools that supports different aspects of software development, within some common framework and user interface.

- **IDEs** are created to support development in a specific programming language such as Java.

❖ **Component/system deployment factors**:

1. **The hardware and software requirements of a component.**
2. **The availability requirements of the system High-availability systems** require components to be deployed on more than one platform.
3. **Component communications** if there is a high level of communications traffic between components then deploy them on the same platform or on platforms that are physically close to one other.
    - This reduces **communications latency.**

❑ <u>Open source development</u>:

It is an approach to software development in which **the source code of a software system is published and volunteers are invited to participate** in the development process.

- ☒ Its roots are in the **Free Software Foundation** (www.fsf.org), which advocates that source code **should not be proprietary** but rather **should always be available** for users to examine and modify as they wish.
- ☒ **Open source software extended this idea** by using the Internet to **recruit (يجند)** a much larger population of volunteer developers. Many of them are also users of the code.
- ☒ **Example:**
  - ❖ **Linux operating system** which is widely used as a server system and, increasingly, as a desktop environment.
  - ❖ **Java**, the **Apache web server** and the **mySQL database management system**.

☒ **<u>Open source issues</u>**:

1. Should the product that is being developed make use of open source components?
2. Should an open source approach be used for the software's development?

☒ **<u>Open source business</u>:**

- ❖ **More and more product companies** are using an open source approach to development.
- ❖ Their business model is **not reliant on selling a software product** but **on selling support for that product**.
- ❖ They believe that involving the open source community will allow software to be developed **more cheaply**, **more quickly** and **will create a community of users** for the software.

☒ <u>Open source licensing</u>:

- ❖ A fundamental principle of open-source development is that source code should be freely available, this does not mean that anyone can do as they wish with that code.
- ❖ **<u>License models</u>:**

  1. **The GNU General Public License (GPL):** called **"reciprocal" license** that means that if you use open source software that is licensed under the GPL license, then you must make that software open source.

  2. **The GNU Lesser General Public License (LGPL)** is a variant of the GPL license where you can write components that link to open source code without having to publish the source of these components.

  3. **The Berkley Standard Distribution (BSD) License.** This is a **non-reciprocal license**, which means you are not obliged to re-publish any changes or modifications made to open source code.

- ❖ **<u>License management</u>:**
  - **Establish a system for maintaining information** about open-source components that are downloaded and used.
  - **Be aware** of the different types of licenses and understand how a component is licensed before it is used.
  - **Educate people** about open source.
  - Have **auditing systems in place**.
  - **Participate** in the open source community.