6. DESIGN, REUSE AND IMPLEMENTATION

Contents:

6.0 Design and Implementation

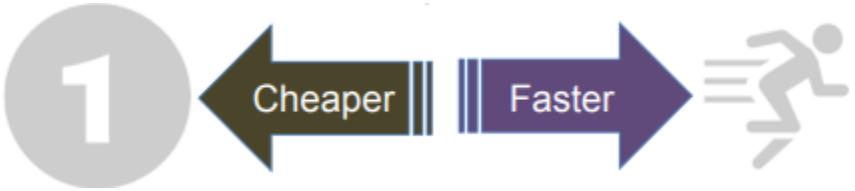
6.0 Introduction: Design and Implementation

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

- Design and implementation activities are 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. These two activities are invariably inter-leaved.

Build or Buy

- In a wide range of domains, it is now possible to buy **commercial off-the-shelf systems** (COTS) that can be adapted and tailored to the users' requirements.
  - For example, 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.



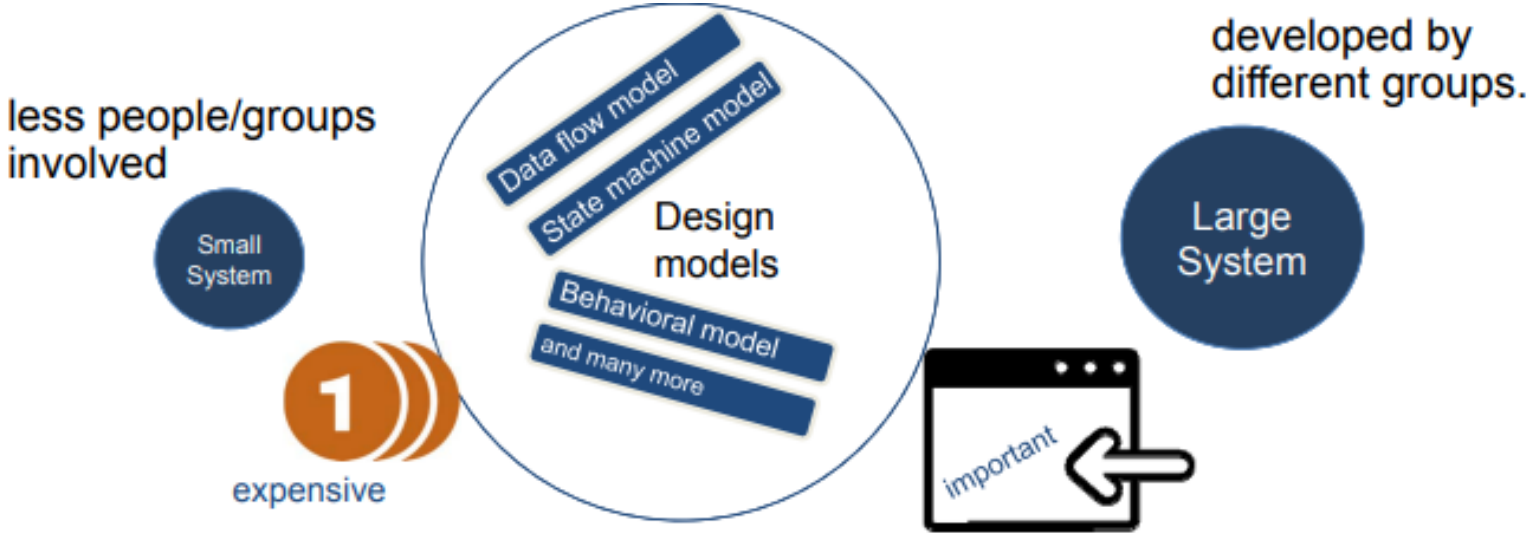
- When you develop an application in this way, the design process becomes concerned with how to use the configuration features of that system to deliver the system requirements.

6.1 Object-oriented design using the UML

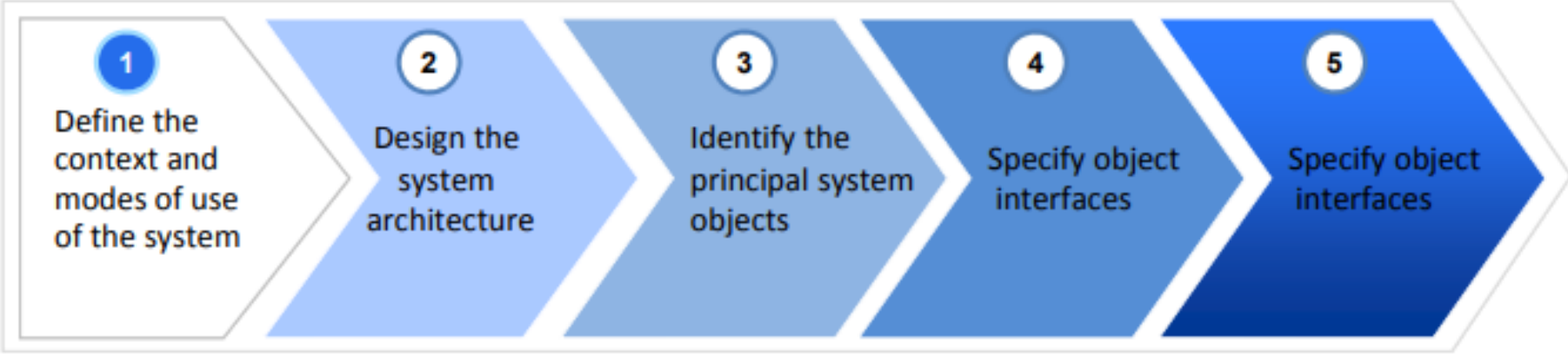
- Use case realization
- Mapping designs to code

6.1 Object-oriented design using the UML

- Structured object-oriented design processes involve developing a number of different **system models**.
- They **require a lot of effort for development and maintenance.**
- For small systems, this may not be cost-effective.
- However, for **large systems** developed by different groups design models require important communication mechanism.



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



i) Use case realization

- Use case realization is the **process of defining how a particular use case within a system is implemented.**
- A use-case realization represents **how a use case will be implemented** in terms of collaborating objects.
- Below figure demonstrates the system context for the weather station.
  - A **system context** model is a structural model that demonstrates 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
- This can be realized using use cases.

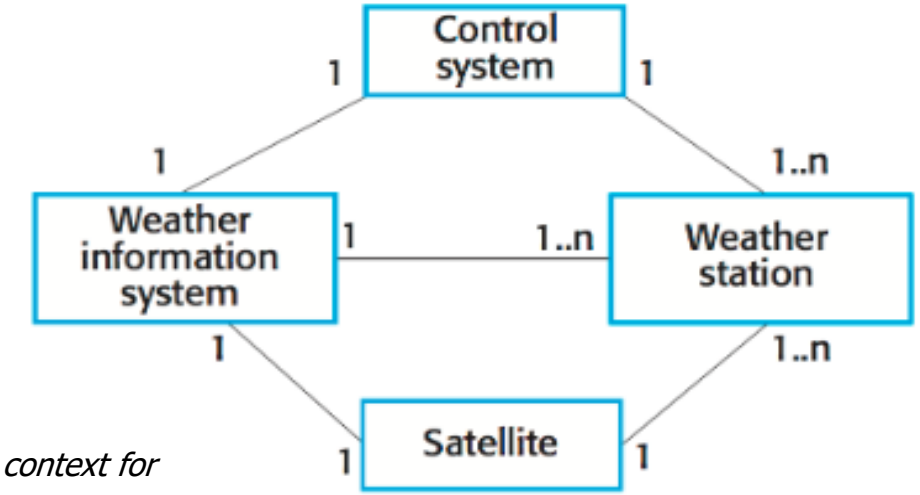


Figure: system context for Weather station

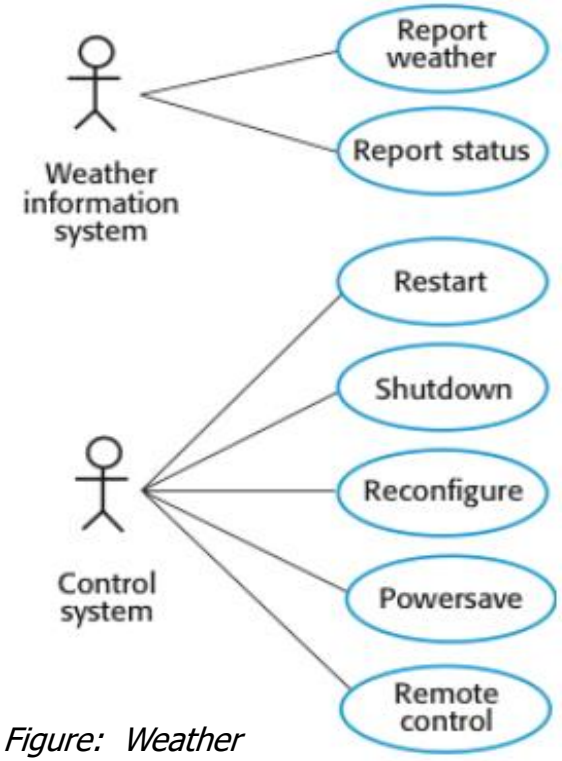


Figure: Weather station Use cases

System	Weather station
Use case	Report weather
Actors	Weather information system, Weather station

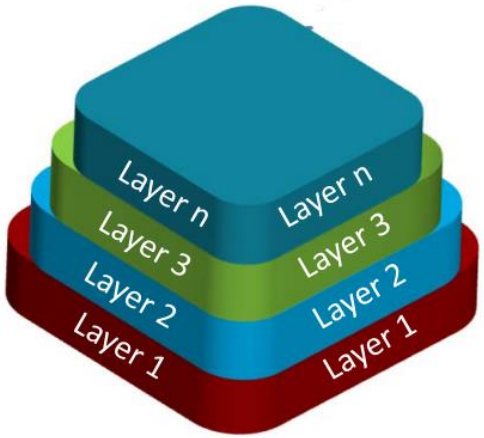
Description	• The weather station sends a summary of the weather data that has been collected from the instruments to the weather information system.
	• The data sent are the;
	  • maximum, minimum, and average ground and air temperatures;
	  • the maximum, minimum, and average air pressures;
	  • the maximum, minimum, and average wind speeds;
	  • the total rainfall; and the wind direction as sampled at five-minute intervals.

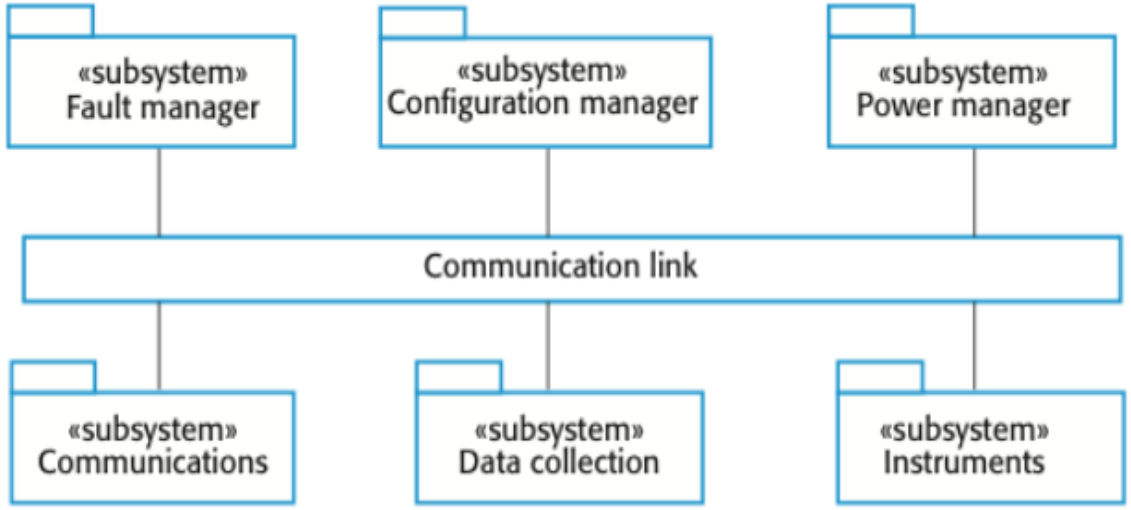
Stimulus	The weather information system establishes a satellite communication link with the weather station and requests transmission of the data.
Response	The summarized data is sent to the weather information system.
Comments	Weather stations are usually asked to report once per hour but this frequency may differ from one station to another and may be modified in the future.

Figure: Use case description- Report weather

ii) Architectural Design

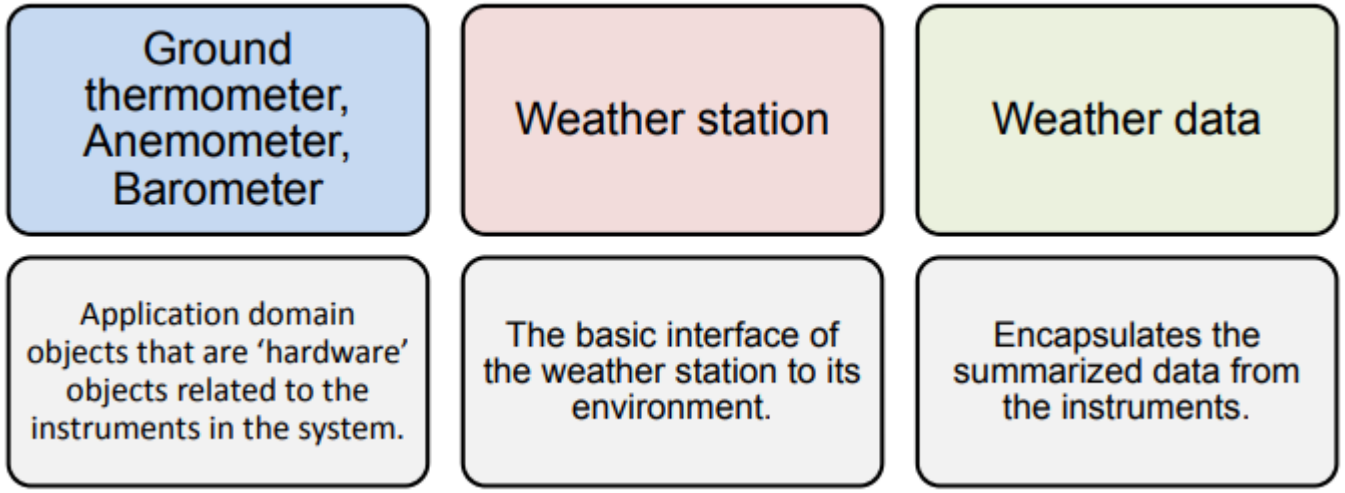
- Once interactions between the system and its environment have been understood, **we use this information for designing the system architecture.**
- We **identify the major components** that make up the system and **their interactions**, and
- then may organize the components using an architectural pattern (e.g. a layered or client-server model).



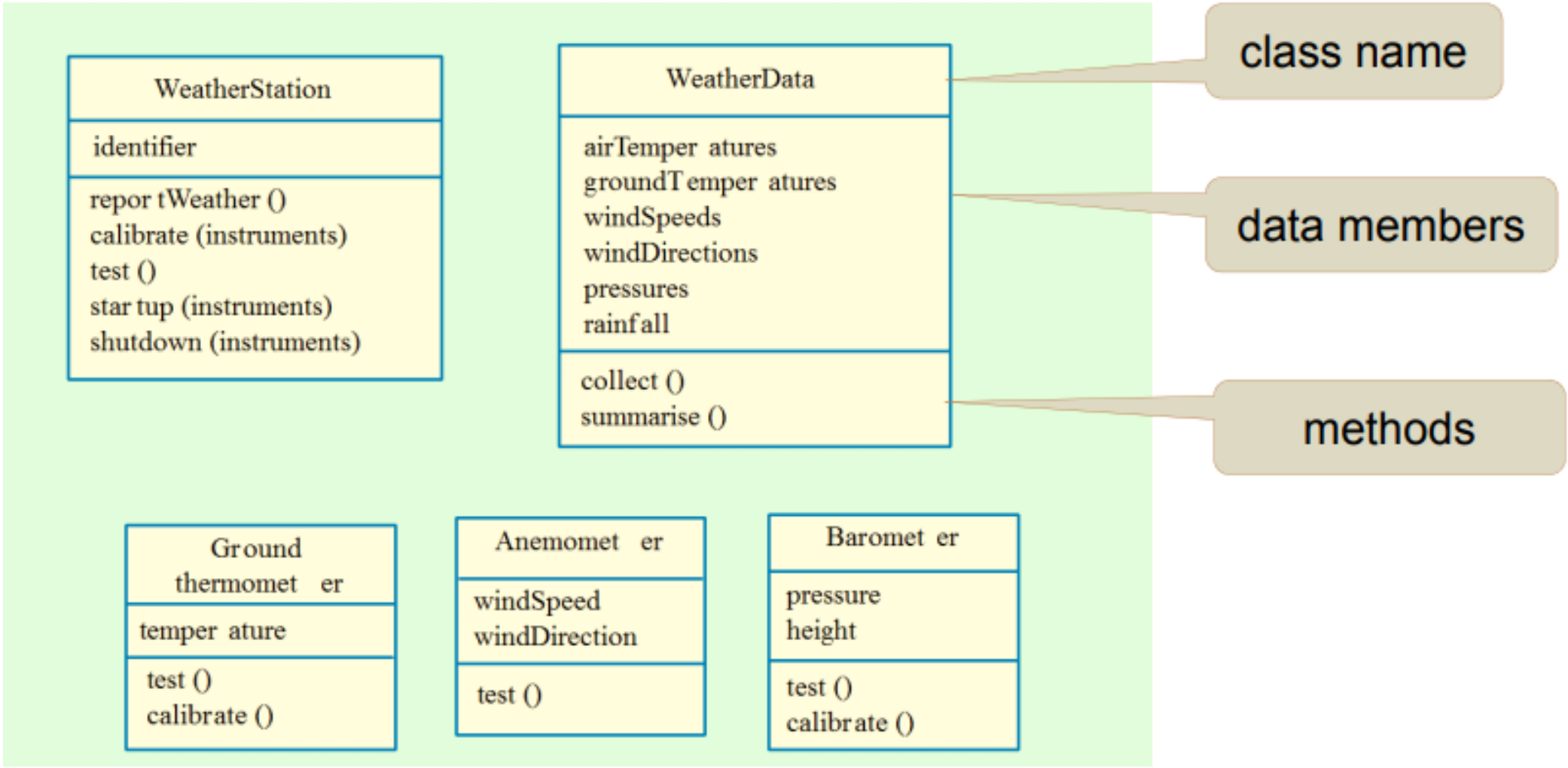


iii) Identifying object classes

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

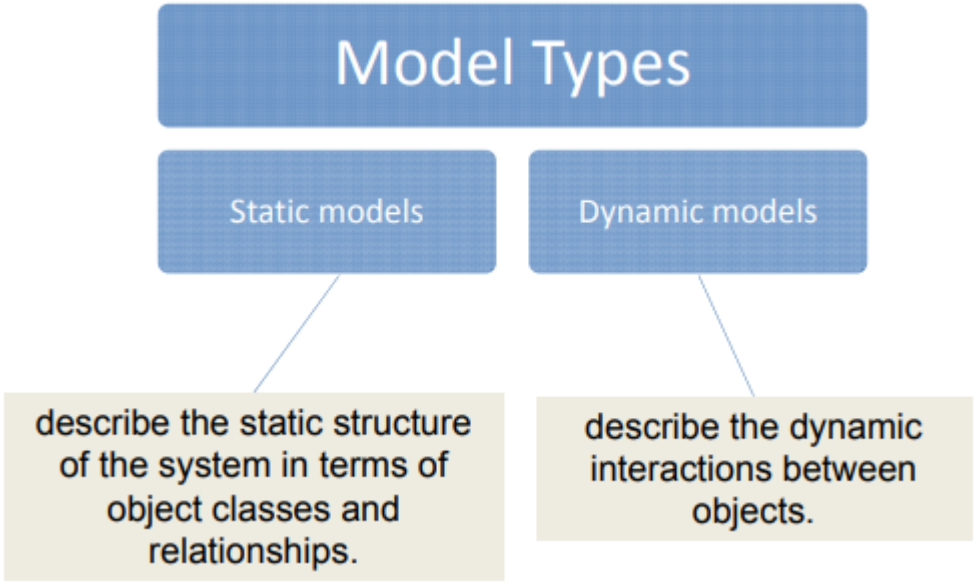


Weather station object classes



iv) Design Models

- Design models show the objects and object classes and relationships between these entities.
- For this, **we use class diagram or sequence diagram**.
- **Static models**
  - describe the static structure of the system in terms of object classes and relationships. Subsystem models are static (structural) models.
  - **Subsystem models** show logical groupings of objects into coherent subsystems.
  - These are **represented using a form of class diagram** with each subsystem shown as a package with enclosed objects.
- **Dynamic models**
  - describe the dynamic interactions between objects.
  - Sequence models are dynamic models.
  - **Sequence models** show the sequence of object interactions.
  - These are represented using a UML **sequence diagram** or a collaboration diagram.



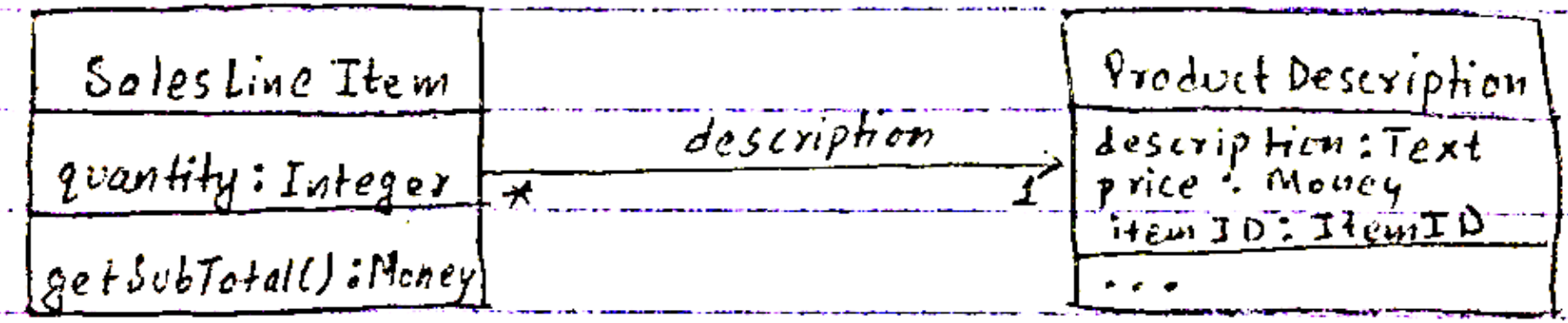
v) Specifying Interface

- Object **interfaces are specified** in order **to provide definition to the methods.**
- The UML uses class diagrams for interface specification.
- It **hides the actual implementation** of the object from the outside world.
- Objects may have several interfaces which are viewpoints on the methods provided.

6.2 Mapping Designs to Code

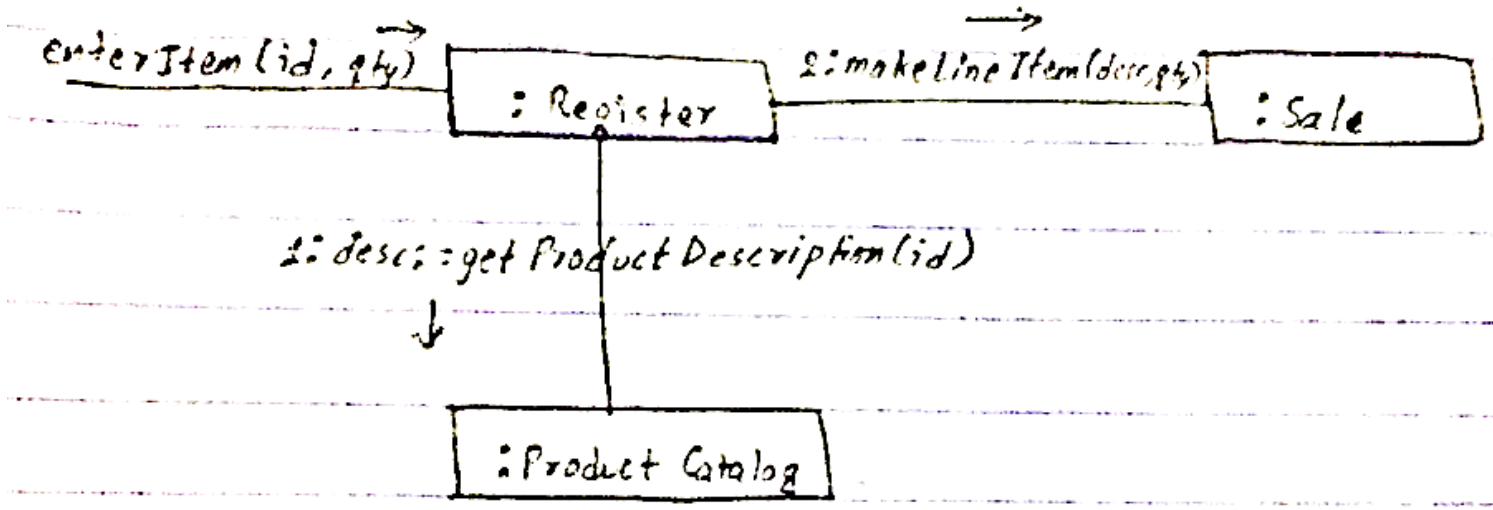
Mapping design to code

- It involves the implementations of object oriented design into the object oriented programming code.
- It requires writing code for:
  a) Class and interface definitions
     - Class definitions are created by mapping design class diagrams to code.
  b) Method definitions
     - Method definitions are created by mapping interaction diagrams to code.




```
public class SalesLineItem
{
    private int quantity;
    private ProductDescription description;
    public SalesLineItem (ProductDescription desc, int qty) {...}
    public Money getSubTotal() {...}
}
```

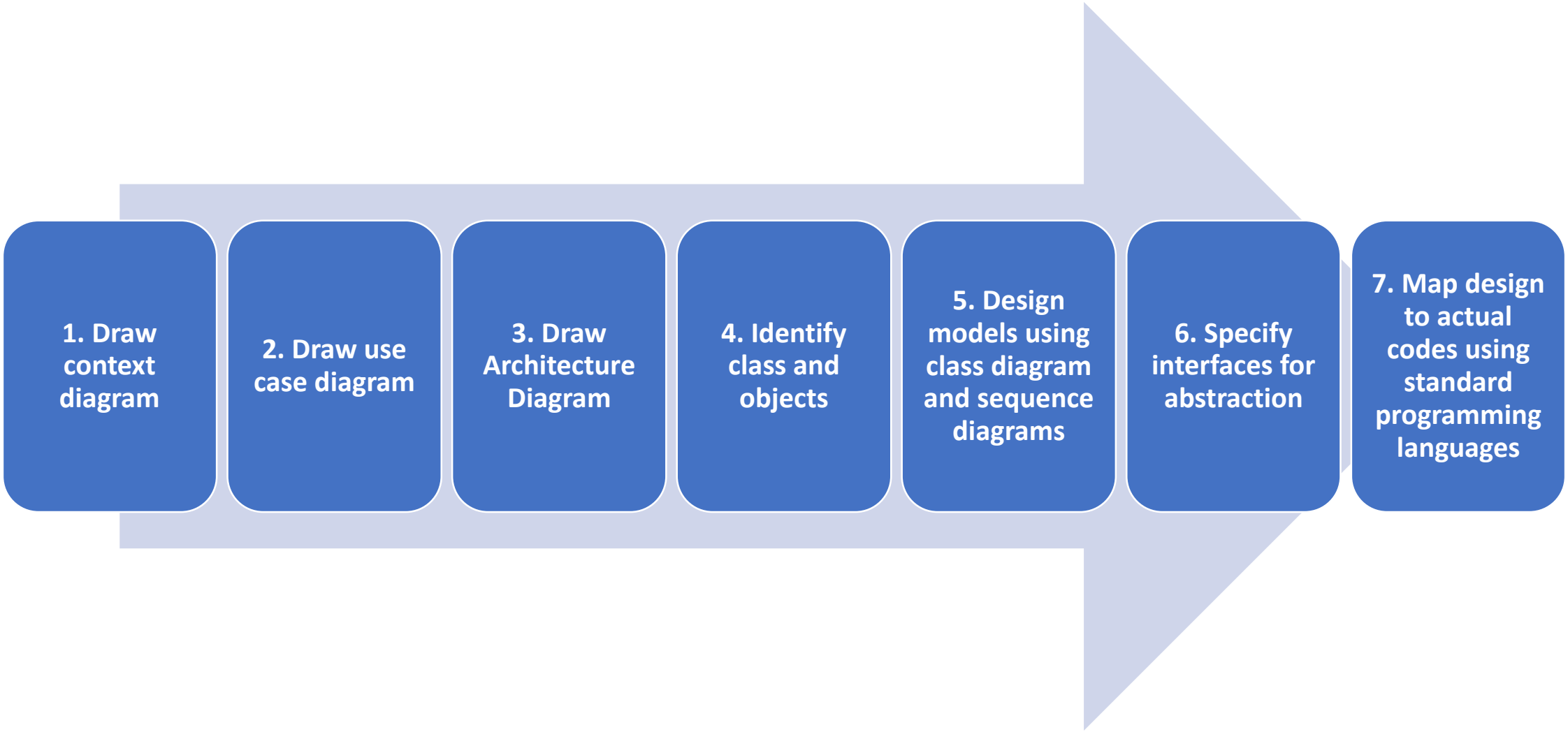
Figure: Creating Class Definition from Design Class Diagram




```
public void enterItem (ItemID id, int qty)
{
    ProductDescription desc = catalog.getProductDescription(id);
    currentSale.makeLineItem(desc, qty);
}
```

Figure: Creating methods from Collaborative diagram

Summary:


1. Draw context diagram
2. Draw use case diagram
3. Draw Architecture Diagram
4. Identify class and objects
5. Design models using class diagram and sequence diagrams
6. Specify interfaces for abstraction
7. Map design to actual codes using standard programming languages

6.3 Design Patterns

6.3 Design Patterns

- The pattern is a description of the problem and the essence of its solution, so that the solution may be reused in different settings.
- In software engineering, a software design pattern is a **general, reusable solution** of how **to solve a common problem** when designing an application or system.
- They **are like pre-made blueprints** that you can customize to solve a recurring design problem in your code.
- **The pattern is not a specific piece of code**, **but a general concept** for solving a particular problem.
- We can **follow the pattern details and implement a solution** that suits the realities of our own program.

Importance of Design pattern

Design patterns offer a best practice approach to support object-oriented software design, which is easier to design, implement, change, test and reuse. These design patterns provide best practices and structures.

1. Proven Solution

Design patterns provide a proven, reliable solution to a common problem, meaning the software developer does not have to “reinvent the wheel” when that problem occurs.

2. Reusable

Design patterns can be modified to solve many kinds of problems – they are not just tied to a single problem.

3. Expressive

Design patterns are an elegant solution.

4. Prevent the Need for Refactoring Code

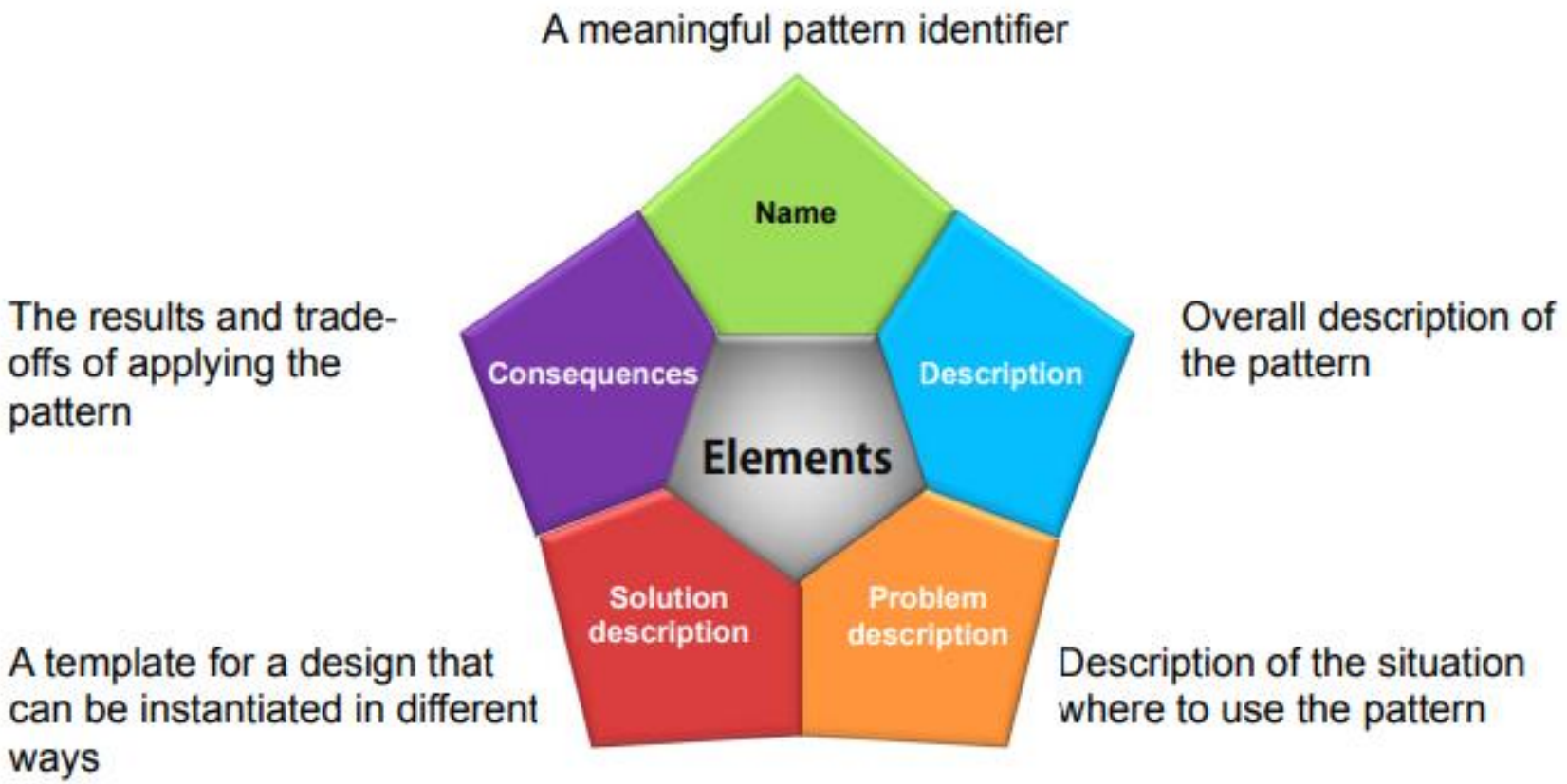
Since the design pattern is already the optimal solution for the problem, this can avoid refactoring.

5. Lower the Size of the Codebase

Each pattern helps software developers change how the system works without a full redesign. Further, as the “optimal” solution, the design pattern often requires less code.

Elements of Design Pattern:

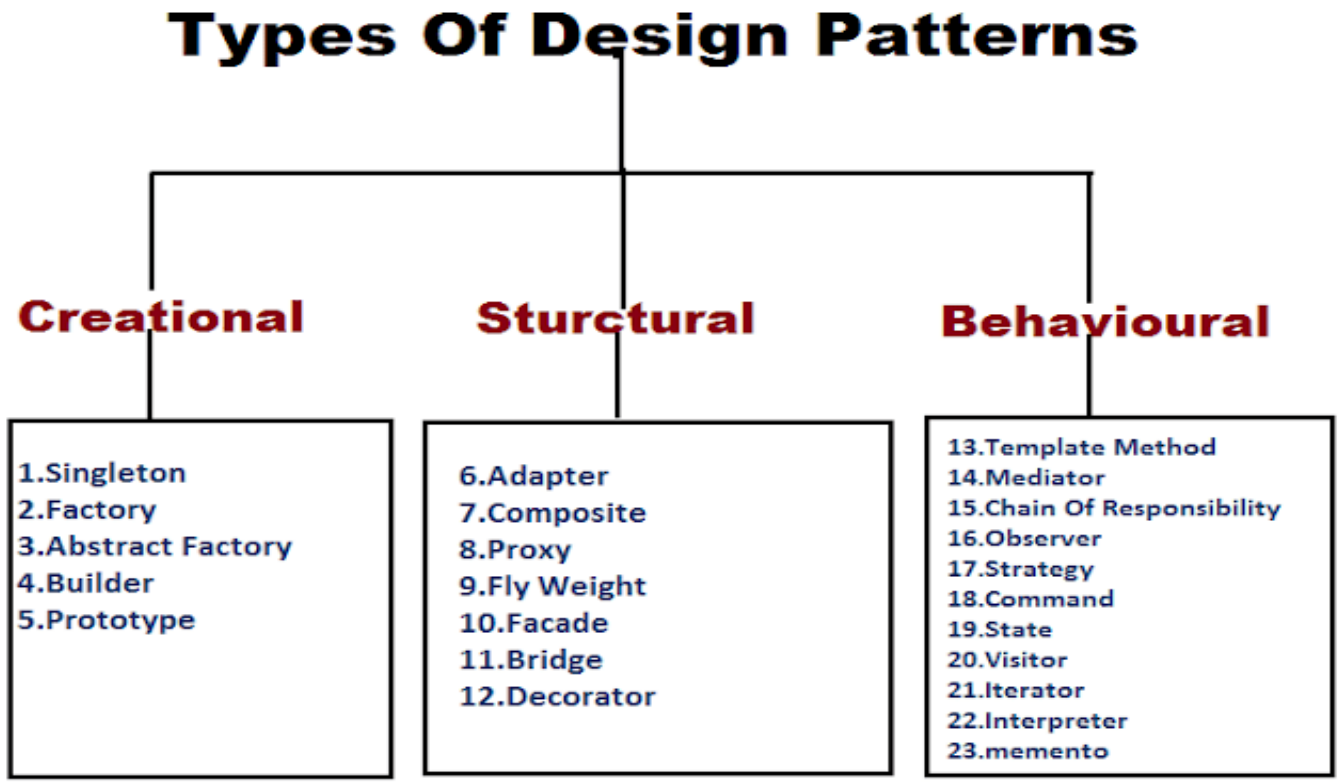
- The Gang of Four defined the four essential elements of design patterns in their book on patterns:



1. **Name**: should be a meaningful reference to the pattern.
2. **Description:** Overall description of the pattern.
3. **Problem Description:** A description of the problem area that explains when the pattern may be applied.
4. **Solution Description:** A solution description of the parts of the design solution, their relationships and their responsibilities. This is often expressed graphically and shows the relationships between the objects and object classes in the solution.
5. **Consequences:** A statement of the consequences—the results and trade-offs—of applying the pattern. This can help designers understand whether or not a pattern can be used in a particular situation

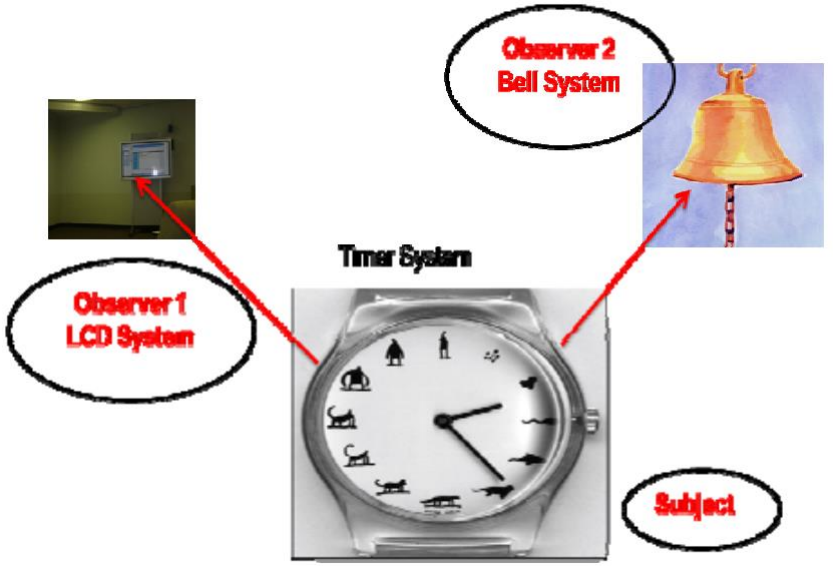
Types of Design Patterns

- Creational patterns
  - concern the process of object creation
- Structural patterns
  - deal with the composition of classes or objects
- Behavioral patterns
  - characterize the ways in which classes or objects interact and distribute responsibility.



The Observer pattern

- Observer is a behavioral design pattern.
- It specifies communication between objects: *observable* and *observers*.
- **An *observable* is an object which notifies *observers* about the changes in its state.**
- For example, a news agency can notify channels when it receives news.
- Receiving news is what changes the state of the news agency, and it causes the channels to be notified.
- This pattern is often used to implement distributed event handling systems.
- An Observer Pattern says that "just define a one-to-one dependency so that when one object changes state, all its dependents are notified and updated automatically".



The Observer pattern

Name

- Observer

Description

- Separates the display of object state from the object itself

Problem description

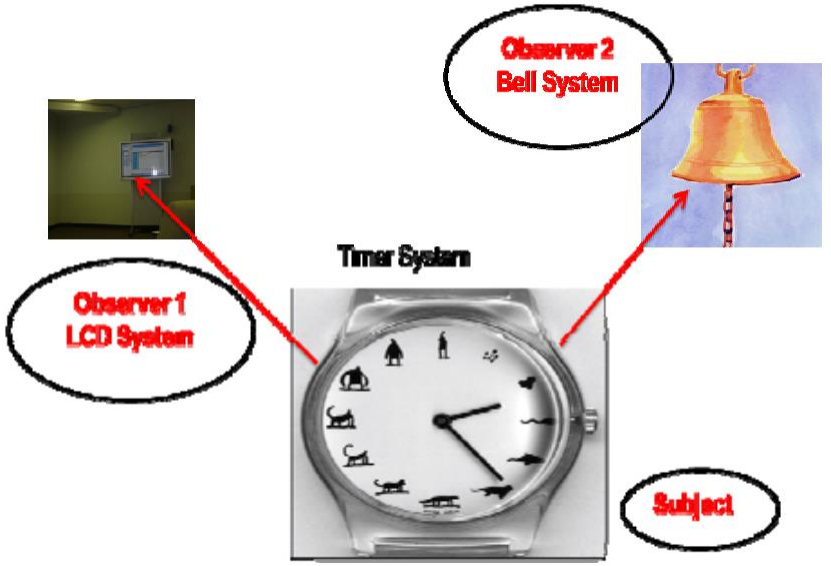
- Used when multiple displays of state are needed

Solution description

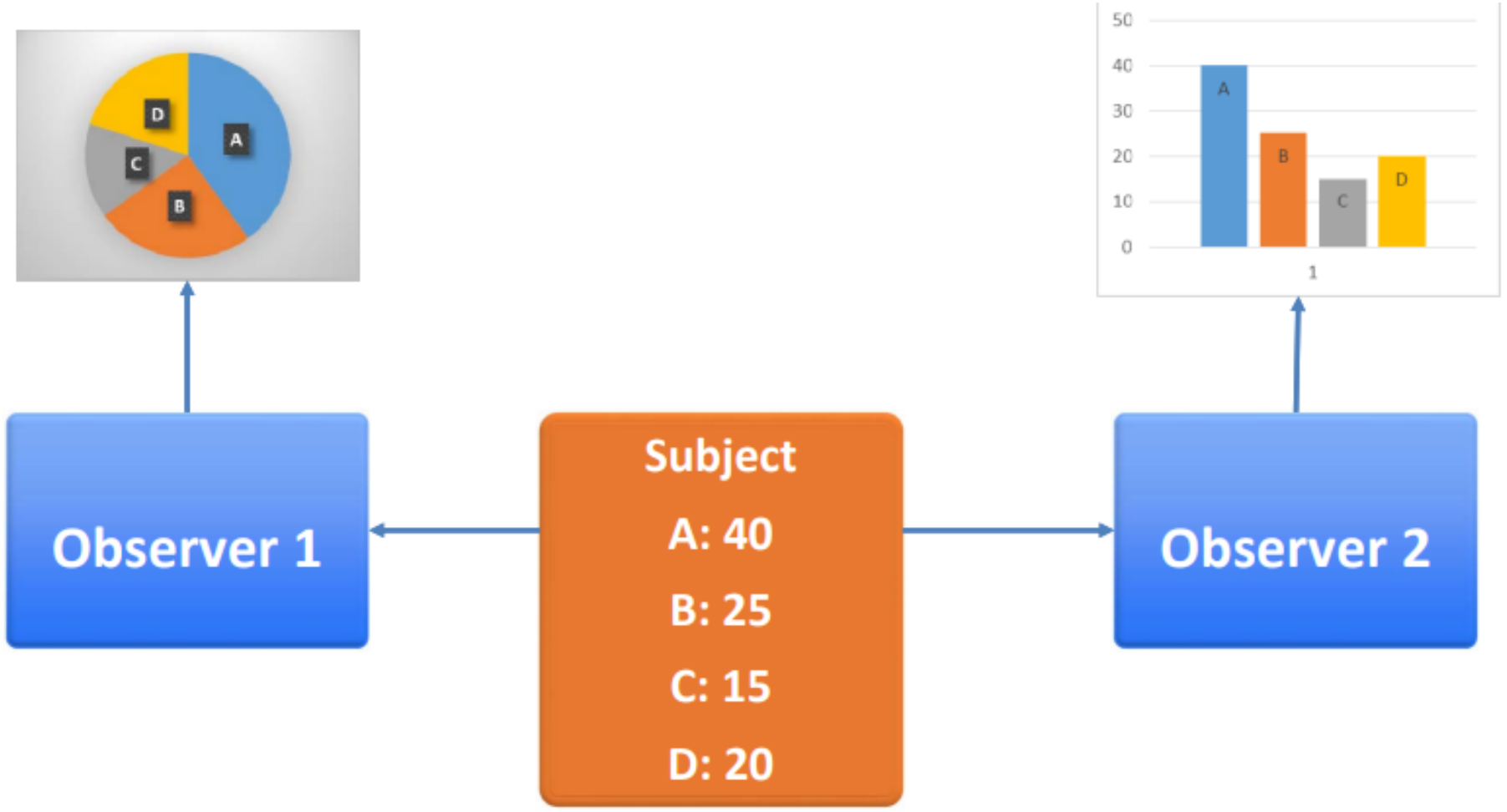
- See slide with UML description

Consequences

- Optimisations to enhance display performance are impractical



Multiple displays using the Observer pattern



6.4 Implementation Issues

6.4 Implementation Issues

- Focus here is not on programming, although this is obviously important, but on other implementation issues that are often not covered in programming texts:
  - Reuse Most modern software is constructed by reusing existing components or systems. When you are developing software, you should make as much use as possible of existing code.
  - Configuration management During the development process, you have to keep track of the many different versions of each software component in a configuration management system.
  - Host-target development Production software does not usually execute on the same computer as the software development environment. Rather, you develop it on one computer (the host system) and execute it on a separate computer (the target system).

6.5 Open Source Development

6.5 Open Source Development

- Open source development 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 a much larger population of volunteer developers. Many of them are also users of the code.
- Open source development refers to the collaborative process of creating and improving software by making the source code freely available to the public.
- This model encourages transparency, collaboration, and community involvement.

Open Source Systems

- Linux OS:
  - The best-known open source product is, of course, the Linux operating system
  - widely used as a server system and, increasingly, as a desktop environment.
- Other important open source products are:
  - Java,
  - Apache web server and
  - mySQL database management system.

Open Source Issues

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

Key aspects and principles of OSS

- **Open Source Software (OSS):**
  - In open source development, the source code of a software project is made accessible to anyone, allowing users to view, modify, and distribute the code.
- **Licensing:**
  - Open source projects are typically released under licenses that grant users the freedom to use, modify, and distribute the software. Common open source licenses include the GNU General Public License (GPL), Apache License, MIT License, and more.
- **Collaboration and Community:**
  - Developers from around the world contribute to projects, sharing their expertise and improving the software.
  - collaboration platforms like GitHub or GitLab.
- **Transparency:**
  - The entire development history, including code changes, discussions, and decisions, is typically accessible to the public.

References:

- https://cs.ccsu.edu/~stan/classes/CS410/Notes16/07-DesignAndImplementation.html

Key points

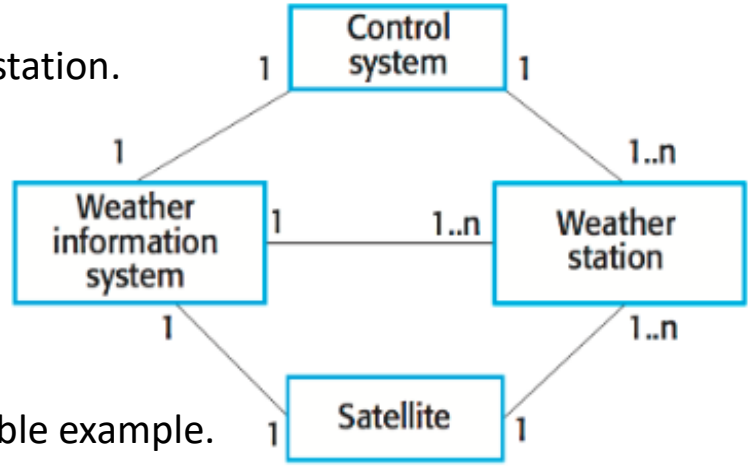
- Software design and implementation are interleaved activities. The level of detail in the design depends on the type of system being developed and whether you are using a plan-driven or agile approach.
- The process of object-oriented design includes activities to design the system architecture, identify objects in the system, describe the design using different object models, and document the component interfaces.
- A range of different models may be produced during an object-oriented design process. These include static models (class models, generalization models, association models) and dynamic models (sequence models, state machine models).
- Component interfaces must be defined precisely so that other objects can use them. A UML interface stereotype may be used to define interfaces.
- When developing software, you should always consider the possibility of reusing existing software, either as components, services, or complete systems.
- Configuration management is the process of managing changes to an evolving software system. It is essential when a team of people is cooperating to develop software.
- Most software development is host-target development. You use an IDE on a host machine to develop the software, which is transferred to a target machine for execution.
- Open-source development involves making the source code of a system publicly available. This means that many people can propose changes and improvements to the software.

Brief Answer Questions:

1. Define software implementation.
2. What do you mean by COTS? When can we use it?
3. What do you mean by use case realization?
4. What are design patterns? Why is it important?
5. Mention the elements of Design pattern.
6. What are the types of design pattern? List them.
7. What is a singleton design pattern? When is it used?
8. What is observer design pattern?
9. What is open source development?
10. Give two examples of open source softwares.
11. What are the disadvantages of open source development?

Short Answer Questions:

1. Realize a use case for below system context for weather station.



2. How design can be mapped to code? Explain with a suitable example.
3. Define Design pattern. Mention the importance of design pattern.
4. Explain the elements of Design patterns.
5. What are the different categories of design pattern? Explain them.
6. Explain the singleton design pattern.
7. Explain the observer design pattern.
8. Explain the different implementation issues.
9. Why do we need design patterns for software development? Explain one example of the design patterns
10. Define design pattern. Differentiate programming paradigm and design pattern.

End of Chapter