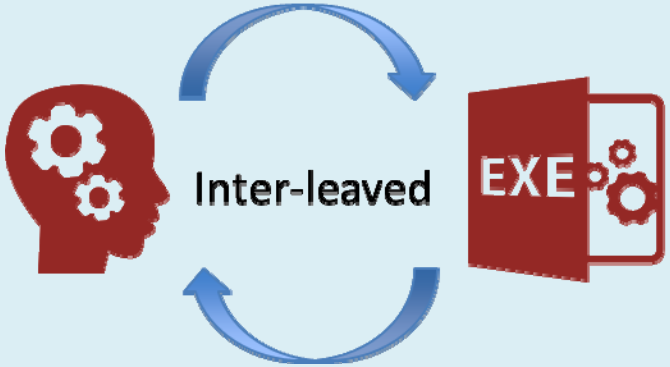
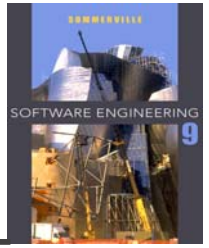
Chapter 7
Design and Implementation

(Lecture 1)

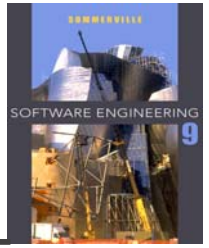


Topics covered

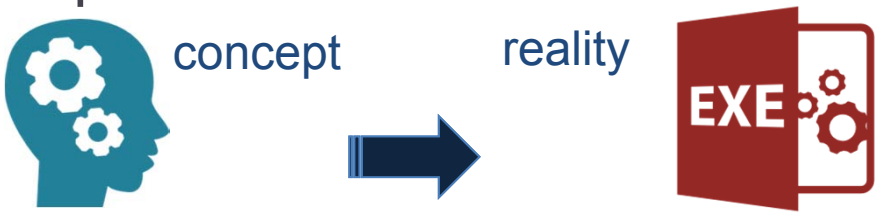


1. Object-oriented design using the UML
2. Design Patterns
3. Implementation Issues
4. Open source development

Design and implementation

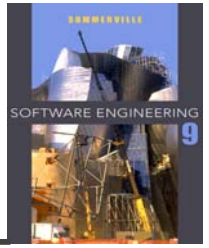


- Software design and implementation is the stage in the software engineering at which an executable software system is developed



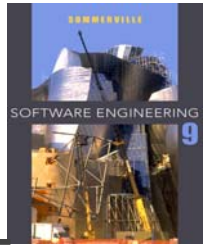
- Design and implementation activities are inter-leaved

Design and implementation

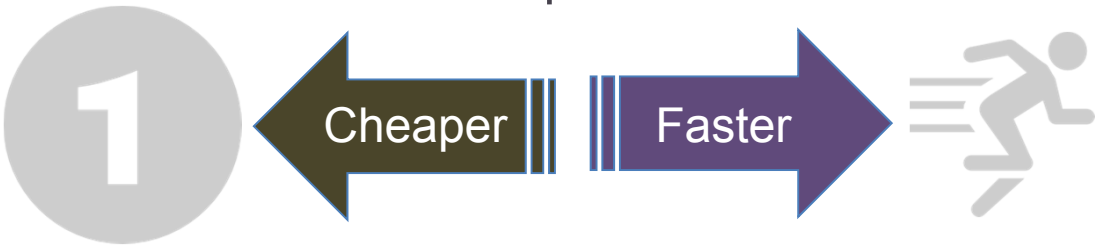


- Software design and implementation is the stage in the software engineering process at which an executable software system is developed.
- Software design and implementation activities are invariably inter-leaved.
  - Software design is a creative activity in which you identify software components and their relationships, based on a customer's requirements.
  - Implementation is the process of realizing the design as a program.

Build or buy

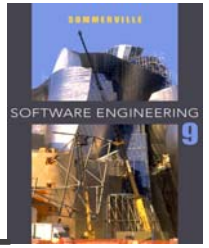


- Buying commercial off-the-shelf systems (COTS) is a norm in today's environment.
  - For example, existing health information systems can be purchased to use in hospitals.



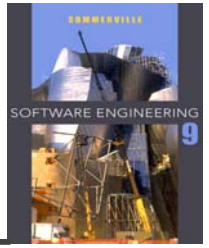
- The main concern with COTS is how to use the configuration features of that system to deliver the system requirements.

Build or buy

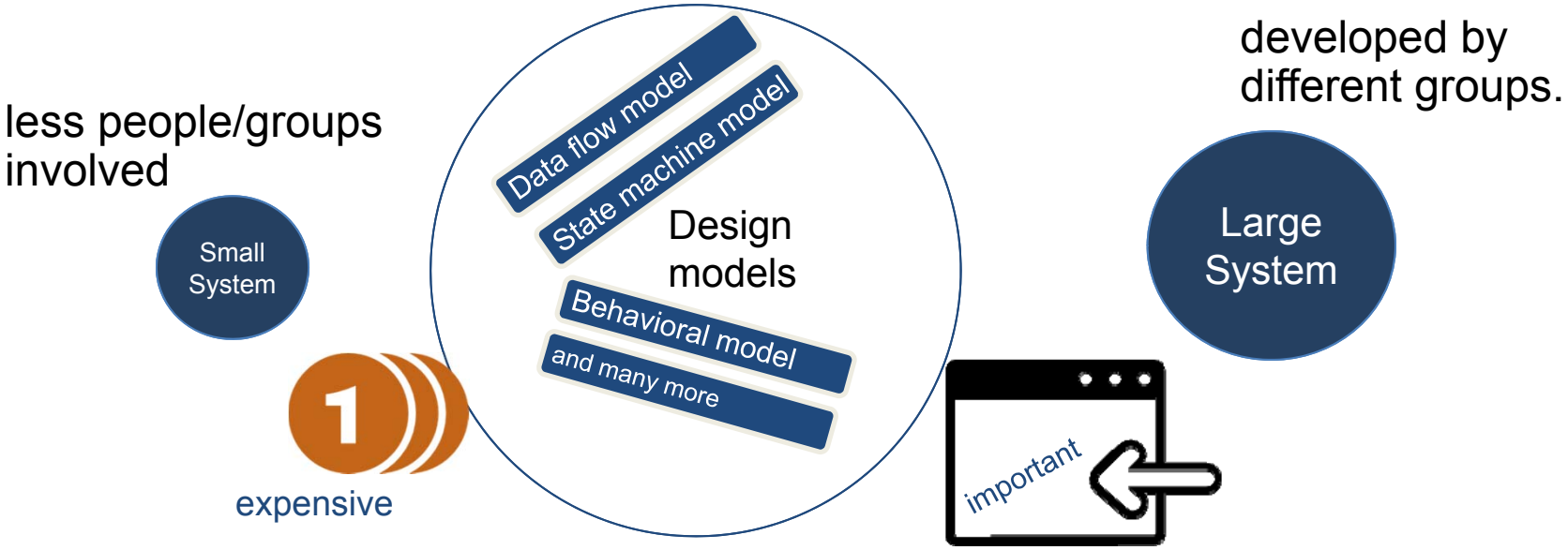


- In a wide range of domains, it is now possible to buy off-the-shelf systems (COTS) that can be adapted and tailored to the users' requirements.
  - For example, if you want to implement a medical records system, you can buy a package that is already used in hospitals. It can be cheaper and faster to 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.

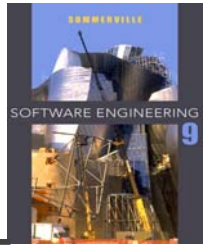
An object-oriented design process



- 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.

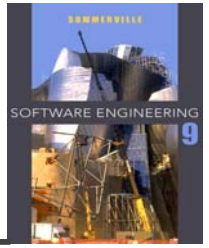


An object-oriented design process

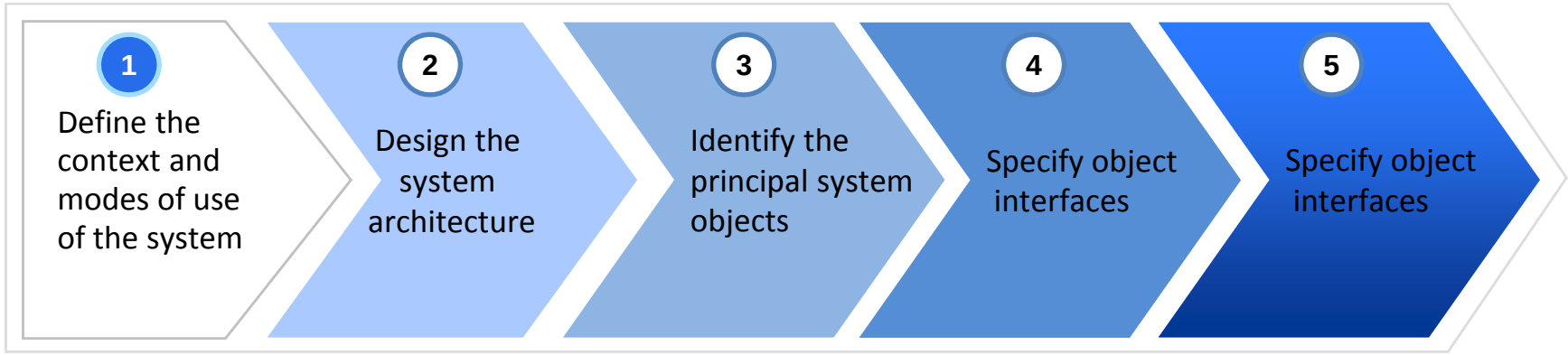


- Structured object-oriented design processes involve developing a number of different system models.
- They require a lot of effort for development and maintenance of these models and, for small systems, this may not be cost-effective.
- However, for large systems developed by different groups design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

Process stages



- There are a variety of different object-oriented design processes that depend on the organization choice.
- Common activities in these processes include:

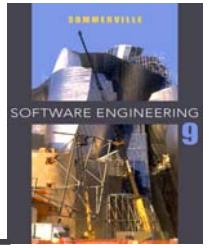


- Process illustrated here using a design for a wilderness weather station.

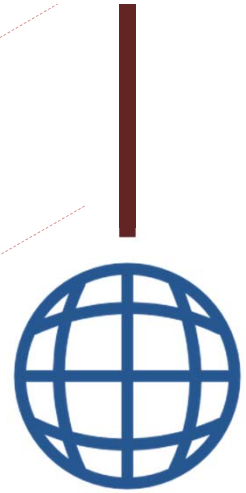
Video: Software Development Process Implementation

Ref: http://www.youtube.com/watch?v=I4ikgjFsdWI

System context and interactions

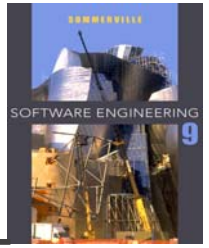


- Understanding the relationships between the software and its external environment is essential for deciding how to;
  - provide the required system functionality
  - structure the system to communicate with its environment.
- Understanding of the context also lets you establish the boundaries of the system.



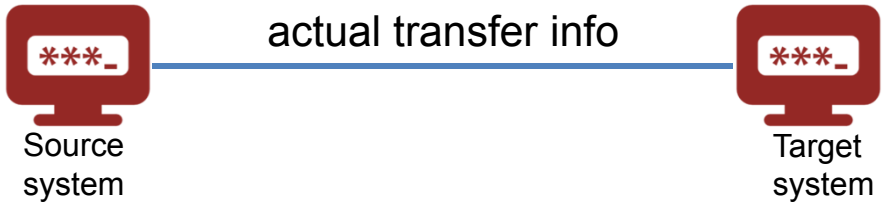
- Context talks about the features
- Interactions defines the communication between a system and its environment.

Context and interaction models

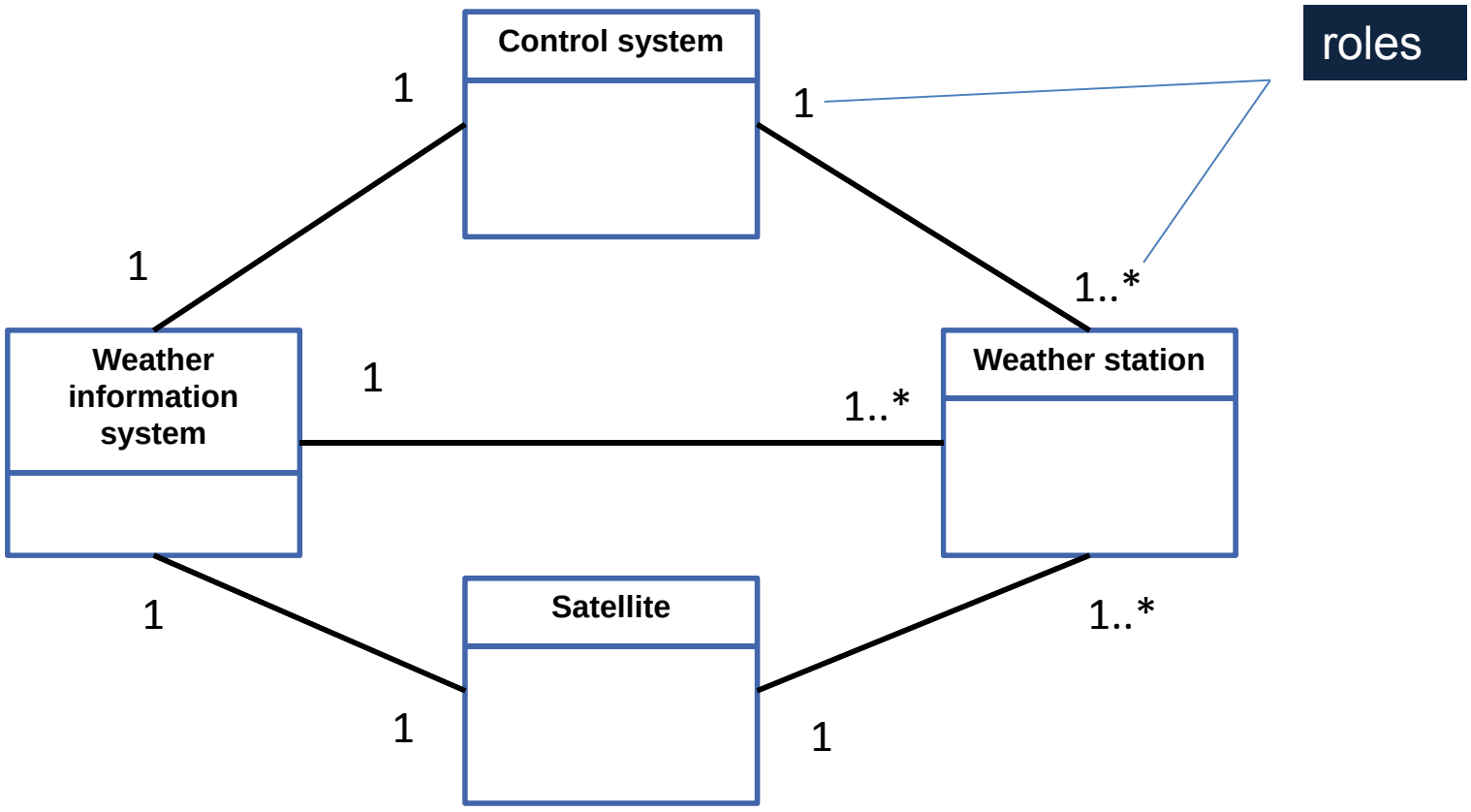


- A system context model is a structural model that demonstrates the other systems in the environment of the system being developed.

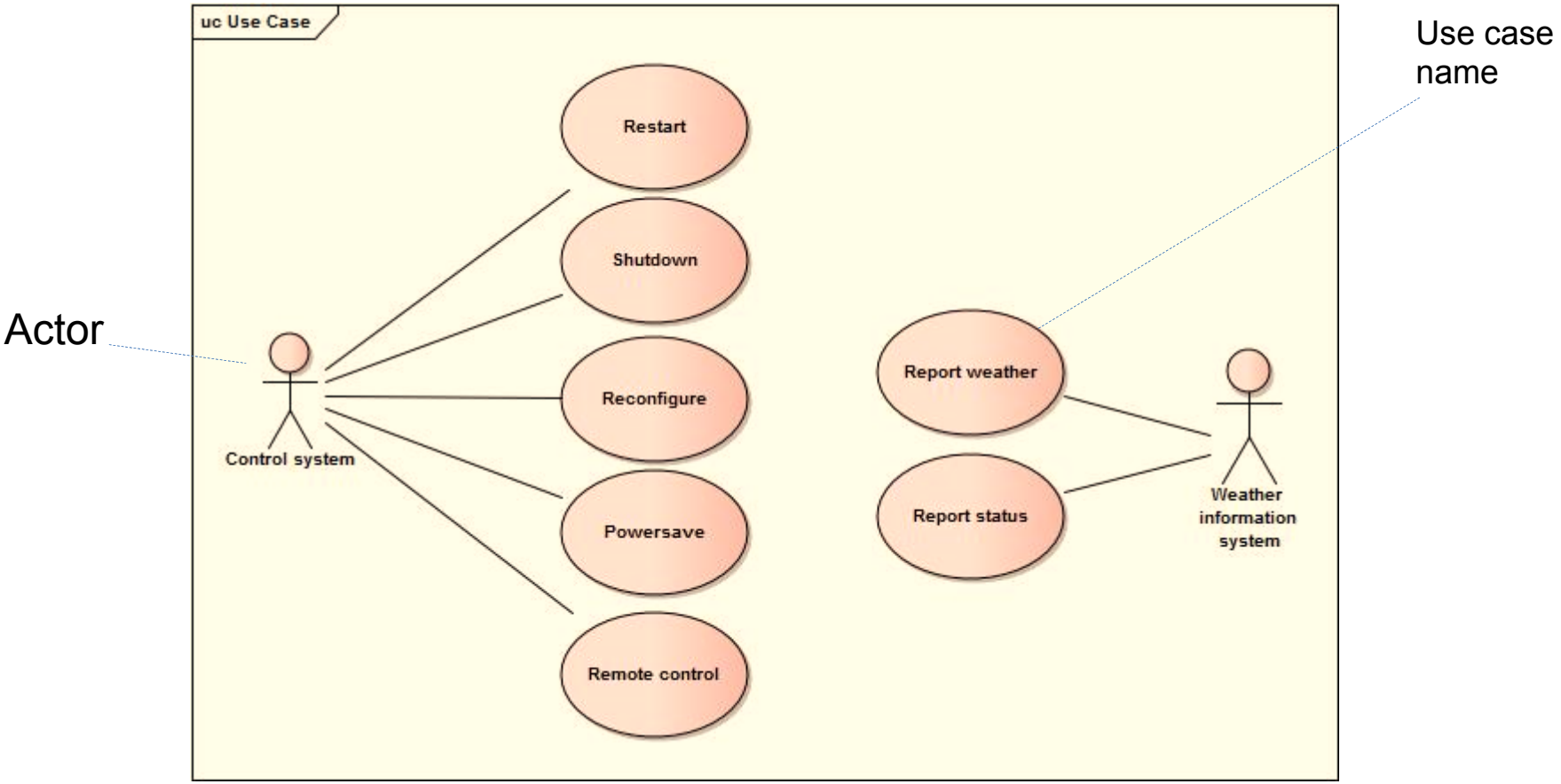
- An interaction model is a dynamic model that shows how the system interacts with its environment as it is used.



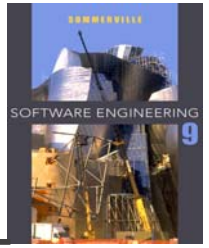
System context for the weather station



Weather station use cases



Use case description—Report weather

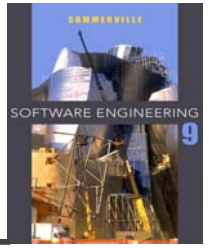


System	Weather station
Use case	Report weather
Actors	Weather information system, Weather station

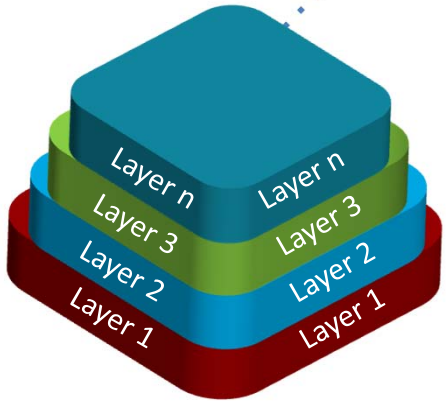
Description	• The weather station sends a summary of the weather data that has been collected from the instruments to the weather information system.
	• The data sent are the;
	  • maximum, minimum, and average ground and air temperatures;
	  • the maximum, minimum, and average air pressures;
	  • the 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.

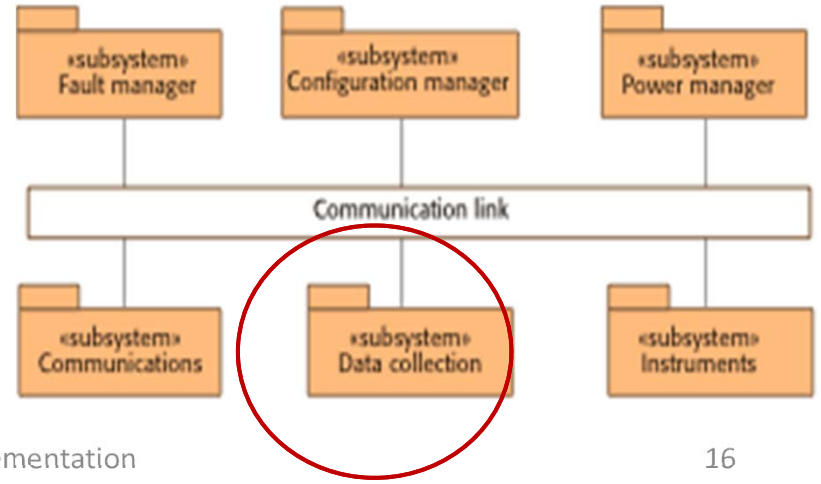
Architectural design



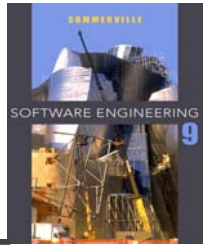
- Interactions between the system and its environment leads to design the system architecture.
- System architecture consists of components
- The components are organized in an architectural pattern such as a client-server or layer model.



Weather station Architecture (High Level)

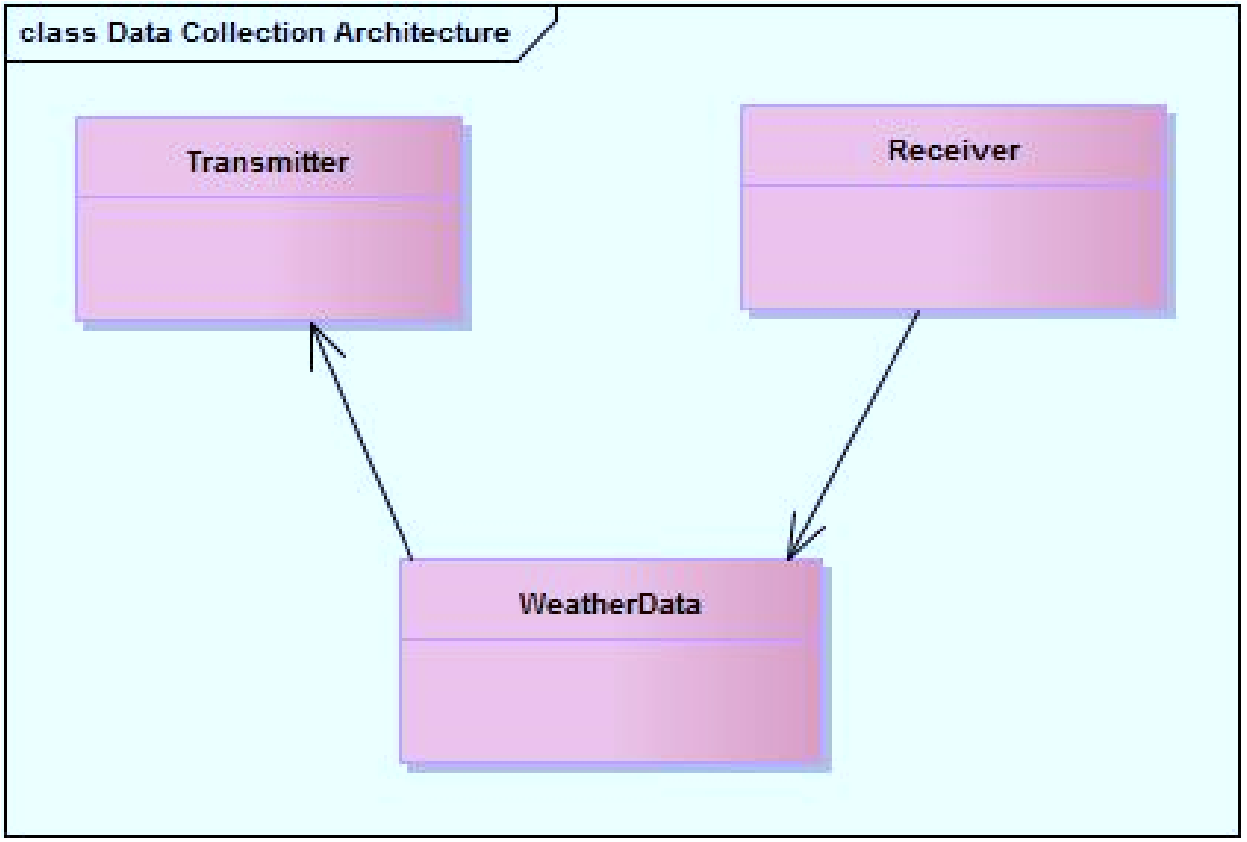


Architectural design

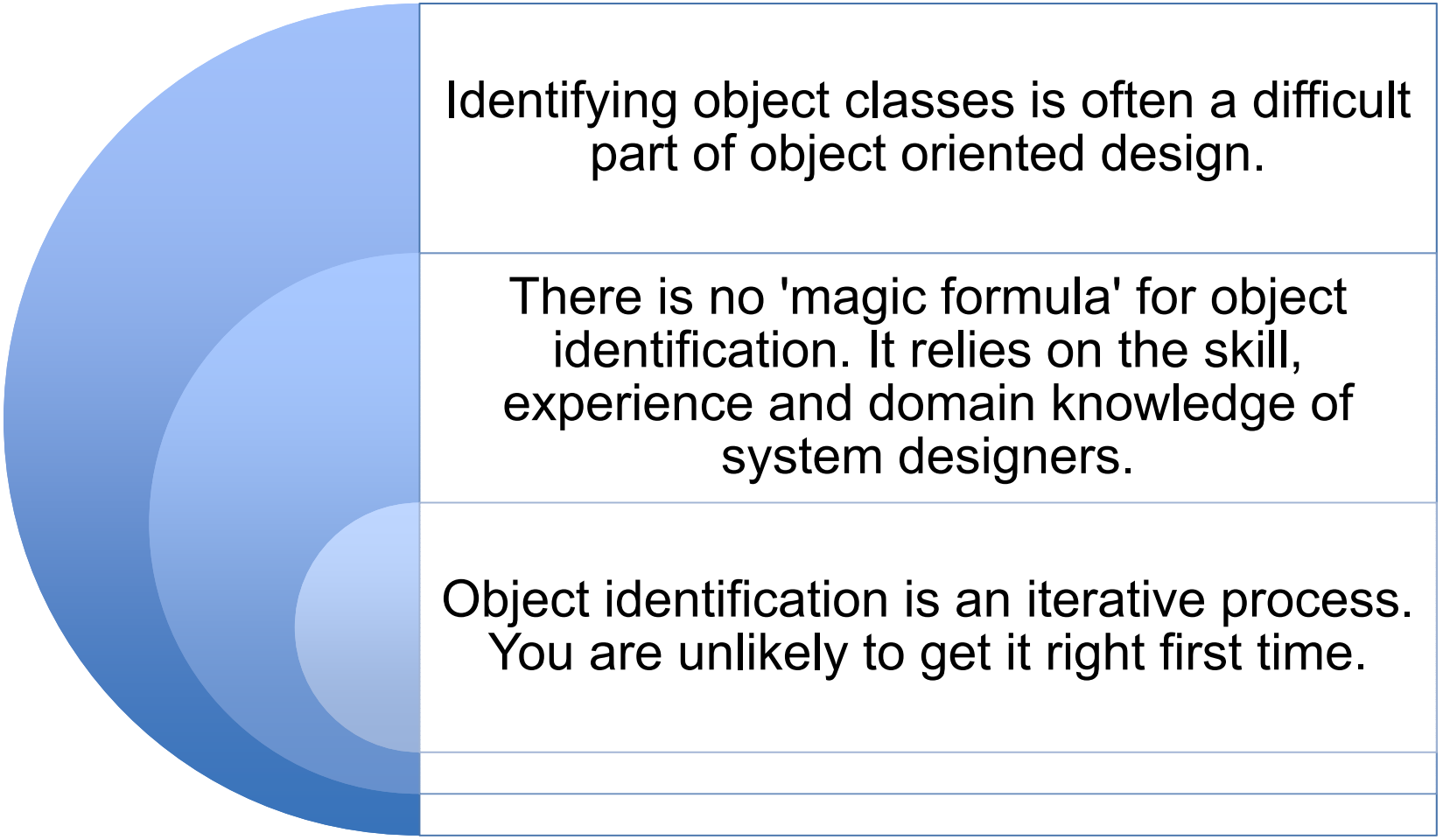


- Once interactions between the system and its environment have been understood, you use this information for designing the system architecture.
- You identify the major components that make up the system and their interactions, and then may organize the components using an architectural pattern such as a layered or client-server model.
- The weather station is composed of independent subsystems that communicate by broadcasting messages on a common infrastructure.

Architecture of data collection system



Object class identification



Approaches to identification

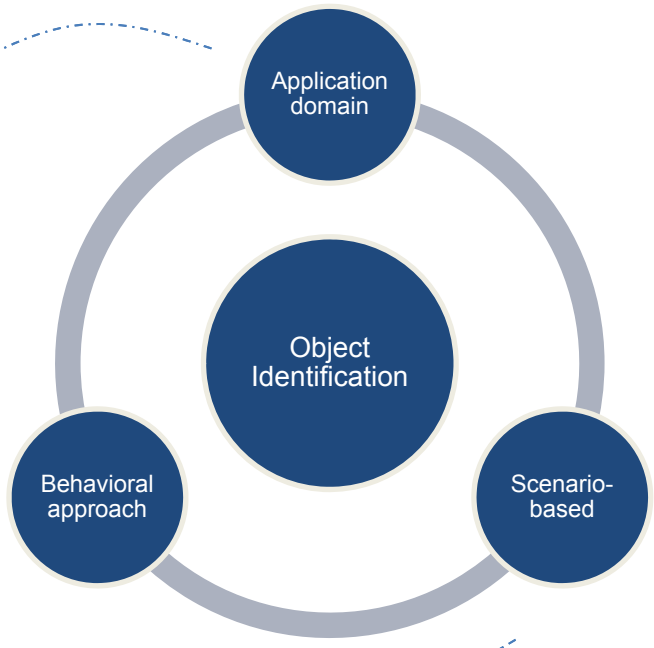
- Use a grammatical approach based on a natural language description of the system used in HOOD method.

Hierarchic Object-Oriented Design

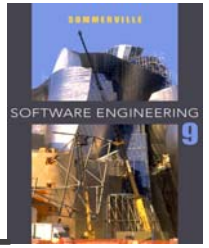
- Base the identification on tangible things in the application domain

- Identify objects based on what participates in what behaviour

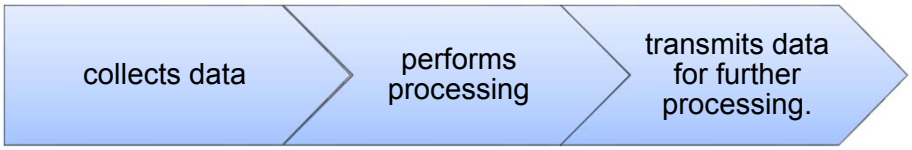
- Identify objects, attributes and methods in each scenario



Weather station description



- A weather station is a package of software controlled instruments which;

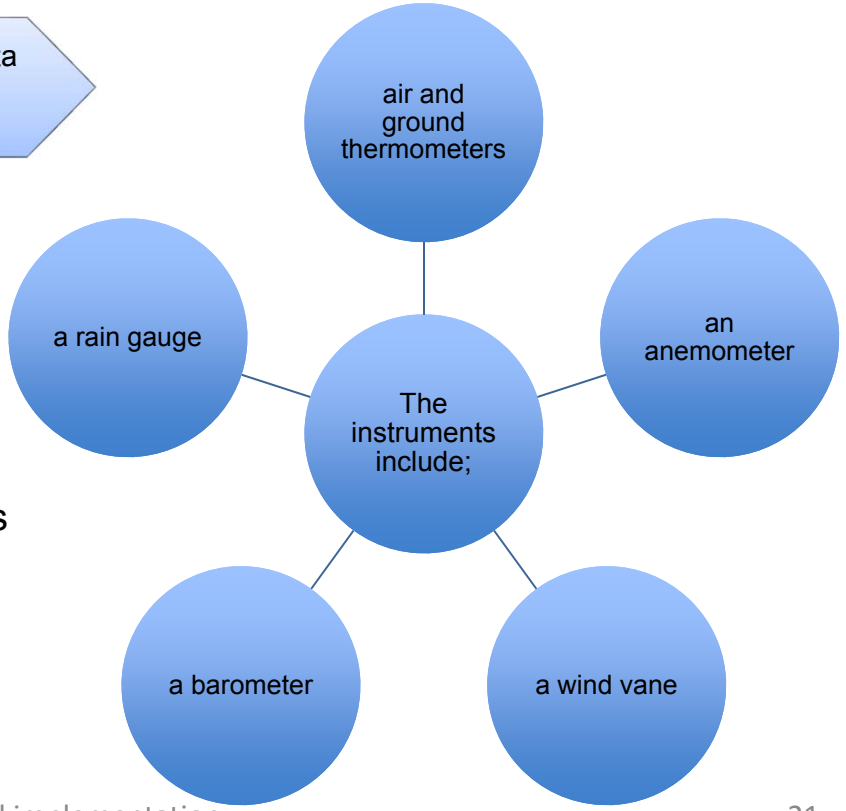


Data is collected periodically.

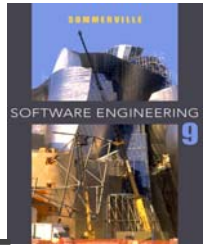
Upon issuing command the weather data is transmitted,

the weather station processes and summarises the collected data.

The summarised data is transmitted to the mapping computer when a request is received.



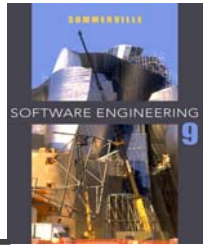
Weather station description



- 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.

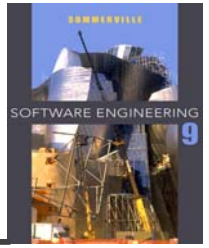
Weather station object classes

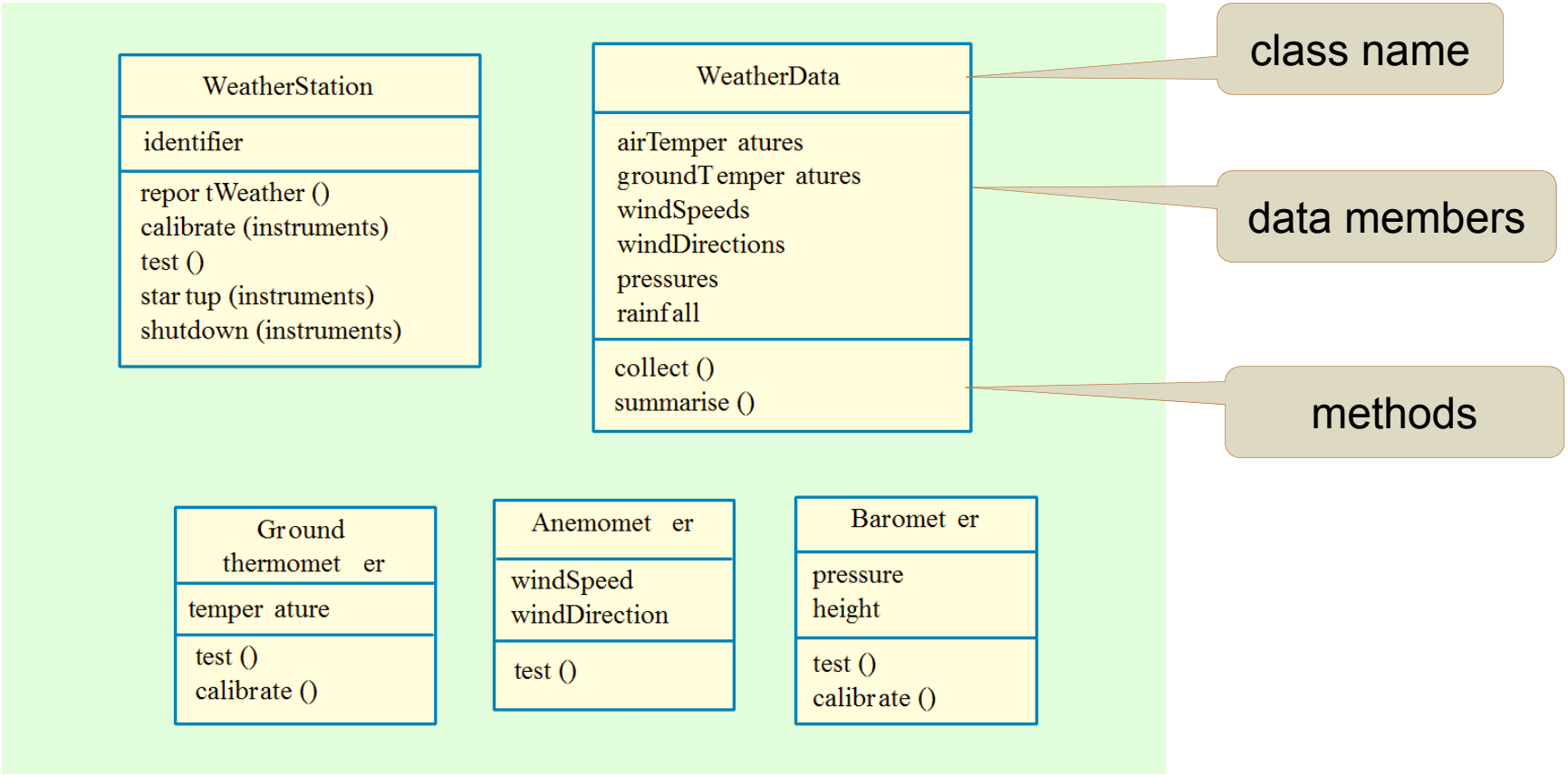


Object class identification in the weather station system may be based on the tangible hardware and data in the system:

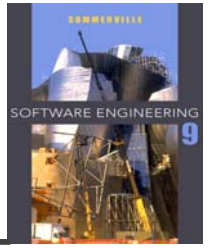
Ground thermometer, Anemometer, Barometer	Weather station	Weather data
Application domain objects that are 'hardware' objects related to the instruments in the system.	The basic interface of the weather station to its environment.	Encapsulates the summarized data from the instruments.

Weather station object classes

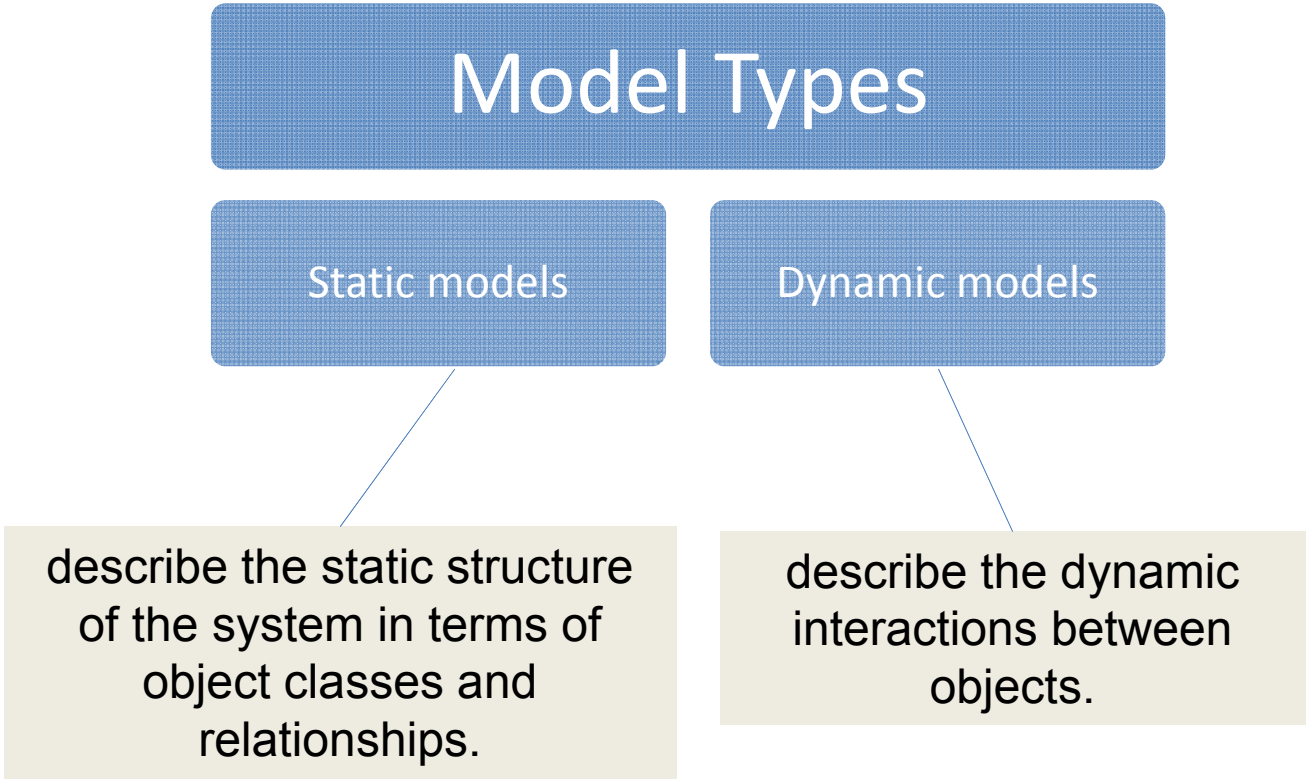




Design models



- Design models show the objects and object classes and relationships between these entities.



Examples of design models

Subsystem models that show logical groupings of objects into coherent subsystems.

Sequence models that show the sequence of object interactions.

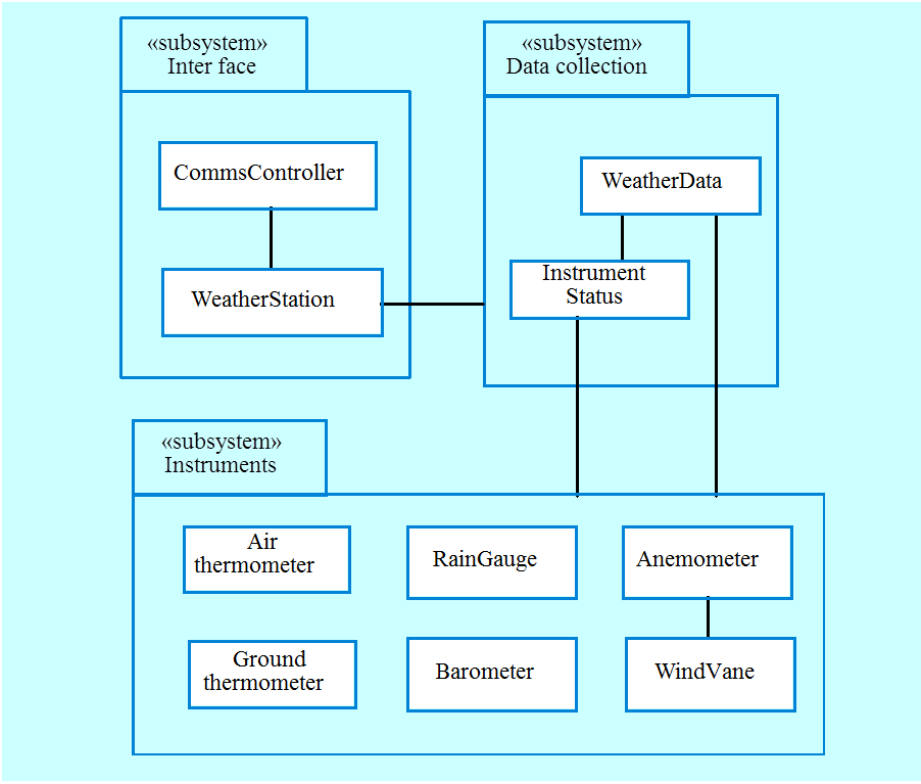
State machine models that show how individual objects change their state in response to events.

Other models include use-case models, aggregation models, generalisation models, etc.

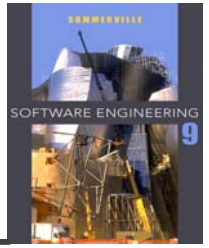
Subsystem models

- Shows how the design is organised into logically related groups of objects.
- In the UML, these are shown using packages – an encapsulation construct
- This is a logical model.
- The actual organisation of objects in the system may be different.

Weather station subsystems

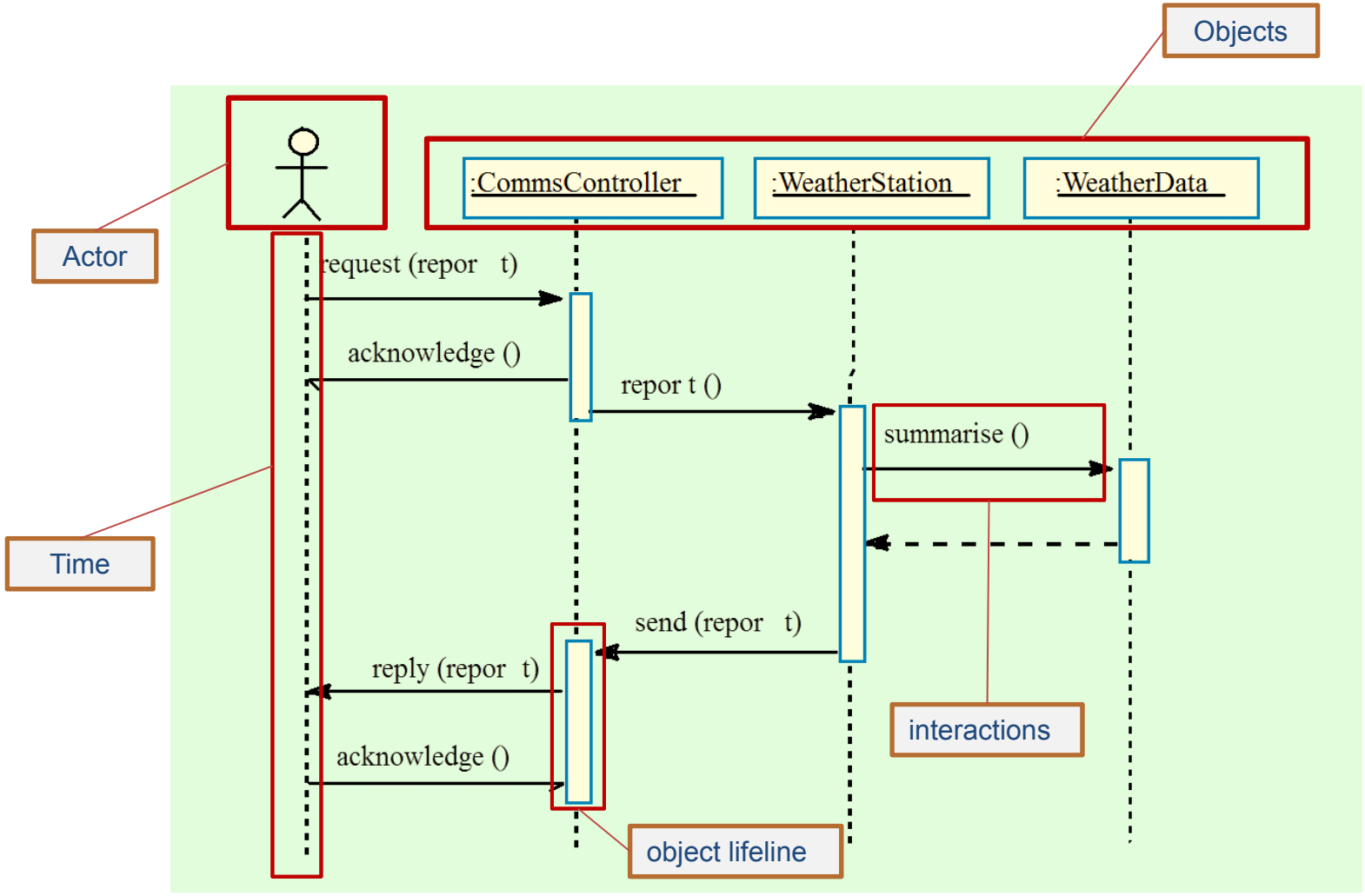


Sequence models

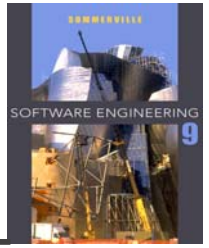


- Sequence models show the sequence of object interactions that take place
  - Objects are arranged horizontally across the top;
  - Time is represented vertically so models are read top to bottom;
  - Interactions are represented by labelled arrows, Different styles of arrow represent different types of interaction;
  - A thin rectangle in an object lifeline represents the time when the object is the controlling object in the system.

Sequence diagram describing data collection



State diagrams



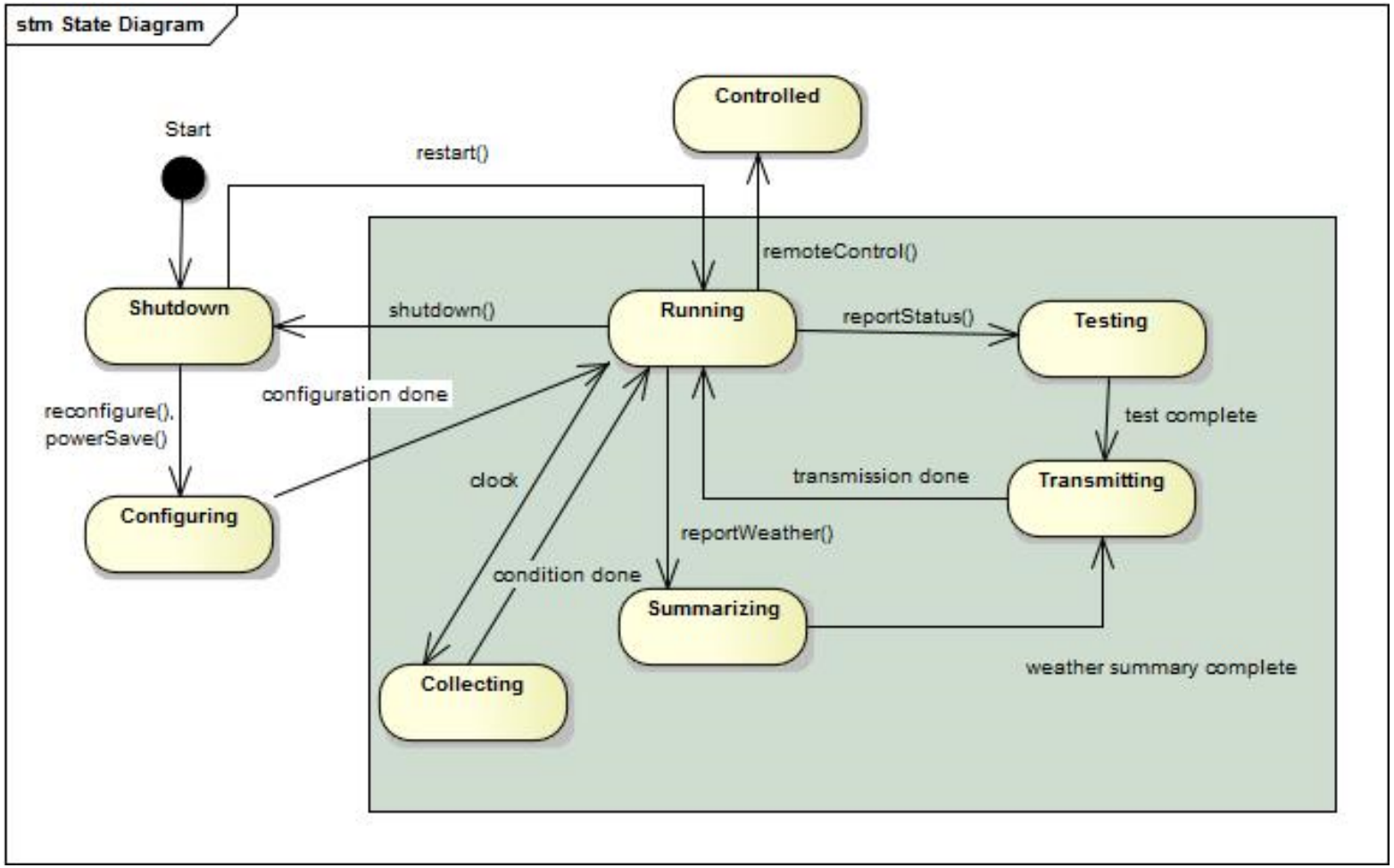
- State diagrams show how objects respond to different service requests

- State diagrams are useful to note object's run-time behavior

  - You don't usually need a state diagram for all of the objects in the system.

  - Many of the objects in a system are relatively simple and a state model adds unnecessary detail to the design.

Weather station state diagram



Interface specification

Object interfaces are specified in order to provide definition to the methods.

It hides the actual implementation of the object from the outside world.

Interface specification

The UML uses class diagrams for interface specification

Objects may have several interfaces which are viewpoints on the methods provided.

Interface

Turn On
Turn off
Volume up
Volume Down
Switch to next Channel
Switch to previous Channel

Current channel
Volume Level

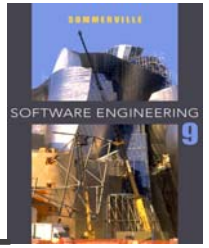
Weather station interfaces


```
interface WeatherStation {

    public void WeatherStation () ;

    public void startup () ;
    public void startup (Instrument i) ;

    public void shutdown () ;
    public void shutdown (Instrument i) ;

    public void reportWeather ( ) ;

    public void test () ;
    public void test ( Instrument i ) ;

    public void calibrate ( Instrument i) ;

    public int getID () ;

} //WeatherStation
```

Interfaces

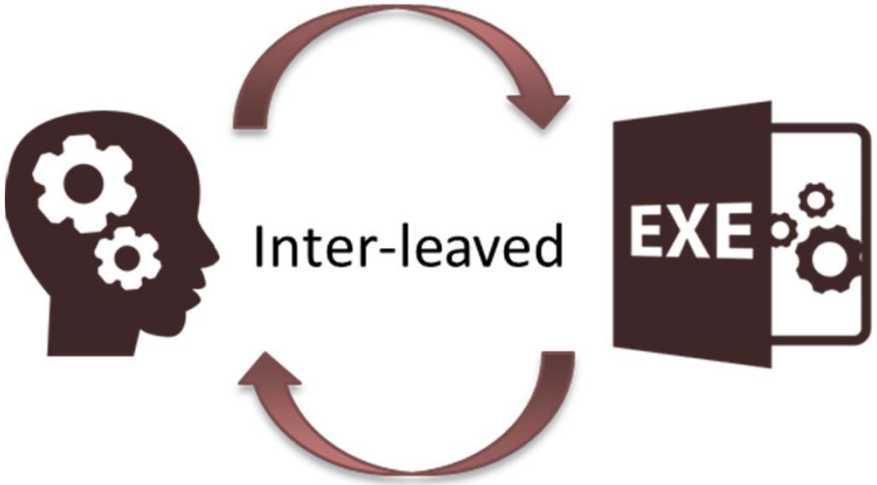
Key points

- Software design and implementation are inter-leaved activities. The level of detail in the design depends on the type of system.
- The process of object-oriented design includes activities to;
  - design the system architecture
  - identify objects in the system
  - designing different object models
  - specify 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
  - **dynamic models**: sequence models, state machine models
- Component interfaces must be defined precisely so that other objects can use them.



Chapter 7 – Design and Implementation

Lecture 2



Design patterns

Describes a proven approach to dealing with a common situation in programming/design

Suggests what to do to obtain an elegant, modifiable, extensible, flexible & reusable system or component

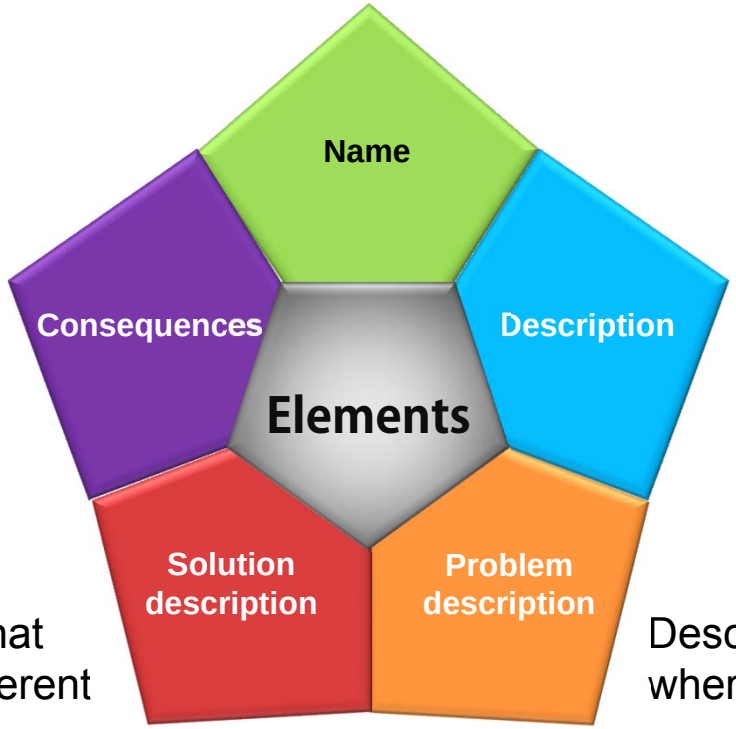
Shows, at design time, how to avoid problems that may occur much later

Is independent of specific contexts or languages

A description of a recurrent problem and of the core of possible solutions.

Pattern elements

A meaningful pattern identifier



The results and trade-offs of applying the pattern

Overall description of the pattern

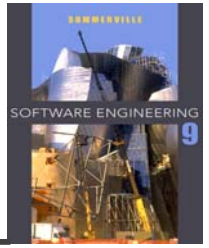
A template for a design that can be instantiated in different ways

Description of the situation where to use the pattern

Pattern elements

- Name
  - A meaningful pattern identifier.
- Problem description.
- Solution description.
  - Not a concrete design but a template for a design solution that can be instantiated in different ways.
- Consequences
  - The results and trade-offs of applying the pattern.

The Observer pattern



Name

- Observer

Description

- Separates the display of object state from the object itself

Problem description

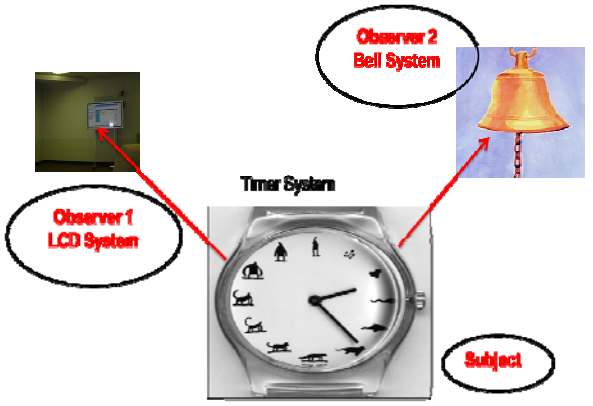
- Used when multiple displays of state are needed

Solution description

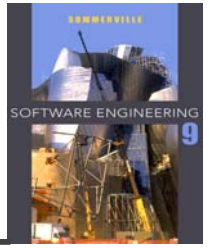
- See slide with UML description

Consequences

- Optimisations to enhance display performance are impractical

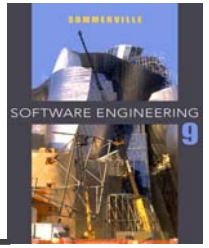


The Observer pattern (1)



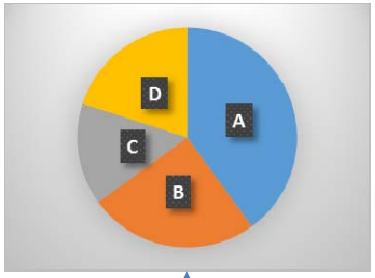
Pattern name	Observer
Description	Separates the display of the state of an object from the object itself and allows alternative displays to be provided. When the object state changes, all displays are automatically notified and updated to reflect the change.
Problem description	In many situations, you have to provide multiple displays of state information, such as a graphical display and a tabular display. Not all of these may be known when the information is specified. All alternative presentations should support interaction and, when the state is changed, all displays must be updated. This pattern may be used in all situations where more than one display format for state information is required and where it is not necessary for the object that maintains the state information to know about the specific display formats used.

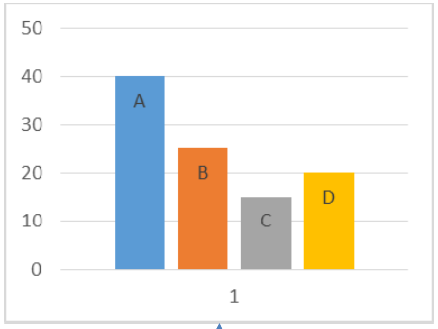
The Observer pattern (2)



Pattern name	Observer
Solution description	▪ This involves two abstract objects, Subject and Observer, and two concrete objects, ConcreteSubject and ConcreteObject, which inherit the attributes of the related abstract objects. ▪ The abstract objects include general operations that are applicable in all situations. ▪ The state to be displayed is maintained in ConcreteSubject, which inherits operations from Subject. ▪ The ConcreteObserver maintains a copy of the state of ConcreteSubject and implements the Update() interface of Observer. ▪ The ConcreteObserver automatically displays the state and reflects changes whenever the state is updated.
Consequences	The subject only knows the abstract Observer and does not know details of the concrete class. Therefore there is minimal coupling between these objects. Because of this lack of knowledge, optimizations that enhance display performance are impractical. Changes to the subject may cause a set of linked updates to observers to be generated, some of which may not be necessary.

Multiple displays using the Observer pattern





Observer 1

Subject

A: 40

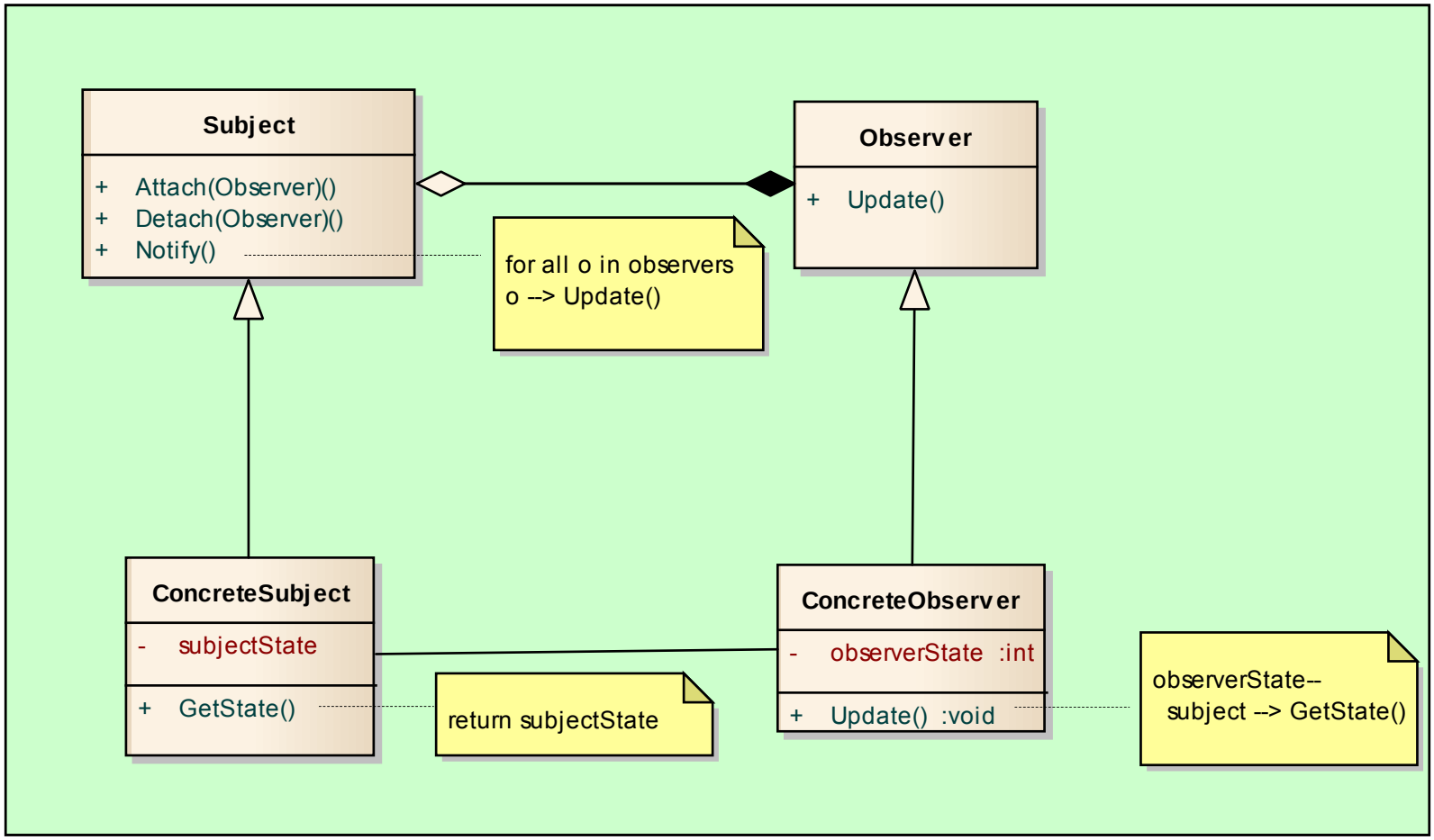
B: 25

C: 15

D: 20

Observer 2

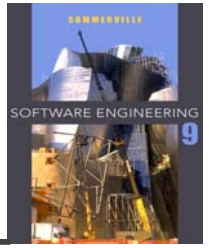
A UML model of the Observer pattern



Observer Pattern Video

Ref: http://www.youtube.com/watch?v=rWvXJo3OAzs

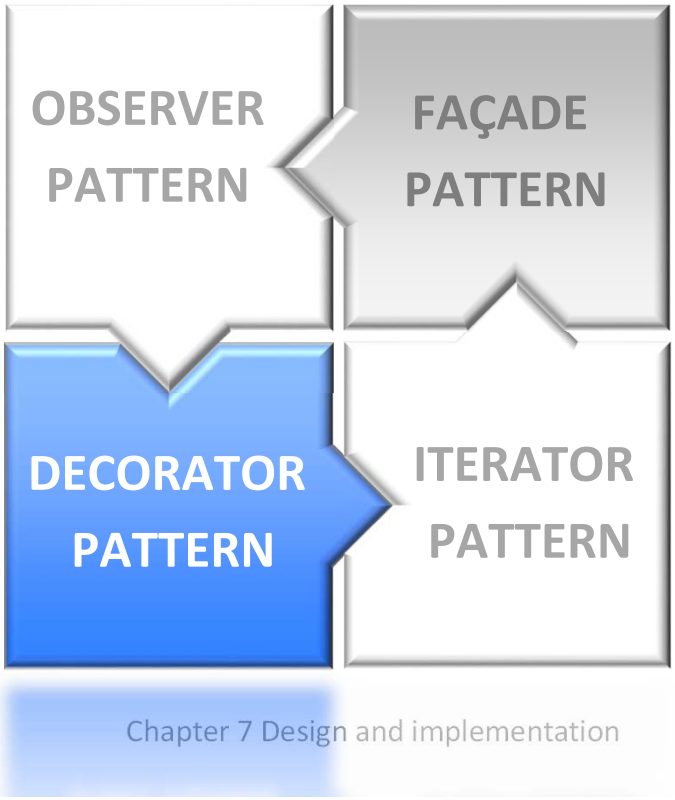
Design problems



To use patterns in your design, you need to recognize that any design problem you are facing may have an associated pattern that can be applied.

DESIGN PATTERNS

Problem
Tell several objects that the state of some other object has changed

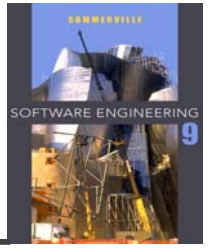


Problem
Tidy up the interfaces to a number of related objects that have often been developed incrementally

Problem
Allow for the possibility of extending the functionality of an existing class at run-time

Problem
Provide a standard way of accessing the elements in a collection, irrespective of how that collection is implemented

Implementation issues



Focus here is not on programming, although this is obviously important, but on 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.

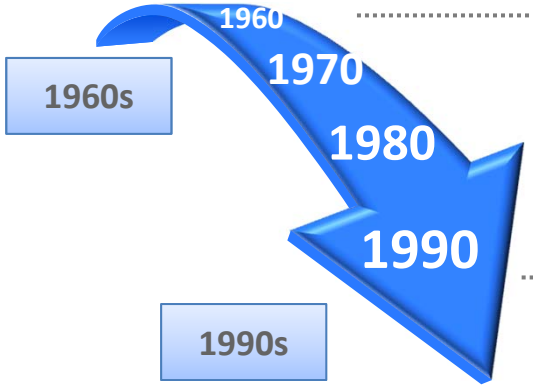
Configuration management

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.

Reuse



1960s – 1990s

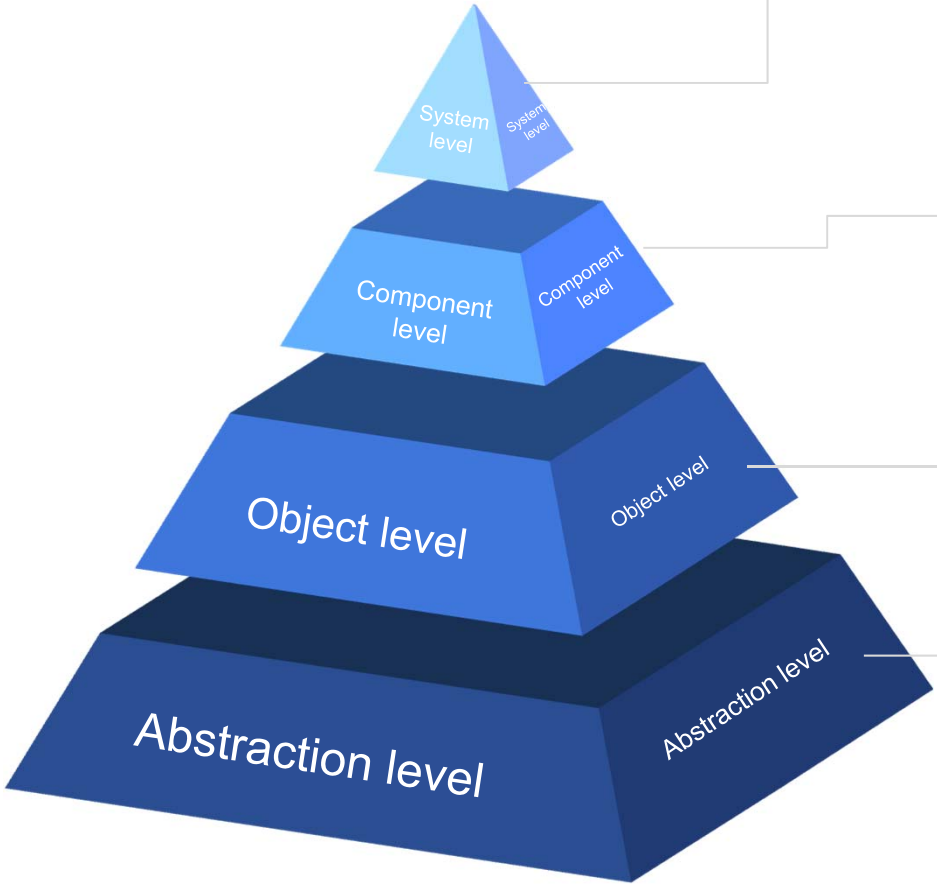
most new software was developed from scratch, by writing all code in a high-level programming language

The only significant reuse or software was the reuse of functions and objects in programming language libraries.

Costs and schedule pressure mean that this approach became increasingly unviable, especially for commercial and Internet-based systems.

An approach to development based around the reuse of existing software emerged and is now generally used for business and scientific software.

Reuse levels



The system level

At this level, you reuse entire application systems.

The component level

Components are collections of objects and object classes that you reuse in application systems.

The object level

At this level, you directly reuse objects from a library rather than writing the code yourself.

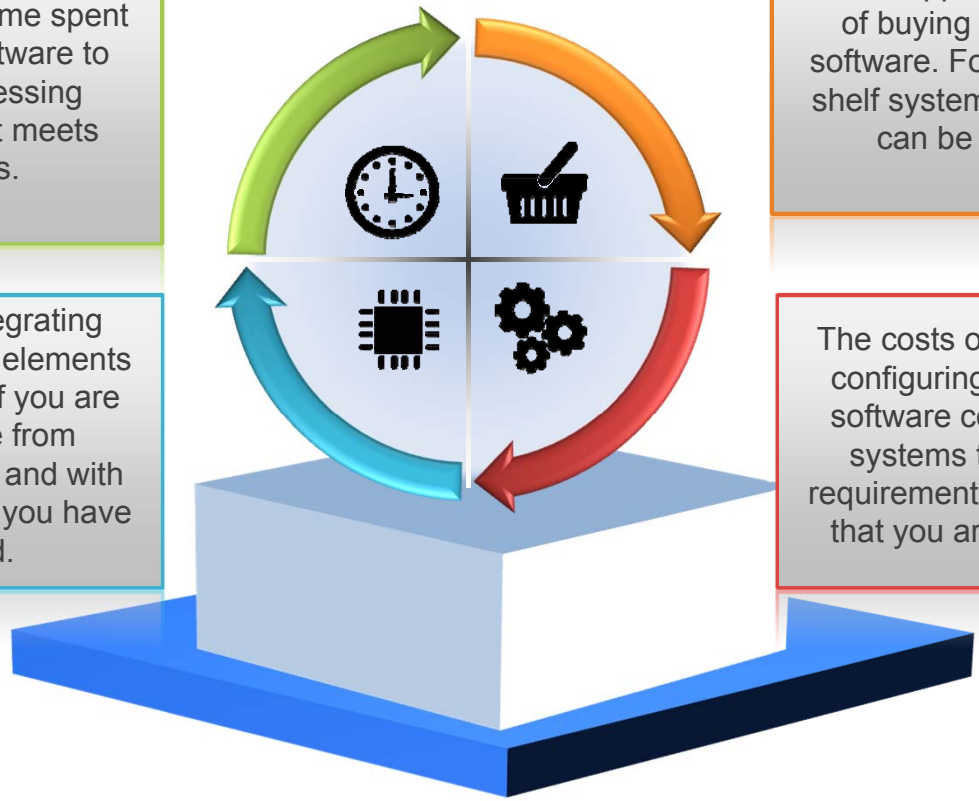
The abstraction level

At this level, you don't reuse software directly but use knowledge of successful abstractions in the design of your software.

Reuse costs

The costs of the time spent in looking for software to reuse and assessing whether or not it meets your needs.

Where applicable, the costs of buying the reusable software. For large off-the-shelf systems, these costs can be very high.



The costs of integrating reusable software elements with each other (if you are using software from different sources) and with the new code that you have developed.

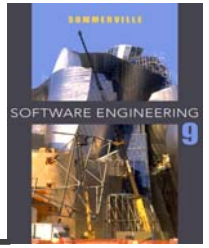
The costs of adapting and configuring the reusable software components or systems to reflect the requirements of the system that you are developing.

Configuration management

Configuration management is the name given to the general process of managing a changing software system.

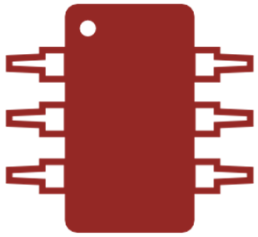
To support the system integration process so that all developers can access the project code and documents in a controlled way, find out what changes have been made, and compile and link components to create a system.

Configuration management activities



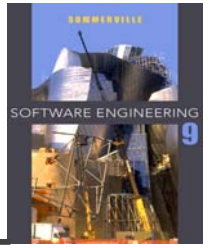
- Version management, where support is provided to keep track of the different versions of software components. Version management systems include facilities to coordinate development by several programmers.

- System integration, where support is provided to help developers define what versions of components are used to create each version of a system. This description is then used to build a system automatically by compiling and linking the required components.

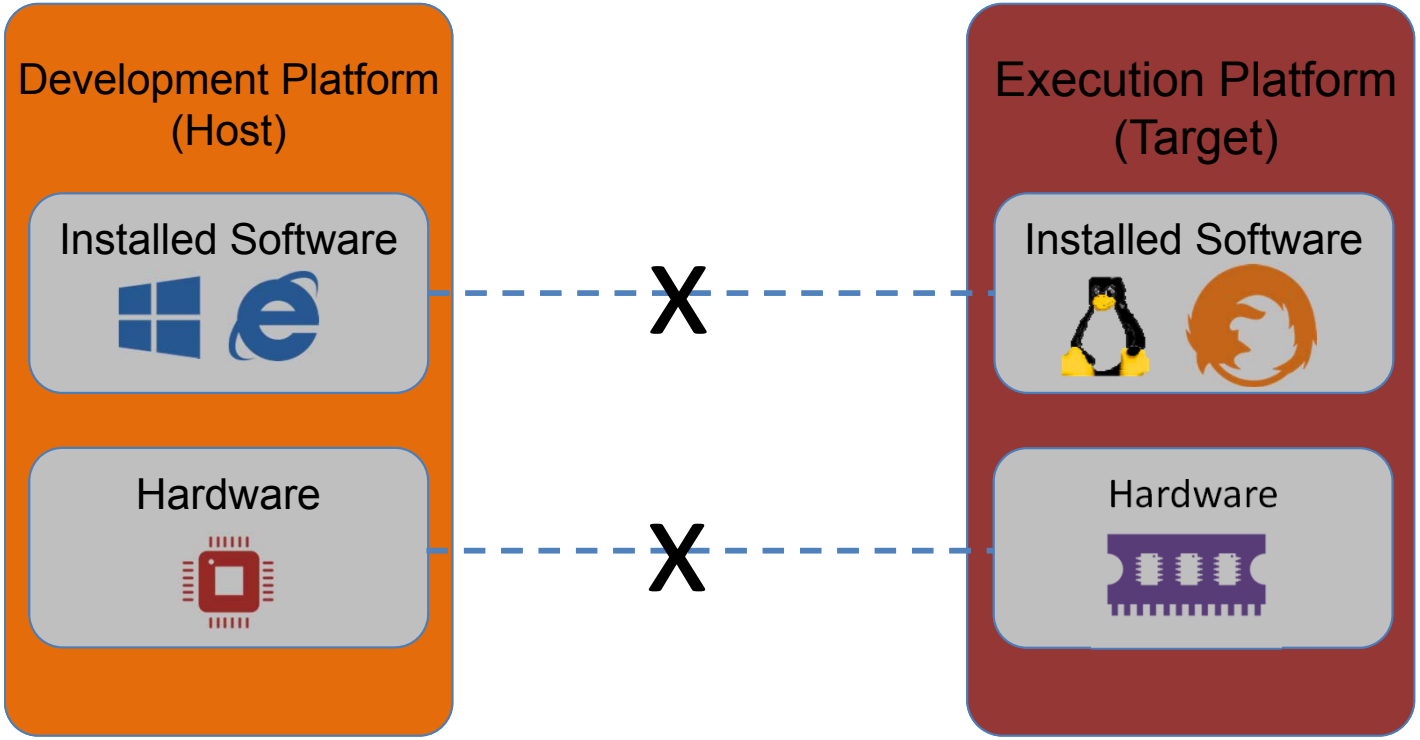


- Problem tracking, where support is provided to allow users to report bugs and other problems, and to allow all developers to see who is working on these problems and when they are fixed.

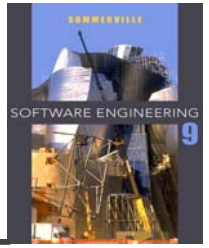
Host-target development



Development platform usually has different installed software than execution platform; these platforms may have different architectures.



Host-target development



- Most software is developed on one computer (the host), but runs on a separate machine (the target).
- More generally, we can talk about a development platform and an 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 or, for development platforms, an interactive development environment.
- Development platform usually has different installed software than execution platform; these platforms may have different architectures.

Development platform tools

An integrated compiler and syntax-directed editing system that allows you to create, edit and compile code.

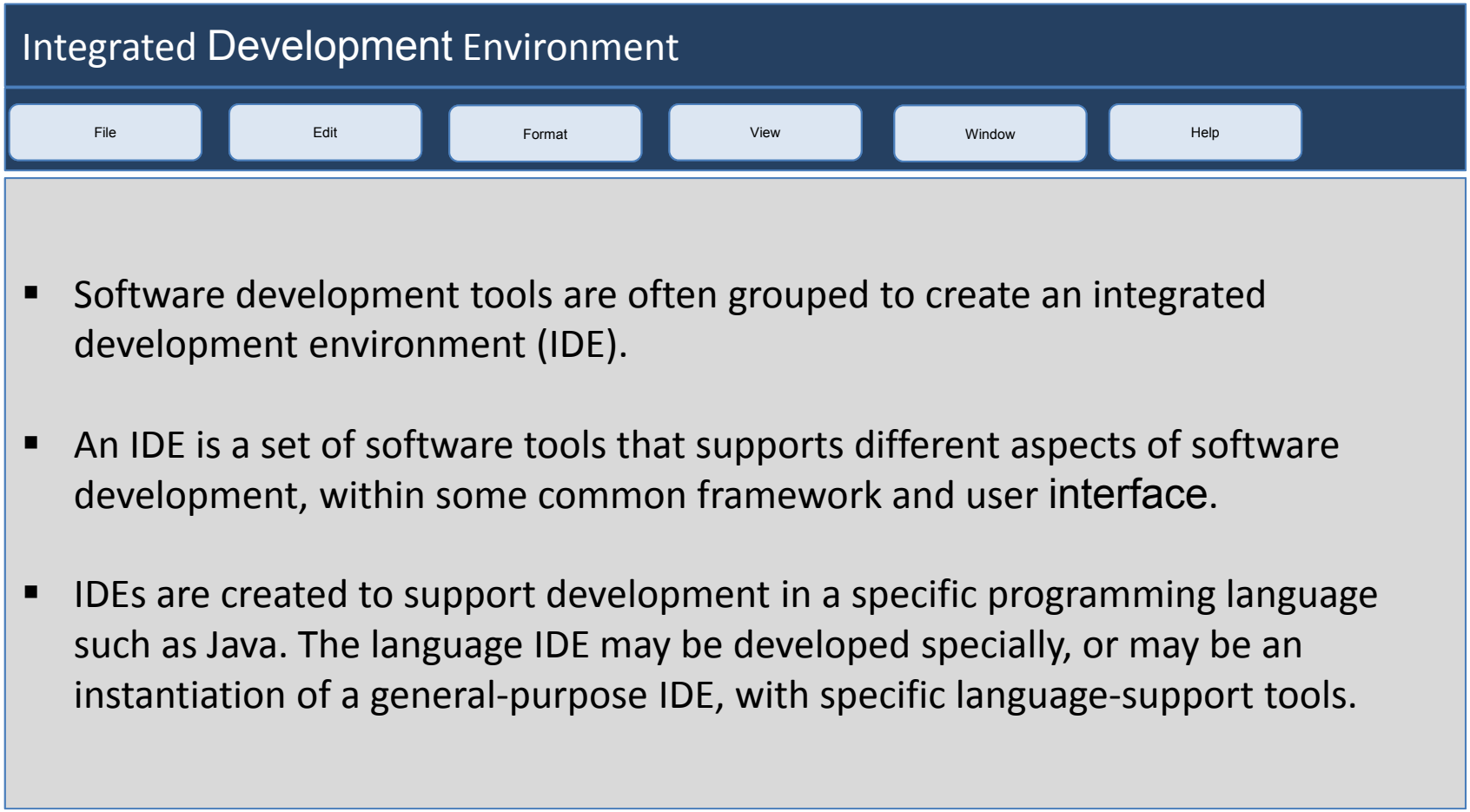
A language debugging system.

Graphical editing tools, such as tools to edit UML models.

Testing tools, such as Junit that can automatically run a set of tests on a new version of a program.

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. The language IDE may be developed specially, or may be an instantiation of a general-purpose IDE, with specific language-support tools.

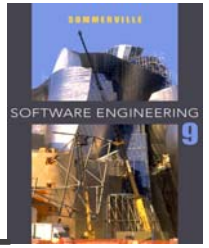
Component/system deployment factors

If a component is designed for a specific hardware architecture, or relies on some other software system, it must obviously be deployed on a platform that provides the required hardware and software support.

High availability systems may require components to be deployed on more than one platform. This means that, in the event of platform failure, an alternative implementation of the component is available.

If there is a high level of communications traffic between components, it usually makes sense to deploy them on the same platform or on platforms that are physically close to one other.

Open source development



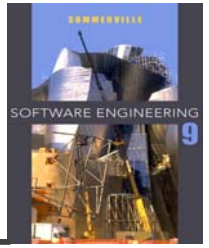
Open source development is an approach to software development in which the source code of a software system is published and volunteers are invited to participate in the development process.

FREE SOFTWARE FOUNDATION

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.

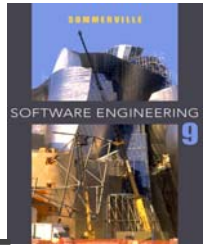
Open source software extended this idea by using the Internet to recruit a much larger population of volunteer developers. Many of them are also users of the code.

Open Source Video



Ref: http://www.youtube.com/watch?v=a8fHgx9mE5U

Open source systems

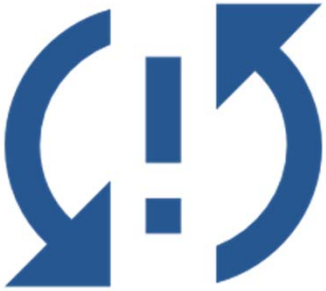


- **Linux operating system**
  - best-known open source product
  - widely used as a server system
  - Increasingly as a desktop environment

- **Other important open source products are:**

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?

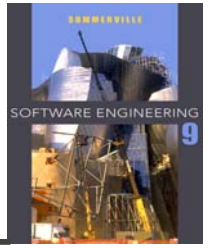


lack of focus on user interface design

lack the complete & accessible documentation

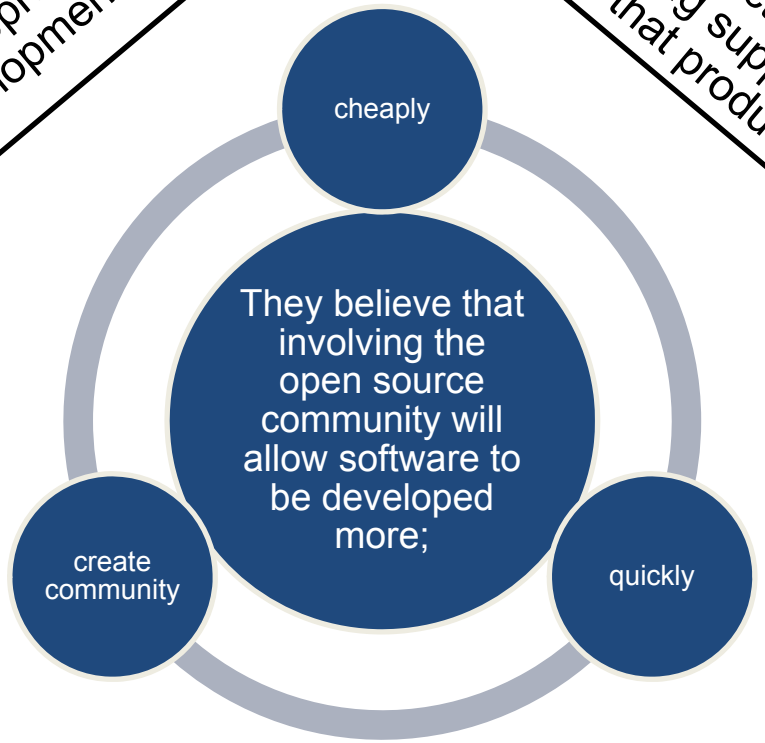
focus on features rather than a solid core

Open source business

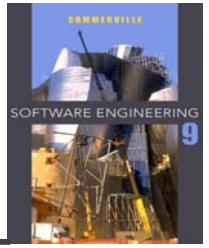


More and more product companies are using an open source approach to development.

Their business model is not reliant on selling a software product but on selling support for that product.

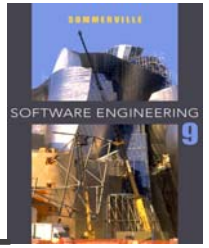


Open source business



- 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.

Open source licensing



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.

Legally, the developer of the code (either a company or an individual) still owns the code. They can place restrictions on how it is used by including legally binding conditions in an open source software license.

Some open source developers believe that if an open source component is used to develop a new system, then that system should also be open source.

Others are willing to allow their code to be used without this restriction. The developed systems may be proprietary and sold as closed source systems.

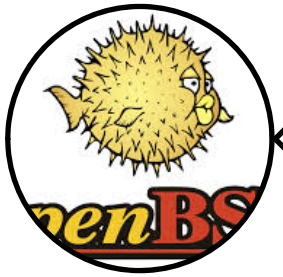
License models

The GNU General Public License (GPL)

- This is a so-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.

The GNU Lesser General Public License (LGPL)

- This 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.



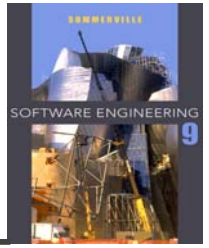
The Berkley Standard Distribution (BSD) License

- This is a non-reciprocal license, which means you are not obliged to re-publish any changes or modifications made to open source code. You can include the code in proprietary systems that are sold.

Open Source vs Closed System

Ref: http://www.youtube.com/watch?v=7NShpY_oSs4

License management



- 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.
- Be aware of evolution pathways for components.
- Educate people about open source.
- Have auditing systems in place.
- Participate in the open source community.

Key points

- When developing software, you should always consider the possibility of reusing existing software, either as components, services or complete systems.

Reuse

- Configuration management is the process of managing changes to an evolving software system.
  - It is essential when a team of people are cooperating to develop software.

Configuration Management

- 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.

Host-Target

- 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.

Open Source