

# SWVE3002-42: Introduction to Software Engineering

## Lecture 7 – Design and Implementation

Sooyoung Cha

Department of Computer Science and Engineering

## Topics covered

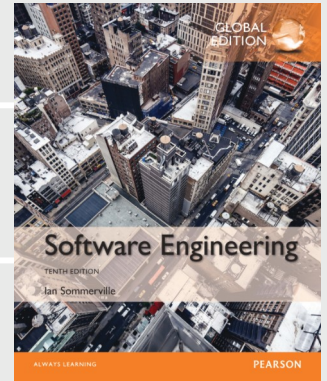
---

**01 | Object-oriented design using the UML**

**02 | Design patterns**

**03 | Implementation issues**

**04 | Open-source development**



Chapter 7. (p.196 ~ p.225)

## [SW Design and Implementation]

- The software engineering process at which **an executable SW system is developed**.
  - SW design: A creative activity in which you **identify SW components and their relationships**, based on a customer's requirements.
  - Implementation: The process of **realizing** the design **as a program**.
- An important implementation decision.
  - Deciding whether to build or **to buy the application software**.  
ex) A medical records system → A package that is already used in hospitals.
  - **Buy → How to configure the system product** to meet the requirements?

### [The goals of this chapter]

- How **system modeling and architectural design** are put into practice in developing an object-oriented SW design?



- What are **the important implementation issues**?
  - Software reuse.
  - Configuration management.
  - Open-source development.

## Object-oriented design using the UML

---

### [The method for developing a system design]

1. Understanding and defining the context and the external interactions with the system.
2. Designing the system architecture.
3. Identifying the principal objects in the system.
4. Developing design models.
5. Specifying interfaces.

## System context and interactions

---

### [1. System context and interactions]

- **The first stage** in any SW design process
  - **Understanding of the relationships** between our SW and its external environment.  
(= Setting the system boundaries.)
- **System context model**
  - **A structural model** that demonstrates the other systems in the environment of the system being developed.
- **Interaction model**
  - A dynamic model that shows how the system interacts with its environment.

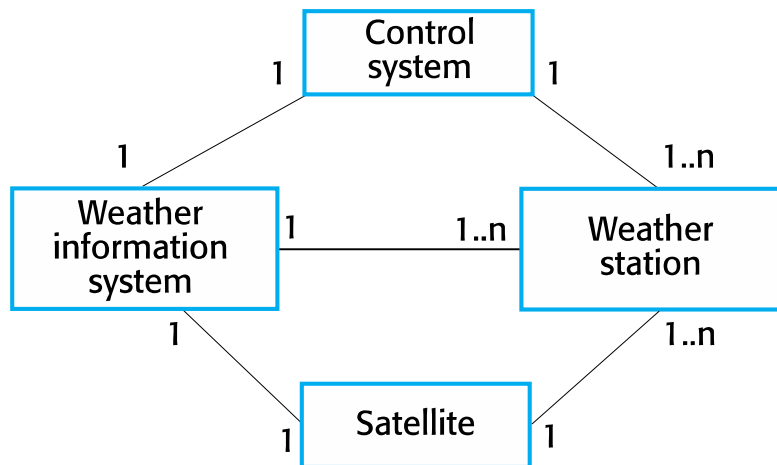
## Chapter 7-1. Object-oriented design using the UML

# System context and interactions

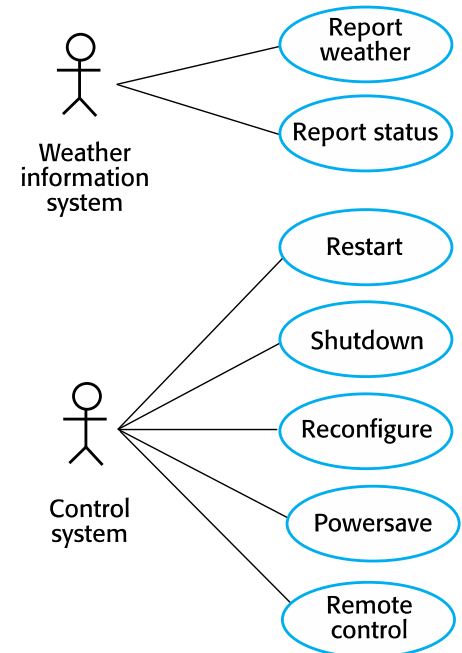
## [A design for the wilderness weather station system]

### ■ Weather station

- Collecting weather data
- Carrying out some initial data processing.
- Transmitting the data to the data management system.



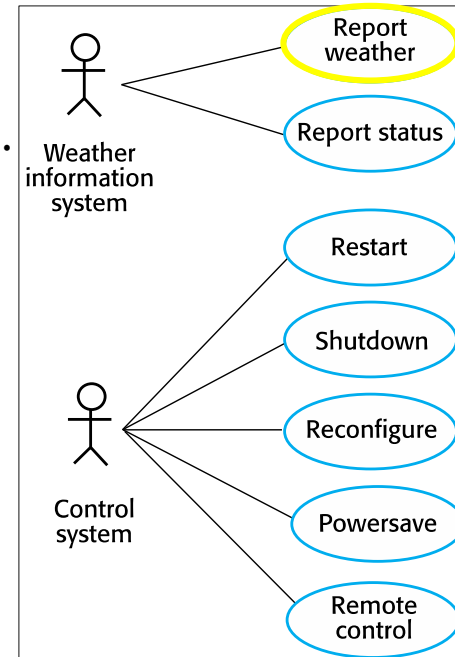
System context for the weather station



Weather station use cases

### [Weather station use cases]

- Report weather & Report status
  - Send weather data to the weather information system.
  - Send status information to the weather information system.
- Restart & Shutdown
- Reconfigure
  - Reconfigure the weather station software.
- Powersave
  - Put the weather station into power-saving mode.
- Remote control
  - Send control commands to any weather station subsystem.





## System context and interactions

---

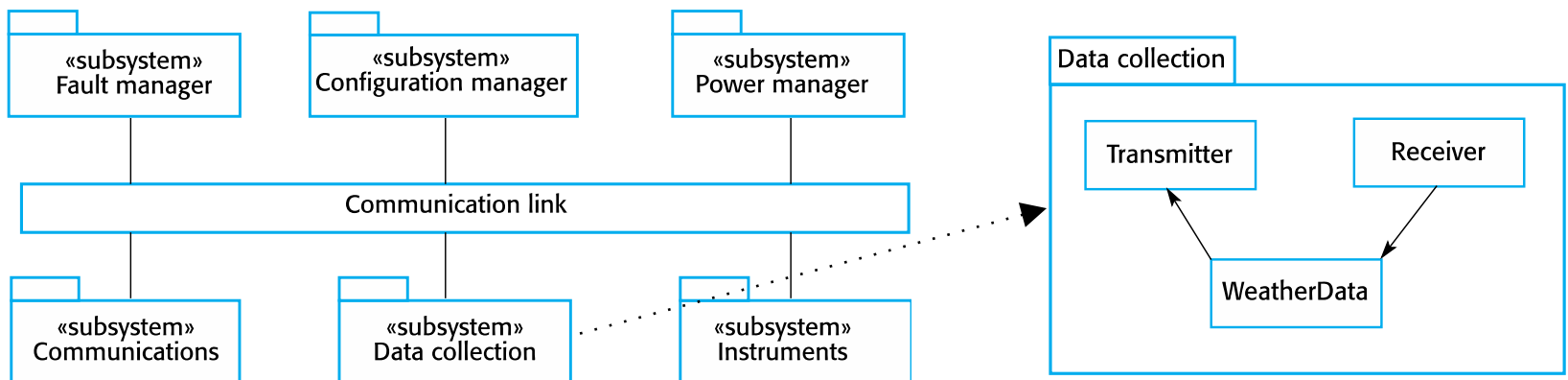
### [Use case description for reporting weather]

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 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 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.

## [2. Architectural design]

- Identifying the major components that make up the system and their interactions.
- Designing the system organization using an architectural pattern.

## [High-level architecture of weather station]



## Object class identification

---

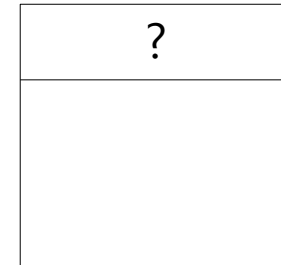
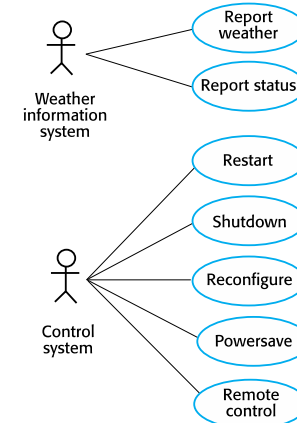
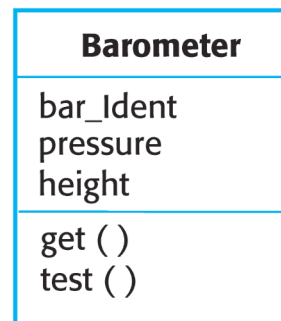
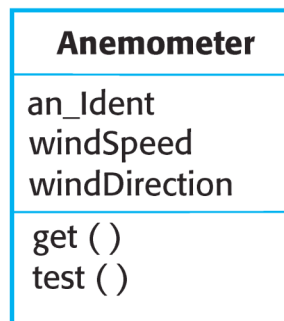
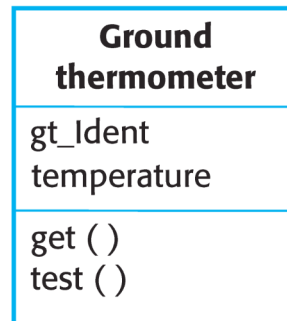
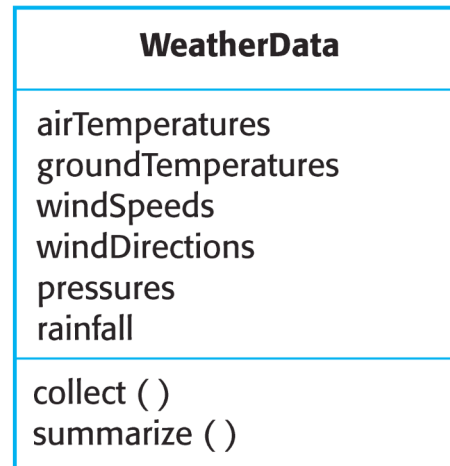
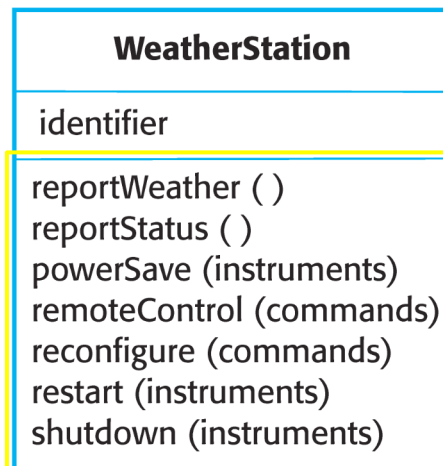
### [3. Object class identification]

- **Identifying high-level system objects** that encapsulate the system interactions defined in the use cases.
  
- Three ways of identifying object classes in object-oriented systems
  - Using **a grammatical analysis of a natural language** description of the system.  
(e.g., objects or attributes = nouns, operations or services = verbs)
  - **Using tangible entities (things).**  
(e.g., domain → aircraft, role → manager, location → office)
  - Using **a scenario-based analysis.**

## Chapter 7-1. Object-oriented design using the UML

# Object class identification

### [Example for weather station objects]



## Object class identification

---

### [Example for weather station objects]

- Weather Station object class
  - Providing the basic interface of the weather station with its environment.
- Weather Data object class
  - Processing the report weather command.  
(Sending the summarized data from the weather station instruments to the weather information system.)
- Ground thermometer, Anemometer, and Barometer object classes
  - Reflecting tangible hardware entities in the system.
  - Operating autonomously to collect data at the specified frequency.
  - Storing the collected data locally.

## Design models

---

### [4.About Design models]

- **Bridge** between the system requirements and its implementation.
  - Showing the objects or object classes in a system.
  - **Showing the associations and relationships** between these entities.
- Deciding **which design models** to use.
  - Use-case, sequence, state, class, and activity diagrams.
- **Level of detail** in a design model **depends on the design process**.
  - Agile development (e.g., whiteboard)
  - Plan-based development (detailed models)
- Minimizing the number of models.
  - **Reducing the costs** of the design and the time.

## Design models

---

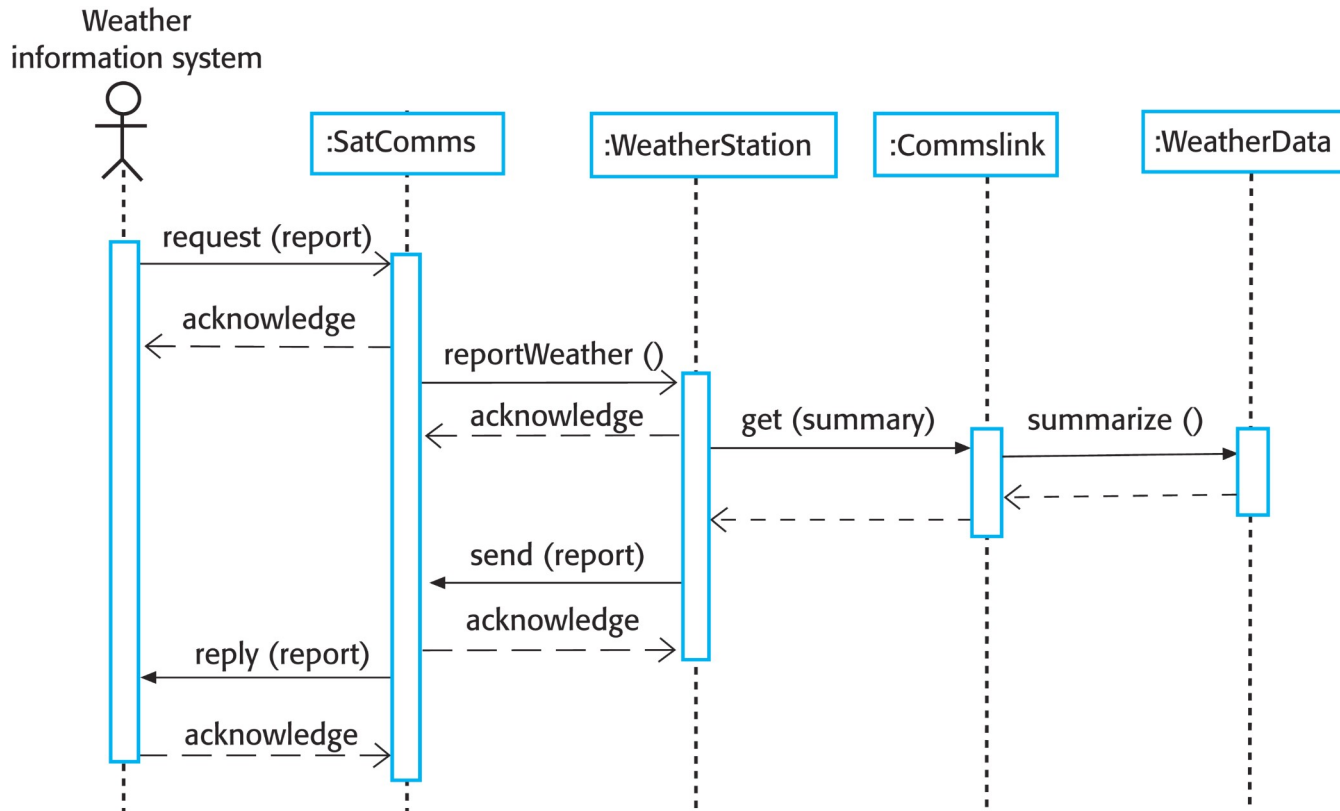
### [Two kinds of design model]

- Structural models.
  - Showing the static structure of the system using object classes and their relationships. (e.g., class diagram)
- Dynamic models.
  - Showing the dynamic structure of the system and the expected runtime interactions between the system objects.
  - (e.g., sequence diagram, state diagram)

## Chapter 7-1. Object-oriented design using the UML

# Design models

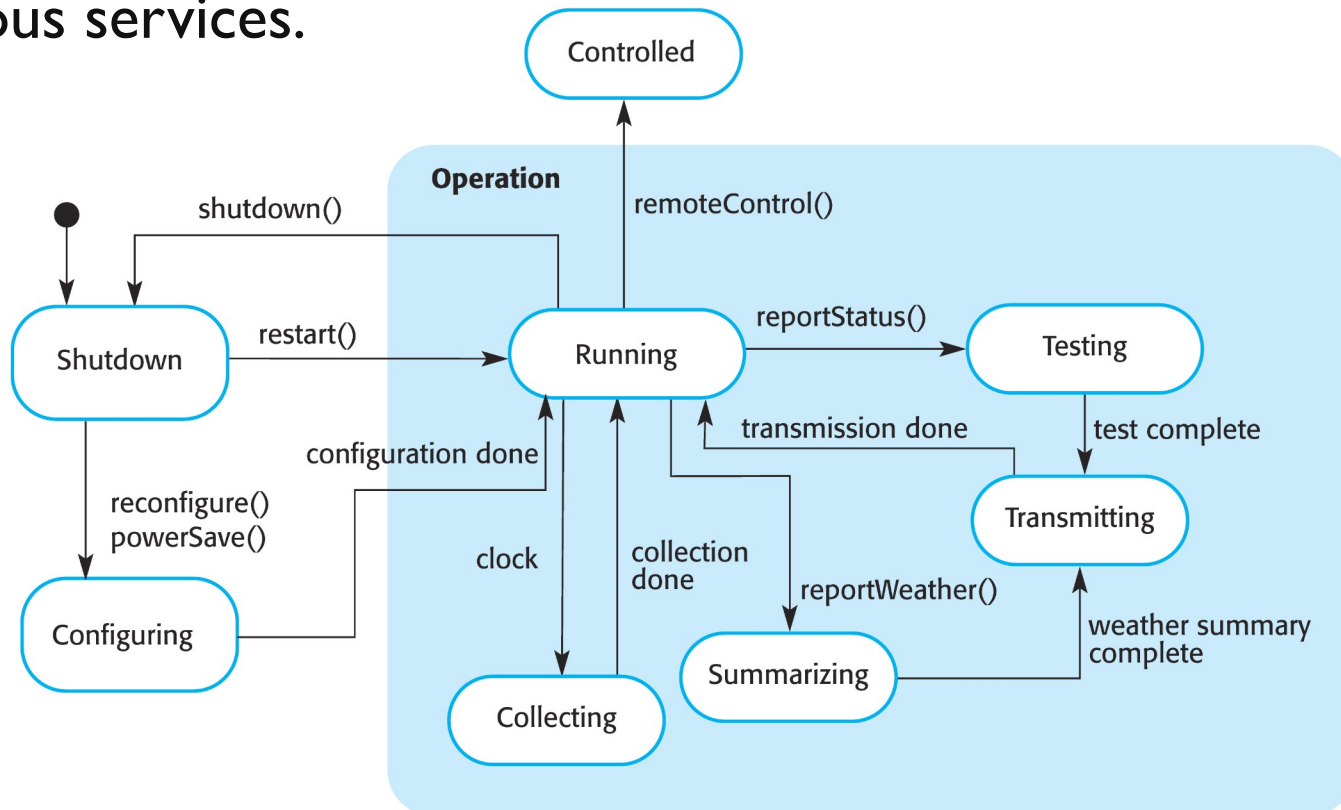
### [Sequence diagram describing data collection]





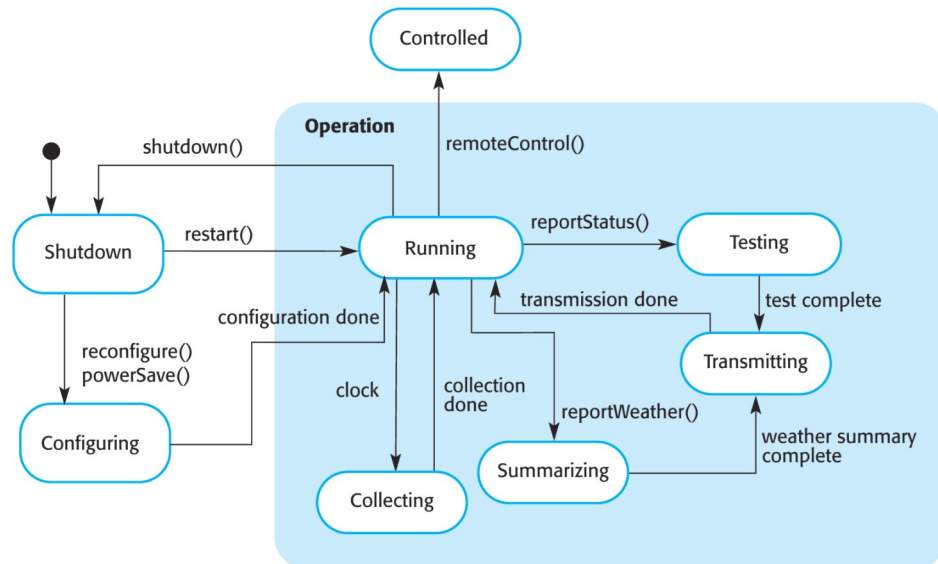
# [Weather station state diagram]

- Showing how the weather station system responds to requests for various services.



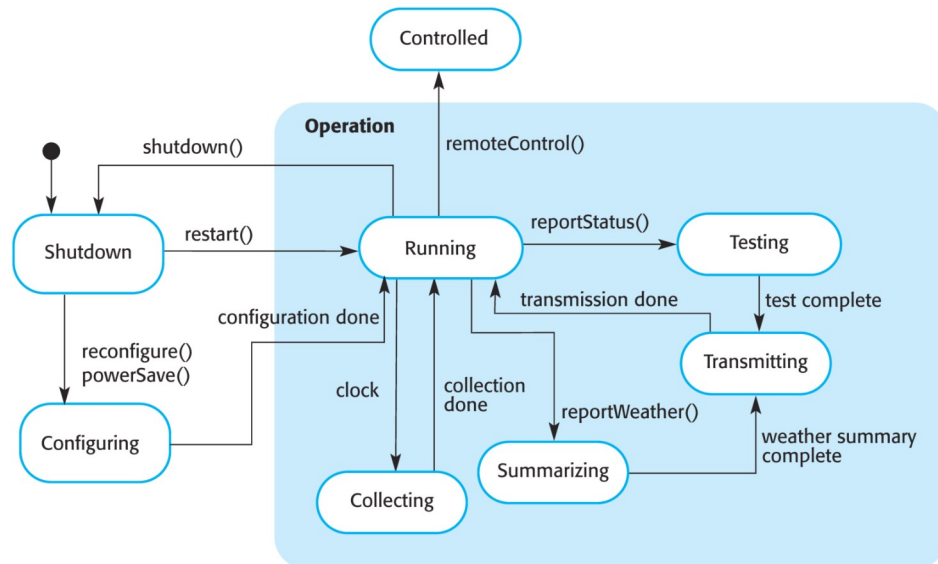
### [Weather station state diagram]

- Shutdown state.
  - The initial state.
  - Responding to a restart(), a reconfigure() or a powerSave() message.
  - Reconfiguration is allowed only if the system has been shut down.



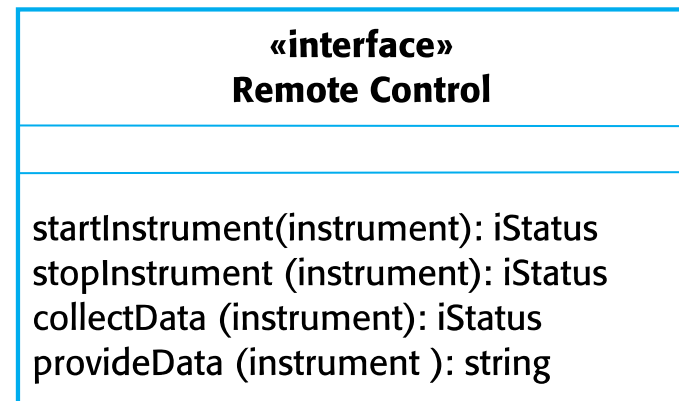
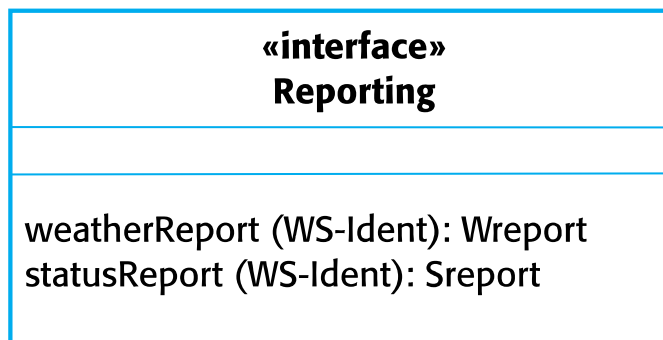
### [Weather station state diagram]

- Running state.
  - The system expects further messages.
  - A shutdown() message → shutdown state.
  - A reportWeather() message → Summarizing state → Transmitting state → Running state.



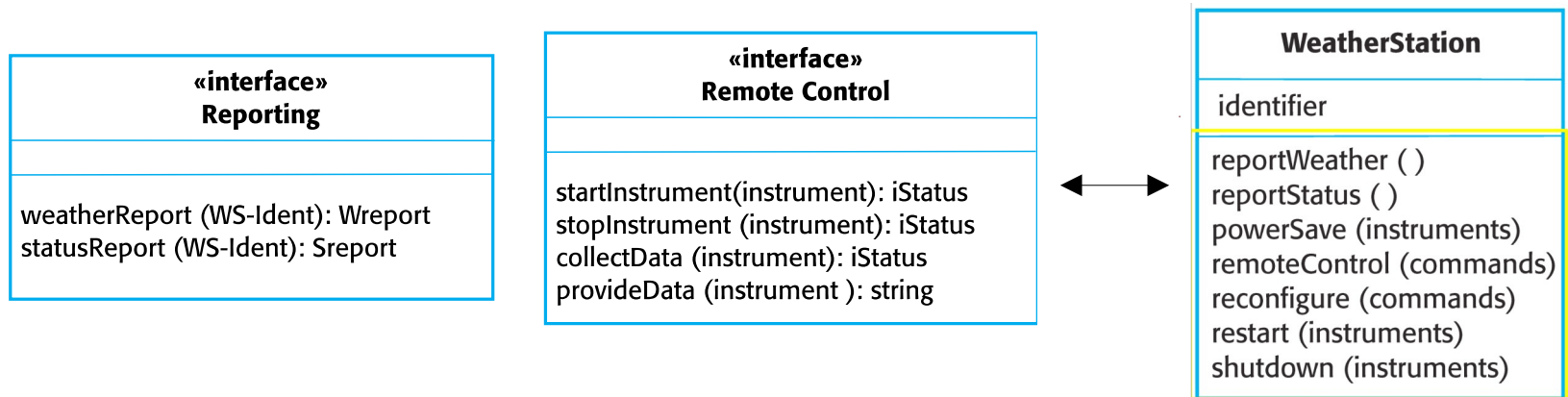
### [5.About interface specification]

- Specifying the detail of the interface to an object or a group of objects.
- Defining the semantics of services provided by a group of objects.
- Not including details of the data representation in an interface design.
  - We can easily change it without affecting the objects that use that data.
- Including operations to access and update data.



## [5.About interface specification]

- Interfaces can be specified in the UML using the same notation as a class diagram.
  - No attribute section.
  - The UML stereotype «interface» inclusion.
  - (Reporting) Weather and status reports map directly to operations.
  - (Remote Control) Four operations map onto a single method in the object.




## Chapter 7-2. Design patterns

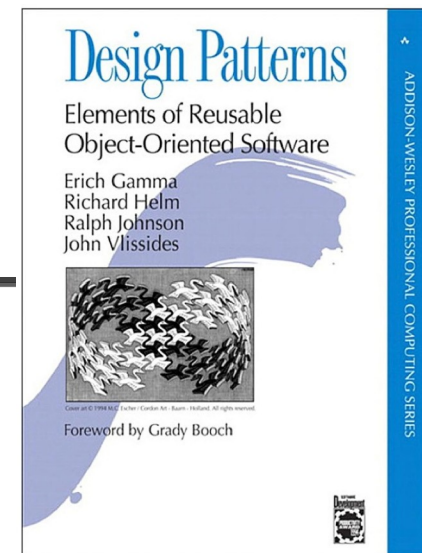
# Design patterns

### [Design patterns]

Christopher Alexander



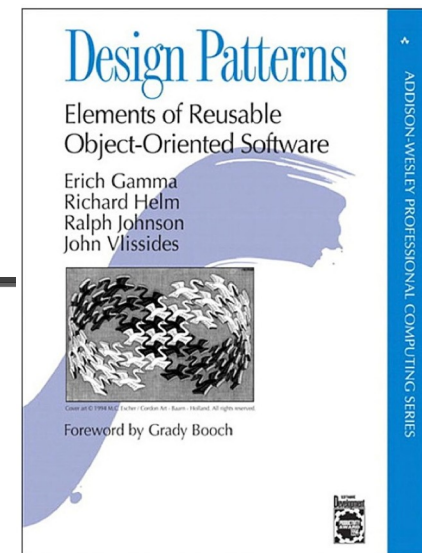
- Motivation 
  - Common patterns of building design that were inherently effective.
- A pattern
  - A description of accumulated wisdom and experience.
  - A well-tried solution to a common problem.
  - A way of reusing the knowledge and experience of other designers.
- Design patterns are associated with object-oriented design.
  - Published patterns often rely on object characteristics such as inheritance.



## Design patterns

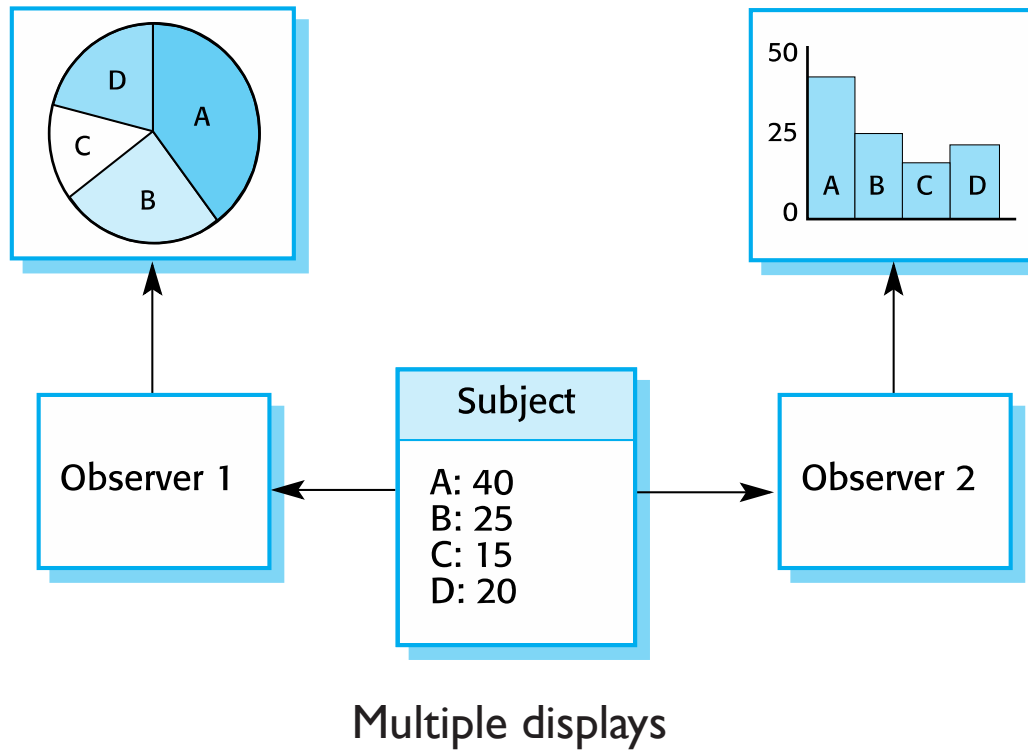
### [Four essential elements of design patterns]

- A name
  - A meaningful reference to the pattern.
- A problem description
  - The problem area that explains **when the pattern may be applied**.
- A Solution description
  - The parts of the design solution, their relationships and their responsibilities.
- A statement of the consequences (The results and trade-offs)
  - Helping designers understand whether or not a pattern can be used in a particular situation.



## Design patterns

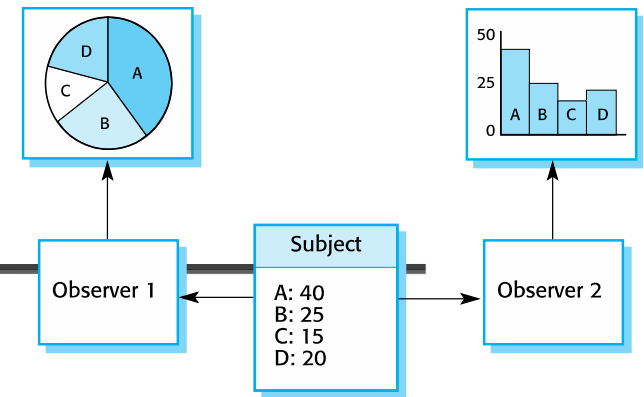
### [Observer pattern]





# Design patterns

## [Observer pattern]



Pattern name	Observer
Description	<p>Separates the display of the state of an object from the object itself and allows alternative displays to be provided.</p> <p>...</p>
Problem description	<p>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.</p> <p>This pattern may be used in all situations where more than one display format for state information is required. ...</p>

## Design patterns

---

### [Observer pattern]

Solution description	<p>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 state to be displayed is maintained in ConcreteSubject, which inherits operations from Subject allowing it to add and remove Observers and to issue a notification when the state has changed.</p> <p>...</p>
Consequences	<p>The subject only knows the abstract Observer and does not know details of the concrete class. Therefore there is minimal coupling between these objects. Changes to the subject may cause a set of linked updates to observers to be generated, some of which may not be necessary.</p>

## Design patterns

---

### [Design patterns]

- **Need to recognize** that your design problem has an associated pattern that can be applied.
  - **Tell several objects** that the state of some other object has changed (Observer pattern).
  - **Tidy up the interfaces** to a number of related objects that have often been developed incrementally (Façade pattern).
  - **Allow** for the possibility of **extending the functionality** of an existing class **at runtime** (Decorator pattern).

...

## [ Implementation]

- Developing programs in high or low-level programming languages.
- **Tailoring off-the-shelf systems** to meet the specific requirements.

## [Important aspects of implementation in SW engineering]

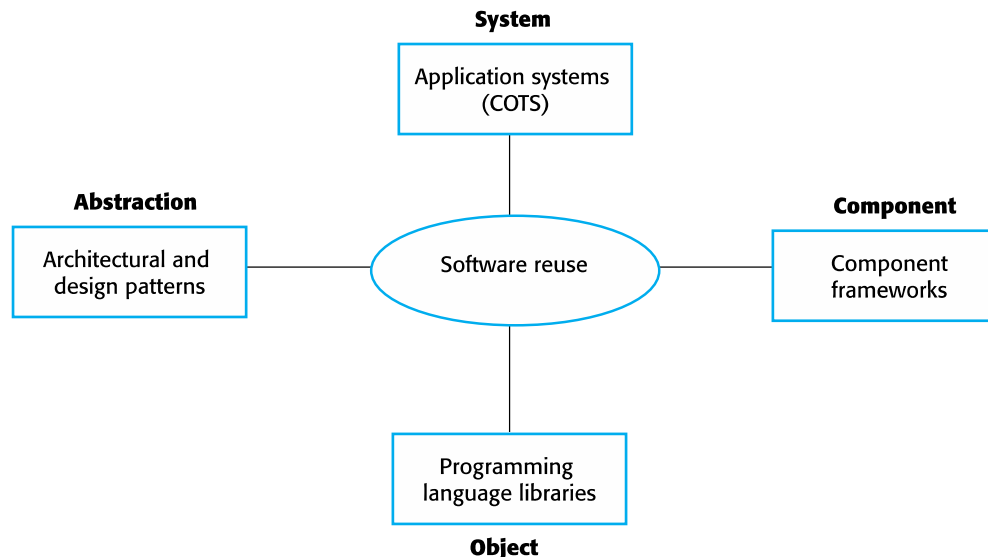
- **Reuse**
- **Configuration management**
  - **Managing many different versions of SW** created during the development process.
- **Host-target development**
  - Developing the SW on one computer (the host system)
  - Executing the SW on a separate computer (the target system)

## Reuse

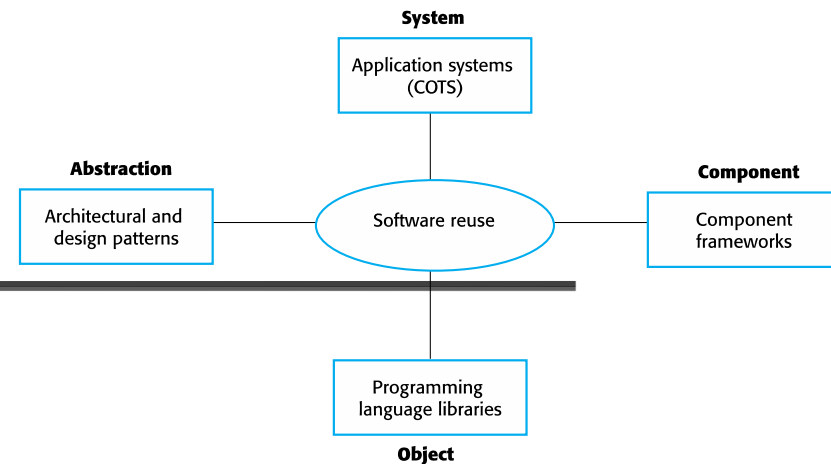
### [Reuse]

- (1960s ~ 1990s) Developing the SW **from scratch**.
- (Now) **Reusing the existing SW** for system development.

### [Different levels of SW reuse]



## Reuse




### [Different levels of SW reuse]

- The abstraction level
  - Don't reuse SW directly but use knowledge of successful abstractions in the design of your SW. (e.g., architectural patterns and design patterns)
- The object level
  - Reuse objects directly by finding appropriate libraries. (e.g., JavaMail library)
- The component level
  - Reuse components, the collections of objects and object classes that operate together to provide related functions and services.
- The system level
  - Reuse entire application systems. (e.g., most commercial systems)

## Reuse

---

### [Reuse costs]

- The costs of the time spent in looking for SW to reuse and assessing whether or not it meets your needs.
- The costs of buying the reusable SW. The Sparrow logo, featuring the word "Sparrow" in a blue script font with a small yellow and blue bird icon to the right.
- The costs of configuring the reusable SW systems to reflect the requirements of the system.
- The costs of integrating reusable software elements with each other and with the new code.




### [Configuration management]

- The general process of **managing a changing software system**.
  - Change management is absolutely essential in SW development.
- Goal
  - **Supporting the system integration process** so that all developers can access the project code in a controlled way. (e.g., conflict in git)
  - Ensuring that everyone can access **the most up-to-date versions of SW**.
  - Finding out **what changes have been made**.



## [4 fundamental configuration management activities]

- Version management.
  - Keeping track of the different versions of software components.
  - Coordinating development by several programmers.
  - Stopping one developer from overwriting code submitted by someone else.

update  sooyoungcha committed on 26 Feb 2021	 72a1665 
update  sooyoungcha committed on 26 Feb 2021	 53c756c 
Merge branch 'master' of <a href="https://github.com/kupl/FSE21">https://github.com/kupl/FSE21</a>  audxo14 committed on 26 Feb 2021	 bdde32b 
update  sooyoungcha committed on 26 Feb 2021	 fb3108f 
Update insight.tex  audxo14 committed on 26 Feb 2021	 be4b215 
Merge branch 'master' of <a href="https://github.com/kupl/FSE21">https://github.com/kupl/FSE21</a>  audxo14 committed on 26 Feb 2021	 7c32c41 

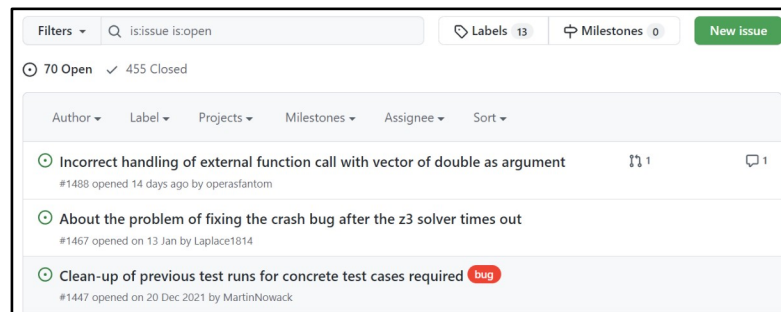
## [4 fundamental configuration management activities]

### ■ System integration.

- Helping developers define **what versions of components are used** to create each version of a system.

### ■ Problem tracking.

- Allowing users to **report bugs and other problems**.
- Allowing all developers to see **who is working** on these problems and **when they are fixed**.






## [4 fundamental configuration management activities]

### ■ Release management.

- Planning the functionality of new releases and **organizing the software for distribution.**

**KLEE 2.3** Latest

 ccadar released this 3 days ago  v2.3  879be79

---

**KLEE 2.3, 4 April 2022**

---

Incorporating changes from 8 December 2020 to March 2022.  
Maintainers during this time span: @ccadar and @MartinNowack  
Documentation at <http://klee.github.io/releases/docs/v2.3>

---

**LLVM support**

---

- Current recommended version is LLVM 11
- Added support for LLVM 12 and 13 (@Izaoral)
- Removed support for LLVM <6
- KLEE 2.3 will be the last version with support for LLVM <9

---

**Options, scripts and KLEE intrinsics added, changed or removed**

---

- Added --max-static-pct-check-delay to specify the number of forks after which the --max-static-\*-pct checks are enforced (@ccadar)
- In klee-stats, added --print-columns to print user-defined columns, e.g. --print-columns 'Path,Instrs,Time(s)' (@251)
- Disabled Doxygen generation by default; it can be enabled via CMake option ENABLE\_DOXYGEN=ON

<https://github.com/klee/klee>

## [Open-source development]

- The source code of a SW system is published and volunteers are invited to participate in the development process.
- The benefits of using open-source SW
  - It is usually cheap or even free to acquire open-source SW.
  - Widely used open-source systems are very reliable.
- The issues when using open-source SW in a company.
  - Should our product make use of open-source components?
  - Should an open-source approach be used for its own software development?

## Chapter 7-4. Open source development

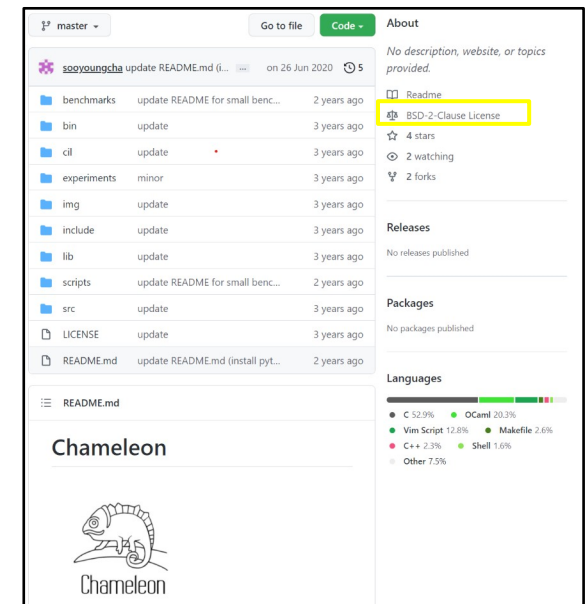
# Open-source development

### [Open-source licensing]

- Source code should be freely available in open-source development, but it does not mean that **anyone can do as they wish with the code**.
  - Legally, the developer of the code owns the code.
  - Developers can place restrictions on how the SW is used.

- Open-source licenses

- The GNU General Public License (GPL)
- The GNU Lesser General Public License (LGPL)
- The Berkley Standard Distribution (BSD) License



## [Open-source licensing]

- The GNU General Public License (GPL)

- A reciprocal license

(if using open-source SW with the GPL, you **must make the SW open source.**)

- The GNU Lesser General Public License (LGPL)

- A variant of the GPL license

(You can use open-source SW **without making your SW open source.**)

(If you change the licensed component, then **you must publish this as open source.**)

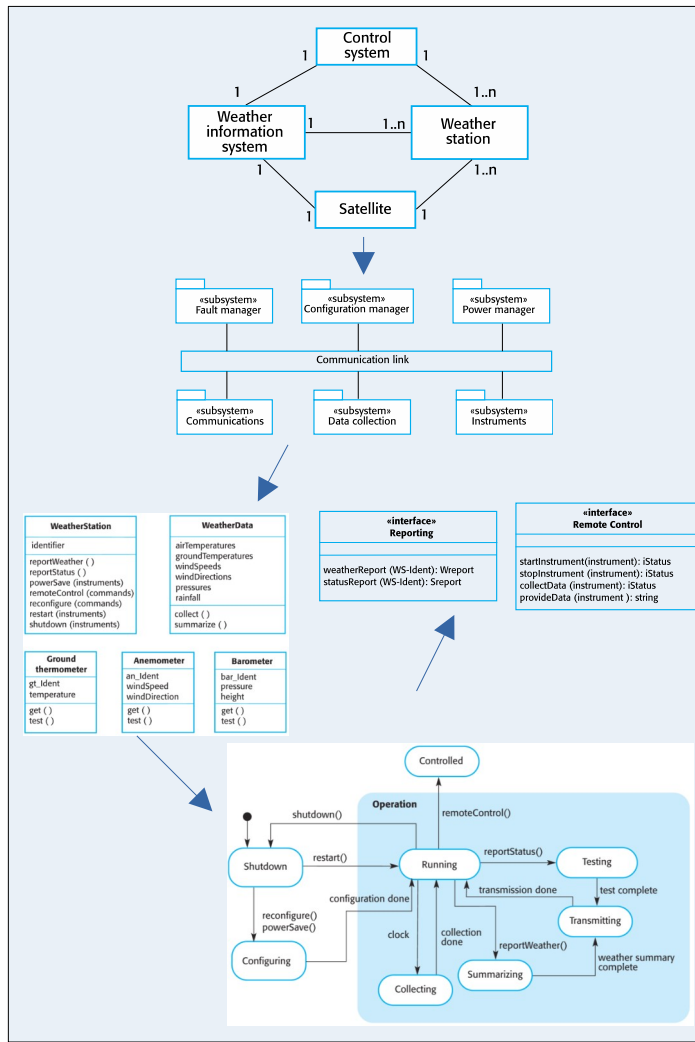
- The Berkley Standard Distribution (BSD) License

- A nonreciprocal license

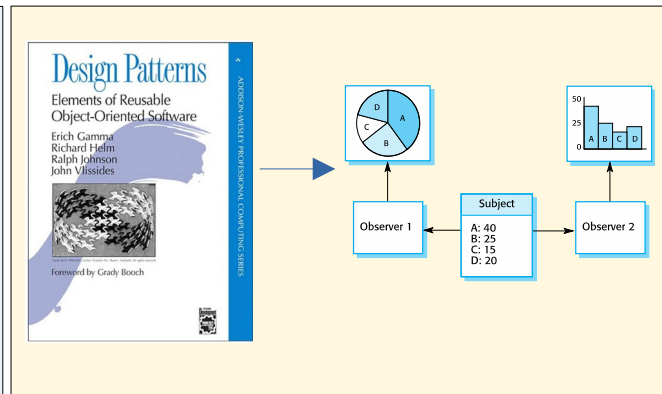
(You are not obliged to re-publish any changes for open-source code.)

# Chapter 7. Design and Implementation Summary

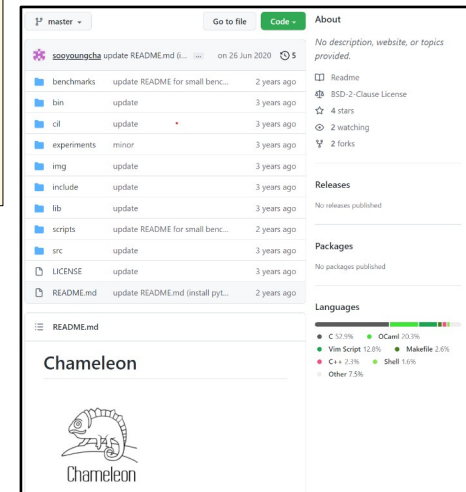
## Object-oriented design using the UML



## Design patterns



## Open-source development



## Implementation Issues

