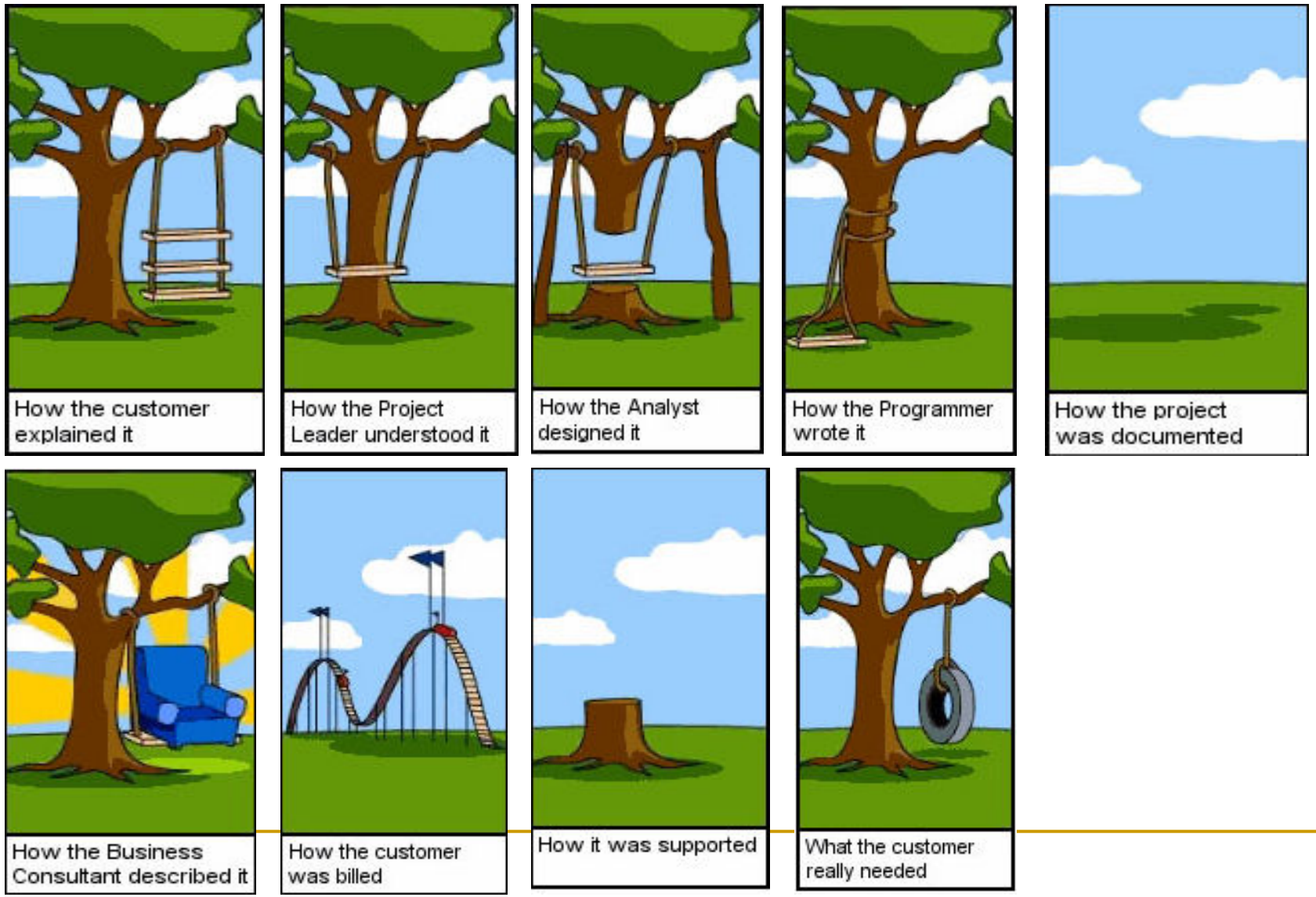
Design

Software Engineering



Design

- The goal is to design a *modular* solution, using the techniques of:
  - Decomposition
  - Abstraction
  - Encapsulation
- Here we focus on the Object-Oriented design which is a powerful modular design.
- In OO design we need to:
  - Identifying the objects in the problem
  - Identifying what actions can be performed on objects, and
  - Determining how the objects interact with each other

Modular design

- Modules must be loosely coupled
  - i.e. be as independent as possible.
  - interact with each other by knowing only *what* they do and not *how* they do it.
  - modifying a module will have no or little effect on the other modules.
  - makes it easy for different people to work on different modules at the same time.
- Modules must be highly cohesive
  - i.e a module should perform one well-defined task, rather than being an incoherent mass of functions.
  - This makes the modules (and therefore the solution) easier to understand and maintain.

Determining the objects (modules).

- How do we determine the objects?
  - It's the subject of entire texts and future course.
  - Here we provide an overview of two general techniques:
    - Abstraction:
      - Procedural abstraction.
      - Data abstraction.
    - Information Hiding

Abstraction

- Abstraction separates the purpose of a module from its implementation
  - Abstraction deals with "*What's*" not "*How's*".
  - The implementation details are ignored (or suppressed) to concentrate on the purpose of a module
- Procedural abstraction
  - Separates the purpose of a method from its implementation
  - Can be specified in pre and post conditions
- Data abstraction
  - Specifies *what* can be done to data in a data collection, not *how* the data is stored or *how* it is done
  - A collection of data specified in this way is called *an abstract data type* (**ADT**)

Procedural Abstraction

- Specifies each method clearly before you implement it.
  - What does the method assume? (Preconditions-input)
  - What actions it takes? (Post conditions-outputs)
- Easier design because you will be able to focus on high-level functionality without the distraction of implementation details.
- Anyone can use a method without “knowing” it’s implementation details.

Procedural Abstraction

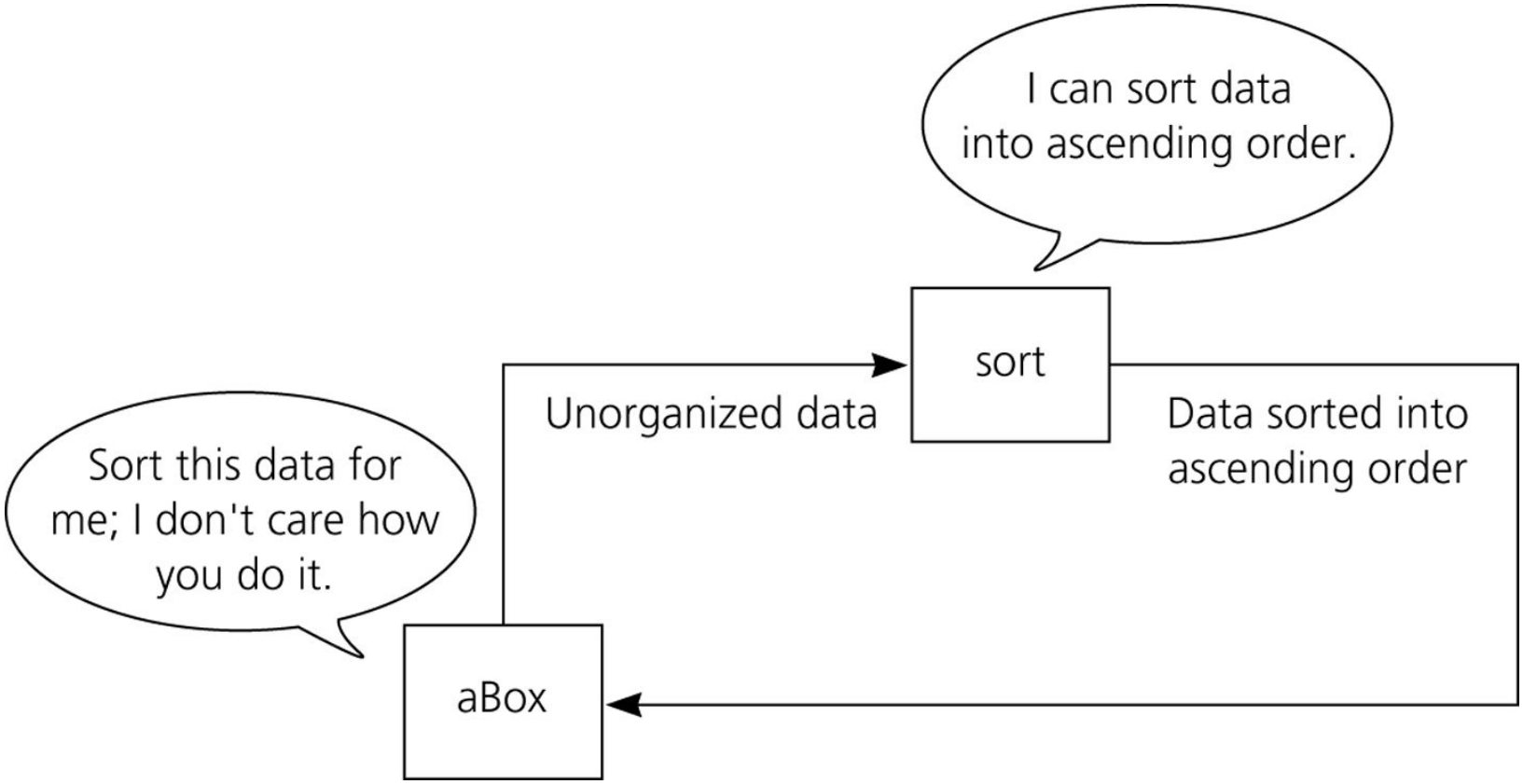


Figure 2-2

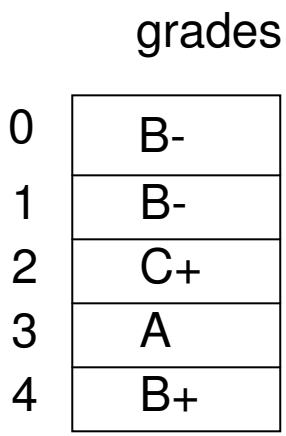
The details of the sorting algorithm are hidden from other parts of the solution.

Data Abstraction.

- Consider a collection of data and a set of operations.
- Data Abstractions focuses on *"what"* are the operations and ignore *"how"* they are implemented or *"how"* the date is actually stored.
- any module can use the Data as long as it knows *"what"* operations can be performed on the data.
- A collection of data specified in this way is called *an abstract data type* (**ADT**).
- Most of this course is about ADT's.

Data Abstraction – Example.

- You have always imagined an array in Java or C++ as follows:



- Is this really *how* an array is implemented?
- Even if you didn't know, you used it many times and had no problem.

Information hiding.

- Abstractions helps you to write specifications that describe each modules outside or public view.
- It also helps you identify details that should be hidden from the public view (these details are private to the module)
- Information hiding tells you not only to hide such details but also ensures that no other module can tamper with these hidden details.