Object Oriented Software Engineering

Sujit Chakrabarti

In object oriented programming, we learned the language constructs that form the basic toolset of a software engineer using object oriented concepts to develop a software. We spoke little about how these constructs can be used to develop great software systems.

What are good software systems? The ISO 9126-1 software quality model identifies 6 main quality characteristics, namely:

1. **Functionality.** The system should have capabilities or features that it promises. It should function properly and should have few or no bugs.

2. **Reliability.** The system should rarely fail.

3. **Usability.** A lay user should be able to interact with the system to get the specified job done.

4. **Efficiency.** The system should use all resources – time, memory, network traffic, DB transactions, power etc. – in an efficient way. We could probably consider humans involved in the development and use of a software software system as resources too. In that case, a system's maintenability, portability and usability also are related to efficiency.

5. **Maintainability.** The ease with which changes to the system can be made to add/enhance/remove features or remove bugs is called its maintenability.

6. **Portability.** The ease of making a system available to be used on a variety of computing platforms is called its portability.

Object oriented concepts aren't a sure-fire prescription to high quality software. They need to be harnessed judiciously to achieve this. Object oriented concepts are powerful tools. And like all powerful tools, they could be used to create harmful results, i.e. bad quality software. Hence, it's important to understand the principles used by expert software designers to best harness OO principles in building great software systems.

Here, we introduce the concept of object-oriented software engineering through two methods:

1. Recurrent themes in software design and solutions which have been useful in solving them. These solutions are called *design patterns*.

2. A complete case study of software design using object oriented concepts

1 Design Patterns

Let's go through a couple of examples of general software design problems: how they are described, how the solutions are proposed in a way that's applicable to all instances of that problem. Being recurrent and reusable, both the problems and their solutions are somewhat abstract. And it may require you to have the vocabulary of object oriented concepts ready on your fingertips. In particular: inheritance and polymorphism, containment, access specifiers, modules and such.

1.1 Composite

Let's return to our favourite example of shapes. Let's say that we wish to add another `Shape` class to the shape class family to represent shapes which contain other shapes as shown in fig. 1.

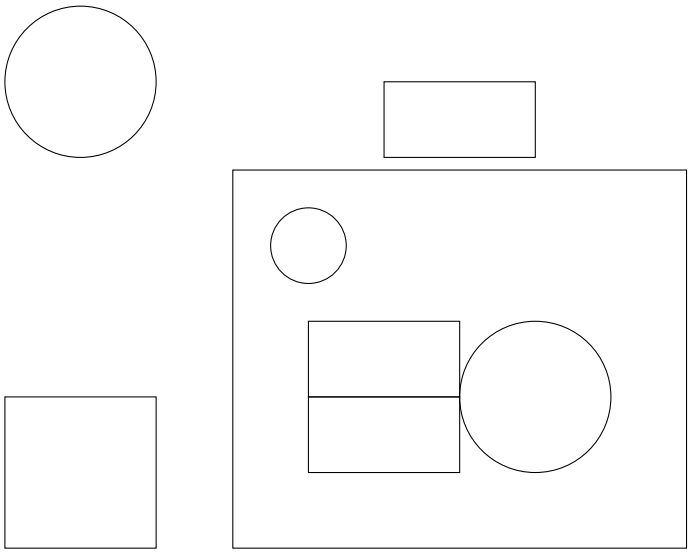

Figure 1: `Shape` family: (a) `Circle`; (b) `Rectangle`; (c) `Square`; (d) `CompositeShape`