4.1 Knowledge of Reusability

一级标题，二级标题，三级标题，四级标题，注，拓展，不懂

目录

4.1.1 Definition

Software reuse is the process of implementing or updating software systems using existing software components.（使用现有的程序元件实现或升级一个软件系统的过程）

4.1.2 Advantage

Reuse is cost-effective and with timeliness（可以较低成本和开发时间）

Increases software productivity by shortening software production cycle time and software developed faster and with fewer people.（降低软件生产周期并且更少人开发人员可以开发程序更加快速）

Does not waste resources to needlessly "reinvent-the-wheel"（不会浪费时间去重新造轮子）

Reduces cost in maintenance (better quality, more reliable and efficient software can be produced)（由于所使用的复用性较好的原件有着更高的质量，可以降低维护费用）

Reuse produces reliable software （经过充分测试，可靠、稳定）

Reusing functionality that has been around for a while and is debugged is a foundation for building on stable subsystems（复用的功能被其他人使用了一段时间并且进行了大量的测试）

Reuse yields standardization （标准化，在不同应用中保持一致）

标准化，使得其他人可以更好的维护你的代码

4.1.3 Disadvantage

生产一个复用程序原件需要大量的工作

Reusable components should be designed and built in a clearly defined, open way, with concise interface specifications, understandable documentation, and an eye towards future use.

The development cost of reusable components is higher than the cost of specific equivalents. This extra reusability enhancement cost should be an organization rather than a project cost. 开发成本高于一般软件的成本：要有足够高的适应性

注：这里所谈的 organization cost 是因为对于一般的复用程序的开发成本一般不会算入到某个项目之中，而是会算入到一个集体中。

对于一个复用程序的使用而言，也可能需要大量的操作

Reuse is costly: it involves spans organizational, technical, and process changes, as well as the cost of tools to support those changes, and the cost of training people on the new tools and changes.（可能需要对复用程序原件进行组织，技术，过程的改变，这需要花费资源，而且做出这些改变的工具需要花费资源）

可能需要对复用程序进行增删改等操作：

A. Extra functionality may have to be added to a component. When this has been added, the new component may be made available for reuse.
B. functionality may be removed from a component to improve its performance or reduce its space requirements
C. The implementation of some component operations may have to be modified.

与一个有特定场景开发的程序相比，复用程序的性能会差一些

Generic components may be less space-efficient and may have longer execution times than their specific equivalents

4.1.4The measure of reusability（复用性的评判）

The frequency in different application scenarios.（复用场合，复用频率）

The cost of reusing.（复用的代价）:

A. Cost to buy the asset and other mandatory libraries 搜索、获取
B. Cost for adapting and extending it 适配、扩展
C. Cost for instantiating it 实例化
D. Cost for changing other parts of the system that interact with it 与软件其他部分的互连的难度

Type Variation（类型可变）

Reusable components should be type-parameterized so that they can adapt to different data types (input, computation, and output);（一个复用性强的复用元件应当是类型可变的,）

A reusable module should be applicable to many different types of element, without requiring developers to perform manual changes to the software text.（可重用的模块应适用于许多不同类型的元素，而无需开发人员对软件文本进行手动更改）

In other words, we need a facility for describing type-parameterized modules, also known more concisely as generic modules（换句话说，我们需要一种工具来描述类型参数化的模块，也更简单地称为通用模块）

Implementation Variation（实现可变）

ADT 有多种不同的实现，提供不同的 representations 和 abstract function，但具有同样的 specification (pre-condition, postcondition, invariants)，从而可以适应不同的应用场景

Routine Grouping（功能任意分组）

提供完备的细粒度操作，保证功能的完整性，不同场景下复用不同的操作及其组合。其所有功能可以灵活组合，以满足不同的需求。

Representation Independence（表示独立）

使得 client 可以忽略掉具体的内部实现，实现隔离，信息隐藏。

Factoring Out Common Behaviors（共性抽取，排除相同行为）

减少重复性的代码书写，DRY 原则

注：将其中的 Type Variation（类型可变），Implementation Variation（实现可变），Routine Grouping（功能任意分组），Representation Independence（表示独立），Factoring Out Common Behaviors（共性抽取，排除相同行为）总结为 External observations of reusability。无法理解，但还是写上吧

拓展：A software asset with high reusability should：

A. Brief (small size) and Simple (low complexity) 小、简单
B. Portable and Standard Compliance 与标准兼容
C. Adaptable and Flexible 灵活可变
D. Extensibility 可扩展
E. Generic and Parameterization 泛型与参数化
F. Modularity 模块化
G. Localization of volatile (changeable) design assumptions 变化的局部性
H. Stability under changing requirements 稳定
I. Rich documentation 丰富的文档和帮助

4.1.4 Types of Reusing

Code or others

A reusable component may be code 最主要的复用是在代码层面

But benefits result from a broader and higher-level view of what can be reused. 但软件构造过程中的任何实体都可能被复用。Such as: Requirements(需求) Design and specifications (设计/规约 spec) Data(数据)Test cases (测试用例) Documentation (文档)

White box reuse or Black box reuse

White box reuse（白盒复用：源代码可见，可修改和扩展）

Reuse of code when code itself is available. Usually requires some kind of modification or adaptation 复制已有代码到正在开发的系统，进行修改

优点: You can customize the module to fit the specific situation, this allows reuse in more situations 可定制化程度高，可以适应不同的情况，软件效率高

缺点: You now own the customized result, so it adds to your code complexity. You requires intrinsic knowledge on component internals. 对其修改增加了软件的复杂度，且需要对其内部充分的了解，看个人

Black box reuse（黑盒复用：源代码不可见，不能修改）

Reuse in the form of combining existing code by providing some “glue”, but without having to change the code itself - usually because you do not have access to the code 只能通过 API 接口来使用，无法修改代码

优点: Simplicity and Cleanliness 简单，清晰

缺点: Many times it is just not possible 适应性差些

4.1.5 Ways to reusability（使用复用类的途径）

Source code reuse

Copy/paste parts/all into your program（直接截取代码到自己的项目中）

问题：需要进行矫正，需要进行选择，存在较大风险，需要 copy 的知识，需要途径获取代码。。。。。这都不是问题

Module-level reuse: class/interface

Characteristics

Source code not necessary, class file or jar/zip.不需要源代码，以 jar 等类型的包的形式进行使用

Documentation very important (Java API)（文档十分的重要）

Versioning, backwards-compatibility still problem（兼容性很难处理，一些版本老化，但

仍然需要保留）

注：Can use javap tool to get a class's public method headers 如果没有该包的文档，可以直接通过 javap（反编译器）工具得到其大致作用

Ways to use Module-level components

inheritance 继承

A. No need to put dummy methods that just forward or delegate work(无需放置仅用于转发或委托工作的虚拟方法)
B. Captures the real world better(可以与现实世界进行良好的对应)
C. Usually need to design inheritance hierarchy before implementation（通常需要良好的层次结构）
D. Cannot cancel out properties or methods, so must be careful not to overdo it（一旦继承无法取消）

delegation 委托

Delegation is simply when one object relies on another object for some subset of its functionality (one entity passing something to another entity) 一个对象依赖于另一个对象的部分功能

Explicit delegation（显式委托）:

passing the sending object to the receiving object.使用函数调用，以参数的形式

Implicit delegation（隐式委托）:

by the member lookup rules of the language 直接在需要对象中构造被需要对象

注：这是一种松耦合的复用关系，这种耦合程度明显低于继承

Ex:

A. Sorter can be reused with arbitrary sort orders
B. Comparators can be reused with arbitrary client code that needs to compare integers

Library-level reuse: API/Package

Libraries:

A set of classes and methods (APIs) that provide reusable functionality

注：The difference of libraries and framework

Libraries:: 开发者构造可运行软件实体，其中涉及到对可复用库的调用

Framework: Framework 作为主程序加以执行，在执行过程中调用开放者所写的程序

System-level reuse: Framework

Frameworks are sub-system design containing a collection of abstract and concrete classes along with interfaces between each class （框架：一组具体类、抽象类、及其之间的连接关系）

A framework is an abstraction in which software providing generic functionality can be selectively changed by additional user-written code, thus providing application-specific

software. 开发者根据 framework 的规约，填充自己的代码进去，形成完整系统（框架是一种抽象，其中可以通过其他用户编写的代码有选择地更改提供通用功能的软件，从而提供特定于应用程序的软件。）

将 framework 看作是更大规模的 API 复用，除了提供可复用的 API，还将这 些模块之间的关系都确定下来，形成了整体应用的领域复用

Whitebox frameworks（白盒框架）

Blackbox frameworks（黑盒框架）

注：这一部分的具体知识参见 4-2

注：Framework 作为主程序加以执行，执行过程中调用开发者所写的程序。

拓展：Frameworks differ from applications

The level of abstraction is different as frameworks provide a solution for a family of related problems, rather than a single one.（框架是为了解决一类问题而设计的，而不同于一个应用仅仅是为了一个问题）

To accommodate the family of problems, the framework is incomplete, incorporating hot spots and hooks to allow customization（为了解决一类问题，框架是不完整的，其中包括热点和挂钩待完成，这些都需要针对具体情况进行分析）