如何保证软件设计的质量

张灿 14126179

学习过软件工程的同学应该都了解软件危机一词，所谓软件危机是指落后的软件生产方式无法满足迅速增长的计算机软件需求，从而导致软件开发与维护过程中出现一系列严重问题的现象。软件危机主要有以下表现：软件开发进度难以预测、软件开发成本难以控制、用户对产品功能难以满足、软件缺少适当的文档资料以及软件产品质量难以保证与维护。因此，建立一套有计划、有系统的方法，来向管理层保证拟定出的标准、步骤、实践和方法能够正确地被所有项目所采用，从而保证软件质量就极为重要。软件质量保证的目的是使软件过程对于管理人员来说是可见的。它通过对软件产品和活动进行评审和审计来验证软件是合乎标准的。软件质量保证组在项目开始时就一起参与建立计划、标准和过程。这些将使软件项目满足机构方针的要求。

无论采取哪种软件生命周期模型，软件设计始终是其中的重要一环。软件设计是从软件需求规格说明书出发，根据需求分析阶段确定的功能设计软件系统的整体结构、划分功能模块、确定每个模块的实现算法以及编写具体的代码，形成软件的具体设计方案。软件设计就是大楼的设计图纸，它把许多事物和问题抽象起来，并且抽象它们不同的层次和角度。将问题或事物分解并模块化使得解决问题变得容易，分解的越细模块数量也就越多。在设计系统的时候，高质量的设计应该具有易理解、易实现、易测试、易修改等特征。

传统的软件设计要素包括软件的结构设计、数据设计、接口设计和过程设计。而面向对象方法是一种运用对象、类、继承、封装、聚合、关联、消息、多态性等概念来构造系统的软件开发方法。面向对象设计就是运用面向对象的方法进行系统设计。因此，从以上几个方面入手，采用面向对象的设计原则对于保证软件设计质量会起到重要的作用。

1 设计基础

要保证软件设计的质量就必须严格遵循设计基础。

1.1 抽象

抽象在最高层次上指的是使用待解决的问题领域内的术语描述的解决方

案。相对较低层次的抽象则更多的面向程序语言，最低层的抽象则是解决方案的可直接实现的方式描述。软件设计的每一个步骤都是对相应层次解决方案的抽象的逐步求精。

1.2 求精

求精又叫做逐步求精指的是通过程序细节连续细化来开发程序体系的策略。分步骤的对程序抽象进行分解直至成为编程语言的过程同时造就了程序的层次结构。在这一点上要对细节多做考虑，这也展示了求精实际上是个苦心经营的过程。

1.3 模块化

模块化是指软件可被分割为分别命名并可寻址的组件（也叫做模块），将模块综合起来又可以满足问题的需求的性质。" 软件的模块化是允许智能化管理程序的唯一属性。" 换句话说，当一个复杂问题被分解为一些小问题时会更容易被解决。

1.4 软件体系（架构）

软件体系涉及到程序的两个重要特性：1）模块的层次结构。2）数据结构。这源自于需求分析时将真实世界问题的含蓄定义与软件解决方案的要素关联起来的分割过程。当问题的每个部分通过一个或多个软件要素得到解决后，与问题的定义和解决相一致软件和数据结构的进化就开始了。这个过程代表了软件的需求分析和设计之间的位置。

1.5 数据结构

数据结构描述了单个数据间的逻辑关系。数据结构规定了数据的组织、访问方法、关联程度、和信息的选择处理。数据结构的组织和复杂性只受限于设计者的灵活性。唯一的限制就是经典数据结构的数量阻碍了更多的久经考验的结构出现。

1.6 软件程序

软件程序着重于处理每个模块的细节并必须提供一个精确的处理规范，包括事件顺序、准确的判定点、重复操作、甚至数据结构。软件的程序表现是分层的，处理方法应该包括其所有子模块的参考。

1.7 信息隐藏

模块应被设计和指定为包含在模块内部且其他模块不可访问的内容对其他模块来说是无需的。隐藏意味着有效的模块性能够通过定义一套独立的模块来实现，这些模块相互之间的通信仅仅包括实现软件功能的所必须的信息。将使用信息隐藏作为设计标准在测试或今后的维护期间需要修改系统时带来了最大的好处。

2 采用面向对象的设计原则

2.1 单一职责原则（SRP）

一个类最好只做一件事，只有一个引起它变化的原因。单一职责原则可以看作是高内聚、低耦合在面向对象原则上的引申。职责过多，可能引起它变化的原因就越多，这将导致职责依赖，相互之间就产生影响，从而极大地损伤其内聚性和耦合度。

2.2 开放封闭原则（OCP）

软件实体应当对扩展开放，对修改封闭。扩展即扩展现行的模块，当我们软件的实际应用发生改变时，出现新的需求，就需要我们对模块进行扩展，使其能够满足新的需求。修改封闭即是在我们对模块进行扩展时，勿需对源有程序代码和 DLL 进行修改或重新编译文件。在设计模块、类以及功能时，允许在不修改现有代码的前提下引入新的功能。采用开闭原则设计的软件具有可复用性强和可维护性好的优点。

2.3 里氏替换原则（LSP）

里氏替换原则是指：子类可以替换父类，并且出现在父类可以出现的任何地方。里氏替换原则解决了有关继承的原则，也就是什么时候应该使用继承， 什么时候不应该使用继承。只有当子类替换父类时，软件的功能不受到影响时，父类才能真正被复用，而子类也能够在父类的基础上增加新的行为。里氏替换原则是对开放封闭原则的补充。

2.4 依赖倒转原则（DIP）

依赖于抽象， 不要依赖于具体。依赖倒转原则要求抽象层是一个系统本质的概括，是战略性的设计。通常传统的系统设计是高层次的模块依赖于低

层次模块，即抽象层依赖于实现层。依赖倒转原则提出依赖关系应当倒转过来，即实现层依赖抽象层。抽象不应当依赖于细节，细节应当依赖于抽象。要求软件设计要面向接口编程，不要对实现编程。面向接口编程是指当程序在需要引用一个对象时，应当尽可能地使用抽象类型作为变量的静态类型。

2.5 接口隔离原则（ISP）

使用多个专门的接口要优于比使用单一的总接口。一个类对另外一个类的依赖性应当是建立在最小的接口上的。人们可以把接口理解成角色，一个接口就只是代表一个角色，每个角色都有它特定的一个接口。在面向对象设计中，恰当的划分角色和角色对应的接口是一个重要的工作。没有关系的接口合并在一起，形成一个臃肿的大接口，这是对角色和接口的污染。

3 采用设计模式

设计模式是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性。毫无疑问，设计模式于己于他人于系统都是多赢的；设计模式使代码编制真正工程化；设计模式是软件工程的基石脉络，如同大厦的结构一样。

3.1 抽象工厂模式

抽象工厂模式提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类，它适用于当有多个抽象角色时。抽象工厂模式可以向客户端提供一个接口，使客户端在不必指定产品的具体的情况下，创建多个产品族中的产品对象。根据 LSP 原则，任何接受父类型的地方，都应当能够接受子类型。因此，实际上系统所需要的，仅仅是类型与这些抽象产品角色相同的一些实例，而不是这些抽象产品的实例。换言之，也就是这些抽象产品的具体子类的实例。工厂类负责创建抽象产品的具体子类的实例。

3.2 建造者模式

建造者模式将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。它适用于当创建复杂对象的算法应该独立于该对象的组成部分以及它们的装配方式时；以及当构造过程必须允许被构造的对象有不同的表示时。

3.3 单例模式

单例模式是一种常用的软件设计模式。在它的核心结构中只包含一个被称为单例类的特殊类。通过单例模式可以保证系统中一个类只有一个实例而且该实例易于外界访问，从而方便对实例个数的控制并节约系统资源。如果希望在系统中某个类的对象只能存在一个，单例模式是最好的解决方案。

3.4 适配器模式

在计算机编程中，适配器模式（有时候也称包装样式或者包装）将一个类的接口适配成用户所期待的。一个适配允许通常因为接口不兼容而不能在一起工作的类工作在一起，做法是将类自己的接口包裹在一个已存在的类中。

3.5 代理模式

代理模式为其他对象提供一种代理以控制对这个对象的访问。在某些情况下，一个对象不适合或者不能直接引用另一个对象，而代理对象可以在客户端和目标对象之间起到中介的作用。

设计模式还有许多许多种，在不同的场合针对不同的情况采用相应的设计模式无疑使程序的架构脉络更清晰，更能保证软件设计的质量。

4 基于复杂性度量改善软件设计质量

设计模式和重构技术对于如何做出设计和代码改进有着一套系统化的方法，但是对于如何探测出系统中的不良设计却依靠开发人员的直觉和经验。如果能利用复杂性度量发现不良设计，再利用设计模式和重构技术改善设计和代码结构，则能够达到很好的互补效果。

耦合度和内聚度是衡量一个系统复杂性的重要指标，同时它们也是度量理论研究的重要内容。在软件设计中应该追求尽可能松散耦合(低耦合度)的关系，在这样的系统中可以研究、测试或维护任何一个模块，而不需要对系统的其它模块有很多了解。此外，由于模块间联系简单，发生在一处的错误传播到整个系统的可能性就小。因此，模块间的耦合程度强烈影响系统的易理解性、可测试性、可靠性和易维护性。内聚度应该是模块成分之间的相互关系。一个高内聚的模块就是具有一个基本功能的模块。一个内聚性很好的模块是很难被分离的。内聚标志着一个模块内各个元素彼此结合的紧密程度，也就是说，它度量单个模块所完成

的诸项任务在功能上相互关联的程度。软件设计力求做到高内聚，理想内聚的模块只完成一项相对完整的任务。通常，模块内的高内聚往往意味着模块间的松耦合。因此内聚和耦合都是进行软件设计的有力工具。

软件设计整个软件生命周期中十分重要的一环，保证软件设计质量对于保证软件质量有着十分重要的意义。在软件设计的过程中需要考虑方方面面，在把握整体架构的同时，尤其要注重一些细节的部分。软件设计相当于设计图纸、人生蓝图，我们在实际学习和工作中一定要重视保证软件设计的质量。