如何保证软件设计的质量

从阿兰·麦席森·图灵为了破解德军在二战时所设计的为无线通信加密的计算机器 Enigma 而开始设计并最终成功研制出 COLOSSUS 而为扭转二战局面立下汗马功劳的一刻起，计算机就已经注定会成为未来人类生活中越来越无法取代的部分，而随着冯诺依曼结构的提出，电脑也更加不可置疑的发展成为人类生活的必需品。而软件也就成为了让越来越多的人更好地控制和使用计算机的强有力的工具。然而在不断的发展过程中硬件的性能逐渐提高，软件的规模也逐渐扩大，无论在开发与使用的过程中，软件质量的优劣越来越突显出其重要性。

正如同人们生活中的衣食住行一样，软件质量的优劣能够直接反映在软件使用者的直观感受上，外国曾对用户的软件使用感受进行调查，从调查结果中发现仅有 23%的用户对自己使用的软件质量评价为良好或者较好，而 10%的用户认为自己使用的软件的质量很差，剩下的近 70%的人认为软件只能勉强使用，这也意味着软件质量问题已经成为了计算机界的严重漏洞，而如何解决这样的问题也成为了软件研究的一个新的重点。

当意识到问题，分析问题的原因成为了解决问题的关键。了解计算机软件具有的特征并分析影响软件质量的可能性因素能够为解决保证软件质量这一问题并研究出解决该问题的技术提供有价值的信息。计算机软件是没有形体的抽象事物，其中的内部结构十分复杂，有人称其为人类创造中最为复杂的实体，就如同人类的大脑一样。同时随着各种编程语言的出现以及每一个编程人员都具有不同的思考和编码方式，也使得软件具有多样性；而在开发过程中以及交付使用后常常会因各种原因需要对软件进行不同程度的修改以及修复，也使得软件具有极强的易变性。与此同时在软件开发前期对需求的说明和确认过程中常会因为表述不清晰，理解有误，沟通问题使得开发人员对用户的需求了解不透致使在开发过程中用户再三要求变更需求，使得软件需求难于把握。这些特征都给开发高质量的软件带来极大困难。通过研究计算机软件的特征，总结软件开发的经验教训，可归纳影响软件开发过程质量的因素有开发人员本身的素质能力，对软件需求分析的透彻程度，软件开发过程中各个环节的衔接是否存在问题，测试具有局限性，以及在整个开发过程中对开发流程、开发规范以及质量标准的把控与实施是否到位。

通过对引起问题的因素进行分析发现，如果能将软件分成模块，如同零件一般进行复用会使软件开发工作进行得更快、更省、更好，而复用的本质就是共享复用构件系统。但被复用的构件必须是成熟的，可靠的，通用的。但是，无论使用多么高明的避开错误技术，也无法做到完美无缺，这就需要采用容错技术，以使错误发生时不致影响系统的特性，对用户的影响也能限制在容许范围内。具有容错功能的软件，在一定程度上，对自身错误的作用有屏蔽能力，也能从错误状态自动恢复到正常状态，发生错误时仍能完成预期的功能。而软件的复用技术对于被复用构件的要求非常高，因此在软件设计的过程中就应该很好的进行架构设计以及高内聚低耦合的结构设计和类设计，在软件过程中能够将软件设计的质量提高对整个软件开发结果的质量保证起着关键的作用。

在面向对象软件设计过程中，我们需要遵循很多原则和设计模式，这些原则和设计模式不但会对整个软件的架构设计起到极大的优化作用，同时还会对代码的重用性有很大的提高，同时也会对整个软件的质量有很大的提高。

在需求不断变化导致软件开发的效率降低是软件开发中很常见的导致软件质量低下的因素，因此如何通过最小的设计改变就可以应对多变的设计需求成为在面向对象设计中设计者极为关注的一个问题，为此面向对象设计者在不断的探索中总结了一些面向对象的设

计原则，这些原则指导面向对象的设计和开发。人们可以通过判断软件设计是否符合这些设计原则来评价面向对象软件设计的质量优劣。

1. 单一职责：在设计过程中应该有一个核心思想即是：“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低耦合、高内聚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. 开闭原则：面向对象设计中“可复用设计”的基石，是面向对象设计中最重要的原则之一。开闭原则指：“软件实体应当对扩展开放，对修改关闭”。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。面向对象设计时采用接口封装机制、抽象机制和多态技术实现开闭原则。

3. Liskov 替换原则（LSP）：Liskov 替换原则是面向对象设计的基本原则之一。是指“任何父类可以出现的地方，子类一定也可以出现”。这个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。Liskov 替换原则是对“开-闭”原则的补充。

4. 依赖倒转原则（DIP）：依赖倒转原则指要“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；而实现层与具体实现技术相关，是战术性的层次。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。

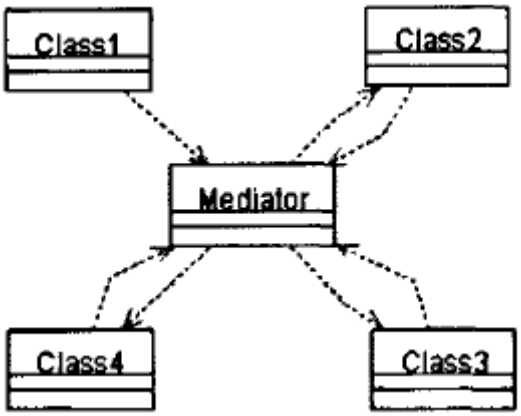
5. 接口隔离原则（ISP）：接口隔离原则是指：“使用多个专门的接口比使用单一的总接口要好”。一个类对另外一个类的依赖性应当是建立在最小的接口上的。人们可以把接口理解成角色，一个接口就只是代表一个角色，每个角色都有它特定的一个接口。在面向对象设计中，恰当的划分角色和角色对应的接口是一个重要的工作。没有关系的接口合并在一起，形成一个臃肿的大接口，这是对角色和接口的污染。

6. 合成/聚合复用原则（CARP）：在面向对象设计中，可以通过合成/聚合和继承这两种方式实现代码的复用。合成/聚合复用原则是指：“尽量使用合成/聚合，而不是使用继承”。因为继承复用破坏类的封装性，继承将父类的实现细节暴露给子类。如果父类发生改变，那么子类的实现也不得不发生改变。而基于合成/聚合的复用所需的依赖较少，复用可以在运行时间内动态进行，新对象可以动态地引用与成分对象类型相同的对象。

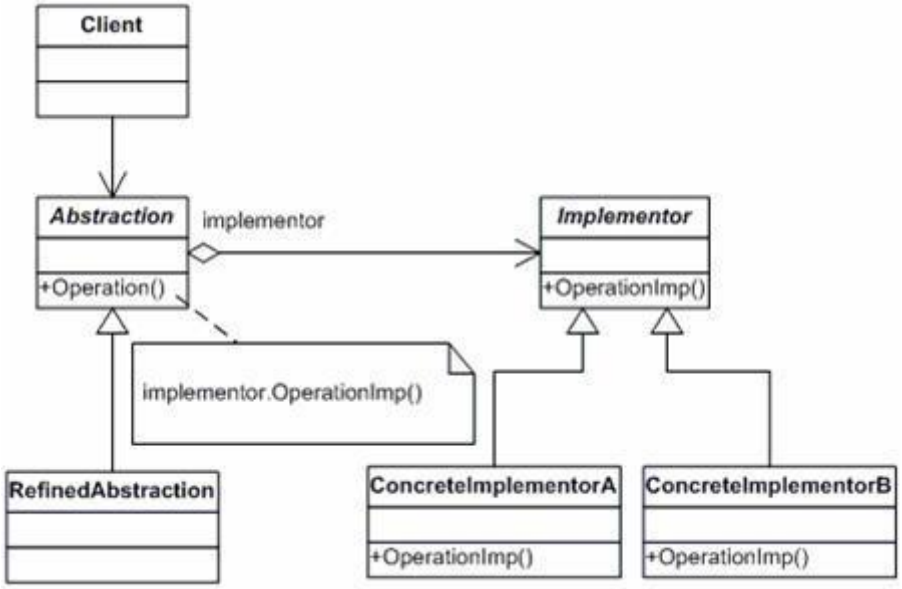
而在不断的研究与实践过程中，形成了一些固定的设计模式，这些相对成熟的设计思想被总结抽象出来，也就形成了现在的设计模。设计模式是提高软件设计质量和软件开发效率的一个重要途径，设计模式描述了软件设计过程中某一类常见问题的一般性解决方案。面向对象设计模式描述了面向对象设计过程中、特定场景下、类与相互通信的对象之间常见的组织关系。其中最著名的面向对象设计模式是 GoF 提出的 23 种经典设计模式。

相比于着眼于整体架构并且具有普适性的设计原则而言，设计模式更倾向于针对某一问题而提供的一种成熟有效的解决方案。模式提供的解决方案不仅能够很好的满足功能性的需求，而且能够很好的满足非功能性的需求，因此模式提供了一种可能的方式来处理非功能的需求。换句话说。模式除了针对某个问题提供基本的、功能性的解决方案外，还使得此方方案获得高质量的、非功能性的提升。每一个模式都有两方面的意图：功能性的意图和非功能性的意图。不同的设计模式对于不同的需求所产生的效果有很大的区别。

Mediator 模式的目的就是为了降低耦合，使得交互集中起来，各对象不需要显示地相互引用。这能够使得单独的类更容易分离并重用，比起重用一组类更加灵活。



桥接模式(bridge)用来将抽象部分与实现部分分开，使他们能够独立变化，并且动态结合。使用"对象间的组合关系"解耦了抽象和实现之间固有的绑定关系，使得抽象和实现可以沿着各自的维度来变化。

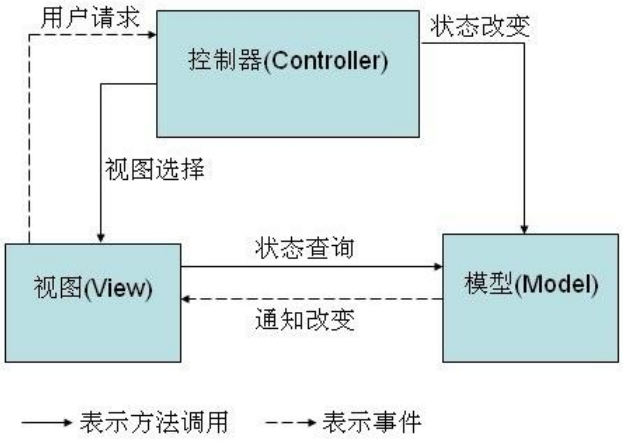


除些之外，还有更多的面向对象设计模式。但是真正能理解并能熟练运用这些模式比较困难，无经验的程序员因不了解设计模式，或不能真正掌握设计模式的精髓，出现软件设计模式缺乏或乱用设计模式，导致设计过度现象。反而为软件质量保证造成了相反的效果，因此在使用设计模式之前遵守设计原则以及遵循需求才是最基本的。

同时在软件设计的过程中通过体系结构模式对整个项目的整体架构进行组织，它是一种高层的抽象，着眼于大规模的组件和子系统、系统全局特性和机制，。它提供一套预定义的子系统，规定他们的职责，并包含用于组织他们之间关系的规则和指南。体系结构模式可作为具体软件体系结构的模版，它是规模较大的模式，它们规定一个应用的系统范围的结构特性。以及对其子系统的体系结构施加的影响。选择体系结构模式是开发软件系统的基本决策。

模型-视图-控制器模式(MVC 框架)就是一种软件体系结构设计模式，这种设计模式通过将整个软件项目进行整体划分，将用户界面，业务逻辑与底层控制分离，弱化了业务逻辑接口和数据接口之间的耦合，使逻辑与呈现相分离，从而让表现层更加独立和更富于变

化，增强了代码的可维护性与扩展性。使得整个项目的代码结构清晰，耦合性低，重用性高，也提高了项目的可维护性并且有利于软件工程化管理。



软件设计的好坏对整个软件的可用性、可移植性、可扩展性等等都有很重要的影响，因此对软件设计的质量的重视也是对整个软件项目质量的重视，保证了软件设计的质量，才能够为整个软件质量提供最基础也最重要的保证，而与此同时对软件测试的质量也应该有所保证，才能在后期对软件成果的测试检验中再一次提高软件的质量。

在软件开发过程中，每一个环节都对最终的成果有着极其重要的作用，只有在每一个环节都遵循开发的原则，才能改善如今软件质量的现状，推动计算机软件更好的发展。