链滴

[阅读] 敏捷软件开发 —— 敏捷设计（一）

作者：lizhongyue248
原文链接：https://ld246.com/article/1552738594515
来源网站：链滴
许可协议：署名-相同方式共享 4.0 国际 (CC BY-SA 4.0)

<img src="https://ld246.com/images/img-loading.svg" alt="" data-src="https://b3logfile
com/bing/20180322.jpg?imageView2/1/w/960/h/540/interlace/1/q/100">
敏捷的开发能够让我们开发出来的软件变得易用且优秀。如果敏捷性是以最小增量的方式构建软
，那么我们如何去设计软件呢？《敏捷软件开发》第二部分就是对敏捷设计的学习，与第一部分不同
是，他注重的更多是设计而不是代码的书写与实践，所以相比起来这章是对思想的一种锤炼。
<h2 id="回顾">回顾</h2>
上周真正的体验了一次 XP 的实战，里面学习到了不少。但是总会有一个疑问，因为多次发现许
重构后提出来的私有方法都是一句话代码，都是 <code>return</code> 直接返回回去了，除了更
读以外似乎没有太多明显的好处了。而在这章开篇就提到了这个问题：
<blockquote>
如果以最小增量的方式构建软件，难道不是打着重构的旗号，而实际上却导致了去多无用的代码
片和返工吗？
</blockquote>
所以也激起来了我的兴趣。
<h2 id="开篇">开篇</h2>
在开始，我们必须知道如何衡量一个软件设计的优劣，因为这样才能够更好的了解如何去评判一
软件。而后则是一写开发的原则，一些经历了时间沉淀下来的经验，能够让我们软件更加优雅的原则

<h2 id="什么是敏捷设计">什么是敏捷设计</h2>
<blockquote>
在按照我的理解方式审查了软件开发的生命周期后。我得出一个结论：实际上满足工程设计标准
唯一软件文档，就是代码清单。
</blockquote>
曾经自己以为的设计，应该是有一个设计图，对每一个类详细的标识出一些方法这些。但是书中
设计，则有很大不同：
<blockquote>
软件项目的设计是一个抽象的概念，他和程序的概括形状、结构以及每一个模块、类和方法的详
形状和结构够管，可以使用许多不同的媒介 uq 描绘他，但是他最终体现为源代码。最后源代码就是
计。
</blockquote>
的确，回想起来，所有的文档都没有源代码具有说服力。文档作为辅助，代码作为主体，相辅相
。
评判一个软件的优劣，往往通过下面的特征：

僵化性：进行一个改动的时候，程序的其他的许多地方都有可能出现问题。
牢固性：设计中包含了对其他地方有用的部分，但是却无法把这些部分从系统中分离出来，或者
要巨大的风险和代价。
粘滞性：

软件：面临一个改动的时候，开发人员往往会有多种改动的方法。其中一些方法会保持设计，而
外一些方法会破坏设计。而当那些可以保持系统设计的方法比那些波坏设计的方法更加难以应用时，
表明设计具有高的粘滞性。
环境：当开发环境迟钝、低效时，就会产生环境的粘滞性。

不必要的复杂性：如果设计中包含当前没有用的组成部分，他就含有不必要的复杂性。
不必要的重复：复制——粘贴 的重复性代码。
晦涩性：模块难以理解。

我们面对一个好的设计，最大的挑战莫过于软件的需求在不断的变化。所以，一个好的团队，必
要找到一种办法，使得设计对于这种变化具有弹性，并且应用一些实践来防止设计腐化。并且最为重
的是，敏捷的团队依靠变化来获取活力，他们几乎不会预先的设计，保持系统设计尽可能的干净、简
。通过许多的单元测试，保持设计的灵活性、易于理解性。
书中为我们举了一个 copy 程序的例子，展现了需求的需求的变化带来的设计的退化</

trong>。然而这就是软件开发最重要的事实之一：需求总在变化。不过，再一强调，对于敏捷开发团队来说，如果我们软件的设计由于需求变化了而退化，那么我们就是敏捷的。
从软件开发的三个方面来发现要做什么。

遵循敏捷实践去发现问题。
应用设计原则去诊断问题。
应用适当的设计模式去解决问题。

软件开发的三个方面间的相互作用就是设计。
敏捷设计是一个过程，不是一个事件。敏捷开发人员不会对一个庞大的预先设计应用些原则和模式。相反，这些原则和模式被应用在一次次的迭代中，力图使代码以及代码所表达的设计持干净。
<h2 id="单一职责原则-SRP-">单一职责原则（SRP）</h2>
<blockquote>
只有佛自己应当担负起公布玄妙秘密的职责
</blockquote>
曾经我们在写保龄球实例的时候，最后的重构中，将一个 <code>Game</code> 给分离到了个类中，一个负责游戏，一个负责积分，各司其职。这就是单一职责。
如果一个类承担的职责过多，那么就会产生耦合，这样的设计往往会造成一个职责的变化可能会响或削弱其他职责的能力。
这里的 职责 怎么理解好呢？应该将它定义为 “变化的原因” 。如果有多于一个机去改变一个类，那么这个类就不符合单一职责原则。这个时候我们就要进行分离。
但是万事皆有特例，有时候可能会由于一些客观原因造成我们没有办法，我们就要采用其他的方进行处理，比如抽象。书中举例了一个类实现两个接口的例子，虽然在类中多于一个职责，但是通过离他的接口，也实现了解耦。
SRP 最简单，也是最难运用的。
<h2 id="开放--封闭原则-OCP-">开放——封闭原则（OCP）</h2>

“对于扩展是开放的”：这意味着模块的行为是可以扩展的，当应用的需求改变时，我们可以对块及性能扩展，使其具有满足那些改变的新行为。换句话说，我们可以改变模块的功能。
“对于更改是封闭的”：对模块行为进行扩展时，不必改动模块的源代码或者二进制代码。

这个还是比较好理解的，最好的实现就是抽象。例子是 C++ 的有点不太明白。书中有一点我非同意：为了防止软件背着不必要的复杂性，我们会允许自己被愚弄一次，这意味着我们在最初的代码写时，假设变化不会发生，当变化发生时，我们就创建抽象来隔离发生的同类变化。
始终遵循该原则才能从面向对象技术中持续地获得最大的益处，例如：可重用性和可维护性。而确切的说，它需要程序我们更专注于将抽象技术应用到程序中那些趋于变化的部分上。
<h2 id="总结">总结</h2>
这周感觉难度不是很大，20 页很快就过了，不过自己没有动手实现。后面应该找事件实践一下而且这周可能是由于电脑系统的原因，自己静不下来好好的看书。要好好调整自己的状态才行。