# 简单设计

## 问

极限追问 011

简单设计怎么做？

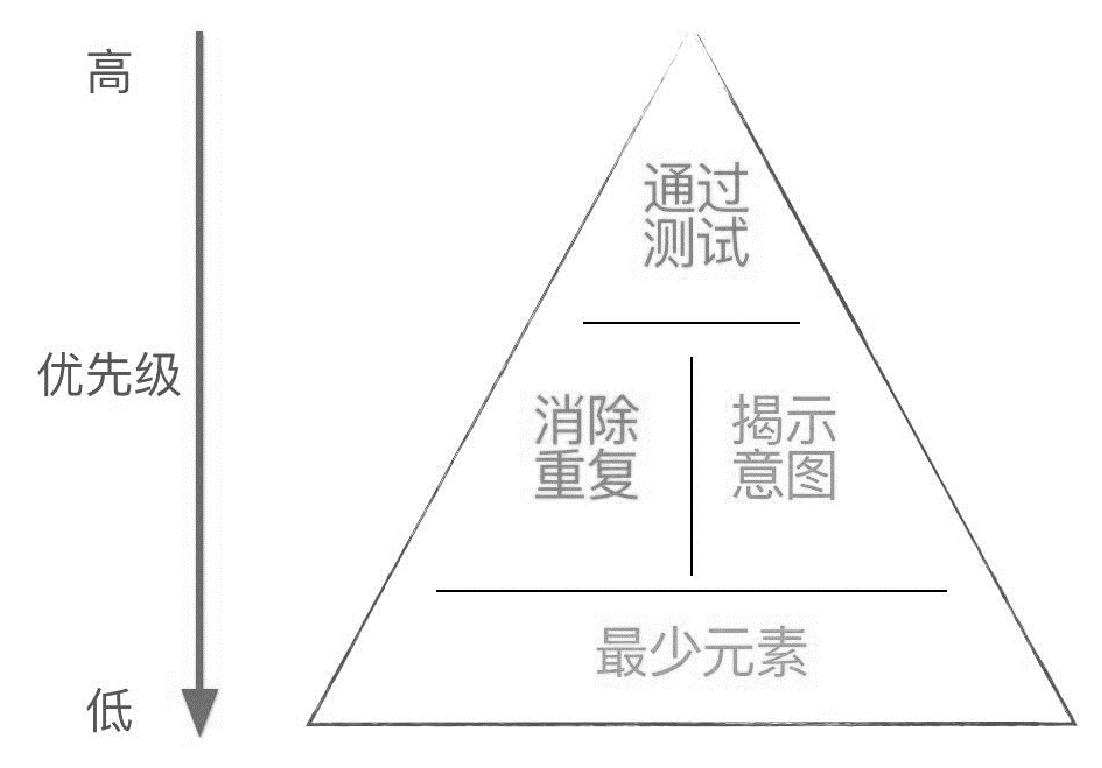
很多人说极限编程没有做软件设计，也有很多人面对问题不知道应该怎么开始写下TDD的第一个测试。

极限编程需要做设计吗？做什么程度的设计？

拿到用户故事，如何转化成TDD的测试？

## 答

### 小人物1）



这个是网上之前看的过的简单设计的图片，说个我学习到的简单设计原则，在设计和重构决策中我们应关注需求实现、可理解性、易修改性 和复杂度四个维度，针对这四个维度，不断问自己四个问题：

测试都通过了吗？

代码理解起来困难吗？

代码存在重复逻辑吗？

能够减少代码元素吗？

我理解所谓简单设计主要是针对编码实践层面的，与其它设计方法和理论配合使用效果会更好

XP 的简单设计并不是不要设计，而且需要认识到设计不是可以在编码之前一次性完成的，那样只是建立在“情况不会发生变化”或者“我们可以预见所有的变化”之类的前提的基础上，而这种前提往往是不成立的。

用户故事如何转化为TDD？ 我现在还是倾向于按照Kent Beck的那种，先理解需求，列待办，同步梳理思路，然后选择自己想的最清楚的部分入手开始TDD, 开发过程中又会不断加深对需求和实现思路的理解，这个过程再不断问自己简单设计的几个问题，不断重构，循环往复，直到故事完成。

应该改成在自己已有经验和条件情况下，一次性把事情尽可能做好，然后还能不断改进，这就是极限编程里说的时间的价值：总有一种更好的方式来设计这个软件，即使是同一领域的软件，随着经验和设计理念的进步，也总会出现更好的设计方式，我们只需要设计出当下足够的东西来得到反馈，然后利用反馈改进我们的设计，不断循环。

所以要权衡，如果有特别明确的可预见的扩展需求，那这个前期设计是必要且值得的，比如大家都用Jira，jira的扩展机制肯定不是TDD出来的，但是这个扩展机制实现后会大大加快后续开发效率，总体来说应该是提高了开发效率而不是只增加了工作量

如果只是类和对象层面的扩展，那这个实际上在开发过程中通过重构会更合适，因为这些具体对象的扩展很难开始就想清楚，这时候开始就投入工作量的投入产出比会很低

//---

我之前做过一个系统，需要兼容所有主流通信设备的日志文件格式，有文本有xml json还有二进制，在遇到具体需求之前我们自己根本不知道会遇到什么奇葩格式，这种情况只能不断通过重构来兼容未来的可能性，一次重构解决一批问题，最后实现基于配置就可以增加各自奇葩文件的处理逻辑，这个系统就是典型的我知道会有扩展需求，但是我真的没法一次搞定，如果开始就让我设计我肯定是设计不出来的。

领域知识掌握的不深刻 自然设计也做的不够深刻。。挖坑是难免的。。

之前我写代码比别人快，我总认为技术比别人好，最后发现实际是因为我的领域知识比别人理解的深，我是站在领域角度在实现，别人是在理解我的实现思路，然后去实现，这个效率能差出好几倍去

Kent Beck一个视频里也说他给别人介绍极限编程时候 都是从 价值观 原则 实践三个层面介绍，如果只介绍实践人们难免陷入教条里去而忽视了价值观和原则，而那个才是最重要的，就TDD来说，想实现的就是测试先行，具体实现手法实际是很多的，没有可以放之四海而皆准的做法

可以看看这个，kentbeck 自己讲的极限编程发展史

<https://youtu.be/cGuTmOUdFbo>

### **小人物2）**

tdd需要的是用户行为层级的用户故事 所以只要符合的就不需要设计 不符合的就要分析成为该层级的用户故事

### **小人物3）**

TDD：测试驱动开发，也可以把最后一个D理解为Design,设计驱动设计，那么这种设计就是JIT设计，刚刚好符合需求，消除了浪费。

但要达到这种“驱动”型设计的前提就是“重构”，通过测试先行，定好输入、输出，完成功能后通过重构，优化架构以达到JIT————功能与架构设计的平衡，这种平衡我称之为：动态平衡，不是稳态的平衡，是面向变化的平衡，准备随时接受变更，但是，这种平衡的先决条件仍然是“重构的功力”。

回到问题，我们做TDD时是否需要设计，这个取决于我们对于任务的把握程度，如果把握程度高，其实不需要单独的“设计环节”，这时无需设计，其实是动用了脑中已有的经验或知识；

另一方面，如果对于相对大型的任务，或影响较大之类的任务把握程度不高，最好需要有设计，但这种设计只需要是概要设计就可以了，能大体帮助自己理清思路即可，不需要做详细设计，做详细设计又回到传统的模式中去了，而且详细设计有较大概率会抑制变更，做简单设计到底多简单，有一个说法是：不超过一张A4纸，理清主体功能模块，剩下的就通过TDD来“驱动”了。

绕了一圈还是回到了“重构”，如果重构能力较高，如果能够“一次把事做对”，才能最大限度的消除浪费。

关于第一个案例，收到任务卡片后，分析用户使用场景，依据场景写出相应的验收条件acceptance criteria，每一个A/C就可以转换成若干个测试用例。

另外一种做法是直接用需求实例化来进行需求分析整理，每一个实例天然就可以转换为测试用例。

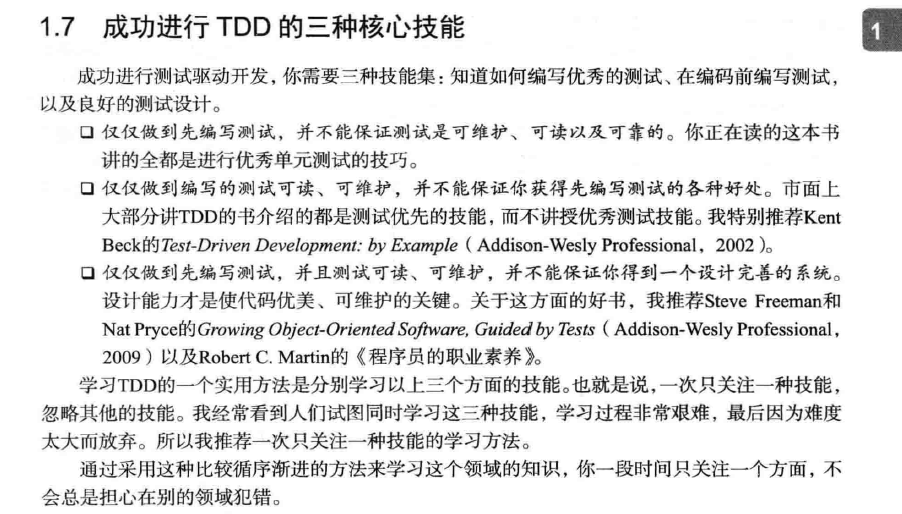
“一次把事做对”是一种希望达到的情况，我们经常会想着说我这个任务先做出来，然后重构，这个事情，通常是不会回来重构的，因为每个人都有一堆的任务在身，尽量吧

单元测试的代码更应该保持简单，如果单元测试中有方法嵌套，有业务逻辑转换，那维护单元测试本身一样要花费大量精力

### **小人物4）**

有一种做法，叫做“只被第一颗子弹击中”。意思就是：对于当前的需求，平铺直叙的去做，不考虑灵活与易扩展；但是当需求变更时，就需要重构并创建抽象来应对这次的变化以及未来的同类型变化。这样相当于在效率和扩展性之间寻求了一个效率平衡点

### **小人物5) 单元测试的艺术**



<https://book.douban.com/subject/25934516/>