01-编写简单脚本

2014年10月17日 星期五　　15:48

```
#编写完代码后，要提权
#chmod 777 ./1.sh
#shell中获取当前时间 $(date +%Y-%m-%d)
#shell中给变量赋值 d=$(date +%Y-%m-%d)
#shell中输出变量  echo $d

#自动关机
#sudo shutdown -h +80
#sudo shutdown -h now
#sudo shutdown -h 12:00

#取消自动关机
#sudo kill pid(shutdown的时候生成的pid)

cd Desktop

d=$(date +%Y-%m-%d)
mkdir $d
cd $d
mkdir code
mkdir 练习
open ./
```

02-显示隐藏文件

2014年10月21日 星期二　　16:32

显示：defaults write com.apple.finder AppleShowAllFiles -bool true
隐藏：defaults write com.apple.finder AppleShowAllFiles -bool false

取消隐藏文件夹：chflags nohidden ./abc
设置隐藏文件夹：chflags hidden ./abc

03-推荐视频

2014年10月21日 星期二　　20:14

一席：方励《感谢你给我机会上场》
http://v.youku.com/v_show/id_XNzg2MDQyNzYw.html?ev=1&from=y1.1-2.10001-0.1-1

乔布斯：访谈1990
http://v.youku.com/v_show/id_XNzkzMzgyMjY4.html?f=22906104&ev=1&from=y1.1-2.10001-0.1-1

【一席】大冰《赶着音乐放牧》
http://v.youku.com/v_show/id_XNDkxMDM5NDE2.html?f=22306468

Adobe联手微软PS触屏化 触控造就未来
http://v.youku.com/v_show/id_XNzk4NDUyMTQ0.html?f=22926986&ev=4
&from=y1.1-2.10001-0.1-1

迈克·马塔斯:新一代数字图书
http://v.youku.com/v_show/id_XNzkzOTQxMzY4.html?firsttime=241

视频：《平凡之路MV》一路向西517-318骑行川藏2013_HDMOV
http://v.youku.com/v_show/id_XNzYwOTA3NjY0.html?from=y1.1-2.20001-0.1-1

04-开启ftp

2014年10月23日 星期四　　下午5:39

开启ftp

sudo -s launchctl load -w /System/Library/LaunchDaemons/ftp.plist

关闭FTP Server

sudo -s launchctl unload -w /System/Library/LaunchDaemons/ftp.plist

05-玩游戏学编程

2014年10月25日 星期六　　23:02

http://codecombat.com/

Up for a fun challenge? Want a sweet job? If you can **beat** Gridmancer, our first developer challenge level, we'll **help you find a programming job** in the San Francisco Bay Area. This is a hard level, so if you can do it, you're probably qualified for some amazing opportunities.
In Gridmancer, you write an algorithm to cover the empty floor spaces in as few rectangles as possible. This is a real problem we had to solve in CodeCombat for both our pathfinding system and for our collision handling optimization system. Our implementation does it with 33 rectangles, but it's possible to do it with fewer--many of our playtesters actually beat our implementation, and a couple found the optimal solution of 29. (We're going to steal one of their algorithms to make CodeCombat itself faster.)

如果说调试是清除bug的过程，那编码是放置bug的过程

目前只有两种方式能写出没有bug的程序，但是第三种方式才能达到预期的效果

http://www.cnbeta.com/articles/347021.htm

06-10句编程箴言 每个程序员都应该知道

2014年10月25日 星期六　　23:41

摘要：所谓谚语，就是用言简意赅、通俗易懂的方式传达人生箴言和普遍真理的话，它们能很好地帮助你处理生活和工作上的事情。也正因如此，我才整理了10句编程谚语，每位开发人员都应该铭记他

导读：原文作者Kevin Pang在kevinwilliampang.com发表一篇《10 Programming Proverbs Every Developer Should Know》。译文由伯乐在线整理编译成《10句编程箴言 每个程序员都应该知道》。文章内容如下：

所谓谚语，就是用言简意赅、通俗易懂的方式传达人生箴言和普遍真理的话，它们能很好地帮助你处理生活和工作上的事情。也正因如此，我才整理了10句编程谚语，每位开发人员都应该铭记他们，武装自己。

1. 无风不起浪

别紧张，这也许只是一场消防演习

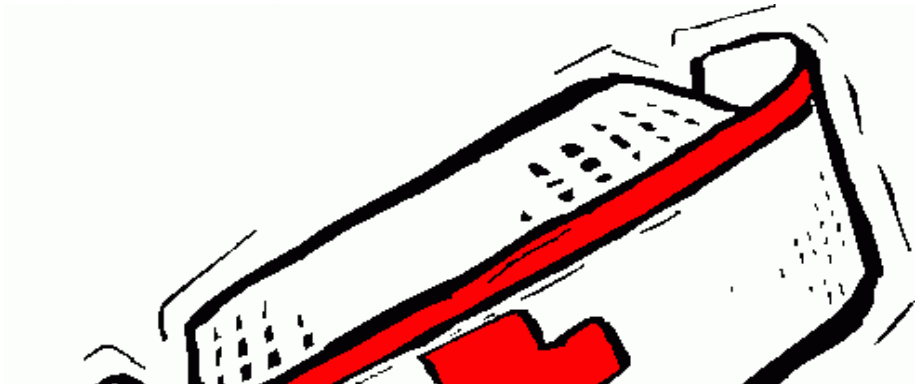
代码设计是否糟糕，从某些地方就可以看出来。比如：

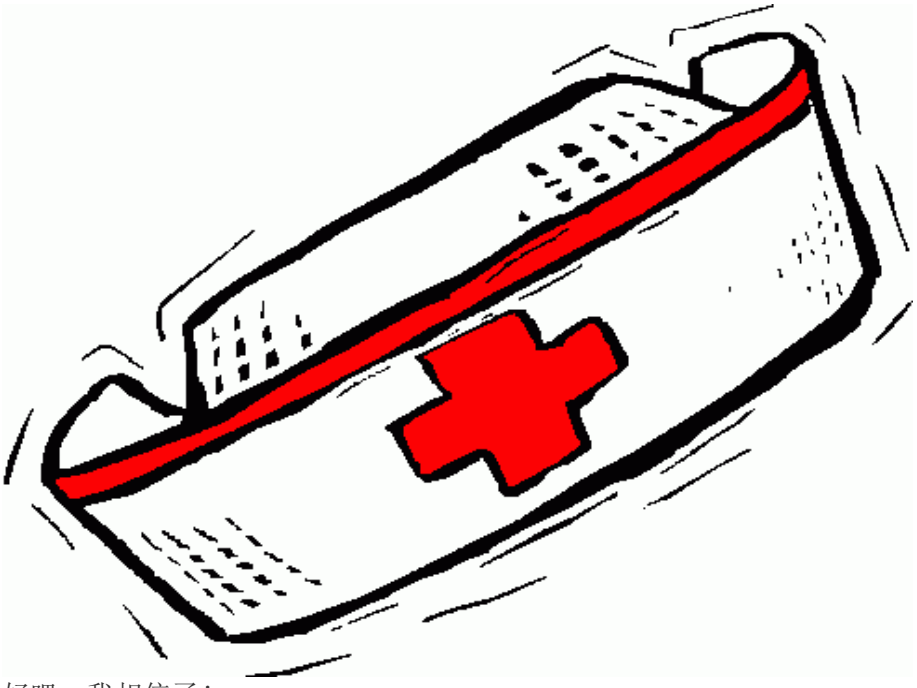
a. 超大类或超大函数
b. 大片被注释的代码
c. 逻辑重复
d. If/else嵌套过深

程序员们通常称它们作代码异味(Code Smell)，但是就我个人认为“代码警报”这个名字更为合适一些，因为它有更高的紧迫感的含义。根本问题处理不当，终将引火烧身。

译注：Code Smell中文译名一般为“代码异味”，或“代码味道”，它是提示代码中某个地方存在错误的一个暗示，开发人员可以通过这种smell（异味）在代码中追捕到问题。

2. 预防为主，治疗为辅





好吧，我相信了！

20世纪80年代，丰田公司的流水作业线因为它在缺陷预防方法上的革新变得出了名的高效。每个发现自己的部门有问题的成员都有权暂停生产。这个方法意在宁可发现问题后马上暂定生产、解决问题，也不能由其继续生产而导致更棘手且更高代价的修复/更换/召回后的问题。

程序员总会做出生产率就等同于快速编码的错误臆断。许多程序员都会不假思索地直接着手代码设计。可惜，这种Leeroy Jenkins式鲁莽的做法多会导致软件的开发过程变得很邋遢，拙劣的代码需要不断的监测和修改——也可能会被彻底地替换。最终，生产率所涉及到的因素就 不仅仅是写代码所消耗的时间了，还要有调试的时间。稍不留神就会“捡了芝麻丢了西瓜”。（因小失大。）

译注：Leeroy Jenkins 行为：WOW游戏中一位玩家不顾大家独身一人迎敌，导致灭团。

3. 不要孤注一掷 （过度依赖某人）

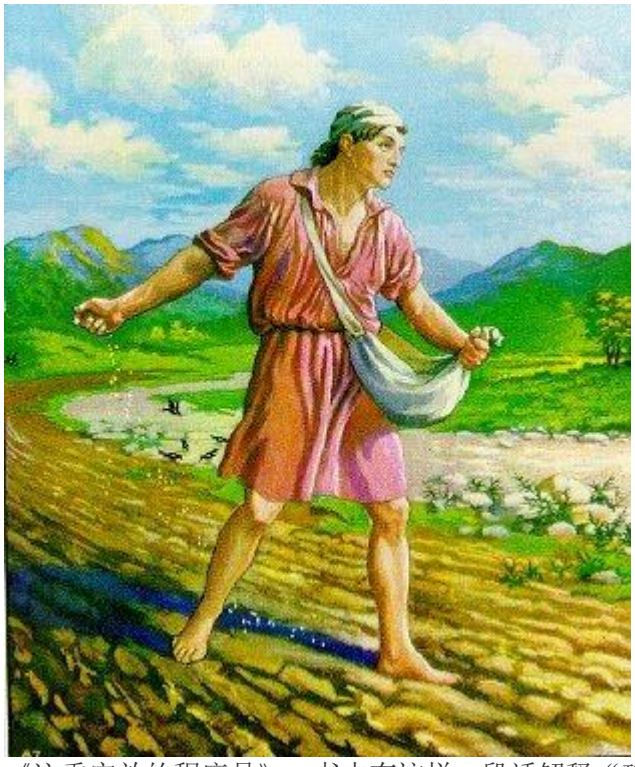
一个软件开发团队的公共要素（bus factor）是指那些会影响整个项目进程的核心开发人员的总数。比如某人被车撞了或某人生孩子或某人跳槽了，项目可能就会无序，甚至会搁置。

译注： bus factor 即指公共要素，比喻了开发过程中的一些共同因素。如果挤上 bus 的 factor 越多，bus 就越不稳定，所以要控制好 bus factor ，以免问题发生。

换句话说，如果你的团队突然失去了一个主力成员，你会怎么办？生意依旧进行还是戛然而止？很不幸，大多数软件团队都陷入了后一种情况。这些团队把他们的开发员培养成了只会处理他们自己专业领域的“领域专家”。起初，这看起来是一个比较合 理的方法。它 对汽车制造装配生产线很适用，但是为什么对软件开发团队就不行呢？毕竟，想让每个成员都掌握所编程序的细微差别也不太可能，对吧？

问题是开发人员不容易轻易替换掉。虽然当每位成员都可用时，“抽屉方法”很有效，但如果当“领域专家”突然因人事变动、疾病或突发事故而无法工作 时，抽屉 方法立马土崩瓦解。（所以，）软件团队有一些看似多余实则重要的后备力量是至关重要。代码复查、结对编程和共有代码可用成功营造一个环境，在这个环境中， 每位开发人员至少表面上是熟悉自己非擅长领域之外的系统部分。

4. 种瓜得瓜，种豆得豆



《注重实效的程序员》一书中有这样一段话解释“破窗理论”：不要留着“破窗户”（低劣的设计、错误的决策或者糟糕的代码）不修。发现一个就修一个。 如果没有足够的时间进行适当的修理，就先把它保留起来。或许你可 以把出问题的代码放到注释中，或是显示“未实现”消息，或用虚拟数据加以替代。采取一些措施，防止进一步的恶化。这表明局势尚在掌控之中。
我们见过整洁良好的系统在出现“破窗”之后立马崩溃。虽然促使软件崩溃的原因还有其他因素（我们将在其他地方接触到），但（对“破窗”）置之不理，肯定会更快地加速系统崩溃。
简而言之，好的代码会促生好的代码，糟糕的代码也会促生糟糕的代码。别低估了惯性的力量。没人想去整理糟糕的代码，同样没人想把完美的代码弄得一团糟。写好你的代码，它才更可能经得住时间的考验。
译注：《注重实效的程序员》，作者Andrew Hunt/David Thomas。该书直击编程陈地，穿过了软件开发中日益增长的规范和技术藩篱，对核心过程进行了审视——即根据需求，创建用户乐于接受的、可工作和易维护 的 代码。本书包含的内容从个人责任到职业发展，直至保持代码灵活和易于改编重用的架构技术。从本书中将学到防止软件变质、消除复制知识的陷阱、编写灵活、动 态和易适应的代码、避免出现相同的设计、用契约、断言和异常对代码进行防护等内容。
译注：破窗理论（Broken Window theory）：是关于环境对人们心理造成暗示性或诱导性影响的一种认识。“破窗效应”理论是指：如果有人打坏了一幢建筑物的窗户玻璃，而这扇窗户又得不到及时的维修，别人就可能受到某些暗示性的纵容去打烂更多的窗户。发现问题就要及时矫正和补救。

5. 欲速则不达

经理、客户和程序员正日益变得急躁。一切都需要做的事，都需要马上就做好。正因如此，快速修复问题变得非常急迫。
没时间对一个新功能进行适当的单元测试？好吧，你可以先完成一次测试运行，然后你就可以随时回来继续测试它。
当访问Y属性时，会不会碰到奇怪的对象引用错误？无论怎样，把代码放到try/catch语句块中。我们要钓到大鱼啦！
是不是似曾相识呢？这是因为我们在以前已经都做到了。并且在某些情况下、它是无可非议的。毕竟，我们有最后期限，还得满足客户和经理。但不要过于频 繁操 作，否则你会发现你的代码不稳定，有很多热修复、逻辑重复、未测试的方案和错误处理。最后，你要么是把事情草草做完，要么是把事情好好做完。

6. 三思而后行

“敏捷开发”这个词最近被频繁滥用，经常被程序员用来掩饰他们在软件开发过程中的糟糕规划/设计阶段。我们是设计者，看到产品朝正当方向有实质进 展，我们理应高兴。但意外的是，UML图和用例分析似乎并不能满足我们的愿望。所以，在不知自己做什么的情况下或者不知自己身处何处时，我们开发人员经常 就稀里糊涂地写代码了。
这就好比你要去吃饭，但你根本没有想好去哪里吃。因为你太饿了，所以你迫不及待地找个餐

馆，定个桌位。然后你上车开车后沿途在想（找地方吃饭）。只 是，这样会耗费更多的时间，因为你要过较多的U型弯道，还在餐馆前停车，也许最后因等待时间过长而不吃了。确切地说，你最后应该能找到地方吃饭，但你可能 吃的饭并不是你想吃的，并且这样花费的时间，可能比你直接在想去的餐馆订餐所花的时间更长。

7. 如果你惟一的工具是一把锤子，你往往会把一切问题看成钉子

看见了吧？我早就说过动态记录在这个项目中很有效

程序员有一种倾向，当一谈到他们工具时，其视野就变狭窄了。一旦某种方法在我们的一个项目上“行得通”，我们就会在接下来所有的项目上都用到它。学 习新东 西仿佛是一种煎熬，有时候甚至会心神不定。从始至终都在想“如果我用之前的方法做、这个就不会这么麻烦了”。一定要摒弃这种想法，按我们所知道的去做，即 使那不是最完美的解决方法。

坚持自己所知很简单，不过从长远的角度讲，选择一个适合这项工作的工具要容易得多。否则，就会与你的职业生涯格格不入。

8. 沉默即赞同

我什么都没看见！没看见！

“破窗理论”与“变成惯性理论”有着宏观的联系。

编程社区就好像一个现实社区。每个作品都是一个开发者的缩影。糟糕的代码发布的越多，就越容易反映现状。如果你不去努力编写优秀、整洁和稳定的代码，那你每天都将和糟糕的代码相伴了。

同样地，如果你看到别人写出了糟糕的代码，你就要跟这个人提出来。注意，这时候机智就应该用上场了。一般情况下，程序员都愿意承认他们在软件开发中还是有不懂的地方，并且会感谢你的好意。互相帮助对大家都有利，而对问题视而不见，只会使问题一直存在。

9. 双鸟在林，不如一鸟在手

如果可以讨论系统架构和重构，那么就差找个时间把事情做完。为了使正常运作的东西更加简洁而做改动，权衡改动的利弊很重要。当然了，简洁是一个理想 目标， 但总会有可以通过重构改进的代码。在编程世界中，为了代码不过时，会频繁简单改动代码。但有时候你又必须保证代码对客户有价值。那么，你面临一个简单窘 境：你不能一石二鸟。你在重构旧代码上所发时间越多，

你编写新代码的时间就越少。在及时改进代码和维护程序之间，也需要找到平衡点。

10. 能力越大，责任越大

毫无疑问，软件已成为我们生活中一个既基本又重要的一部分。正因如此，开发优秀软件格外重要。乒乓球游戏中的Bug是一回事，航天飞机导向系统或者 航空交通管制系统中的Bug是另外一回事。Slashdot曾发表一文，讲述了单单Google News的一个小失误使一家公司股票蒸发11.4亿美元。其他例子参见《软件Bug引发的十次严重后果》。这些例子便说明了我们正行使着多大的权利。你今 天写的代码，无论你是否有意，说不定有朝一日在重要的应用程序中派上用场，这想想都令人害怕。编写正确合格的代码吧！

07-面向对象设计原则

2014年10月25日 星期六　　23:44

在使用面向对象的思想进行系统设计时，前人共总结出了7条原则，它们分别是：单一职责原则、开闭原则、里氏替换原则、依赖注入原则、接口分离原则、迪米特原则和优先使用组合而不是继承原则。

1．单一职责原则（SRP）

单一职责原则的核心思想就是：系统中的每一个对象都应该只有一个单独的职责，而所有对象所关注的就是自身职责的完成。它的英文缩写是SRP，英文全称是Single Responsibility Principle。

其实单一职责原则的意思就是开发人员经常说的“高内聚、低耦合”。也就是说，每个类应该只有一个职责，对外只能提供一种功能，而引起类变化的原因应该只有一个。在设计模式中，所有的设计模式都遵循这一原则。

2．开闭原则（OCP）

开闭原则的核心思想就是：一个对象对扩展开放，对修改关闭。它的英文缩写是OCP，英文全称是Open for Extension，Closed for Modification。

其实开闭原则的意思就是：对类的改动是通过增加代码进行的，而不是改动现有的代码。也就是说，软件开发人员一旦写出了可以运行的代码，就不应该去改变它，而是要保证它能一直运行下去，如何才能做到这一点呢？这就需要借助于抽象和多态，即把可能变化的内容抽象出来，从而使抽象的部分是相对稳定的，而具体的实现层则是可以改变和扩展的。

3．里氏替换原则（LSP）

里氏替换原则的核心思想就是：在任何父类出现的地方都可以用它的子类来替代。它的英文缩写是LSP，英文全称是Liskov Substitution Principle。

其实里氏替换原则的意思就是：同一个继承体系中的对象应该有共同的行为特征。里氏代换原则关注的是怎样良好地使用继承，也就是说不要滥用继承，它是继承复用的基石。

4．依赖注入原则（DIP）

依赖注入原则的核心思想就是：要依赖于抽象，不要依赖于具体的实现。它的英文缩写是DIP，英文全称是Dependence Inversion Principle。

其实依赖注入原则的意思就是：在应用程序中，所有的类如果使用或依赖于其他的类，则都应该依赖于这些其他类的抽象类，而不是这些其他类的具体实现类。抽象层次应该不依赖于具体的实现细节，这样才能保证系统的可复用性和可维护性。为了实现这一原则，就要求开发人员在编程时要针对接口编程，而不针对实现编程。

5．接口分离原则（ISP）

接口分离原则的核心思想就是：不应该强迫客户程序依赖它们不需要使用的方法。它的英文缩写是ISP，英文全称是Interface Segregation Principle。

其实接口分离原则的意思就是：一个接口不需要提供太多的行为，一个接口应该只提供一种对外的功能，不应该把所有的操作都封装到一个接口当中。

6．迪米特原则（LOD）

迪米特原则的核心思想就是：一个对象应当对其他对象尽可能少的了解。它的英文缩写是LOD，英文全称是Law of Demeter。

其实迪米特原则的意思就是：降低各个对象之间的耦合，提高系统的可维护性。在模块之间，应该只通过接口来通信，而不理会模块的内部工作原理，它可以使各个模块耦合程度降到最低，促进软件的复用。

7．优先使用组合而不是继承原则（CARP）

优先使用组合而不是继承原则的核心思想就是：优先使用组合，而不是继承。它的英文缩写是CARP，英文全称是Composite/Aggregate Reuse Principle。

其实优先使用组合而不是继承原则的意思就是：在复用对象的时候，要优先考虑使用组合，而不是继承，这是因为在使用继承时，父类的任何改变都可能影响子类的行为，而在使用组合时，是通过获得对其他对象的引用而在运行时刻动态定义的，有助于保持每个类的单一职责原则。

面向对象设计的SOLID原则

S.H.I.E.L.D

S.O.L.I.D是面向对象设计和编程(OOD&OOP)中几个重要编码原则(Programming Priciple)的首字母缩写。

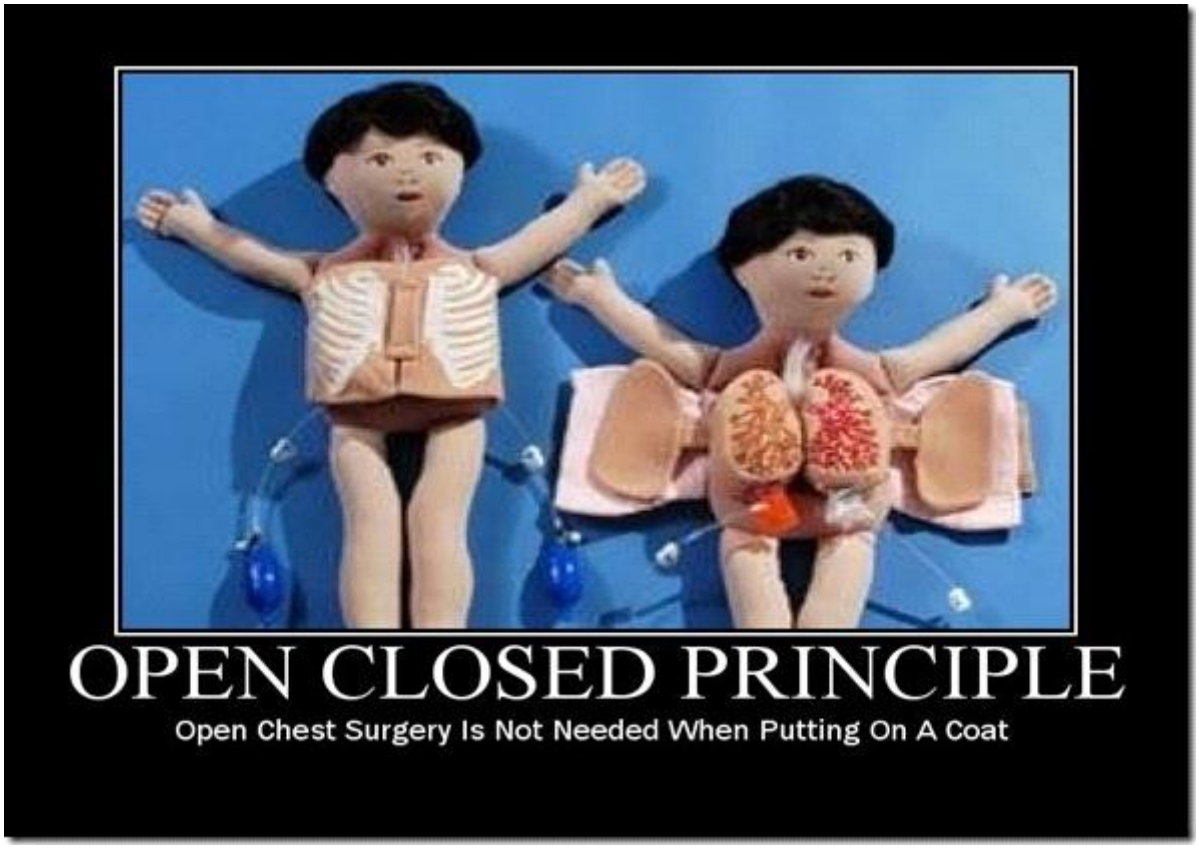
SRP	The Single Responsibility Principle	单一责任原则
OCP	The Open Closed Principle	开放封闭原则
LSP	The Liskov Substitution Principle	里氏替换原则
DIP	The Dependency Inversion Principle	依赖倒置原则
ISP	The Interface Segregation Principle	接口分离原则

单一责任原则：

当需要修改某个类的时候原因有且只有一个（THERE SHOULD NEVER BE MORE THAN ONE REASON FOR A CLASS TO CHANGE）。换句话说就是让一个类只做一种类型责任，当这个类需要承当其他类型的责任的时候，就需要分解这个类。

开放封闭原则

软件实体应该是可扩展，而不可修改的。也就是说，对扩展是开放的，而对修改是封闭的。这个原则是诸多面向对象编程原则中最抽象、最难理解的一个。



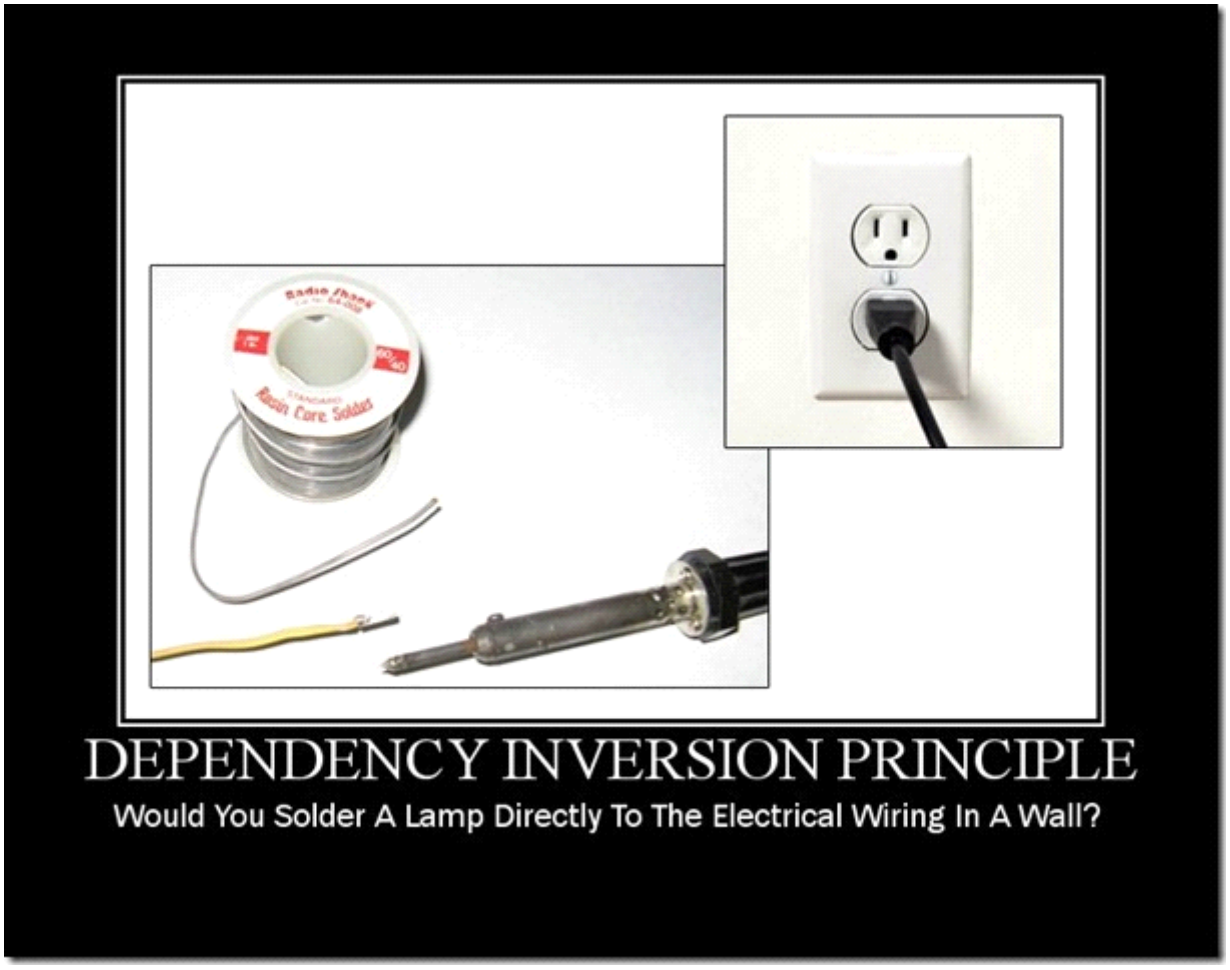
里氏替换原则

当一个子类的实例应该能够替换任何其超类的实例时，它们之间才具有is-A关系

依赖倒置原则

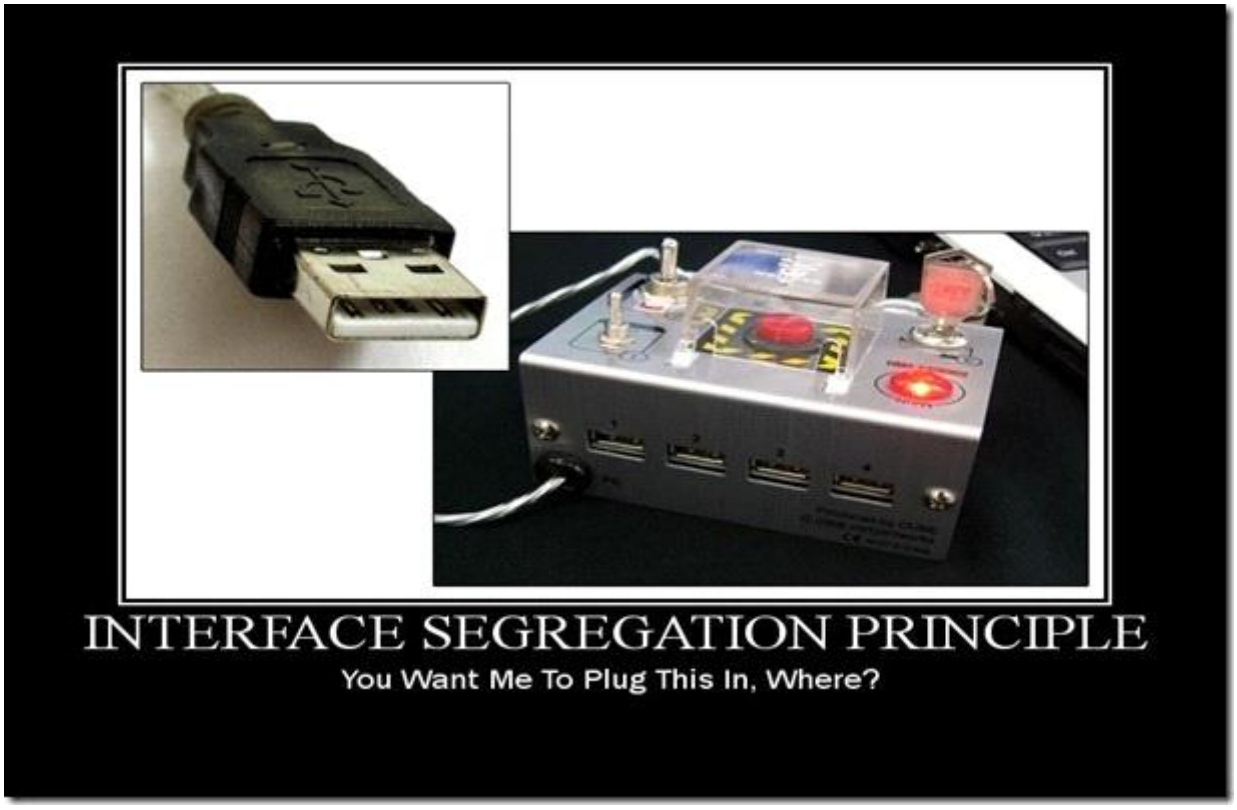
1. 高层模块不应该依赖于低层模块，二者都应该依赖于抽象

2. 抽象不应该依赖于细节，细节应该依赖于抽象



接口分离原则

不能强迫用户去依赖那些他们不使用的接口。换句话说，使用多个专门的接口比使用单一的总接口总要好。



这几条原则是非常基础而且重要的面向对象设计原则。正是由于这些原则的基础性，理解、融汇贯通这些原则需要不少的经验和知识的积累。上述的图片很好的注释了这几条原则。