DRY原则的误区

很多编程的人，喜欢鼓吹各种各样的“原则”，比如KISS原则，DRY原则…… 总有人把这些所谓原则奉为教条或者秘方，以为兢兢业业地遵循这些，空喊几个口号，就可以写出好的代码。同时，他们对违反这些原则的人嗤之以鼻——你不知道，不遵循或者藐视这些原则，那么你就是菜鸟。所谓“DRY原则”（Don’t Repeat Yourself，不要重复你自己）就是这些教条其中之一。盲目的迷信DRY原则，在实际的工程中带来了各种各样的问题，却经常被忽视。

简言之，DRY原则鼓励对代码进行抽象，但是鼓励得过了头。DRY原则说，如果你发现重复的代码，就把它们提取出去做成一个“模板”或者“框架”。对于抽象我非常的在行，实际上程序语言专家做的许多研究，就是如何设计更好的抽象。然而我并不奉行所谓DRY原则，并不是尽一切可能避免“重复”。“避免重复”并不等于“抽象”。有时候适当的重复代码是有好处的，所以我有时候会故意的进行重复。

抽象与可读性的矛盾

代码的“抽象”和它的“可读性”（直观性），其实是一对矛盾的关系。适度的抽象和避免重复是有好处的，它甚至可以提高代码的可读性，然而如果你尽“一切可能”从代码里提取模板，甚至把一些微不足道的“共同点”也提出来进行“共享”，它就开始有害了。这是因为，模板并不直接显示在“调用”它们的位置。提取出模板，往往会使得阅读代码时不能一目了然。如果由此带来的直观性损失超过了模板所带来的好处时，你就应该考虑避免抽象了。要知道，代码读的次数要比写的次数多很多。很多人为了一时的“写的快感”，过早的提取出不必要的模板，其实损失了读代码时的直观性。如果自己的代码连自己都不能一目了然，你就不能写出优雅的代码。

举一个实际的例子。奉行DRY原则的人，往往喜欢提取类里面的“共同field”，把它们放进一个父类，然后让原来的类继承这个父类。比如，本来的代码可能是：

```
class A {
  int a;
  int x;
  int y;
}

class B {
  int a;
  int u;
  int v;
}
```

奉行DRY原则的人喜欢把它改成这样：

```
class C {
  int a;
}

class A extends C {
  int x;
  int y;
}

class B extends C {
  int u;
  int v;
}
```

后面这段代码有什么害处呢？它的问题是，当你看到`class A`和`class B`的定义时，你不再能一目了然的看到`int a`这个field。“可见性”，对于程序员能够产生直觉，是非常重要的。这种无关紧要的field，其实大部分时候都没必要提出去，造出一个新的父类。很多时候，不同类里面虽然有同样的`int a`这样的field，然而它们的含义却是完全不同的。有些人不管三七二十一就来个“DRY”，结果不但没带来好处，反而让程序难以理解。

抽象的时机问题

奉行DRY原则的人还有一个问题，就是他们随时都在试图发现“将来可能重用”的代码，而不是等到真的出现重复的时候再去做抽象。很多时候他们提取出一个貌似“经典模板”，结果最后过了几个月发现，这个模板在所有代码里其实只用过一次。这就是因为他们过早的进行了抽象。

抽象的思想，关键在于“发现两个东西是一样的”。然而很多时候，你开头觉得两个东西是一回事，结果最后发现，它们其实只是肤浅的相似，而本质完全不同。同一个`int a`，其实可以表示很多种风马牛不及的性质。你看到都是`int a`就提出来做个父类，其实反而让程序的概念变得混乱。还有的时候，有些东西开头貌似同类，后来你增添了新的逻辑之后，发现它们的用途开始特殊化，后来就分道扬镳了。过早的提取模板，反而捆住了你的手脚，使得你为了所谓“一

致性”而重复一些没用的东西。这样的一致性，其实还不如针对每种情况分别做特殊处理。

防止过早抽象的方法其实很简单，它的名字叫做“等待”。其实就算你不重用代码，真的不会死人的。时间能够告诉你一切。如果你发现自己仿佛正在重复以前写过代码，请先不要停下来，请坚持把这段重复的代码写完。如果你不把它写出来，你是不可能准确的发现重复的代码的，因为它们很有可能到最后其实是不一样的。

你还应该避免没有实际效果的抽象。如果代码才重复了两次，你就开始提取模板，也许到最后你会发现，这个模板总共也就只用了两次！只重复了两次的代码，大部分时候是不值得为它提取模板的。因为模板本身也是代码，而且抽象思考本身是需要一定代价的。所以最后总的开销，也许还不如就让那两段重复的代码待在里面。

这就是为什么我喜欢一种懒懒的，笨笨的感觉。因为我懒，所以我不会过早的思考代码的重用。我会等到事实证明重用一定会带来好处的时候，才会开始提取模板，进行抽象。经验告诉我，每一次积极地寻找抽象，最后的结果都是制造一些不必要的模板，搞得自己的代码自己都看不懂。很多人过度强调DRY，强调代码的“重用”，随时随地想着抽象，结果被这些抽象搅混了头脑，bug百出，寸步难行。如果你不能写出“可用”（usable）的代码，又何谈“可重用”（reusable）的代码呢？

谨慎的对待所谓原则

说了这么多，我是在支持DRY，还是反对DRY呢？其实不管是支持还是反对它，都会表示我在乎它，而其实呢，我完全不在乎这类原则，因为它们非常的肤浅。这就像你告诉我说你有一个重大的发现，那就是“1+1=2”，我该支持你还是反对你呢？我才懒得跟你说话。人们写程序，本来自然而然就会在合适的时候进行抽象，避免重复，怎么过了几十年后，某个菜鸟给我们的做法起了个名字叫DRY，反而他成了“大师”一样的人物，我倒要用“DRY”这个词来描述我一直在干的事情呢？所以我根本不愿意提起“DRY”这个名字。

所以我觉得这个DRY原则根本就不应该存在，它是一个根本没有资格提出“原则”的人提出来的。看看他鼓吹的其它低劣东西（比如Agile，Ruby），你就会发现，他是一个兜售减肥药的“软件工程专家”。世界上有太多这样的肤浅的所谓原则，我不想对它们一一进行评价，这是在浪费我的时间。世界上有比这些喜欢提出“原则”的软件工程专家深邃很多的人，他们懂得真正根本的原理。