10 | 软件设计的目的：糟糕的程序员比优秀的程序员差在哪里？

2019-12-13 李智慧

讲述：李智慧
时长 10:02　大小 9.20M

有人说，在软件开发中，优秀的程序员比糟糕的程序员的工作产出高 100 倍。这听起来有点夸张，实际上，我可能更悲观一点，就我看来，有时候，后者的工作成果可能是负向的，也就是说，因为他的工作，项目会变得更加困难，代码变得更加晦涩，难以维护，工期因此推延，各种莫名其妙改来改去的 bug 一再出现，而且这种局面还会蔓延扩散，连那些本来还好的代码模块也逐渐腐坏变烂，最后项目难以为继，以失败告终。

如果仅仅是看过程，糟糕的程序员和优秀的程序员之间，差别并没有那么明显。但是从结果看，如果最后的结果是失败的，那么产出就是负的，和成功的项目比，差别不是 100 倍，而是无穷倍。

程序员的好坏，一方面体现在编程能力上，比如并不是每个程序员都有编写一个编译器程序的能力；另一方面，体现在程序设计方面，即使在没有太多编程技能要求的领域下，比如开发一个订单管理模块，只要需求明确，具有一定的编程经验，大家都能开发出这样一个程序，但优秀的程序员和糟糕的程序员之间，依然有巨大的差别。

在软件设计开发这个领域，好的设计和坏的设计最大的差别就体现在应对需求变更的能力上。而好的程序员和差的程序员的一个重要区别，就是对待需求变更的态度。差的程序员害怕需求变更，因为每次针对需求变更而开发的代码都会导致无尽的 bug；好的程序员则欢迎需求变更，因为他们一开始就针对需求变更进行了软件设计，如果没有需求变更，他们优秀的设计就没有了用武之地，产生一拳落空的感觉。这两种不同态度的背后，是设计能力的差异。

一个优秀的程序员一旦习惯设计、编写能够灵活应对需求变更的代码，他就再也不会去编写那些僵化的、脆弱的、晦涩的代码了，甚至仅仅是看这样的代码，也会产生强烈的不舒服的感觉。记得一天下午，一个技术不错的同事突然跟我请假，说身体不舒服，需要回去休息一下，我看他脸色惨白，有气无力，就问他怎么了。他回答：刚才给另一个组的同事 review 代码，代码太恶心了，看到中途去厕所吐了，现在浑身难受，需要休息。

惊讶吗？但实际上，糟糕的代码就是能产生这么大的威力，这些代码在运行过程中使系统崩溃；测试过程中使 bug 无法收敛，越改越多；开发过程使开发者陷入迷宫，掉到一个又一个坑里；而仅仅是看这些代码，都会使阅读者头晕眼花。

糟糕的设计

糟糕的设计和代码有如下一些特点，这些特点共同铸造了糟糕的软件。

僵化性

软件代码之间耦合严重，难以改动，任何微小的改动都会引起更大范围的改动。一个看似微小的需求变更，却发现需要在很多地方修改代码。

脆弱性

比僵化性更糟糕的是脆弱性，僵化导致任何一个微小的改动都能引起更大范围的改动，而脆弱则是微小的改动容易引起莫名其妙的崩溃或者 bug，出现 bug 的地方看似与改动的地方毫无关联，或者软件进行了一个看似简单的改动，重新启动，然后就莫名其妙地崩溃了。

如果说僵化性容易导致原本只用 3 个小时的工作，变成了需要三天，让程序员加班加点工作，于是开始吐槽工作的话，那么脆弱性导致的突然崩溃，则让程序员开始抓狂，怀疑人生。

牢固性

牢固性是指软件无法进行快速、有效地拆分。想要复用软件的一部分功能，却无法容易地将这部分功能从其他部分中分离出来。

目前微服务架构大行其道，但是，一些项目在没有解决软件牢固性的前提下，就硬着头皮进行微服务改造，结果可想而知。要知道，微服务是低耦合模块的服务化，首先需要的，就是低耦合的模块，然后才是微服务的架构。如果单体系统都做不到模块的低耦合，那么由此改造出来的微服务系统只会将问题加倍放大，最后就怪微服务了。

粘滞性

需求变更导致软件变更的时候，如果糟糕的代码变更方案比优秀的方案更容易实施，那么软件就会向糟糕的方向发展。

很多软件在设计之初有着良好的设计，但是随着一次一次的需求变更，最后变得千疮百孔，趋向腐坏。

晦涩性

代码首先是给人看的，其次是给计算机执行的。如果代码晦涩难懂，必然会导致代码的维护者以设计者不期望的方式对代码进行修改，导致系统腐坏变质。如果软件设计者期望自己的设计在软件开发和维护过程中一直都能被良好执行，那么在软件最开始的模块中就应该保证代码清晰易懂，后继者参与开发维护的时候才有章法可循。

一个设计腐坏的例子

软件如果是一次性的，只运行一次就被永远丢弃，那么无所谓设计，能实现功能就可以了。然而现实中的软件，大多数在其漫长的生命周期中都会被不断修改、迭代、演化和发展。淘宝从最初的小网站，发展到今天有上万名程序员维护的大系统；Facebook 从扎克伯格一个人开发的小软件，成为如今服务全球数十亿人的巨无霸，无不经历过并将继续经历演化发展的过程。

接下来，我们就来看一个软件在需求变更过程中，不断腐坏的例子。

假设，你需要开发一个程序，将键盘输入的字符，输出到打印机上。任务看起来很简单，几行代码就能搞定：

复制代码

```
void copy()
{
  int c;
  while ((c=readKeyBoard()) != EOF)
    writePrinter(c);
}
```

你将程序开发出来，测试没有问题，很开心得发布了，其他程序员在他们的项目中依赖你的代码。过了几个月，老板忽然过来说，这个程序需要支持从纸带机读取数据，于是你不得不修改代码：

复制代码

```
bool ptFlag = false;
// 使用前请重置这个 flag
void copy()
{
  int c;
  while ((c=(ptFlag? readPt() : readKeyBoard())) != EOF)
    writePrinter(c);
}
```

为了支持从纸带机输入数据，你不得不增加了一个布尔变量，为了让其他程序员依赖你的代码的时候能正确使用这个方法，你还添加一句注释。即便如此，还是有人忘记了重设这个布尔值，还有人搞错了这个布尔值的代表的意思，运行时出来 bug。

虽然没有人责怪你，但是这些问题还是让你很沮丧。这个时候，老板又来找你，说程序需要支持输出到纸带机上，你只好硬着头皮继续修改代码：

复制代码

```
bool ptFlag = false;
bool ptFlag2 = false;
// 使用前请重置这些 flag
void copy()
```

```
{
  int c;
  while ((c=(ptFlag? readPt() : readKeyBoard())) != EOF)
    ptFlag2? writePt(c) : writePrinter(c);
}
```

虽然你很贴心地把注释里的”这个 flag“改成了”这些 flag “，但还是有更多的程序员忘记要重设这些奇怪的 flag，或者搞错了布尔值的意思，因为依赖你的代码而导致的 bug 越来越多，你开始犹豫是不是需要跑路了。

解决之道

从这个例子我们可以看到，一段看起来还比较简单、清晰的代码，只需要经过两次需求变更，就有可能变得僵化、脆弱、粘滞、晦涩。

这样的问题场景，在各种各样的软件开发场景中，随处可见。人们为了改善软件开发中的这些问题，使程序更加灵活、强壮、易于使用、阅读和维护，总结了很多设计原则和设计模式，遵循这些设计原则，灵活应用各种设计模式，就可以避免程序腐坏，开发出更强大灵活的软件。

比如针对上面这个例子，更加灵活，对需求更加有弹性的设计、编程方式可以是下面这样的：

复制代码

```
public interface Reader {
  int read();
}

public interface Writer {
  void write(int c);
}

public class KeyBoardReader implements Reader {
  public int read() {
    return readKeyBoard();
  }
}

public class Printer implements Writer {
  public void write(int c) {
    writePrinter(c);
  }
```

```
19 }
20
21 Reader reader = new KeyBoardReader();
22 Writer writer = new Printer():
23 void copy() {
24   int c;
25   while(c=reader.read() != EOF)
26     writer(c);
27 }
```

我们通过接口将输入和输出抽象出来，copy 程序只负责读取输入并进行输出，具体输入和输出实现则由接口提供，这样 copy 程序就不会因为要支持更多的输入和输出设备而不停修改，导致代码复杂，使用困难。

所以你能看到，应对需求变更最好的办法就是一开始的设计就是针对需求变更的，并在开发过程中根据真实的需求变更不断重构代码，保持代码对需求变更的灵活性。

小结

我们在开始设计的时候就需要考虑程序如何应对需求变更，并因此指导自己进行软件设计，在开发过程中，需要敏锐地察觉到哪些地方正在变得腐坏，然后用设计原则去判断问题是什么，再用设计模式去重构代码解决问题。

我在面试过程中，考察候选人编程能力和编程技巧的主要方式就是问关于设计原则与设计模式的问题。

我将在”软件的设计原理“这一模块，主要讲如何用设计原则和设计模式去设计强壮、灵活、易复用、易维护的程序。希望这部分内容能够帮你掌握如何进行良好的程序设计。

思考题

你在软件开发实践中，是否曾经看到过一些糟糕的代码？这些糟糕的代码是否符合僵化、脆弱、牢固、粘滞、晦涩这些特点？这些代码给工作带来了怎样的问题呢？

欢迎你在评论区写下你的体验，我会和你一起交流，也欢迎你把这篇文章分享给你的朋友或者同事，一起交流进步一下。

上一篇　09 | 软件设计实践：如何使用UML完成一个设计文档?

下一篇　11 | 软件设计的开闭原则：如何不修改代码却能实现需求变更?

精选留言 (8)

写留言

Paul Shan
2019-12-13

僵化性代码的例子是滥用了继承，导致添加一个小功能，所有的基类和派生类都要修改。脆弱性代码的例子是引入全局依赖，导致意外的修改扩散。每当我看到很多全局变量的时候，对程序的掌控感荡然无存。
牢固性代码的例子是超大类，由于类内部是可以任意访问，所有的巨量函数和属性组成了一个巨大完全图，牵一发而动全身，根本不知道从哪里下手。...

展开

12

难得糊涂ck
2019-12-13

A：可以说脏话嘛?
B：不能。

A：那我没什么好说的

展开

2

观弈道人

2019-12-13

想了解下智慧老师是如何提问考察应聘者编程能力和编程技巧

2 2

探索无止境

2019-12-13

优劣设计案例做对比，最能让人理解到文字所阐述的点，希望老师可以举更多的例子，这样更有收获

1

靠人品去赢

2019-12-18

不用看别人，我的代码就很有问题，主要问题有一，命名，代码的命名是门大学问，看到一本书说是好的命名相当于完成来一部分代码，看点指导性的书还有一些具体的最佳实践，比如说阿里自己的编程规范，github上有，在他那个插件里面。
第二个，设计模式用的不够好，总是if else来写代码，实际上可以借助文中的例子，采用一些设计模式，像工厂模式借助Java的父类和子类，接口解耦来搞一下，防止出现厄运金...

展开

云川

2019-12-15

刚刚改造后的例子中，从纸带读取数据不是也要去实现自带的类吗。如果不同程序员对输入和输出有不同的要求，是不是让他们自己实现输入输出接口然后作为参数传入进来？

golangboy

2019-12-14

1. 在写代码前，把逻辑理清楚；
2.分离变化和不变的过程，将数据的传输控制和解析处理分离。对于变的地方，函数式编程中用不同的函数去灵活替换，对象编程的话，就用不同对象去替换。这样程序设计面对需求时就灵活了；
3.函数适量的拆分，不要太长，太长太丑、看起来累，也花时间；...

展开

sunfuwen

2019-12-13

Reader reader = new KeyBoardReader();
Writer writer = new Printer():
void copy()
{
int c; ...

展开