16 | 熊节：什么代码应该被重构？

2021-02-04 熊节

代码之丑 进入课程 >

讲述：正霖

时长 10:50 大小 9.93M

你好，我是郑晔。

代码坏味道的说法源自《重构》这本书，坏味道和重构这两个概念几乎是如影随形。提及《重构》这本书，在国内谁还能比《重构》两版的译者熊节更了解它呢？所以，这一讲，我就请来了我的老朋友熊节，谈谈在他眼中看到的重构和坏味道。有请熊节老师！

你好，我是熊节。

自从翻译了《重构》以后，很多公司找我去做重构的培训，光是华为一家，这个主题在...个不同的部门就培训过好些次。每次讲这个主题，我都觉得挺为难的：重构这事有什么可培训的呢，不就是一个无脑模式匹配的事吗！然而跟各家公司的读者们一交流，我就发现事情并没有那么简单。

很多人一说到重构，就聊到虚无缥缈的事上了，像什么架构啦、文化啦，等等。我不得不先把他们拉住仔细问问，他们是怎么读《重构》这本书的？这一问我就发现，原来很多读者（恐怕是绝大多数读者），还没弄明白这本书到底应该怎么读。

什么代码应该被重构?

《重构》这本书，以及重构这门手艺，提纲挈领的部分，都在一个关键的问题上：**什么代码应该被重构**。

你可能会说，质量不好的代码需要被重构。没错，可是代码的质量到底应该如何评判呢？

首先我们要明确的是，代码的好与坏不应当用个人好恶、“含混的代码美学”来表达，因为这会带来两个困难：

第一，每个人对于“好”或“美”的观念可能相当不同；

第二，对于坏代码缺乏明确的“症状”判断，也就很难提出明确的改进措施。

即便是一些经典的程序设计原则，也有同样的问题。例如“高内聚低耦合”，尽管这是所有人都赞同的设计原则，但究竟什么样的代码呈现了“低内聚”、什么样的代码呈现了“高耦合”、“低内聚”与“高耦合”是否总是同时出现、应该以何种办法提高内聚降低耦合……这些问题仍然是悬而未决的。

因此，对于真正在一线工作的人来说，“高内聚低耦合”很多时候就成了一句咒语，念完咒语后，呼唤出的其实还是每个人原本的编程习惯与风格，并不真正指导任何行为的改变。

而当我们去观察“低内聚高耦合”带来的问题时，事情就变得明朗了。比如，当我们仔细阅读《重构》第三章时，我们会发现，“低内聚”会直接引发的现象是“霰弹式修改（Shotgun Surgery）”：

> 每当需要对某个事情做出修改时，你都必须在许多不同的类内做出许多小修改，那么就可以确定，你所面临的坏味道是霰弹式修改。当需要修改的代码分散在多处，你不但很难找到它们，也很容易忘记某个重要的修改。

而“高耦合”直接引发的现象则是有某种相似性、但又表现不同的“发散式变化（Divergent Change）”：

> 如果某个类经常因为不同的原因在不同的方向上发生变化，发散式变化就出现了。

我再举另一个设计原则的例子。“迪米特原则”也是常被提及的面向对象设计原则之一，然而知道这个名称是一回事，知道如何识别不符合迪米特原则的代码，则又需要更多的个人经验。《重构》第三章则把这个原则表述为两个非常直观的症状：“过长的消息链（Message Chains）”和“中间人（Middle Man）”。

> 如果你看到用户向一个对象请求另一个对象，而后者再次请求另一个对象，然后再请求另一个对象……这就是消息链。在实际代码中，你看到的可能是一长串取值函数，或者一长串临时变量。

> 人们可能过度运用委托。你也许会看到某个类接口有一半的函数都委托给其他类，这样就是过度运用。

你发现没？对于开始我们提到的“什么代码应该被重构”这个关键问题，虽然《重构》作者 Martin Fowler 和 Kent Beck 非常客气地声称：“并不试图给你一个何时必须重构的精确衡量标准”，实际上，《重构》给出的 24 项“坏味道”（在《重构》第一版中是 22 项）已经形成了一个非常明确的代码质量检查清单。

尽管这本书从未声称这是一份完备的坏味道清单，但在实际工作中，还不用说完全识别并消除这份列表中的全部坏味道，只要能做到命名合理、没有重复、各个代码单元（类、函数等）体量适当、各个代码单元有明确且单一的职责、各个代码单元之间有恰当的交互，这就已经是质量相当高的代码了。

更重要的是，伴随着对具体症状的了解，对症的解决办法也变得明确。在《重构》第三章里非常明确地讲到：

> 对于“霰弹式修改”，解决的办法是使用“搬移函数”和“搬移字段”，把所有需要修改的代码放进同一个模块；
>
> 对于“发散式变化”，解决的办法是首先用“提炼函数”将不同用途的逻辑分开，然后用“搬移函数”将它们分别搬移到合适的模块；

对于“过长的消息链”，你应该使用“隐藏委托关系”；对于“中间人”，对症的疗法则是“移除中间人”，甚至直接“内联函数”。

这就是我前面所说的“无脑模式匹配”。

讲到这里，你也就明白了，对于绝大多数程序员而言，阅读和使用《重构》这本书的正确方法就应该是：

1. 打开任意一段代码（可以是自己刚写完的或者马上要动手修改的）；
2. 翻开《重构》第三章，遍历其中的每个坏味道：
   a. 识别这段代码中是否存在上述坏味道；
      i. 如有，则遵循该坏味道所列的重构手法，对该段代码进行重构；
      ii. 如无，则继续遍历代码。

上述过程不需要玄妙的理论和含混的代码美学，只需要机械的重复和简单的模式匹配。正因为此，重构才是一项完完全全具备可操作性、能够在任何遗留代码库上实践的技术。

培养对“坏味道”的判断力

当然，每位实践者仍然“**必须培养出自己的判断力，学会判断一个类内有多少实例变量算是太大、一个函数内有多少行代码才算太长”。**

就在最近，我看到某大厂的一位“代码委员会理事”在文章里说，某段代码“挺好的，长度没超过 80 行，逻辑比较清晰”。而在我看来，一个函数超过 7 行就已经是“太长”（这还是在考虑到 Java 语法比较啰嗦的前提下）。这就是不同实践者“自己的判断力”所体现的差异。

尽管从来没有明确指定对每个函数或类的代码行数要求，但“对象健身操”这篇文章（见于《ThoughtWorks 文集》）提出的 9 项规则已经有非常明确的指向：

方法中只允许使用一级缩进；

不允许使用 else 关键字；

封装所有的原生类型和字符串；

……

在这样的规则约束下，写出一个超过 10 行的函数将是相当困难的（实际上在“规则 6：保持实体对象简单清晰”中已经明确提出，每个类的长度不能超过 50 行）。

正如“对象健身操”这篇文章的作者 Jeff Bay 自己所说，这套“健身操”的意义在于：“在一个简单的项目里尝试一些比以前严格得多的编码标准……会迫使你更为严格地以面向对象的风格编写代码”，从而“以一种全新的方式思考你的代码”。

不过这得需要你刻意练习。正所谓“台上一分钟，台下十年功”，**缺乏在受控环境下的刻意练习，很难通过工作中的自然积累提升判断力。**

另外，对正确的代码构造足够熟悉，也是很重要的一个基本功，这个观点最早是 Kent Beck 的《实现模式》这本书中提到的。什么意思呢？

传说旧时民间古董店的学徒需要先在仓库里看真货，看得多了，见到假货时就会本能地提起警觉。对于代码也是一样：程序员需要熟悉正确的代码构造，在看到有问题的代码构造时才会本能地提起警觉。并且，**“正确的代码构造”并非无穷无尽，实际上在单线程编程中，几十个常见的模式已经几乎能够完全覆盖所有场景。**

Kent Beck 在前言里说“这是一本关于‘如何编写别人能懂的代码’的书”，尽管他还谦虚地说这本书“不是模式书籍”，但实际上《实现模式》充分地展现了“模式”的本意：它提供了一整套“用代码表述意图”的模式语言，这套语言能让程序员在最短的时间内学会如何写出具有表现力的代码，并且自然而然地远离坏味道。

从一开始就以合理的方式编程，从而使坏味道不要出现，我想这才是负责任的程序员应该采取的工作方式。

当然，极限编程的各种实践，尤其是工程技术实践彼此紧密相关。例如自动化测试、持续集成、集体代码所有制的缺失，都会导致代码的坏味道更容易堆积。而从另一个角度来看，这些实践从任何一个切入，又都会自然地引导出其他相关的实践。

一位“知行合一”的程序员最终会发现，极限编程是唯一合理且有效的软件开发方法。最终，只有采用以可工作的软件为核心的软件开发方法，才能得到高质量的可工作的软件，

这就是《敏捷宣言》第二句关于坏味道的终极答案。

郑老师说

好了，熊节老师的分享就到这里，我是郑晔。

熊节老师对于问题的分析总是这么一针见血。重构就是一个模式匹配的过程，识别出坏味道，运用对应的重构手法解决问题。坏味道是一切重构的起点，而识别坏味道不是靠个人审美，而要依赖通用的标准。

我的这个专栏就是把一些坏味道用更直接的代码形式展现在你面前，让你可以日常的工作中，不断地锻炼自己的代码嗅觉。

思考题

经过熊节老师的讲解，你是不是对重构和坏味道有了新的认识呢？欢迎在留言区分享你更新过的想法。

感谢阅读，我们下一讲再见。

提建议

上一篇 15 | 新需求破坏了代码，怎么办?

下一篇 17 | 课前作业点评：发现“你”代码里的坏味道

精选留言 (7)

写留言

Geek_3b1096
2021-02-05

专栏更贴近真实场景

展开

作者回复: 这是建议还是评价呢?

1 3

adang
2021-02-04

读书不等于掌握；掌握了不等于正在执行；执行了不等于一直在践行；识别出坏味道并践

行重构的能力就像武功一样，不用就会慢慢荒废。难的不是掌握这个能力，最难的是坚持践行合一。

作者回复: 知行合一，谈何容易。

2

NiceBen
2021-02-04

这种极限拆分，对于很多业务代码，拆分成很多方法，更加零散一些，而且有些拆出来的方法几乎不会被复用。这样还有必要拆分么？

展开

作者回复: 举个不是特别恰当的例子，这就像现在还在吃糠咽菜，就担心钱多了该怎么花。

先把函数拆小，能不能复用才能看出来，没有拆，何谈复用。再者，更重要的是，拆出来的目标不一定要复用，而是为了让代码更清晰。

1 1

升级打怪兽
2021-02-04

静态代码扫描，其实也不治根，规约然后说了，但是就是存在，还是得从意识上去唤醒这种细节点，毕竟只能要求自己，团队太难驱动

展开

作者回复: 先做好自己，再去影响他人。

1

qinsi
2021-02-04

这种机械和简单的模式匹配最好能交给机器去做

展开

作者回复: 道理上是这样的，问题是，现在这种模式匹配机器暂时还没法识别，也许是一个值得探索的方向。

2 1

Jxin

2021-02-05

喜欢郑老师的课程。每次都能get到好的新主题，进而课后可以做专项的主题学习。课程内容反而是附赠的。

本章:
1.get: 缺乏在受控环境下的刻意练习，很难通过工作中的自然积累提升判断力。...
展开

作者回复: 哈哈，欢迎订阅赠品。

Edison Zhou

2021-02-05

就我经历过的团队（传统企业信息中心）来看，不论是初级还是中级的同事，总是喜欢学习和研究分布式架构相关的知识点，而不喜欢阅读如《重构》、《代码整洁之道》一类的提高程序员最本质的手艺-写代码。然而，分布式的东西对我所经历的团队来说并不重要，因为实际上能用上的并不多。然而，每次Code Review总会有一些让人摸不着头脑要讲半天的代码，虽然也加了静态代码复杂度检查之类的，但是还是层出不穷。或许是这个浮...
展开

作者回复: 这是一个有趣的角度，现在大部分人其实用不到复杂的分布式知识，但这东西却成了主流，这与大公司面试方式有关。和当年受微软影响，各公司面试智力题有异曲同工的作用。

微软后来发现，这种选拔人才的方式有问题，需要改进，估计国内公司也需要经过一个漫长的过程之后，发现他们用这种方式选出来的人，与他们需要的人之间存在差距。

毕竟，有大量用户规模的事，只有一些大公司的核心团队才会遇到，而大部分人需要的是，写好代码。而大部分人学习分布式只是学习屠龙术，空有一身本事，无处施展。

1