<<代码之道>>

图书基本信息

书名：<<代码之道>>

13位ISBN编号：9787111251675

10位ISBN编号：7111251679

出版时间：2009-1

出版时间：机械工业出版社

作者：Eric Brechner

页数：192

译者：陆其明

前言

长期以来，我一直在阅读Eric Brechner以I. M. Wright为笔名撰写的栏目。
当我第一次见到作者本人的时候，我费了好大的劲才让自己相信，我是在跟同一个人说话。
Wright先生非常地自以为是，然而站在我面前说话的人却那么谦虚、彬彬有礼、非常友好，他看起来更像Clark Kent。
（译者注：Clark Kent是“超人”的名字，他具有超强的本领，是一个虚构的超级英雄，美国漫画中的经典人物。
）我很关注微软内部团队在软件开发的过程中，他们是如何去处理技术与人际交流之间的关系的；这类栏目总是我的最爱。
看到大量的公司内幕被写了出来，我常常会感到吃惊——我不知道还有多少不为人知的故事没有说出来。
大型项目中的软件工程管理者面临着3个基本的问题。
第一个是，程序代码太容易被改变了。
跟机械或土木工程不一样，它们在现有系统上做一次改变总是要付出实实在在地拆毁某些东西的代价，而软件程序的改变只需要敲敲键盘就行了。
如果对一座桥的桥墩或一架飞机的引擎做一个错误的结构性更改，由此产生的后果，即使不是专家也很容易就能看出来。
然而，如果在一个现有程序上做修改，对于其风险，即使经验丰富的软件开发者进行了充分的讨论，其结果常常还是错的。
建筑隐喻实际上可以很好地适用于软件。
基于程序代码在系统中所处的层次，它们可以被比作为“基础、框架和装饰”。
“基础”代码具有高度的杠杆作用，它们的改动常常会引起严重的连锁反应。
“装饰”代码比较容易改动，而且也需要被经常改动。
问题是，累积了几年的改变之后，复杂的程序就跟历经过几次装修的房子差不多了——电源插座躲到了橱柜的后面，浴室风扇的出风口通向了厨房。
再做任何改变的话，其副作用或最终的代价都是很难预知的。
第二个基本问题是，软件行业还太年轻，关于可复用组件的正确标准实际上还没有被发现或建立起来。
大头钉是否应该放在离开16英寸的地方，以同时适应水平或垂直的4x8英尺的干垒墙或夹板？
我们不仅在这类问题上还没有取得一致意见，甚至我们还没有决定，是否像大头钉、干垒墙和夹板这样的组合更可取，还是我们要去发明像泥浆、稻草、石头、钢铁和碳化纤维这样的组合。
最后一个问题实际上是第二个问题的另一种表现形式。
每个项目中重复发明的软件组件，它们也被重复命名了。
软件行业里对现有的概念发明新的名字是很常见的，即使用的名字相同，这些名字也以新的方式被重用。
行业里有一个心照不宣的秘密：关于软件开发最佳方法的相当多的讨论，参与的实际上都是同一群人，只不过他们用了不同的名字，他们甚至对彼此正在说的东西都没有一个哪怕是很朦胧的想法。
表面上看来，这些都是很简单的问题。
建立一些标准，然后强制实行它们。
在快速进步的大容量、高价值、低成本的软件世界里，这可是一个让你的业务落败的捷径。
实际情况是，软件最大的工程障碍，同时也是它最大的优势。
无处不在的软件（运行在低成本的个人电脑和互联网上），已经使得以惊人的步伐去创新成为可能。
随着微软的成长，公司已经不再能在最佳工程实践的研究方面大量地投入，然后经过深思熟虑，挑选出其中具有最好质量的方法。
个人电脑和Windows的成功，已经把公司从按传统方式做些小项目的形态转变出来，转而要去谱写开发有史以来最庞大、最复杂软件的新篇章。

为了能够创建出平衡风险与效率、创新的最佳系统，微软面临着持续不断的挣扎。
考虑到我们的一些项目有着极度的复杂性，这些努力甚至可以称得上“英勇无畏”。
在过去的一段时间以来，我们已经设置了专员、建立了专门的组织，他们都一心一意、致力于这个行业里最困难的事情——“软件发布”。
我们已经学会了很多的民间传说、风俗、文化、工具、过程和大拇指规则（译者注：Rules of Thumb，是指没有经过科学实验、直接从实践中总结出来的方法和规则；它们在很多情况下都有用，但并不是放之四海皆准），那些都有助于我们建造和发布这个世界上最复杂的软件。
但与此同时，每天都处理这些问题难免也让人心惊胆战、士气受挫。
Eric的栏目正是大家跟我们一起分享和学习的极好方式。

<<代码之道>>

内容概要

本书以一位微软内部人士的视角，揭示了关于软件编码、软件测试和项目管理中的方方面面问题。作者文笔犀利，见解独到，对软件行业内的很多常见问题提出了解决方案，并提供了最佳实践。本书详细介绍怎样提高软件的质量和价值；切合实际地管理项目的时间表、风险和规范书；为常见的低效率开发瘦身；应用过程改进方法，避免固执盲从；规划一个成功的、令人满意的职业生涯；发展并管理一个欣欣向荣的团队等。

本书是作者对过去在软件行业6个不同的公司、28年的工作经验的一次总结。

本书不仅是微软内部员工的必读之书，也同样适合于软件行业内其他所有工程师和管理者阅读。

作者简介

Eric Brechner，微软公司“卓越开发”部门的总监，在软件行业已经积累了20多年的经验。
他从2001年开始写“Hard Code”栏目，作为一种资源提供给微软的员工。
自那以后，其观点栏目在微软内部成千上万的软件开发者之间，激起了无休无止的关于最佳实践的讨论——如今，这些观点

<<代码之道>>

书籍目录

<<代码之道>>

章节摘录

第1章 项目的不当管理本章内容：2001年6月1日：“ 开发时间表、飞猪和其他幻想 ”2001年10月1日：“ 竭尽所能：再论开发时间表 ”2002年5月1日：“ 我们还开心吗？
分诊的乐趣 ”2004年12月1日：“ 向死亡进军 ”2005年10月1日：“ 揭露真相 ”我的第一个栏目是在2001年6月刊的微软内部网络杂志《Interface》上发表的。
为了进入I.M.Wright的人物角色，我需要一个真正能让我伤脑筋的主题。
而工作的时间安排和进度跟踪再好不过了。
项目管理的伟大神话至今都让我疯狂，它的威力远胜过其他任何主题。
这些神话是：1.人们能够按期交付被要求实现的功能（事实上，项目可以按期交付，但人们按期交付功能的概率不会高于击中曲棍球的概率）。
2.有经验的人估计日期比较准（事实上，他们能够较好地估计工作，但不是日期）3.人们必须按照项目预定的日期按时交付项目（事实上，因为人们不能按期交付被要求实现的功能，而你若想你的项目能够按期交付的话，你必须进行管理风险、范围和通过沟通来减轻人性的弱点可能给项目带来的负面影响）。
在本章中，I.M.Wright讨论了如何通过管理风险、范围和进行沟通，来保障你的项目能够按时完成。
前两个栏目专门讨论开发工作的时间安排，接着讨论善后事宜的管理（我们称之为“ Bu9分诊 ”），最后是一篇对死亡行军的声讨，以及一个关于人们为什么要撒谎的哲学栏目。
不得不提的是：通过我在微软多年的工作经历，以及对我所在组织的观察，我发现，项目管理行为和方法在不同规模和抽象层次的组织中，其表现大不相同。
这些层次包括：团队或功能层次（10人左右），项目层次（50～5000人，他们一起致力于某个特定的产品发布），以及产品层次（由高层人员主管的多次产品发布）。
敏捷方法在团队这个层次能够很好地发挥作用，组织方法在项目这个层次比较适用，而长远的战略规划方法在产品这个层次功效显著。
然而，一个人几乎不可能同时在多个层次上工作。
时间长河为每个人将这些经历错开了。
当一个人从一个层次转到另一个层次工作的时候，他可能会想，在以前那个层次上有效的方法在其他层次上应该也同样有效。
灾难就这么产生了！
原因很简单：小型、紧凑的群体跟大型、松散的机构在运转方式上是不同的。
因此要因地制宜，选择最适合的方法。
——Eric2001年6月1日：“ 开发时间表、飞猪和其他幻想 ”一匹马走进酒吧，说道：“ 我能在两天内完成那个功能。
”开发成本计算和时间安排是个玩笑。
相信它的人，要么是傻瓜，要么是初出茅庐的项目经理。
这不是模糊科学，纯粹是杜撰。
不错，的确有人相信编码可以被分割成一个可预见进度和质量的可重复的过程，那我儿子至今还相信牙仙子呢！
事实上，除非你只需编写10行那么长的代码，或者代码可以直接从以前的工作中复制过来，否则你不可能知道编码会花费你多久时间。
作者注：项目经理（ProgramManager，PM）有很多职责，其中最主要的是负责说明最终用户体验和跟踪项目的整体进度。
这种角色是必要的，但他们常常不讨开发者的喜欢，因而也很少得到开发者的尊重。
真遗憾，项目经理是一份很难做好的工作。
但是，对于Wright先生来说，做好项目经理仍然是一个有趣并且容易达到的目标。
里氏震级估计当然，你可以估计，但估计出来的时间是成对数比例的。
有些事情需要花费几个月，有些事情需要几周，有些需要几天，有些需要几个小时，有些则只需几分

钟。
而我跟我的部门项目经理（GroupProgramManager，GPM）一起给一个项目做时间安排时，我们对每个功能使用“困难／中等／容易”3个等级来评估。
“困难”意味着一个全职开发人员需要花费整个里程碑时间；“中等”意味着一个全职开发人员需要花费2～3周时间；“容易”意味着一个全职开发人员需要花费2～3天时间。
这里没有中间等级，也不做精确的时间表。
为什么呢？
因为我们俩知道，我们已经不可能知道得再精确了。
在我的记忆里，除了一系列里程碑、测试版、正式版发布等“项目日期”外，我没有在开发时间表上为各个功能规定交付日期。
一个好的开发时间表应该是这样的——它只是简单地列出在每个里程碑期间需要实现的功能。
那些“必须有”的功能放在第一个里程碑内，并且标上开发人员的数量和“困难／中等／容易”等级；“最好有”的功能放在第二个里程碑内；“希望有”的功能放在第三个里程碑内。
除此之外的所有功能统统不做。
通常情况下，如果到了第三个里程碑内的第二周，你仍然有较多“最好有”、“希望有”的功能没有实现，这时候大家都很惶恐，你就要把所有“希望有”的功能扔掉，并且将“最好有”的功能也只保留一半。
作者注：里程碑的设定因团队而异，也因产品而异。
典型情况下，一个里程碑跨越6～12周不等。
它们被认为是“项目日期”，是组织（50-5000人）用于同步工作和复审项目计划的时间点。
在里程碑期间，各个团队（3-10人）可能使用他们自己的方法来跟踪具体的工作，比如简单的工作条款清单或bum-down图。
风险管理这才是我要引出的主题。
开发成本计算和时间安排不能只盯着日期或时间不放，而应该关注的是风险管理。
我们通过软件的功能和特性来取悦客户，而不管这个软件产品是零售包还是网络服务。
这里的风险是，我们否能在合适的时间、将包含必要的功能集合、并且达到一定质量要求的软件产品交付到客户手中。
一个好的开发时间表通过优先处理关键功能来管理风险。
这些关键功能是能让客户满意的最小功能集合。
通过“困难／中等／容易”这种评级方法，可以判断出在这个最小集合中包含哪些功能才是切实可行的。
其他功能按照优先顺序和一致性原则依次加入。
然后你开始编写代码，并且选择功能实现从困难转向容易，或者从容易转向困难。
你通过平衡资源，以降低不能按时交付高质量的“必须有”的功能的风险。
其他都是次要的，有则锦上添花，没有也无妨，况且你还可以设立不重要、但又不失挑战性的项目交给实习生去做。
作者注：具有讽刺意味的是，几乎所有工程师和经理都赞同要优先处理“必须有”的功能，但事实上很少有人真的这么做，因为“必须有”的功能通常是乏味的，比如安装、建造、向后兼容性、性能和测试套件等。
然而没有这些功能，你的产品根本就发布不了。
因此，产品发布往往是因为这些领域的问题一拖再拖。
一定要破除“功能交付日期”的神话，因为开发人员专注于这种日期的时候会破坏风险管理。
真正要关心的日期只能是“项目日期”，比如各个里程碑、测试版发布等，而绝不应该是“功能交付日期”。
项目日期之间一般都有较长时间的间隔，而且不会很多。
管理这几个日期要容易得多。
如果要求开发人员在某个日期之前一定要实现某个功能，当他们不能按时完成时他们往往不会告诉你

，而是对你说“我正在加紧做……我会加班……”之类的话。
在软件开发过程中进行风险管理，我们还要特别注意以下几个因素：一个是过度劳累的员工，一个是匆匆忙忙实现的、质量很差的功能，再一个就是你花费几周的时间、动用2～3位甚至更多的高级开发人员去解决一个棘手的问题。
如果你的开发人员是在围绕“功能交付日期”付出大量的努力，而不是帮助你在产品的关键功能上实现降低风险，那么那些时间真的就被浪费了。
客户赢了一个产品的成功与否，取决于你对关键功能的风险管理能力。
当你给开发团队解释清楚这一点之后，情况就完全不一样了。
当然，额外的功能可以锦上添花，但最关键的还是要专注于存在风险的领域，充分沟通，并一起努力把它们解决掉。
当所有人都理解了目标，所有人都能比以前工作得更好。
每个艰巨任务的完成都能鼓舞士气，即使新员工也会因为成熟的决议而得到回报。
最终，我们的客户是大赢家，因为他们得到了真正想要的功能，并且产品质量也是他们当初所期望的，而不是一些勉强实现的、质量不能保证的垃圾。
顺便提一句，我对开发时间安排的所有论述，对于测试时间安排同样适用。
2001年10月1日：“竭尽所能：再论开发时间表”该对我6月份的那个栏目（“开发时间表、飞猪和其他幻想”）的评论做出一些回应了。
其实，大部分评论都是恭维之词，这里就不再赘述，因为没有必要再次证明我有多么正确。
我这里要做的是，去帮助一下那些对那个栏目还在无知中徘徊、但又非常热情的读者朋友们。
软件工程绝对是含糊的我对关于不能也不应该对一个功能的开发做时间安排的论断表示怀疑。
文中的论述精确地描绘了“编码”活动。
不幸的是，这是初中生干的事情，类似于他们拼凑一个VB程序来解密信息、相互通信。
我们可是软件工程师啊，不是电脑苦工。
——一个充满怀疑的无知者作者注：这是我仅有的一个“邮件箱”栏目，收集了我对一些读者来信的回复。
我还在持续不断地收到读者对我的栏目的大量“反馈”，但一旦一个栏目很受欢迎，很多新的话题便会涌现出来；讨论那些新话题的价值要远远超过对一个老话题邮件的回复。
不管怎么样，当我回顾这个早期的栏目时，我意识到，可能Wright先生应该再次清空他的邮件箱了。
我经常听到这种说法，但请就此打住。
银行经理并不管理银行，软件工程师也不在软件上做工程。
他们开发软件，定制软件，通常随意性很大，对所谓的操作范围、公差、故障率、压力条件等没有明确的度量标准。
的确，我们的系统有这些标准，但这些标准不是为软件编码准备的。
我曾到一个工程学校进修过。
我的朋友当中也有很多是电力、基建、航空、机械等方面的工程师。
工程师做的项目，其建造模块和建筑流程都经过了很好的定义和提炼，而且都是可预测的。
虽然有时候为了达到客户的要求需要一个优雅的设计，像写小说一样把各个模块创造性地组合在一起，但最标新立异的建筑也会符合一定的公差要求，并且具有严格的可控质量和功能。
但对软件开发来说，情况就不一样了，尽管很多人竭力想让这两者达成一致。
软件的各个构造模块太底层了，变数太多。
它们之间的交互影响太难预料了。
像Windows、Office、VisualStudio、MSN等大型软件系统的复杂度，已经远远超过了工程的正常范围，以至于哪怕只在这些系统中做微小的功能改动，也无法粗略估计出这些改动所引起的“平均失效时间”。
因此无论好坏，还是抛开痴心妄想和崇高理想，回到现实中来吧！
我们必须承认，我们是开发者，而不是工程师。
我们不能奢望轻易得到传统的工程领域积累了成百上千、甚至成千上万年的经验才能做到的“可预测

性 ”。
这无异于我们奢望：不跟电脑说什么，而电脑却能按照我们心里的想法去做事。
我们还办不到！
作者注：在我写下这个栏目6年后的今天，微软已经对其很多软件进行了“ 平均失效时间 ”的评估。
除此之外，把编程当作工程看待的各种方法也逐渐出现了。
这个我会在第5章的“ 软件发展之路 ”栏目中再次介绍。
纵然如此，我仍然认为本栏目很好地见证了软件开发作为一个领域，他已经走过了幼年，但跟他早已长大成人的传统工程兄弟相比，他还只是个10几岁的小朋友的现状。

<<代码之道>>

编辑推荐

《代码之道》是《代码大全》姊妹篇！
是微软公司内部所有工程师的必读之书！
了解未经掩饰的真相：怎样提高软件的质量和价值，从设计到安全；怎样切合实际地管理项目的时间表、风险和规范书，怎样为常见的低效率开发瘦身，怎样应用过程改进方法、避免固执盲从，怎样驱动一个成功的、令你自己满意的职业生涯，怎样不变成暴君、发展并管理一个欣欣向荣的团队！

版权说明