PEARSON Addison Wesley

12th Annual SOFTWARE Development PRODUCT EXCELLENCE AWARD

软件工艺

Software Craftsmanship

Foreword by Dave Thomas

〔美〕Pete McBreen 著
熊节 译

人民邮电出版社
POSTS & TELECOM PRESS

软件工艺

[美] Pete McBreen 著

熊节 译

人民邮电出版社

软 件 工 艺

◆ 著　　[美] Pete McBreen
译　　熊　节
责任编辑　俞　彬　陈冀康

◆ 人民邮电出版社出版发行　北京市崇文区夕照寺街 14 号
邮编　100061　电子函件　315@ptpress.com.cn
网址　http://www. ptpress.com.cn
读者热线　010-67132705
北京汉魂图文设计有限公司制作
北京鸿佳印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：880×1230　1/32
印张：8.625
字数：171 千字　　2004 年 5 月第 1 版
印数：1-6 000 册　　2004 年 5 月北京第 1 次印刷

著作权合同登记　图字：01 - 2001 - 3675 号
ISBN 7-115-12243-1/TP • 3954

定价：19.80 元

内容提要

本书针对软件开发，提出了一些相当棘手和敏感的问题，并给出了颇具争议性的结论：从一个数百年来一直兴旺发达的系统——工艺学中获得启示，寻找答案。

本书用5个部分共19章的篇幅，系统地阐述作者的观点，并试图回答一直困扰着软件行业的难题——我们应该如何重组软件构造的过程，使其能够如我们所愿地有效运转？第1部分共4章，对传统的观点提出质疑——软件工程真的是解决软件开发问题的灵丹妙药吗？第2部分共2章，这一部分提出了本书的观点，即以软件工艺的视角看待软件开发。第3部分以7章的篇幅，从不同的角度全面地展现了软件工艺理论所带来的主要变化，以及如何实践这个观念。第4部分共3章，对比了软件工艺与软件工程，并为各自适用的范畴重新划定了界限。第5部分共3章，分别讨论软件开发中的权宜之计和长期问题。

本书荣获2002年度Jolt图书大奖。阅读本书，有助于引发读者在软件开发问题上的独立思考，本书适合软件行业的所有从业人员阅读参考。

中译本序

看见“软件工艺”的思想在全世界得到日益广泛的认同，我欣喜若狂。这本《软件工艺》，是为软件开发的工艺学送上的赞歌。在过去的 30 年中，很多人试图把软件开发变成一种机械化的行为。尽管如此，优秀的开发者们仍然知道：真正起决定性作用的，还是编写软件的人，是他们的技能和经验。没错，现在有很多精良的工具和技术，但最关键的还是使用这些工具的人，是他们的才华。

从事软件开发的企业为什么会把这一切置之脑后呢？我不知道。不止一次，我看到早期的文献中强调“软件编写者的能力”的重要性。软件工艺的思想毫不新鲜，在我着手撰写本书之前，Jim Coplien 和 Steve McConnell（以及其他很多人）早已在各自的著作中提到过它。掌握软件开发中的工具和技术需要耗费很多时间，但在那之后，开发者还需要学会并习惯交付优秀的软件。

在过去的一段时间里，人们似乎太热衷于追寻一蹴而就的终南捷径，静听“掌握软件开发技艺”的传统似乎已经不再流行。但是，时间早已证明，只有精通自己的技艺，才是获得成功的不二法门。对于软件开发之外的很多行业，这个道理是众所周知的。即便是批量生产的标准件取代了手工作坊，人们在内心深处依然坚信：哪怕是司空见惯的日常用品，也只有工艺大师才能做出精品。

自出版以来，本书已经先后被翻译为日文和韩文，现在又有了中文译本。本书中的思想与西方的工艺学传统、学徒传统有着紧密的联系，看着这些思想被如此贴切地翻译到另一种文化中，是一件颇有兴味的事情。曾经有很多人错误地认为软件开发是一项机械性的行为，希望本书能帮助你避免重蹈覆辙——这就是我给本书读者的祝愿。

Pete McBreen

2003 年 8 月

译者序

时隔一年之后，当我再次到 AMAZON 寻找这本《软件工艺》时，它的评价不知何时已经悄悄地变成了四星半，销售排行也已经到了 5 000 多名——熟悉 AMAZON 的读者应该知道，在 AMAZON 能得到四星半评价的已经是精品好书，四位数的销售排行就更能证明它的品质。可是，一年前看到的批评仍然历历在目："过时的例子"、"混乱的逻辑"、"东拉西扯不知所云"……当这些尖锐的言辞与"所有软件管理者的必读书目"一类溢美之词并列时，我不得不再次认真地思索：这究竟是一本怎样的书？

Pete McBreen 说，这本《软件工艺》是一本"极具煽动性"的著作。而在我看来，用"煽动性"来评价它只嫌太客气，或许"颠覆性"会是一个更贴切的词。自 1968 年 NATO 会议以后，软件工程的话语就始终把持着软件业（以及软件学科）的言路。一切的软件问题都不由自主地被归咎于"软件危机"，同样自然地，一切的解决方案都不由自主地被划入"软件工程"的范畴。从记事以来……呃，我是说，从上大学以来，我们的一切讨论都围绕着软件工程展开：这样做是否符合软件工程？如何对软件工程加以改进？企业应该如何开展软件工程？凡此种种。在言词的不断重复与变调之间，"软件工程"逐渐被捧上了神坛，成为一种信仰，并因此失去了它旧有的价值与意义。君不见，即便是软件工程的反对者，也只能说出"我们不要软件工程"这样的话——仍然未

脱软件工程的话语霸权。

Pete McBreen 是一个极其敏锐的人，并且对语词背后的意蕴有着深刻的体会和认识，这或许与他年少时在英国所受的教育有关吧。他一针见血地指出：软件并不是工程，“软件工程”仅仅是一个多少有些不够贴切的隐喻而已。是的，一个隐喻，正如我们常说的“桌子腿”、“针眼”一样，这是一个深入人心、渗透极广的隐喻。但是，尽管每个人都知道绣花针并没有长眼睛，我们却常常忘记了软件工程作为隐喻的本来身份，真心诚意地把软件作为一种工程来对待了。软件工程的困境与读到这本《软件工艺》时本能的拒斥，殆出于此。

钱钟书曾说，反其道以行也是一种模仿。而对于目前软件工程的反对者们，另一个更恰当的比喻是“反转的胶片”——胶片的颜色与照片完全相反，但两者记载的信息却是毫无二致。作为一个真正意义上的颠覆者，Pete McBreen 为软件开发找到了另一个隐喻（以及随之而来的另一套语词），那就是本书的标题——软件工艺（software craftsmanship）。

对于这套工艺学的语词，我一直有着淡淡的隐忧。在西方，工艺学传统多半与文艺复兴的人本主义联系在一起，而在对现代性的反动中，文艺复兴一直是吸引大众的旗帜。因此，可以想象，“软件工艺”这样一个隐喻对于欧美程序员有着不可抵挡的诱惑。而在中国，“学而优则仕”的传统价值观固有地鄙薄工艺学的实践者，“五四”以后，现代性的话语又早已根深蒂固，所以我实在不敢乐观地期望这个译本的读者能够让“软件工艺”的思想畅行无阻。好在 Pete McBreen 并不是一个太喜欢夸夸其谈的作者，书中

的论述虽然大胆，却是有理有据。既然 Fred Brooks 的“没有银弹”已经深入人心，相信以逻辑思维见长的软件开发者们能够抛开对软件工程的迷信，随作者一道认识软件工程的局限，并由此生发对软件工艺的思索。倘如此，这本书也就算不辱使命了。

在你开始正式阅读本书之前，请允许我给你打一针预防针：这本书可能颠覆你浸淫其中数年甚至十年的软件观，所以书中的很多观点可能让你感到出离惊奇，甚至出离愤怒。请你不要马上把它扔到墙角去，阅读的过程也就是习惯一种话语方式的过程。当你逐渐习惯软件工艺的话语方式之后，或许能从中找到一些自己需要的东西。

为这个译本，我要感谢 UMLChina 的站长潘加宇，是他把这本书硬塞到我的手上，让我没有与这本精彩的著作失之交臂。我还要感谢我的女友马姗姗给予的帮助，她优雅的文笔弥补了我言辞的生涩，她的支持与鼓励让我能够在工作之余顺利完成此书的翻译。谢谢你，亲爱的姗姗。

最后，希望你能从这本《软件工艺》中找到别样的触动和欣喜——就像我曾经的阅读体验。

熊节

2003 年 8 月 9 日星期六 凌晨

杭州

作者简介

Pete McBreen 是一名独立顾问，对软件开发情有独钟。尽管将很多时间用于写作、教学和顾问工作，但他仍然坚持每年至少在一个真实项目中亲手从事编程工作。Pete 特别善于为软件开发者面临的问题找到创造性的解决方案。在过去的很多年中，他参与了各种正式和非正式的过程改进活动，所以他能够以超然的态度看待软件业普遍存在的问题，并敏锐地意识到："软件开发理应有其乐趣。否则，开发过程就是错的。"Pete 住在加拿大亚伯达省的小镇考昆，没有再回到大城市居住的计划。

译者简介

熊节，普通程序员，喜编程，乐此而不疲。酷爱读书，好求新知。记性好忘性大，故凡有所得必记诸文字，有小得，无大成。胸有点墨，心无大志，惟愿宁静淡泊而已。夜阑人静，一杯清水，几本闲书，神交于各方名士，献曝于天下同好，吾愿足矣。

序　言

本书提出了一些相当棘手的问题。

对于工作量少于 100 人年（developer-year）的项目，软件工程仍然适用吗？软件工程与生俱来的专业化趋势真的是一个好主意吗？甚至，软件的开发真的可以用工程学的术语来描述吗？

本书也提出了一些相当敏感的问题：那些缺乏经验的开发者，是否享受了过高的薪资待遇？那些资深的开发者，是否应该比企业中绝大多数人都拿得更多？那些出现至今还不足十年的工具，是否应该在长期的项目中使用？

在这一切的最核心处，本书提出了一个最大的问题：我们应该如何重组软件构造的过程，使其能够如我们所愿地有效运转？

对于这些问题，本书也提出了一些颇具争议性的答案：本书认为，我们一直对一个简单的事实视而不见——编写软件的，不是硕大无朋的方法学，也不是一丝不苟的组织结构，而是人。在当今软件开发的领域中，我们正在面临一个日益严重的危机。为了扭转这种不利的局面，我们需要培养出优秀的软件开发者。而为了达到这个目的，Pete[1]回顾了一个数百年来一直兴旺发达的系统，并期望从中获得启示——那就是工艺学。

---

[1] 译注：本书作者 Pete McBreen。

“工艺”这个词是一种象征，象征着高质量的产品。但它的含义却还不仅限于此。从完整的意义上来说，工艺学是一个自给的系统。在这个系统中，师傅负责安排学徒的工作和培训，而学徒也有可能取代师傅，每个人地位的高下完全取决于他生产出的产品的优劣。学徒、工人和技师，所有人都在同一个团队中一起工作，并彼此学习。顾客则根据团队的声望来做出选择，而团队也只接受那些他们认为有可能提高他们声望的工作。

在我们的产业中，这个完整的工艺学系统能起作用吗？说实话，我也不知道。许多根深蒂固的习惯都必然会反对这种工艺学的观点。不过，我的确知道，作为学徒接受师傅的言传身教而最终成为高手，这样的成长途径是行之有效的，因为我就经历了这样的成长过程。

应该说我很幸运，进入了一所优秀的大学，在那里我学到了很多的理论（不过，在我上大学的时候，需要学的理论比现在可少多了）。但是，真正让我的大学生活显得与众不同的，还是那段学徒经历。那时，一位研究生带着我一起工作。他并没有明确地要教给我什么，但他用实际行动让我看到了一个优秀程序员的思维方式。月复一月，我在他的身边工作，从他那里吸收了大量实用的知识。不论是设计、编码还是调试，我从他那里学到的知识比在任何一门课上学到的都多。

后来，我又在伦敦加入了一个刚刚开工的项目。在那个项目中，我扮演了另外一种学徒的角色。我的新老板用他的所作所为告诉我：软件开发不仅仅是一项技术工作，同时也是一项人文工作。他帮助我理解了软件开发中与商业相关的这一部分，并教

给我如何在拥有一定技术实力的基础上建立起良好的企业人际关系。

从这两次截然不同的学徒经历“毕业”之后，作为一个开发者，我已经比开始时强了太多太多。由于拥有这样一段经验，所以我完全相信：如果要学习一门手艺，与高水平的师傅一起工作是最佳的途径。

正如我刚才说的，本书在“如何训练下一代开发者”方面提出了很有价值的观念。不仅如此，本书还涉及到了软件的哲学。工艺学所针对的是个人的能力：对你的工作，对你个人的发展，对于你的职业，工艺学能给你更多的帮助。它并不干涉你开发软件的工作方式。你可能在通过了 CMM 5 级的企业中朝九晚五循规蹈矩地工作；也可能为了实现又一个惊天动地的新点子而一周工作 100 小时，靠咖啡因来驱散倦意。你可以使用 RUP、XP 或者 SCRUM——或者根本不使用任何标准的过程。不管工作如何组织，只有当技艺高超的开发者编写出高质量的、合乎需要的代码时，只有当他们满足客户的需要时，整个软件开发过程才是真正有价值的。方法学无法造就技艺高超的开发者；只有给予工艺学以足够的认可并按照工艺学的指导去实践（当然，还需要留意本书中的其他观点），技艺高超的开发者才可能出现。如果你能花一些时间来阅读本书，来思考 Pete McBreen 提出的难题，必定会对你自己和你的职业生涯大有裨益。

David Thomas

The Pragmatic Programmers

致　谢

或许你不会相信，本书的灵感来自软件工程研究所（Software Engineering Institute，SEI）。在20世纪90年代早期，我四处寻找提高小型团队生产率的办法，能力成熟度模型（Capability Maturity Model，CMM）看上去是一个可行的起点。但是，我最终认识到：对于绝大多数项目，人的天赋比使用的过程更重要。

很快，我又发现，那并不真的是天赋。优秀的开发者之所以优秀，是因为他们知道如何创建优秀的软件，并且不断学习开发优秀软件的新方法。随后我意识到自己的思想得益于很多优秀的软件开发者和技术作家：Gerald Weinberg、Luke Hohmann、Capers Jones、Jim Coplien……他们的著作占据着我的书架每次读他们的书，总会让我不时掩卷沉思“如何开发优秀软件”这个问题，他们是我心目中的明星。

这本书的写作历时经年。是Alistair Cockburn最早让我明白：软件开发者能够、也应该写书。Alistair还鼓励我成为Addison-Wesley的一名审稿人，这也最终间接促成了本书的问世。

另一个间接的影响来自Catapulse Inc.的Rebecca Bence，她鼓励我在Catapulse网站上撰写了“软件工程 vs. 软件工艺”的文章。在这篇文章中，我对“工程”和“工艺”这两个隐喻做了对比。本书第3章的很多思想就来自这篇文章。

非常感谢Addison-Wesley的编辑团队——Mike Hendrickson、

Heather Olszyk、Ross Venables 和 Marcy Barnes。本书的审稿人包括 Ken Auer、Jim Bird、Bruce Brennan、Martin Fowler、Alastair Handley、Andy Hunt、Kim Kelln、Greg Klafki、Michael Le Feuvre、Miroslav Novak、Dave Thomas、Bruce Wampler 和 Frank Westphal，他们给了我无数的好建议，也对他们表示感谢。

最后，我要感谢我的妻子 Lesley，是她让我明白：应该停止抱怨软件工程的现状，潜心针对这个问题写一些东西。感谢你的爱和支持。

Pete McBreen

前　言

用“工艺学”来比喻软件开发，这可以看成是对软件开发的一次追根溯源：优秀的软件开发者们一直都知道，编程的确就是一门工艺技巧。不论软件开发者拥有多少纷繁芜杂、晦涩难懂的知识，但最终左右着应用程序开发的仍是那种不可言说的感觉和经验。举个最简单的例子：也许有人能够了解 Java 语言所有深奥幽暗的技术细节，但除非这个人能培养起自己对于软件的审美感觉，否则他永远无法真正精通应用程序的开发。而与此相反，一旦某个人获得了那种软件开发的感觉，特定的技术细节就变得几乎无关紧要了。优秀的开发者总是在不断地学习、使用最新的技术和技巧，对于他来说，学习一门新的技术只是软件开发者生涯中的家常便饭而已。

“软件工程”这个词是由 NATO 属下的一个研究组在 1967 年提出的，这个研究组还提议召开一次会议，专门讨论“软件所面临的问题”。1968 年，由 NATO 科学委员会主办的这次会议在德国 Garmish[1]召开，会议提交的报告就被命名为《软件工程》[2]。在这份报告中，Peter Naur 和 Brian Randell 这样写道：“我们之所以选择了‘软件工程’这个颇具争议性的词，是为了暗示这样

---

[1] 译注：Garmish，位于阿尔卑斯山区，德国度假胜地。

[2] Peter Naur、Brian Randell 编辑，《软件工程》(Software Engineering: A Report on a Conference Sponsored by the NATO Science Committee)，NATO，1969。

一种意见：软件的生产有必要建立在某些理论基础和实践指导之上——在工程学的某些成效卓著的分支领域中，这些理论基础和实践指导早已成为了一种传统教义。”

和他们一样，本书之所以选择了这样一个颇具争议性的书名（并提出了很多颇具争议性的观点），是为了暗示这样一种意见：软件开发的实践者们有必要开始关注软件开发的工艺。软件工艺是如此重要，因为它让我们摆脱“制造业”的隐喻（这个隐喻正是软件工程所带来的），并让我们开始关注从事软件开发工作的人。软件工艺带来了另一种隐喻：拥有技术的软件开发者抱定决心要掌握自己所从事的工艺，对自己的劳动成果负责并以之为荣。

软件工艺并非与软件工程或者计算机科学针锋相对，格格不入。与科学和工程学相比，软件工艺是另一种完全不同的教义，但又能与这两者很好地共存，并从中获益。现代的铁匠能够因为更好的工具、原料和知识而获益；同样，软件工艺也能够因为更好的计算机、可复用的组件和更先进的编程语言而获益。铁匠们在自己的工作中融入了技巧和艺术，从而超越了科学和工程学的范畴；同样，软件工艺能够指导开发者生产出优秀的应用程序及系统，因此也可以超越计算机科学和软件工程学的范畴。对于这一论点，最好的佐证大概就是 UNIX 和现在的 GNU Linux 了——这两个系统之所以能够获得如此巨大的成功，完全是因为它们的创建者拥有精巧的手艺、高超的技术和无私的奉献精神。

很长的时间里，人们一直试图强迫商用软件的开发适应软件工程的要求。这种削足适履的做法引发了不少的问题，而软件工