软件工程师实战基本功

侯清富　郭　岗　编著

人民邮电出版社
POSTS & TELECOM PRESS

软件工程师实战基本功

侯清富　郭岗　编著

人民邮电出版社

图书在版编目（CIP）数据

软件工程师实战基本功 / 侯清富，郭岗编著. —北京：人民邮电出版社，2005.1
ISBN 7-115-12856-1

Ⅰ. 软… Ⅱ. ①侯… ②郭… Ⅲ. 软件开发—工程技术人员—基本知识 Ⅳ. TP311.52

中国版本图书馆 CIP 数据核字（2004）第 134318 号

内 容 提 要

本书旨在指导从事软件编程工作不久的软件工程师，在实际工作中通过学习积累经验并掌握技能，成为一名称职的软件工程师。本书内容包括适应软件过程的要求、设计技术方案、编写高质量代码、代码缺陷复查、程序调试与优化、编写高质量文档、版本控制、软件质量控制和团队协作等基本功。本书的每一章对应于软件工程师要掌握的一项基本功。

本书以软件工程为指导，讲解各项基本功的关键技术要点，具有很强的实用性和可操作性，适合于信息专业在校高年级学生、软件工程师和软件项目管理者阅读。

软件工程师实战基本功

◆ 编　　著　侯清富　郭　岗
责任编辑　陈万寿

◆ 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　010-67129258
北京市朝阳展望印刷厂印刷
新华书店总店北京发行所经销

◆ 开本：720×980　1/16
印张：8.5
字数：195 千字　　2005 年 1 月第 1 版
印数：1-4 000 册　　2005 年 1 月北京第 1 次印刷

ISBN 7-115-12856-1/TN・2365

定价：16.00 元

前　言

像所有的职业一样，软件工程师在不同的职业阶段，有着不同的内心冲突。青年学生在刚刚走出大学校门时，尽管已掌握了计算机原理、程序设计和软件工程的知识，并写过不少程序，但由于缺乏实际工作经验，较难找到称心如意的职位；等有了职位后，一般需要面临半年甚至更长时间的考察期。在这半年多的时间里，只好谨小慎微地工作。在企业里工作两年左右，胜任工作没有问题，可个人能力却没有长进和提高，为个人的职业发展感到担忧。

本书的目的就是希望能帮助年轻的软件工程师，在实际项目中锻炼两年左右，在现职的岗位上自我充电，尽快成长为一名称职的软件工程师。

本书的全部内容来自于作者的工作笔记。在近5年的时间里，作者一直领导一支40人左右的核心队伍，从事大型通信软件系统的设计与开发工作。在这5年期间，研发的系统取得了极大成功，上百套软件系统在互联网的重要通信设备上运行，为全国各地的网络用户传送IP电话、视频点播、网络游戏等多媒体信息。在行业竞争的过程中，一支能征善战的软件开发队伍同时被锤炼出来。在1999年刚组建软件项目团队时，作者很快发现几乎所有的新员工，尽管有学士或硕士学位，对如何按软件工程的要求编写软件却知之甚少。当时曾请过国内知名的软件工程专家进行培训，讲授面向对象的设计方法，软件配置管理等专业知识，但收效甚微。后来发现，是因为培训的内容不切实际，不少软件工程师并不是缺乏书本上的知识，而是不知道如何将书本知识联系工作实际。迫于工作的实际需要，作者和同事们坚持在战斗中学习战斗，通过自身的实践摸索出实用的、可操作的、书本之外的软件工程经验。

2002年，作者主持了一个以个体软件过程为主题的内部培训，自己编写培训教材。这次培训满足了软件工程师的实际需要，深受大家的欢迎。

2003年，作者在原讲稿的基础上，对内部教材进行了结构性的调整。相关专家直接参与了新版培训提纲的修改和审订，并组织了专门的评审会。作者重新编写了培训教材，增补了一些代表性的工程案例，同时将修订后的教材用于内部培训。读者所看到的这本书，是作者在2003年培训的基础上，吸取了同行专家的意见后编撰而成的。作者认为这些内容已经经受了实践的检验，可以供软件业的同行参考。

本书的内容强调通俗易懂，可实际操作性强，能学以致用。它的重点在于帮助有关信息工程项目的软件工程师了解项目开发中所面对的任务和困难，并学会运用已掌

握的知识，完成这些任务，克服面临的困难。作者过去5年多的实践，已经使上百人受益，这为本书的实用价值提供了一个很有说服力的注解。

作　者

目　录

第 1 章

软 件 过 程

“软件过程”4 个字，被一些软件工程师当成一个不实用的词汇。他们认为，“软件过程”是死板、苛刻而没有效率的。

可能超出他们的预料，软件工程师的第一项基本功就是适应软件过程。

1.1 软件开发基本功

在软件工程师的求职简历中，很多人都强调自己会多种编程语言，写过上千行的程序代码，声称自己有丰富的软件开发经验。真是如此吗？是否写过代码就会开发软件？

可以肯定地说，会编程不等于会开发软件。从编程与开发软件的关系来看，编程是开发软件的一个环节，不是开发软件的全部。开发软件除了编程之外，还有需求分析、方案设计、系统联调、系统测试、版本发行、质量控制、计划和进度安排等等重要事务。编程只是软件开发整体工作中的一小部分。

要更清楚地理解编程和开发软件的实质性不同，下面我们不妨做一个简单的比较。

开发软件和编程所追求的目标不同。开发软件在满足用户需求的同时，要通过实施软件过程，保证按时交付软件、控制质量和降低成本。项目延期意味着财力和物力的投入有超出预算的风险，质量没能达到要求必然导致软件维护困难和增加投入成本，有效地控制成本才能获得更高的收益。编程追求的目标是程序能完全正确地实现和执行设计者的意图。在软件工程师实际的工作中，让程序能正确地工作比较容易，做到完全符合设计者的意图就很难。

从取得成功的角度来说，要使开发软件取得成功，关键在于有适合团队特点的软件过程。因为一个人的能力毕竟有限，团队的动作要靠软件过程来协调和统一。相对来说，编程主要靠个人的经验和技巧，个人的因素起主导性的作用。

软件工程师要胜任自己的岗位，除熟练使用编程语言外，还需掌握更多的基本功，

包括具有编写有质量保证代码、复查代码缺陷、调试代码缺陷的技术技能，以及团队协作的非技术技能，等等。如果把这些基本功比作演员具备的技能，软件过程就是支持他们表演的舞台。

软件工程师的第一项基本功是适应软件过程。

1.2 软件过程的作用

“软件过程”这个词被一些软件工程师当成一个不实用的词汇。这些人认为“软件过程”是死板、苛刻而没有效率的。这种看法的论点是，最好的项目执行方式是聘请能找到的最佳人才，给予他们要求的所有资源，然后放手让这些人做他们最擅长的事情。照这种观点，不受任何软件过程约束的项目才能够特别有效率。抱有这种论点的人承认“工作脱节”或非生产性的工作占有一些份量。工程师会做错事，他们承认这一点，可是这些错误可以很快有效地弥补过来，并且这比执行“软件过程”所花费的成本要少。依据这种观点，在项目中加上软件过程约束不但是多余的，还会耗去生产性工作时间。

这种说法有着直觉性的吸引力。在项目开始，对执行软件过程的关注会用掉一些生产性工作时间，如果这种情形从头到尾持续下去，继续花时间去执行软件过程就不合理了。不过，根据软件业界的经验，在中型项目中，开始没有建立有效的软件过程的项目，在一段时间之后还是被迫建立软件过程，而且花费时间更多，而得到的好处却更少。

一个起初就对软件过程漠不关心的项目管理者，会让工程师在此后觉得他们把时间花费在开会跟修正错误上，而非用在加强软件功能上。这些软件工程师知道项目进度脱节了，当他们发现不能满足时间底线时，他们的自我保护念头开始萌发，使他们退回“单独开发模式”——只求满足个人的时间低限，避免没完没了的加班。他们不再跟项目经理、客户、测试人员等开发团队的其他人打交道，使得项目纪律荡然无存。

当工程师最具生产力时，他们的感觉也最好。好的项目要建立清楚的目标远景，并利用软件过程让工程师们觉得自己拥有不可思议的生产力。工程师讨厌阻碍工作的各种困扰，这些障碍大部分是因为眼高手低、毫无章法所造成的，软件过程起到了一个清理障碍的作用。正如 IBM 的资深经理人所言：“尽管人们还是觉得在开发项目时应保持一定程度的自由，不过我认为，一些基本的做法应渗透到整个团队。正是这些基本做法推动整个项目前进与发展。”

一个软件项目必须为产品开发确定一个软件过程。事实上，由于产品竞争的客观需要，任何企业也承受不了因重复设计错误、漠视用户抱怨与建议或纵容项目进度失

控所付出的代价。软件产品市场很难使软件企业能承受起一次软件产品出品的失败。正如比尔•盖茨所说：“开发和软件发行永远是一对矛盾。开发软件的目的是发行软件。除了产品质量外，还会有许多市场的考虑会影响软件发行决策。致命的错误以及由之引起的版本升级不但会损坏公司的信誉，并可能导致法律诉讼。当我们送软件光盘去生产，只是生产大量的软件拷贝。DOS6 被发布去生产时，我们只是生产了 200 万份软件拷贝。我们把这一产品以大约 45 美元的价格进入销售渠道。如果我们必须把它更新，那么所有的利润会‘呼’地一声烟消云散。如果不得不经历一次产品回收，那是很荒谬的。”确定一个软件过程，就是要将产品开发的成本和风险控制在许可的范围内。

对微软来说，要让软件产品足够可靠以使其他公司能够购买，还要让产品足够简单以使初级消费者都能理解，这是一个极大的挑战。微软产品开发的关键之处是坚持被称为“同步与稳定”的软件过程。每日通过产品构造保持同步，定期实行里程碑式稳定、不断的测试。微软的“同步与稳定”的软件过程，既使自己比其他公司更好地适用了对 PC 软件的爆炸性需求，又使庞大的公司能像开发小组一样有效地工作，有足够的灵活性快速响应市场反馈。

1.3　瀑布式软件过程

所有软件工程的书，都会讲到瀑布式软件过程。尽管瀑布式软件过程不是很实用，但能让软件工程师便于理解和掌握开发软件的规律。

在过去的 10 多年中，研究者们指出了开发软件所要涉及的主要工作，大致上分为 7 个环节。

第 1 个环节是需求分析。主要是了解用户的需求，最终要落实到一份或一套需求文件中，在文件中描述用户所要的软件系统是一个什么样子。

第 2 个环节是结构设计。通过结构设计，提出一个解决方案，来满足用户的需求。

第 3 个环节是详细设计。对于需要组建一个团队来开发的软件系统，一个系统可划分成子系统，子系统要划分成模块。详细设计就是定义模块中的接口、数据结构和算法。

第 4 个环节是编程和调试。这是软件工程师在大学里学到的最主要技能。

第 5 个环节是单元测试。确保所做的模块是可以用的，基本上没有错误，或没有常见的错误。

第 6 个环节是系统测试。两个模块都没有错误，但它们合起来并不能肯定没有问题。通过了系统测试，软件就可以交付给用户使用了。

第 7 个环节是软件维护。用户使用软件后，仍会发现软件有各种各样的问题。一

个可能的原因，是软件的错误以前没有发现，在用户使用过程中出现了，需要排除这些错误。还有一个可能是市场发生了变化，用户又有了新的需求。软件维护阶段需要及时有效地处理这些问题。

对于经验不多的软件工程师，总会感觉到实际需要做的事情比想象的还要多。当不能理解到为什么有这么多事情的时候，最好的选择是接受而不是排斥这些事情。如果排斥它们的话，你的进步就会慢一些。因为软件过程反映软件开发本身的规律，规律本来就是这个样子，排斥它们只会阻碍自己的进步。所以这里给读者的一个建议是，先去接受这些还不太理解的东西，之后在具体的实际工作中去体会和消化，这样进步就会快一些。

瀑布式软件过程理解起来比较单纯，却不便于在实际工作中操作和执行。原因有多个方面，其中一个主要原因是瀑布式软件过程有一个假设，就是前一个环节的工作全部完成之后，才开始第 2 个环节的工作。但这样的假设与实际情况是不相符的。以需求分析为例，做需求分析是很困难的，很难说得清楚需求分析的结果是否已没有问题了。如果等到需求分析没有问题了，才开始结构设计，那就不知道要等到何时才能开始。现实的情况是，需求分析中总会存在各种各样的大大小小的问题，甚至有很严重的问题，如果按瀑布过程去操作的话，是根本不可能的。

在做实际的商业软件时，没有哪个开发团队会完全按照瀑布软件过程去做事情的。基本上可以说，瀑布式软件过程是不可管理的，无法有效地控制项目开发的质量和进度。

针对瀑布式软件过程的局限性，人们提出了增量式软件过程。

1.4 增量式软件过程

在增量式软件过程中，将瀑布式软件过程的前 6 个环节联结成一个封闭的环形。这个封闭环的意思是，在开始需求分析时，尽管不能把握好全部的需求，但总是可以把握好其中的一部分。例如，要开发一个通信网关软件系统，尽管开始时不太可能把计费的需求搞得很清楚，但把两个网络联通起来的需求还是比较容易把握好的。针对已经把握好的这一部分需求，去做结构设计、详细设计、编写代码、做好调试，最终将系统交付给用户使用。用户在使用过程中，发现网络联通来起来没有问题了，但发现有时网络的速度不快，忙时流量拥塞比较严重。这时进入第 2 次循环，分析需要加进一些什么样的功能，以保证网络 24 小时通畅，不会出现流量拥塞。循环一次，就是增量一次。通过一次循环，解决一部分问题，满足一部分用户需求；再循环一次，解决一部分新的问题，满足用户新的需求。增量式软件开发的用户需求是逐步增加的。

增量式软件过程是20世纪70年代早期由软件工程之父Mills提出的。Mills提出，任何软件系统都应按增量的方式来开发功能。也就是说，系统首先能运行，即使它无任何实用功能，只是调用了一系列伪子系统。接着，系统的功能一点一点地被充实，子系统轮流被开发，或者是在更低的层次调用程序、模块、伪程序（Stub）等。

与瀑布式软件过程相比，增量式软件过程表现出很多工程操作和实施方面的优点如下所述：

一是有效地控制进度。因为要满足的需求已经很规范，并且被软件工程师正确理解了，因而能比较准确地估计开发进度，而不是像瀑布式过程那样猜测开发进度。

二是能有效地吸纳用户的反馈意见。如果没有一个系统供用户使用，用户就会比较随便地提出自己的需求，因为他们自己也并不十分清楚需要一个什么样的软件系统。但是有一个系统给用户使用时，他们提出的需求会更有针对性，描述起来也会准确得多。

三是有效地控制质量。如果采用增量式过程进行项目开发，方便进行统计质量管理和变更管理。在瀑布式软件过程中，这一点是做不到的或很难做到的。

在开发大型软件系统时，增量式软件过程的优点会体现得更明显。因而，增量式软件过程受到工程界的喜爱和重视。在《人月神话》一书中，作者弗雷德里克•布鲁克斯对增量式软件过程给予了极高的评价。其中特别提到，增量式软件过程对开发士气的推动是非常直接的。当有一个可运行的系统，即使是一个非常简单的系统出现时，开发人员的热情就迸发出来了。当一个新图形软件的第一幅图案出现在屏幕上时，即使是一个简单的长方形，开发人员工作的动力也会成倍地增长。在开发过程的每一个阶段，总有可运行的系统，即使在短短的4个月内，采用增量式开发的效果也是非常明显的。

1.5　软件过程的具体体现

软件过程最终会落实到一系列具体的技术规定、质量要求和其他规章制度等。其中与技术开发工作直接相关的制度性文件，就是作业指导书了。简单地说，作业指导书是规定软件过程中某些子过程的执行步骤。例如《软件编程作业指导书》，会重点规定软件编程的作业流程，就编程前的准备工作、编程进行中的注意事项以及编程完成后的质量验收等内容给出指导性的要求。

可能大家有一个顾虑，就是按作业指导书来做事情的话，会极大地限制做事情的自由。所以，很多时候开发人员不太愿意按照作业指导书来做，往往视作业指导书为一个负担，而不是前进道路上的一根拐杖。

可以从两个方面来分析这样一个顾虑。一个方面的原因是，作业指导书本身的可操作性不够好，所以操作起来不方便，常常碍手碍脚。另一个方面的原因是，有些做事情的好方法自己还没有体会到，与自己的想法不一致。他们因为没有完全地理解和体会，所以不太愿意接受作业指导书所要求的做法。

个人如何对待这些作业指导书？这里给读者提的一个建议是：尽管感觉作业指导书不太好用，仍要坚持按作业指导书来做手头的工作。为什么呢？因为写一份作业指导书并不容易，它要归纳以往的工作经验，特别是吸取犯过错误的教训。如果抛开作业指导书，很容易犯以前别人犯过的错误，这是不值得的。反反复复地犯别人或自己犯过的错误，这样的现象在软件开发中并不罕见。开发人员创造能力强，会非常有热情去探索自己的道路，不愿意走别人已走过的路。这本身并没有什么不好，但积极有效地吸纳前人的经验教训也非常关键。其中重要的一点，就是知道别人已经有了什么成功的经验和失败的教训，从而吸取他人成功的经验，避免别人的教训在自己的身上再重复一次或多次。作业指导书能够做到这一点。

作业指导书也是保证开发质量的措施。事实上，执行作业指导书的一个重要出发点，就是要保证软件开发的质量。质量管理的一个基本思想是，在做任何一件事情时，如果落实了所要要求的质量管理要素，质量就能得到保证。作业指导书已将质量管理要素与具体的技术操作结合起来形成了作业步骤。如果不按作业指导书操作，意味着某些质量要素将得不到落实，也就无法保证软件质量。

作业指导书还有一个非常重要的作用是限制了沟通中可能出现的误解。开发人员开发软件过程中，做到有效的沟通是相当的不容易。因为他们脑子里所想的与嘴里所说的，说出去的意思与别人听到后理解到的意思，常常相距很远。通过约定的步骤来约束做事情的方式方法，沟通起来就会方便得多，也是提高工作效率的简单可行的办法。

最后一点是，作业指导书会规定做事的统一风格。以编写技术文档为例，如果没有作业指导书的指导，技术文档的形式和内容就会是五花八门。作业指导书会提供一些文档模板，从而约定了做事的统一风格。

第 2 章

软件系统设计

“软件设计”理解起来并不困难，是指把项目经理手中的一份软件系统的定义文件（称为技术规范或产品定义文件）转变成可以在计算机上运行的软件系统。软件设计的目标是，在满足软件系统定义的前提下找到一个比较好的技术方案。

为了达到这个目标，设计所要解决的主要问题是降低系统的复杂性，使实际的系统变得尽量简单和可靠。

2.1 设计基本手段

对于本来就比较简单的系统，凭个人的直觉和经验就能找到技术方案，但对于复杂的系统，凭直觉和经验是无法办到的。对于简单的设计问题，在脑子里或便笺上构思一下，在键盘上就能完成。对于复杂的设计，在键盘上完成设计是完全不可能的。此时，必须遵循系统化的软件设计方法，降低问题的复杂性。

软件设计的基本手段有 3 种：分解、层次化和抽象。

分解就是分而治之，将整体化解为局部。降低软件设计的复杂性可以通过将系统分成子系统，将子系统分解成模块，将模块分解成程序，将程序分解成子程序等方式来实现。对于好的设计方案，复杂性应能得到最大程度的降低。

层次化与分解一样，也是很自然的降低复杂性的手段。当我们画一个复杂的物体（如房子）时，我们也是分层画出来的。首先画房子的轮廓，然后是窗户和门，最后是其他细节。我们并不是将房子一块砖一块砖、一片瓦一片瓦、一颗钉一颗钉地画出来的。软件系统的层次结构一般有操作系统层次、高级语言应用程序层次和用户界面层次。对于高级语言设计软件工程师，仅需知道高级编程和用户界面即可。操作系统可使他们免受与机器指令打交道以及对最底层的操作调用的麻烦。在设计软件系统时，除了有现成可用的层次之外，还可以通过设计来创建新的层次。