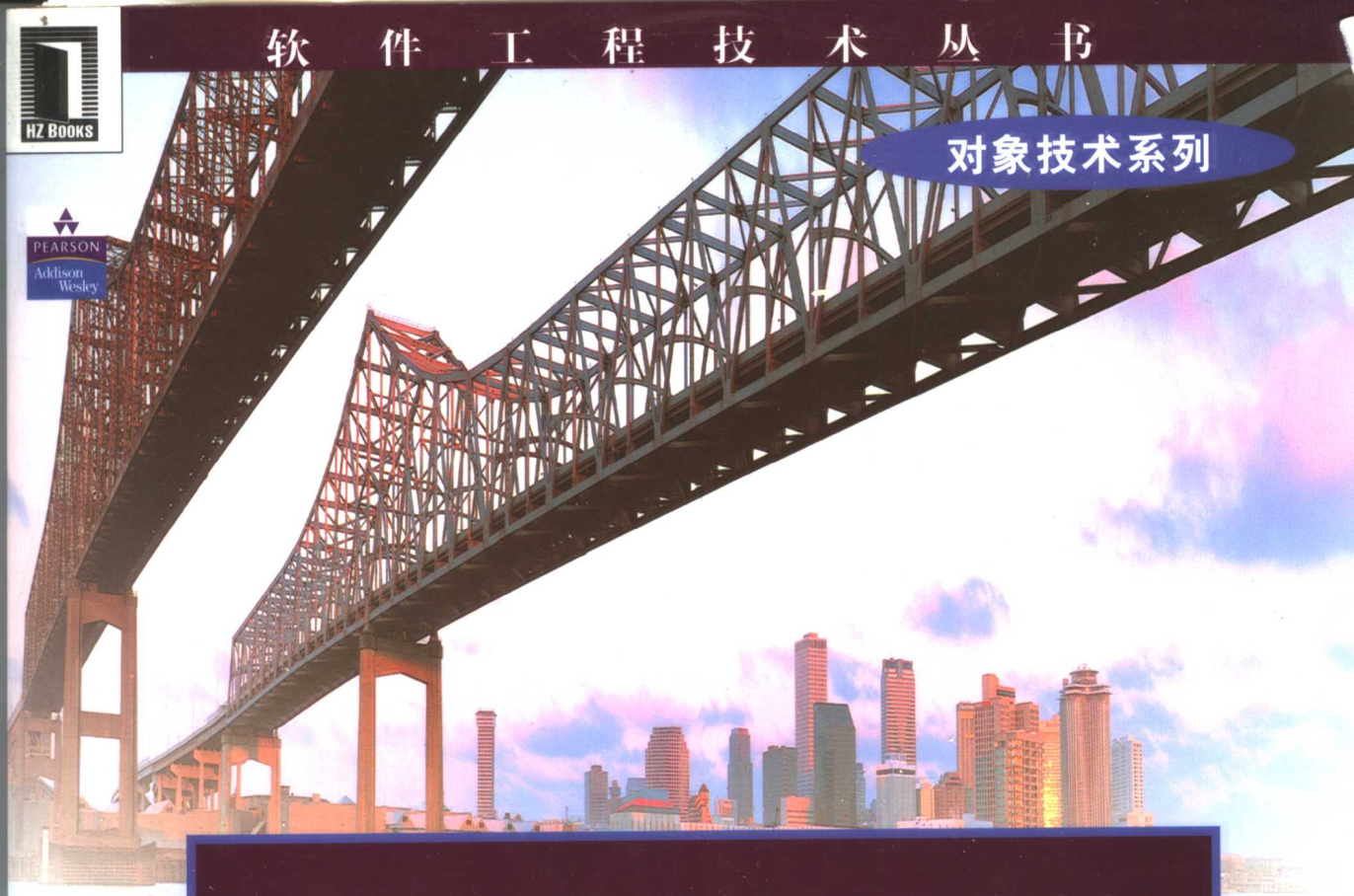
软 件 工 程 技 术 丛 书



对象技术系列

面向对象项目的解决方案

Object Solutions
Managing the Object-Oriented Project

（美） Grady Booch 著　　邢春丽 冯学民 张丽梅 译

机械工业出版社
China Machine Press

软 件 工 程 技 术 丛 书

对象技术系列

面向对象项目的解决方案

Object Solutions

Managing the Object-Oriented Project

（美） Grady Booch 著 / 邢春丽 冯学民 张丽梅 译

机械工业出版社
China Machine Press

本书是享誉软件工程领域的科学家 Grady Booch 的力作之一，主要从产品、过程、人员、规划等诸方面详细介绍面向对象项目管理的经验和原则。书中列举了作者多年来参与数以百计的面向对象项目所积累的成功和失败经验，呈现给读者许多使面向对象项目成功的推荐性做法和经验性法则。

本书对于初次接触面向对象项目的读者和在面向对象方面经验丰富的读者都是十分宝贵的资源。读者对象包括项目经理、高级程序设计人员以及希望成为有经验的程序开发人员的新手。

本书版权登记号：图字：01-2003-2274

图书在版编目（CIP）数据

面向对象项目的解决方案/（美）布奇（Booch, G.）著；邢春丽等译．-北京：机械工业出版社，2003.8

（软件工程技术丛书　对象技术系列）

书名原文：Object Solutions: Managing the Object-Oriented Project

ISBN 7-111-12309-3

Ⅰ.面…　Ⅱ.①鲍…②邢…　Ⅲ.面向对象语言-软件开发　Ⅳ.TP311.52

中国版本图书馆 CIP 数据核字（2003）第 043719 号

机械工业出版社（北京市西城区百万庄大街 22 号　邮政编码　100037）

责任编辑：张金梅

北京中加印刷有限公司印刷·新华书店北京发行所发行

2003 年 8 月第 1 版第 1 次印刷

787mm×1092mm 1/16 · 18.75 印张

印数：0 001-5 000 册

定价：39.00 元

译者序

面向对象思想的出现可以追溯到20世纪60年代末期，但直到20世纪80年代Smalltalk-80语言之后，这一技术才得到迅猛的发展（一段时间里，先后出现了一批面向对象程序设计语言，如Object-C、C++、Eiffel等），面向对象技术经过不断改进和完善而逐渐走向成熟和实用，成为当前软件开发的主流技术之一。

目前，面向对象技术已经拓展到业界的各个领域，单就软件开发过程来说，已经涉及面向对象分析、面向对象设计、面向对象编程、面向对象测试等方面。面向对象技术在国内真正热起来也就是最近几年的事情，但其在应用方面还处于起步阶段，远没有达到广泛应用和熟练运用的程度，与美国、印度等软件工程应用水平较高的国家相比仍存在着很大的差距。

面向对象技术引入了一种新的认知和表示世界的方法，其开发思想更符合人们的思维习惯。同时，面向对象技术在对问题领域的刻画、软件重用、适应需求变更等方面较传统开发方式有了长足的进步。

另一方面，在从传统的方式向面向对象技术转型时，软件开发组织必然要对软件开发过程中的一系列辅助行为——管理、测试、维护等进行适时的改进，以适应这种开发技术在软件开发过程中的成功运用。

单就管理来说，与传统开发方式相配套的一些方式、方法、规则等等将不再适用，取而代之的应该是一整套从实践中摸索出来的一般性的做法和经验性法则。

本书是软件工程大师Grady Booch先生的力作之一，书中对软件开发过程涉及到的各种要素——产品、过程、项目、人员等——逐一进行了详尽而透彻的阐述。因为本书主要讨论面向对象项目管理方面的问题，所以，本书的内容是从管理者的角度出发，侧重于如何控制项目走向成功，对软件开发过程中应该完成的工作和需要生成的制品进行了详细说明，并将这些过程中应该注意到的各种问题

一一列举出来，从而保证面向对象的项目能够达到预期目标。

作者多年来一直致力于软件工程方面的研究和实践，亲身经历过不计其数的软件项目（传统方式的和面向对象的）的开发和管理，因此，在编写此书时作者引入了大量的项目管理实例，总结出了成功的面向对象项目的惯常做法和应该遵循的规则，也适时地给出了一些经验性的法则。由于这些内容来自于大师本人的实践（当然有成功也有失败），并通过大师之手进行了甄选和提炼，因此对管理面向对象项目无疑会大有裨益。

本书的第1、2、5、6章由邢春丽翻译，第3、4章、前言和词汇表由冯学民翻译，第7章由张丽梅翻译，全书由冯学民统稿、整理和审校。另外，李丽韫、张欣彤、曹明东、余洁、沈黎敏、张俊刚、孙立刚等也参与了本书初期的部分翻译工作。

本书的翻译经历了一段较长的时间：一方面是因为我们在翻译过程中力求准确和通顺，另一方面则是因为对书中某些大师欲表达的深邃观点和含义无法理解到位（其中有中西方文化差异的原因，但更主要的在于译者对于书中所涉及技术的认知程度），需反复推敲。即便如此，对于作者某些地方所表达的意义还是难免出现理解上的偏差，故敬请广大读者对翻译不妥之处予以批评、指正。

真心希望能够把作者关于面向对象管理的一些思路和观点呈现给大家，以期对国内在面向对象技术方面的进步提供一定帮助。

译　者

2003年3月于北京航空航天大学

前　言

最初采用面向对象技术的人们坚信，面向对象是一种好东西，它让人看到了改善软件开发的某些不良方面的希望。这些基本的努力有些成功了，有些失败了，但总的来说，很多这样的项目开始逐步体验到对象的预期收益：更快的上市时间、优良的质量、更大的适应变化的柔性，以及增强的可重用级别。当然，有些新技术只是在短时间内玩玩而已。在业界，确实有一部分人热衷于追逐软件开发的最新时尚，然而，对于有些成熟技术，真实的业务案例表明它们对实际项目提供了可度量和持续增长的效益。

在世界范围的众多应用中，面向对象技术体现了它的价值。我已经看到了面向对象的语言和方法成功地运用到各种不同的问题领域，如证券交易、医疗电子、企业范围的信息管理、空中交通管制、半导体生产、交互式视频游戏、电信网络管理以及天文研究等。实际上，可以说，在每个发达国家和每个可以想到的应用领域，都多多少少涉及到了面向对象技术的一些应用。面向对象将无可争议地成为计算技术主流的一部分。

人们在应用面向对象技术的项目中积累了丰富的经验并且还在不断地增长，这些经验——无论是正面的，还是反面的——对指导新项目都大有裨益。我从所有这样的项目中得到了一个重要的结论：面向对象对软件开发可能有十分积极的影响，但项目成功需要的不止是面向对象的修饰。程序设计者无须因为对象的缘故而彻底放弃自己的开发原则。同样，管理者必须清楚对象带给传统经验的微妙影响。

范围

在我所接触的几乎每个项目中，往往人员并不多，但是受外部影响却很大，一些常见的问题是总要面对的：如何将开发组织迁移到面向对象的开发上来？应该设法继续控制什么制品？如何组织我的职员？如何度量处于开发中的软件的质

量？如何协调程序开发人员的创造性需要和管理的稳定性及可预测性需要之间的矛盾？面向对象能帮助我和我的客户更好地表达他们的真正需要吗？这些都是合理的问题，它们的答案均涉及到面向对象技术的核心——与传统的方法相比，面向对象技术有哪些不同和特殊之处。

本书通过对推荐性做法提出实用的意见和介绍成功项目所使用的经验性法则来回答上述这些问题以及很多其他相关的问题。

本书不是理论性的书籍，其目的也不是解释面向对象分析、设计和编程的所有疑难问题。我以前的著作 *Object-Oriented Analysis and Design with Applications*[⊖]，可以满足上述目标：它阐述了所有面向对象事物的理论基础，并给出了面向对象分析和设计统一方法的全面参考。

本书提供了直接和折中处理管理面向对象项目中所有重要问题的方法。我曾经参与过数以百计的项目，本书总结了其中众多的经验。目的是解释应该做什么，不应该做什么，以及如何对两者加以区分。

读者

本书的读者包括项目管理者和希望将面向对象技术成功地应用于他们的项目而避免可能发生的常见错误的高级程序设计人员。本书也适用于经验丰富的程序开发人员，可以使他们了解将面向对象的代码转化为实际产品时的很多问题，还有助于理解管理者的做法。希望成为有经验的程序开发人员的学生会进一步理解为什么实际的软件开发不都是有条不紊的，以及工业项目如何处理这种无序状况。

组织

我是从管理面向对象项目的各种功能方面来组织本书的。因此，读者可以按照顺序从头至尾阅读本书或有选择地阅读某些章节。为了使书中的内容更易于理解，我的编写风格是提出问题，进行讨论得到其结论，接着提供一些推荐性做法和经验性法则。为了在正文中加以区分，我使用了如下的印刷约定：

⊖ 此书即将由机械工业出版社引进出版。——编者注

这是一个关于项目管理中某些功能范围的问题，通常以一个提问后跟答案的形式表述。

这是一条推荐性做法，代表解决给定问题的一般可以接受的方法。

P#

这是一条经验性法则，代表有关某个特定做法的一些可以度量的方法。

R#

我按顺序为这些做法和规则编了号，以易于在后面引用。

为了强化某些经验教训，我列举了一些从大量面向对象项目产品中抽取出来的例子，但为了避免侵权，其细节已经进行了改动。采用以下形式来突出显示这些例子：

这是一个从某个面向对象项目中抽取出来的例子。

致谢

作为面向对象知识的概论，本书很大程度上应该归功于那些职业管理人员和程序开发人员，他们的贡献提高了面向对象技术的现状。

特别感谢下述人员，在成书的过程中他们审阅了本书并提出了很多有益的意见和建议：Gregory Adams，Glen Andert，Andrew Baer，Dave Bernstein，Mike Dalpee，Rob Daly，Mike Devlin，Richard Dué，Jim Gillespie，Jim Hamilton，Larry Hartweg，Philippe Kruchten，Brian Lyons，Joe Marasco，Sue Mickel，Frank Pappas，Jim Purtilo，Rich Reitman，Walker Royce，Dave Tropeano，Mike Weeks 和 William Wright 博士。

特别感谢我的妻子 Jan，她使我能同时编写另一本书并保持清醒的头脑，而且使我感到，除了面向对象之外还有丰富多彩的生活。

目　录

第 1 章

首要原则

自然界只使用最长的丝线来编织图案，因此每一小片织物都能揭示整个织锦的结构。

理查德·费曼

首先，是一些好消息。

从软件开发人员的角度看，我们生活在一个十分有趣的时代。想想支持工业社会中很多习以为常的活动的所有软件：打电话、购买公共基金股份、驾驶汽车、看电影、进行医疗检查等等，复杂的软件已经十分普遍，而且它仍然不断深入到社会的方方面面，这使得有创意的构架设计师、抽象派艺术家和实现人员供不应求。

还有一些坏消息。可以毫不夸张地预测将来的软件会更加复杂。确实，两个关键性的因素决定了这种趋势：分布系统不断增加的关联性，高性能计算系统和众多用户对访问信息更加直观的预期。第一个因素——增加的关联性——由于不断增加的高带宽信息通路的出现而成为可能，并由于规模经济而得以实现。第二个因素——众多用户的预期——是任天堂时代公众意识到自动化的创造性潜能的最直接结果。在这两个因素的影响下，消费者希望通过有线电视定购的电影可以直接到他/她的账户进行结算的想法是合理的。科学家希望在线访问远端实验室中的信息是合理的。构架设计师希望评审异地合作者创建的虚拟蓝图是合理的。玩家与游戏交互时几乎无法与现实相区分是合理的。零售业希望在其任务关键的系统中，将顾客购买商品的事件与公司采购人员（对顾客的喜好作出及时的反应）的活动以及公司市场部门（必须向增长的特定用户群提供商品）的活动无缝地关联起来也是合理的。在这样的系统中出现疏漏，或当用户询问“为什么我不能做某事”的时候，都说明我们还没有把握住某个特定领域的复杂性。

即使我们忽略花费在软件维护上的大量资源，全球软件开发人力资源也会很容易地消耗在由于前面两个简单因素而导致的更多软件编写活动上。如果再考虑到微处理器之争、操作系统之争、程序设计语言之争，甚至于方法论之争等毫无收益的因素，我们会发现，可用于发现或创建新的强力（killer）应用的资源少得可怜。最终，遭殃的是计算机用户。

另外，积极的一面是，现在的软件开发很少再受限于硬件。与十几年前相比，当前很多应用所处的环境非常优越：充裕处理能力（百万条指令）的处理器、足够的内存和廉价的连接等。当然，这种优势也有两面性。一方面，这意味着硬件不再明显地制约我们精心设计的软件构架。另一方面，这种过度的充裕也鼓励了人们对于软件功能无法满足的欲望。

因此，我们面对的是一个简单但基本的真理：

> 我们对复杂应用的想像能力总会超越我们开发它们的能力。

实际上，这是最为积极的形式：我们设想的需求驱使我们去不断提高设计软件的方法。

1.1 好的项目陷入困境时

大多数软件项目始于如下原因：满足市场需要，向最终用户群提供某种十分必要的功能，探究某些待证明的理论。对很多人来说，编写软件是他们业务必须的或不可回避的部分，换句话说，是仅次于其业务的。银行的业务是管理钱而非软件，然而要达到这一目的软件是其必须的手段。从事零售业的公司通过相应的软件达到生产和分销上的竞争优势，而其主营业务可能只是向消费者提供最新潮的服装。对另外一些人来说，编写软件是他们的生命、爱好和乐趣，当他们编写软件时，其他人还专注于日常的繁杂琐事。另外，开发软件是很花费时间和精力的。

> 为什么软件项目会失败？最通常的原因在于：
> - 未能恰当地管理风险。
> - 构造出了错误的东西。
> - 受到技术的局限。

不幸的是，随着项目组工作的展开，很多项目迷失了方向。多数项目失败的原因在于缺乏成熟的监督和管理[1]。于是，很多不切实际的进度表和计划陆续出台，不断积累了一系列的谎言，而没有人鼓起勇气再去坚持和确认事实。空中楼阁形成了。每个问题均被视为“一种简单的编程问题”，而不是作为系统构架或开发过程中一个更为系统化的问题反映出来。由组内最不受欢迎的人确定项目的方向和活动，因为对于管理层来说，让这组人自行其事比问题出现时做出困难的抉择更容易。像这种无管理的项目最终将陷入没有任何人承担责任、大家只是坐

[1] 我并非夸大其词。这些项目的失败不是因为管理很差，而是因为根本就没有管理。

等结果的僵局。通常，在这种情形下要做的最为明智的事情是，在这个项目彻底崩溃前中止它。

某公司要履行一份较大的合同：提供一系列通用的开发工具套件，最终由合同另一方用于任务关键性的空间项目。该公司做了很多正确的事情：选择了经验丰富的构架设计师，精心培训了公司的成员，选择了一些很优秀的工具，甚至为项目配备了专门的设备以便管理人员随时调整开发过程。但是，一旦项目开始实施，高层管理者实际上会处于毫无意义的地位而任由项目的技术人员随意行事。如果没有任何清晰的目标和严格的时间表约束功能递增的版本，那么程序设计者便会追求他们觉得很酷的东西，任由具体实现的结果与系统相背离。当项目管理者把时间用在行政管理上时，程序设计者们仍然按照他们的时间表行事，这主要是因为不存在交付任何实际产品的内部压力的事实造成的。而与此同时，最终用户不断要求得到高优先级的交付版本，却很快遭到漠视，因为他们认为一旦项目组构造完了所有必须的底层功能，这种版本最终就会得以实现。测试组更关心性能，警告说正在精心设计的框架一旦面临实际用户的需求，会由于其自身的问题而最终崩溃。在耗费了数千万美元之后，项目被取消了，但几乎没有任何可交付的软件来表明项目的成果。

从这个项目的失败中可以汲取一些简单的经验教训：

管理人员必须积极着手解决项目的风险，否则风险便会令你措手不及。

P1 项目会迷失方向，因为它们经常毫无目标地进入到未知的区域。没有解决问题的通用答案。项目组对于最终的目标毫无头绪，因此不断进行摸索，将珍贵的精力投入到表面上看来彷佛是最重要的技术问题上，而一旦最终用户最后看到该产品，才发现这个问题实际上并不重要。没有人用这些时间去和最终用户或领域专家确认到底要构造什么系统。有时候，所谓的分析师捕获了系统实际需求的要素，但由于很多政治和社会原因，那些要素从未通知到设计和实现该系统的人员。一种缺乏理解的感觉弥漫整个项目，在用户拒绝接受交付的在完全真空状态下精心制作的软件时，每个人都感到很惊讶。

某公司被选中开发一个大型州际交通控制系统。初期的规模估算提议需要数百个开发人员（其实是一种最早的预警信号）。于是，两栋建筑拔地而起，一栋提供给项目的系统分析师使用，另一栋提供给项目设

计人员和实现人员使用。不必惊讶，在用户需求到项目的程序设计者间进行交流时，这些人为形成的物理界限引入了大量的干扰。备忘录大战盛行，系统分析师将包含着他们对问题观点的报告隔墙抛给可怜的、孤立无援的设计者，而设计者不时用他们的报告予以还击。项目的程序设计者很少直接面对实际的最终用户；他们忙于编码，而且按照项目一贯的传统，他们一向认为大多数项目的程序设计者不知道如何与用户打交道。随着时间一天天地过去，项目的真正需求变得越来越清晰，而将这些需求变更从用户通过系统分析师再通知设计者时，存在相当大的延迟，其结果是造成进度表的严重滞后，使客户/供应商间的关系出现裂痕。

根据从这个项目（很多其他项目与此类似）中得到的经验，提出下面建议性的做法：

将真实用户融入到整个软件开发过程中，他们的存在时刻提醒开发者为什么和为谁开发软件产品。

P2

有时候，项目失败是由于受限于用来构造软件的技术。工具在最不该失效的时候失效，缺乏处理复杂性成倍增加的项目的能力。有时，项目的工具本来就存在明显的错误，需要程序设计者进行异常处理来避免这些缺陷。有时候，第三方软件供应商并没有交付他们最初承诺的产品，经常缺少一些预期的功能，或者性能比预期的差很多。最糟糕的情况是第三方供应商干脆关门歇业，完全置项目于不顾。技术的负面效应最常见于超出项目本身控制的市场力量从底层改变了技术规则：硬件平台提供商停止生产某种产品，操作系统界面和功能也由其供应商改变，超过了项目可以保持同步的速度；最终用户的喜好改变了而且他们的期望值也提高了，因为其中某个人看到了最新的行业杂志里提到的其他某个非常精巧的程序（尽管那种产品事后证明只是一种雾件）。尽管项目选用的最新语言/工具/方法可能承诺了很多好处，但获得上述好处通常比它最初看起来要困难得多。当技术意外出错时，通常已没有足够的时间从头再来，无法减少项目曾经承诺交付的系统功能。最后的情况对于软件开发组织来说十分窘迫：作为专业人士，他们怎么也想不到竟会被自己的技术所害。

某安全贸易公司在面向对象工具方面付出了很大的投资，为其属下所有开发人员购买最流行品牌的工作站和程序设计工具。不久以后，该种工作站提供商决心转型成为一家软件公司，并决定停止生产其所有的

硬件产品。

通常，底层技术所引发的问题会导致项目的失败，但真正的肇事者其实是非技术的，也就是说，是缺乏积极有效的管理——首先应该预见到或计划好出现技术风险的可能性。毕竟我们不是生活在一个完美的世界里，因此：

P3

尽可能不要将你的项目局限于任何独家拥有的技术，但如果必须如此（例如，当上述技术提供了某些令人瞩目的优点足以面对其存在的风险时），就应该在你的构架和过程中构造“防火墙”以便项目不会因技术出现问题而失败。

1.2 确定项目的核心

尽管存在上述三种失败的原因，但出于各种合理的理由而启动的很多软件项目确实获得了一定程度的成功。然而，即使最成功的项目，似乎也花费了较长的时间，投入了更多的精力，而且需要实施比预先想到的更多的风险管理。不幸的是，如同 Parnas 指出的，我们永远不会有一个完全合理的开发过程，因为：

- 系统的用户通常不能确切知道他们需要什么，不能清晰地表达他们所知道的一切。
- 即使确定了系统的所有需求，但还有很多有关系统的细节只能在具体实现时才能发现。
- 即使我们了解了所有这些细节，但人类目前处理复杂情况的能力还有很大的局限性。
- 即使我们可以掌握所有这些复杂性，但还存在很多外部力量，远远超出项目的控制范围，它们会导致需求的变化，有些可能会推翻早期做出的决定。
- 人类构造的系统总会出现人为错误。
- 在开始启动每个新的项目时，总会引入前面项目设计的一些思想，还有对软件在经济方面的考虑，这两方面会影响我们的决定，使之偏离系统的真正需求。

Parnas 还提道“因为所有这些原因，软件设计人员以理性的、无差错的方式按照需求说明进行设计的想法很不现实”。幸运的是，正如 Parnas 所说的和我将

在后续的章节中进一步讨论的，有可能而且确实值得做的，就是模拟一个系统。采用各种方法建立一个近似合理的设计过程是每个成功项目必须做到的。

每个成功项目也需要制定很多技术决策。有些决策为构造中的系统提出了总的建议，例如采用某种客户机/服务器拓扑结构的决策，采用特定的窗口框架的决策，采用关系数据库的决策等。这些决策称为战略决策，因为每个决策都意味着一种基本的构架模式。其他决策实际上更多针对项目的局部，如对迭代使用特定的程序设计风格的决策，确定个别类的接口的决策，采用特定供应商的关系数据库产品的决策等等。这些决策称为战术决策。战略和战术决策合起来构成了待开发系统的完整结构。

然而，技术决策自己并不能消除某个项目失败的起因。管理风险、构造恰当的产品并预防技术性灾难都是主要的经济性决策，而非技术决策。当然，项目所采用的技术确实会影响到每个经济性决策，通过建立一系列选项和不同的成本、风险值将有助于每个决策的制定。

因此，在启动任何新的项目前，最根本的就是要明确哪些主要特征可以用来制定上述经济决策。这些特征包括如下标准：

- 上市时间。
- 完整性。
- 性能。
- 质量。
- 容错性。
- 可度量性。
- 扩展性。
- 可移植性。
- 构架复用性。
- 总体成本。

优化一个系统使之同时符合所有这些标准是不可能的。如果上市时间是项目最关心的，这在竞争激烈的软件产品领域更常见，那么完整性、性能和其他各种特性很可能会无法兼顾。如果质量和容错性是核心，常见于很多人命关天的系统，如空中交通管制和医疗仪器等，那么上市时间就必须退而求其次。这并不是说项目不应该或不能有多个目标。当然，整个开发组必须认识到，对于每个技术