

第一部分 软件需求： 是什么和为什么

第1章 基本的软件需求

“喂，是Phil吗？我是人力资源部的Maria，我们在使用你编写的职员系统时遇到一个问题，一个职员想把她的名字改成Sparkle Starlight，而系统不允许，你能帮帮忙吗？”

“她嫁给了一个姓Starlight的人吗？”Phil问道。

“不，她没有结婚，而仅仅是要更改她的名字，”Maria回答。“就是这问题，好像我们只能在婚姻状况改变时才能更改姓名。”

“当然是这样，我从没想过谁会莫名其妙地更改自己的姓名。我不记得你曾告诉我系统需要处理这样的事情，这就是为什么你们只能在改变婚姻状况对话框中才能进入更改姓名的对话框。”Phil说。

Maria说：“我想你当然知道每个人只要愿意都可以随时合法更改他（她）们的姓名。但不管怎样，我们希望在下周五之前解决这个问题，否则，Sparkle将不能支付她的账单。你能在此前修改好这个错误吗？”

“这并不是我的错！我从来不知道你需要处理这种情况。我现在正忙着做一个新的性能检测系统，并且还要处理职员系统的一些需求变更请求”（传来翻阅稿纸的声音）。“我还有别的事。我只可能在月底前修改好，一周内不行，很抱歉。下次若有类似情况，请早一些告诉我并把它们写下来。”

“那我怎么跟Sparkle说呢？”Maria追问道，“如果她不能支付账单，那她只能挂帐了。”

“Maria，你要明白，这不是我的过错。”Phil坚持道，“如果你一开始就告诉我，你要能随时改变某个人的名字，那这些都不会发生。因此你不能因我未猜出你的想法（需求）就责备我。”

Maria不得不愤怒地屈从：“好吧，好吧，这种烦人的事使我恨死计算机系统了。等你修改好了，马上打电话告诉我，行吧？”

如果作为客户有过类似的经验，你一定知道：一个不能进行一项基本操作的软件产品是多么令人烦恼。尽管开发者最终会满足你的要求，你也不会感谢他。但从开发者角度来看，在整个系统已经完成后，用户再提出对功能的进一步要求是多么烦人的事。同时，修改系统的请求迫使你放下当前的项目，而且往往修改请求还要求你优先处理，也是令人很不愉快的。

其实，在软件开发中遇到的许多问题，都是由于收集、编写、协商、修改产品需求过程中的手续和作法（方法）失误带来的。例如上面的Phil和Maria，出现的问题涉及到非正式信息的收集，未确定的或不明确的功能，未发现或未经交流的假设，不完善的需求文档，以及突发的需求变更过程。

对大多数人来说，若要建一幢20万美元的房子，他一定会与建房者详细讨论各种细节，

他们都明白完工以后的修改会造成损失，以及变更细节的危害性。然而，涉及到软件开发，人们却变得“大大咧咧”起来。软件项目中百分之四十至百分之六十的问题都是在需求分析阶段埋下的“祸根”(Leffingwell 1997)。可许多组织仍在那些基本的项目功能上采用一些不合规范的方法，这样导致的后果便是一条鸿沟（期望差异）——开发者开发的与用户所想得到的软件存在着巨大期望差异。

在软件工程中，所有的风险承担者（stakeholder）都感兴趣的就是需求分析阶段。这些风险承担者包括客户、用户、业务或需求分析员（负责收集客户需求并编写文档，以及负责客户与开发机构之间联系沟通的人）、开发人员、测试人员、用户文档编写者、项目管理者和客户管理者。这部分工作若处理好了，能开发出很出色的产品，同时会使客户感到满意，开发者也倍感满足、充实。若处理不好，则会导致误解、挫折、障碍以及潜在质量和业务价值上的威胁。因为需求分析奠定了软件工程和项目管理的基础，所以所有风险承担者最好是采用本书提供的有效的需求分析过程。

本章将帮助你：

- 了解软件需求开发中使用的一些关键名词。
- 警惕在软件项目中可能出现的与需求相关的一些问题。
- 知道优秀的需求规格说明应该具有的特点。
- 明白需求开发与需求管理之间的区别。

1.1 软件需求的定义

软件产业存在的一个问题就是缺乏统一定义的名词术语来描述我们的工作。客户所定义的“需求”对开发者似乎是一个较高层次的产品概念。而开发人员所说的“需求”对用户来说又像是详细设计了。实际上，软件需求包含着多个层次，不同层次的需求从不同角度与不同程度反映着细节问题。

IEEE软件工程标准词汇表（1997年）中定义需求为：

- (1) 用户解决问题或达到目标所需的条件或权能（Capability）。
- (2) 系统或系统部件要满足合同、标准、规范或其它正式规定文档所需具有的条件或权能。
- (3) 一种反映上面(1)或(2)所描述的条件或权能的文档说明。

1.1.1 一些关于“需求”的解释

IEEE公布的定义包括从用户角度（系统的外部行为），以及从开发者角度（一些内部特性）来阐述需求。

关键的问题是一定要编写需求文档。我曾经目睹过一个项目中途更换了所有的开发者，客户被迫与新的需求分析者坐到一起。分析人员说：“我们想与你谈谈你的需求。”客户的第一反应便是：“我已经将我的要求都告诉你们前任了，现在我要的就是给我编一个系统”。而实际上，需求并未编写成文档，因此新的分析人员不得不从头做起。所以如果只有一堆邮件、贴条、会谈过几次或一些零碎的对话，你就确信你已明白用户的需求，那完全是自欺欺人。

另外一种定义认为需求是“用户所需要的并能触发一个程序或系统开发工作的说明”(Jones 1994)。需求分析专家 Alan Davis (1993) 拓展了这个概念：“从系统外部能发现系统

所具有的满足于用户的特点、功能及属性等”。这些定义强调的是产品是什么样的，而并非产品是怎样设计、构造的。而下面的定义则从用户需要进一步转移到了系统特性（Sommerville and Sawyer 1997）：

需求是……指明必须实现什么的规格说明。它描述了系统的行为、特性或属性，是在开发过程中对系统的约束。

从上面这些不同形式的定义不难发现：并没有一个清晰、毫无二义性的“需求”术语存在，真正的“需求”实际上在人们的脑海中。任何文档形式的需求（例如：需求规格说明）仅是一个模型，一种叙述（Lawrence 1998）。我们需要确保所有项目风险承担者在描述需求的那些名词的理解上务必达成共识。

1.1.2 需求的层次

下面这些定义是需求工程领域中常见术语的定义说明。

软件需求包括三个不同的层次——业务需求、用户需求和功能需求——也包括非功能需求。业务需求（business requirement）反映了组织机构或客户对系统、产品高层次的目标要求，它们在项目视图与范围文档中予以说明。用户需求（user requirement）文档描述了用户使用产品必须要完成的任务，这在使用实例（use case）文档或方案脚本（scenario）说明中予以说明。功能需求（functional requirement）定义了开发人员必须实现的软件功能，使得用户能完成他们的任务，从而满足了业务需求。所谓特性（feature）是指逻辑上相关的功能需求的集合，给用户提供处理能力并满足业务需求。软件需求各组成部分之间的关系如图 1-1 所示。

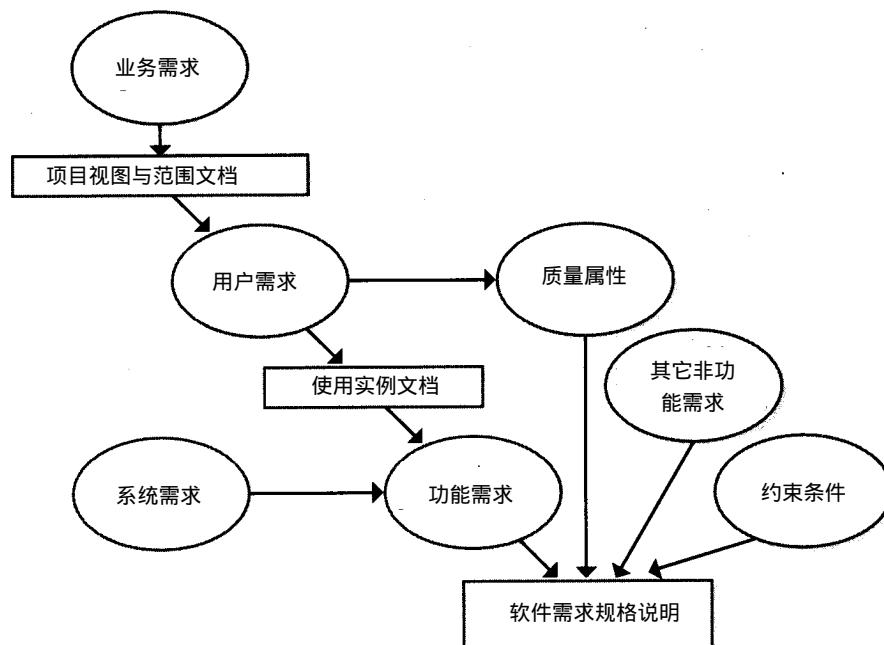


图 1-1 软件需求各组成部分之间的关系

在软件需求规格说明（software requirements specification，SRS）中说明的功能需求充分描述了软件系统所应具有的外部行为。软件需求规格说明在开发、测试、质量保证、项目管

理以及相关项目功能中都起了重要的作用。对一个复杂产品来说，软件功能需求也许只是系统需求的一个子集，因为另外一些可能属于软件部件。

作为功能需求的补充，软件需求规格说明还应包括非功能需求，它描述了系统展现给用户的行为和执行的操作等。它包括产品必须遵从的标准、规范和合约；外部界面的具体细节；性能要求；设计或实现的约束条件及质量属性。所谓约束是指对开发人员在软件产品设计和构造上的限制。质量属性是通过多种角度对产品的特点进行描述，从而反映产品功能。多角度描述产品对用户和开发人员都极为重要。

下面以一个字处理程序为例来说明需求的不同种类。业务需求可能是：“用户能有效地纠正文档中的拼写错误”，该产品的包装盒封面上可能会标明这是个满足业务需求的拼写检查器。而对应的用户需求可能是“找出文档中的拼写错误并通过一个提供的替换项列表来供选择替换拼错的词”。同时，该拼写检查器还有许多功能需求，如找到并高亮度提示错词的操作；显示提供替换词的对话框以及实现整个文档范围的替换。

管理人员或市场分析人员会确定软件的业务需求，这使公司运作更加高效（对信息系统而言）或具有很强的市场竞争力（对商业软件产品而言）。所有的用户需求必须与业务需求一致。用户需求使需求分析者能从中总结出功能需求以满足用户对产品的要求从而完成其任务，而开发人员则根据功能需求来设计软件以实现必须的功能。

从以上定义可以发现，需求并未包括设计细节、实现细节、项目计划信息或测试信息。需求与这些没有关系，它关注的是充分说明你究竟想开发什么。项目也有其它方面的需求，如开发环境需求或发布产品及移植到支撑环境的需求。尽管这些需求对项目成功也至关重要，但它们并非本书所要讨论的。

1.2 每个项目都有需求

Frederick Brooks在他1987年的经典的文章“*No Silver Bullet : Essence and Accidents of Software Engineering*”中充分说明了需求过程在软件项目中扮演的重要角色：

开发软件系统最为困难的部分就是准确说明开发什么。最为困难的概念性工作便是编写出详细技术需求，这包括所有面向用户、面向机器和其它软件系统的接口。同时这也是一旦做错，将最终会给系统带来极大损害的部分，并且以后再对它进行修改也极为困难。

每个软件产品都是为了使其用户能以某种方式改善他们的生活，于是，花在了解他们需要上的时间便是使项目成功的一种高层次的投资。这对于商业最终用户应用程序，企业信息系统和软件作为一个大系统的一部分的产品是显而易见的。但是对于我们开发人员来说，并没有编写出客户认可的需求文档，我们如何知道项目于何时结束？而如果我们不知道什么对客户来说是重要的，那我们又如何能使客户感到满意呢？

然而，即便并非出于商业目的的软件需求也是必须的。例如软件库、组件和工具这些供开发小组内部使用的软件。当然你可能偶尔勿需文档说明就能与其他人意见较为一致，但更常见的是出现重复返工这种不可避免的后果，而重新编制代码的代价远远超过重写一份需求文档的代价。

近来，我遇到一个开发小组开发包括流程图工具和源代码编辑器在内的一套计算机辅助软件工程工具。不幸的是，在他们开发完这个工具后，发现这个工具不能打印出源代码文件，而

用户当然希望有这个功能。结果这个小组只好手工抄写源代码文档以供代码检查。这说明那怕需求明确无误并构思准确，如果我们没有编写文档，软件达不到期望目标也只能是咎由自取了。

相反的情况，我曾为一个要集成到“商业错误跟踪系统”中的简单电子邮件界面写了一页需求说明。而 Unix 系统管理员在为处理电子邮件写脚本时发现简单的一张需求清单竟是如此有用。我依据需求对系统进行测试时，此系统不仅非常清晰地实现了所有必需功能，而且未发现任何错误。

1.3 什么情况将会导致好的群体发生不合格的需求说明

不重视需求过程的项目队伍将自食其果。需求工程中的缺陷将给项目成功带来极大风险，这里的“成功”是指推出的产品能以合理的价格、及时限在功能、质量上完全满足用户的期望。下面将讨论一些需求风险。第 5 章将介绍怎样应用软件风险管理防止与需求有关的风险的出现。

不适当的需求过程所引起的一些风险

- 用户不多导致产品无法被接受。
- 用户需求的增加带来过度的耗费和降低产品的质量。
- 模棱两可的需求说明可能导致时间的浪费和返工。
- 用户增加一些不必要的特性和开发人员画蛇添足 (gold-plating)。
- 过分简略的需求说明以致遗漏某些关键需求。
- 忽略某类用户的需求将导致众多客户的不满。
- 不完善的需求说明使得项目计划和跟踪无法准确进行。

1. 无足够用户参与

客户经常不明白为什么收集需求和确保需求质量需花费那么多功夫，开发人员可能也不重视用户的参与。究其原因：一是因为与用户合作不如编写代码有意思；二是因为开发人员觉得已经明白用户的需求了。在某些情况下，与实际使用产品的用户直接接触很困难，而客户也不太明白自己的真正需求。但还是应让具有代表性的用户在项目早期直接参与到开发队伍中，并一同经历整个开发过程。

2. 用户需求的不断增加

在开发中若不断地补充需求，项目就越变越庞大以致超过其计划及预算范围。计划并不总是与项目需求规模与复杂性、风险、开发生产率及需求变更实际情况相一致，这使得问题更难解决。实际上，问题根源在于用户需求的改变和开发者对新需求所作的修改。

要想把需求变更范围控制到最小，必须一开始就对项目视图、范围、目标、约束限制和成功标准给予明确说明，并将此说明作为评价需求变更和新特性的参照框架。说明中包括了对每种变更进行变更影响因素分析的变更控制过程，有助于所有风险承担者明白业务决策的合理性，即为何进行某些变更，相应消耗的时间、资源或特性上的折中。

产品开发中不断延续的变更会使其整体结构日渐紊乱，补丁代码也使得整个程序难以理解和维护。插入补丁代码使模块违背强内聚、松耦合的设计原则，特别是如果项目配置管理工作不完善的话，收回变更和删除特性会带来问题。如果你尽早地区别这些可能带来变更的特性，你就能开发一个更为健壮的结构，并能更好地适应它。这样设计阶段需求变更不会直接导致补丁代码，同时也有利于减少因变更导致质量的下降。

3. 模棱两可的需求

模棱两可是需求规格说明中最为可怕的问题 (Lawrence 1996)。它的一层含义是指诸多读者对需求说明产生了不同的理解；另一层含义是指单个读者能用不止一个方式来解释某个需求说明。

模棱两可的需求会使不同的风险承担者产生不同的期望，它会使开发人员为错误问题而浪费时间，并且使测试者与开发者所期望的不一致。一位系统测试人员曾告诉我，她所在的测试组经常对需求理解有误，以致不得不重写许多测试用例并重做许多测试。

模棱两可的需求带来不可避免的后果便是返工——重做一些你认为已做好的事情。返工会耗费开发总费用的 40%，而 70% ~ 85% 的重做是由于需求方面的错误所导致的 (leffingwell 1997)。想像一下如果你能减少一半的返工会是怎样的情况？你能更快地开发出产品，在同样的时间内开发更多、更好的产品，甚至能偶尔回家休息休息。

处理模棱两可需求的一种方法是组织好负责从不同角度审查需求的队伍。仅仅简单浏览一下需求文档是不能解决模棱两可问题的。如果不同的评审者从不同的角度对需求说明给予解释，但每个评审人员都真正了解需求文档，这样二义性就不会直到项目后期才被发现，那时再发现的话会使得更正代价很大。其它检测模棱两可需求的技术由 Gause 和 Weinberg (1989) 给予介绍，本章的后面也有所涉及。

4. 不必要的特性

“画蛇添足”是指开发人员力图增加一些“用户欣赏”但需求规格说明中并未涉及的新功能。经常发生的情况是用户并不认为这些功能性很有用，以致在其上耗费的努力“白搭”了。开发人员应当为客户构思方案并为他们提供一些具有创新意识的思路，具体提供哪些功能要在客户所需与开发人员在允许时限内的技术可行性之间求得平衡，开发人员应努力使功能简单易用，而不要未经客户同意，擅自脱离客户要求，自作主张。

同样，客户有时也可能要求一些看上去很“酷”，但缺乏实用价值的功能，而实现这些功能只能徒耗时间和成本。为了将“画蛇添足”的危害尽量减小，应确信：你明白为什么要包括这些功能，以及这些功能的“来龙去脉”，这样使得需求分析过程始终是注重那些能使用户完成他们业务任务的核心功能。

5. 过于精简的规格说明

有时，客户并不明白需求分析有如此重要，于是只作一份简略之至的规格说明，仅涉及了产品概念上的内容，然后让开发人员在项目进展中去完善，结果很可能出现的是开发人员先建立产品的结构之后再完成需求说明。这种方法可能适合于尖端研究性的产品或需求本身就十分灵活的情况 (McConnell 1996)。但在大多数情况下，这会给开发人员带来挫折（使他们在不正确的假设前提和极其有限的指导下工作），也会给客户带来烦恼（他们无法得到他们所设想的产品）。

6. 忽略了用户分类

大多数产品是由不同的人使用其不同的特性，使用频繁程度也有所差异，使用者受教育程度和经验水平也不尽相同。如果你不能在项目早期就针对所有这些主要用户进行分类的话，必然导致有的用户对产品感到失望。例如，菜单驱动操作对高级用户太低效了，但含义不清的命令和快捷键又会使不熟练的用户感到困难。

7. 不准确的计划

“上述是我对新产品的看法，好，现在你能告诉我你什么时候能完成吗？”许多开发人员都遇到这种难题。对需求分析缺乏理解会导致过分乐观的估计，而当不可避免的超支发生时，会带来颇多麻烦。据报道，导致需求过程中软件成本估计极不准确的原因主要有以下五点：频繁的需求变更、遗漏的需求、与用户交流不够、质量低下的需求规格说明和不完善的需求分析（Davis 1995）。

对不准确的要求所提问题的正确响应是“等我真正明白你的需求时，我就会来告诉你”。基于不充分信息和未经深思的对需求不成熟的估计很容易为一些因素左右。要作出估计时，最好还是给出一个范围（如最好的情况下，很可能的，最坏情况下）或一个可信赖的程度（我有90%的把握，我能在8周内完成）。未经准备的估计通常是作为一种猜测给出的，听者却认为是一种承诺。因此我们要尽力给出可达到的目标并坚持完成它。

1.4 高质量的需求过程带来的好处

实行有效的需求工程管理的组织能获得多方面的好处。最大的好处是在开发后期和整个维护阶段的重做的工作大大减少了。Boehm (1981)发现要改正在产品付诸应用后所发现的一个需求方面的缺陷比在需求阶段改正这个错误要多付出68倍的成本。近来很多研究表明这种错误导致成本放大因子可以高达200倍。强调需求质量并不能引起某些人的重视，他们错误地认为在需求上消耗多少时间就会导致产品开发推迟多少时间。传统的质量成本角度分析揭示了需求及其它早期质量工作的重要性（Wiegers 1996a）。

正确的请求过程强调产品开发中的通力合作，包括在整个项目过程中多方风险承担者的积极努力。收集需求能使开发小组更好地了解市场，而市场因素是任何项目成功的一个关键因素。在产品开发前了解这些比在遭到客户批评后才意识到要节约很多成本。让用户积极参与需求收集过程能使产品更富有吸引力，而且能拥有忠实的客户关系。通过了解用户的任务需求而不仅仅局限于一些“华丽”的特性，你能避免在无用功能上白耗精力，并且用户的参与能弥补用户期望和开发者实际开发之间的“鸿沟（期望差异）”。

将选定系统的需求明确地分配到各软件子系统，强调采用产品工程的系统方法。这样能简化硬软件的集成，也能确保软硬件系统功能匹配适当。有效的变更控制和影响分析过程也能降低需求变更带来的负面影响。最后，将需求编写成清晰、无二义性的文档将会极大地有利于系统测试，确保产品质量，以使所有风险承担者感到满意。

1.5 优秀需求具有的特性

怎样才能把好的需求规格说明和有问题的需求规格说明区别开来？下面讨论单个需求陈述说明的几个特点（Davis 1993；IEEE 1998）。让风险承担者从不同角度对SRS需求说明进行认真评审，能很好地确定哪些需求确实是需要的。只要你在编写、评审需求时把这些特点记在心中，就会写出更好的（尽管并不十分完美）需求文档，同时也会开发出更好的产品。在第9章中，我们将使用这些特点找到一些需求陈述中的问题并改进之。

1.5.1 需求说明的特征

1. 完整性

每一项需求都必须将所要实现的功能描述清楚，以使开发人员获得设计和实现这些功能

所需的所有必要信息。

2. 正确性

每一项需求都必须准确地陈述其要开发的功能。做出正确判断的参考是需求的来源，如用户或高层的系统需求规格说明。若软件需求与对应的系统需求相抵触则是不正确的。只有用户代表才能确定用户需求的正确性，这就是一定要有用户的积极参与的原因。没有用户参与的需求评审将导致此类说法：“那些毫无意义，这些才很可能是他们所要想的。”其实这完全是评审者凭空猜测。

3. 可行性

每一项需求都必须是在已知系统和环境的权能和限制范围内可以实施的。为避免不可行的需求，最好在获取（elicitation）需求（收集需求）过程中始终有一位软件工程小组的组员与需求分析人员或考虑市场的人员在一起工作，由他负责检查技术可行性。

4. 必要性

每一项需求都应把客户真正所需要的和最终系统所需遵从的标准记录下来。“必要性”也可以理解为每项需求都是用来授权你编写文档的“根源”。要使每项需求都能回溯至某项客户的输入，如使用实例或别的来源。

5. 划分优先级

给每项需求、特性或使用实例分配一个实施优先级以指明它在特定产品中所占的分量。如果把所有的需求都看作同样重要，那么项目管理者在开发或节省预算或调度中就丧失控制自由度。第13章将更详细地讨论如何划分优先级。

6. 无二义性

对所有需求说明的读者都只能有一个明确统一的解释，由于自然语言极易导致二义性，所以尽量把每项需求用简洁明了的用户性的语言表达出来。避免二义性的有效方法包括对需求文档的正规审查，编写测试用例，开发原型以及设计特定的方案脚本。

7. 可验证性

检查一下每项需求是否能通过设计测试用例或其它的验证方法，如用演示、检测等来确定产品是否确实按需求实现了。如果需求不可验证，则确定其实施是否正确就成为主观臆断，而非客观分析了。一份前后矛盾，不可行或有二义性的需求也是不可验证的。

1.5.2 需求规格说明的特点

1. 完整性

不能遗漏任何必要的需求信息。遗漏需求将很难查出。注重用户的任务而不是系统的功能将有助于你避免不完整性。如果知道缺少某项信息，用 TBD（“待确定”）作为标准标识来标明这项缺漏。在开始开发之前，必须解决需求中所有的 TBD 项。

2. 一致性

一致性是指与其它软件需求或高层（系统，业务）需求不相矛盾。在开发前必须解决所有需求间的不一致部分。只有进行一番调查研究，才能知道某一项需求是否确实正确。

3. 可修改性

在必要时或为维护每一需求变更历史记录时，应该修订 SRS。这就要求每项需求要独立标出，并与别的需求区别开来，从而无二义性。每项需求只应在 SRS 中出现一次。这样更改时

易于保持一致性。另外，使用目录表、索引和相互参照列表方法将使软件需求规格说明更容易修改。

4. 可跟踪性

应能在每项软件需求与它的根源和设计元素、源代码、测试用例之间建立起链接链，这种可跟踪性要求每项需求以一种结构化的，粒度好（fine-grained）的方式编写并单独标明，而不是大段大段的叙述。第18章将详细说明需求的可跟踪性。

1.6 需求的开发和管理

需求中名词术语的混淆将导致对科目（规范，discipline）叫法的不一致。一些作者把整个需求范围称之为“需求工程”，另一些则称之为“需求管理”。我认为可把整个软件需求工程研究领域划分为需求开发（本书的第二部分）和需求管理（本书第三部分）两部分更合适，如图1-2所示：

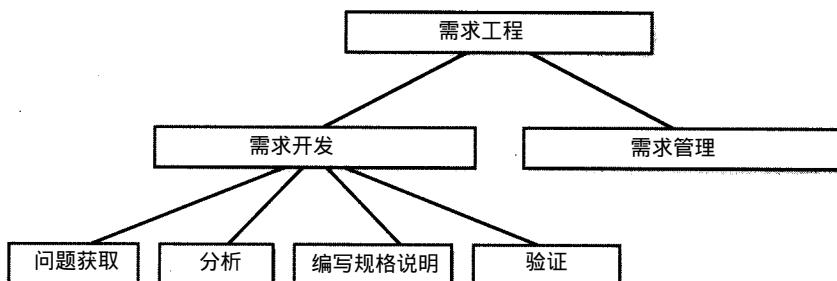


图1-2 需求工程域的层次分解示意图

需求开发可进一步分为：问题获取（elicitation）、分析（analysis）、编写规格说明（specification）和验证（verification）四个阶段（Thayer and Dorfman 1997）。这些子项包括软件类产品中需求收集、评价、编写文档等所有活动。需求开发活动包括以下几个方面：

- 确定产品所期望的用户类。
- 获取每个用户类的需求。
- 了解实际用户任务和目标以及这些任务所支持的业务需求。
- 分析源于用户的信息以区别用户任务需求、功能需求、业务规则、质量属性、建议解决方法和附加信息。
- 将系统级的需求分为几个子系统，并将需求中的一部份分配给软件组件。
- 了解相关质量属性的重要性。
- 商讨实施优先级的划分。
- 将所收集的用户需求编写成规格说明和模型。
- 评审需求规格说明，确保对用户需求达到共同的理解与认识，并在整个开发小组接受说明之前将问题都弄清楚。

需求管理需要“建立并维护在软件工程中同客户达成的契约”（CMU/SEI 1995）。这种契约都包含在编写的需求规格说明与模型中。客户的接受仅是需求成功的一半，开发人员也必须能够接受他们，并真正把需求应用到产品中。通常的需求管理活动包括：

- 定义需求基线（迅速制定需求文档的主体）。

- 评审提出的需求变更、评估每项变更的可能影响从而决定是否实施它。
- 以一种可控制的方式将需求变更融入到项目中。
- 使当前的项目计划与需求一致。
- 估计变更需求所产生影响并在此基础上协商新的承诺（约定）。
- 让每项需求都能与其对应的设计、源代码和测试用例联系起来以实现跟踪。
- 在整个项目过程中跟踪需求状态及其变更情况。

由图1-3中可以从另一个角度来看需求开发和需求管理之间的区别：

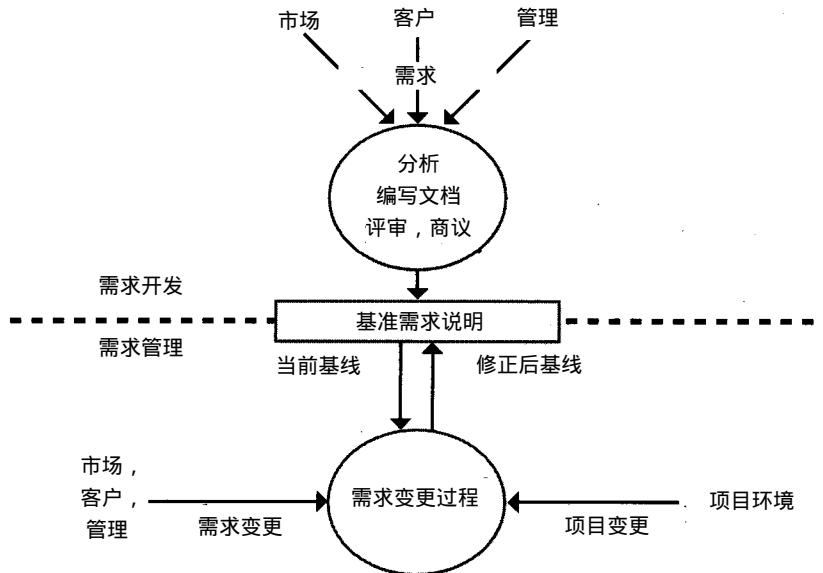


图1-3 需求开发与需求管理之间的界限

下一步：

- 记录你在当前项目或以前项目中所遇到的与需求相关的问题。指明每一个问题是需求开发问题还是需求管理问题，以及这些问题带来的影响及其产生的根本原因。
- 与你的组员和其他风险承担者（客户，市场调查人员，项目管理者）一起讨论当前或以前项目中的需求问题，及其产生的根源和带来的影响。向所有参与者指明，如果想解决这些困难，必须正视它，大家是否为此做好准备了呢？
- 整理出对整个项目人员一天训练用的软件需求课程，人员要包括重要的客户，市场人员和管理人员。训练是一种有效的团队学习与合作的方法。大家将会在训练中达成术语与技术上的共识，有利于相互交流沟通与协作。

第2章 客户的需求观

Contoso 制药公司的高级管理长官 Gerhard ,会见Contoso公司的信息开发小组的新管理员 Cynthia。“我们需要建立一套化学制品跟踪信息系统 ”, Gerhard说道。“该系统可以记录库房或某个实验室中已有的化学药品，这样，化学专家可以直接从楼下的某人那里拿到所需的药品，而不必再买一瓶新的。另外，卫生保健部门也得为联邦政府写些关于化学药品的使用报告。你们小组能在五个月内开发出该系统吗？”

“我已经明白这个项目的重要性了， Gerhard ” Cynthia说，“但在我制定计划前，我们必须收集一些系统的需求。”

Gerhard觉得很奇怪“你的意思是什么？我不是刚告诉你我的需求了吗？”

“实际上，你只说明了整个项目的概念与目标，” Cynthia解释道，“这些高层次的业务需求并不能为我们提供足够的详细信息以确定究竟要开发什么样的软件，以及需要多长时间。我需要一些分析人员与一些知道系统使用要求的化学专家进行讨论，然后才能真正明白达到业务目标所需的各种功能和用户的要求。我们甚至并不需要开发一个新的软件系统，这样可节省许多钱。”

Gerhard此前还从未遇到过与这位系统开发人员类似的看法。“那些化学专家都非常忙”他坚持道，“他们没有时间与你们详细讨论各种细节，你不能让你的手下的人说明要做的系统吗？”

Cynthia尽力解释从使用新系统的用户处收集需求的合理性。“如果我们只是凭空猜想用户要求，结果不会令人满意。我们只是软件开发人员，而并非化学专家。我们并不能真正明白化学专家们需要这个化学制品跟踪系统做些什么。我曾经尝试过，未真正明白这些问题就匆忙开始编码，结果没有人对产品满意。”

“行了，行了，我们没有那么多时间” Gerhard坚持道。“我来告诉你需求，请马上开始开发系统。随时将你们的进展情况告诉我。”

像这样的对话经常出现在软件开发过程中。要求开发一个新信息系统的客户通常并不懂得从系统的实际用户处得到信息的重要性。市场人员在有了一个很不错的新产品想法后，也就自认为能充分代表产品用户的兴趣要求。然而，直接从产品的实际用户处收集需求有着不可替代的必要性。通过对 8380 个项目的调查发现，导致项目失败的最主要的两个原因是缺乏用户参与和不完整的需求以及不完整的规格说明 (Standish 1995)。

引起需求问题的一部分原因是对不同层次需求（业务、用户、功能）的混淆所致。Gerhard说明了一些业务需求，但他并不能描述用户需求，因为他并不是“化学制品跟踪系统”的实际使用者。只有实际用户才能描述他们要用此系统必须完成的任务。但他们又不能指出完成这些任务所有具体的功能需求。

本章说明客户与开发人员之间的关系，它对软件项目开发的成功极为关键。我建议写一份软件客户需求权利书和相应的软件客户需求义务书，以强调客户（和实际用户）参与需求开发过程的重要性。

2.1 谁是客户

通常意义上，客户是指直接或间接从产品中获得利益的个人或组织。软件客户包括提出要求、支付款项、选择、具体说明或使用软件产品的项目风险承担者 (stakeholder)或是获得产品所产生的结果的人。

Gerhard代表支付、采购(procure)或投资软件产品的这类客户，处于 Gerhard层次上的客户有义务说明业务需求。他们应阐明产品的高层次概念和将发布产品的主要业务内容。在第 6章中讨论到业务需求应说明客户、公司和想从该系统获利的风险承担者或从系统中取得结果的用户所要求的目标。业务需求为后继工作建立了一个指导性的框架。其它任何说明都应遵从业务需求的规定，然而业务需求并不能为开发人员提供许多开发所需的细节说明。

下一层需求——用户需求——必须从使用产品的用户处收集。因此这些用户(通常称作最终用户)，构成了另一种软件客户。他们能说清楚要使用该产品完成什么任务和一些非功能性的特性，而这些特性会对使用户很好接收具有该特点的产品是重要的。

说明业务需求的客户有时将试图替代用户说话，但通常他们根本无法准确说明用户需求。因为对信息系统、合同(contract)或是客户应用程序的开发、业务需求应来自风险承担者，而用户需求则应来自产品的真正使用、操作者。

不幸的是，这两种客户可能都觉得他们没有时间与(收集、分析与编写需求说明)需求分析者讨论。有时客户还希望分析人员或开发人员无须讨论和编写文档就能说出用户的需求。除非遇到的需求极为简单，否则不能这样做。如果你的组织希望软件成功，那必须要花上数天时间来消除需求中模糊不清的地方和一些使程序人员感到困惑的方面。

商业软件开发的情况有些不同，因为通常其客户就是用户。正如市场部这类客户代理，可能想确定究竟软件产品的购买者会喜欢什么。但即使是商业软件，也应该让实际用户参与到收集需求的过程中来(第 7章将谈及)。如果你不这样做，那产品很可能会因缺乏足够用户提供的信息而出现不少隐患。

2.2 客户与开发人员之间的合作关系

优秀的软件产品是建立在优秀的需求基础之上的。而高质量的需求来源于客户与开发人员之间有效的交流与合作。通常，开发人员与客户或客户代理人，如市场人员间的关系反而会成为一种对立关系。双方的管理者都只想自己的利益而搁置用户提供的需求从而产生摩擦，在这种情况下，不会给双方带来一点益处。

只有当双方参与者都明白要成功自己需要什么，同时也应知道要成功合作方需要什么时，才能建立起一种合作关系。由于项目压力与日渐增，所有风险承担者有着一个共同的目标这一点容易被遗忘。其实大家都想开发出一个既能实现商业价值，又能满足用户需要，还能使开发者感到满足的优秀软件产品。

软件客户需求权利书列出了十条关于客户在项目需求工程实施中与分析人员、开发人员交流时的合法要求。每一项权利都对应着软件开发人员、分析人员的义务。而软件客户需求义务书也列出了十条关于客户在需求过程中应承担的义务。如果愿意，可以将其作为开发人员的权利书。

软件客户需求权利书

客户有如下权利：

1. 要求分析人员使用符合客户语言习惯的表达。
2. 要求分析人员了解客户系统的业务及目标。
3. 要求分析人员组织需求获取期间所介绍的信息，并编写软件需求规格说明。
4. 要求开发人员对需求过程中所产生的工作结果进行解释说明。
5. 要求开发人员在整个交流过程中保持和维护一种合作的职业态度。
6. 要求开发人员对产品的实现及需求都要提供建议，拿出主意。
7. 描述产品使其具有易用、好用的特性。
8. 可以调整需求，允许重用已有的软件组件。
9. 当需要对需求进行变更时，对成本、影响、得失（trade-off）有个真实可信的评估。
10. 获得满足客户功能和质量要求的系统，并且这些要求是开发人员同意的。

软件客户需求义务书

客户有下列义务：

1. 给分析人员讲解业务及说明业务方面的术语等专业问题。
2. 抽出时间清楚地说明需求并不断完善。
3. 当说明系统需求时，力求准确详细。
4. 需要时要及时对需求做出决策。
5. 要尊重开发人员的成本估算和对需求的可行性分析。
6. 对单项需求、系统特性或使用实例划分优先级。
7. 评审需求文档和原型。
8. 一旦知道要对项目需求进行变更，要马上与开发人员联系。
9. 在要求需求变更时，应遵照开发组织确定的工作过程来处理。
10. 尊重需求工程中开发人员采用的流程（过程）。

当为内部集团使用而开发软件时，这些权利和义务可以直接应用于客户。这也适用于那些有合同关系或者有明确的主要客户集的情况。对普遍市场产品的开发，这些权利和义务更适于像市场部这样的客户代理者。

作为项目计划的一部分，客户和开发人员应评审上述两张列表并达成共识。一些很忙的客户可能不愿意积极参与需求过程（也即，他们不太接受软件客户需求义务书），而缺少客户参与将很可能导致不理想的产品。故一定要确保需求开发中的主要参与者都了解并接受他们的义务。如果遇到分歧，通过协商以达成对各自义务的相互理解，这样能减少今后的摩擦（如一方要求而另一方不愿意或不能满足要求）。

2.2.1 软件客户需求权利书

权利#1：要求分析人员使用符合客户语言习惯的表达

需求讨论应集中于业务需要和任务，故要使用业务术语，你应将其教给分析人员，而你不一定要懂得计算机的行业术语。

权利#2：要求分析人员了解客户的业务及目标

通过与用户交流来获取用户需求、分析人员才能更好地了解你的业务任务和怎样才能使产品更好地满足你的需要。这将有助于开发人员设计出真正满足你的需要并达到你期望的优秀软件。为帮助开发人员和分析人员，可以考虑邀请他们观察你或你的同事是怎样工作的。

如果新开发系统是用来替代已有的系统，那么开发人员应使用一下目前的系统，这将有利于他们明白目前系统是怎样工作的，其工作流程的情况，以及可供改进之处。

权利#3：要求分析人员编写软件需求规格说明

分析人员要把从你和其他客户那里获得的所有信息进行整理，以区分开业务需求及规范、功能需求、质量目标、解决方法和其它信息。通过这些分析就能得到一份软件需求规格说明。而这份软件需求规格说明（software requirements specification, SRS）便在开发人员和客户之间针对要开发的产品内容达成了协议。SRS可以用一种你认为易于翻阅和理解的方式组织编写。要评审编写出的规格说明以确保它们准确而完整地表达了你的需求。一份高质量的软件需求规格说明能有助于开发人员开发出真正需要的产品。

权利#4：要求得到需求工作结果的解释说明

分析人员可能采用了多种图表作为文字性软件需求规格说明的补充。因为如工作流程图那样的图表能很清楚地描述出系统行为的某些方面。所以需求说明中的各种图表有着极高的价值。虽然它们不太难于理解，但是你很可能对此并不熟悉。因此可以要求分析人员解释说明每张图表的作用或其它的需求开发工作结果和符号的意义，及怎样检查图表有无错误及不一致等。

权利#5：要求开发人员尊重你的意见

如果用户与开发人员之间不能相互理解，那关于需求的讨论将会有障碍，共同合作能使大家“兼听则明”。参与需求开发过程的客户有权要求开发人员尊重他们并珍惜他们为项目成功所付出的时间。同样，客户也应对开发人员为项目成功这一共同目标所作出的努力表示尊重与感激。

权利#6：要求开发人员对需求及产品实施提供建议，拿出主意

通常，客户所说的“需求”已是一种实际可能的实施解决方案，分析人员将尽力从这些解决方法中了解真正的业务及其需求，同时还应找出已有系统不适合当前业务之处，以确保产品不会无效或低效。在彻底弄清业务领域内的事情后，分析人员有时就能提出相当好的改进方法。有经验且富有创造力的分析人员还能提出增加一些用户并未发现的很有价值的系统特性。

权利#7：描述产品易使用的特性

你可以要求分析人员在实现功能需求的同时还要注重软件的易用性。因为这些易用特性或质量属性能使你更准确、高效地完成任务。例如，客户有时要求产品要“用户友好”或“健壮”或“高效率”，但这对于开发人员来说，太主观了并无实用价值。正确的应是：分析人员通过询问和调查了解客户所要的友好、健壮、高效所包含的具体特性（第 11 章将详细讨论它）。

权利#8：调整需求，允许重用已有的软件组件

需求通常要有一定的灵活性。分析人员可能发现已有的某个软件组件与你描述的需求很相符。在这种情况下，分析人员应提供一些修改需求的选择以便开发人员能够在新系统开发中重用一些已有的软件。如果有可重用的机会出现，同时你又能调整你的需求说明，那就能降低成本和节省时间，而不必严格按原有的需求说明开发。所以说，如果想在产品中使用一些已有的商业常用组件，而它们并不完全适合你所需的特性，这时一定程度上的需求灵活性就显得极为重要了。

权利#9：要求对变更的代价提供真实可信的评估

有时人们面临更好、也更昂贵的方案时，会做出不同的选择。而这时，对需求变更的影响进行评估从而对业务决策提供帮助，是十分必要的，所以，你有权利要求开发人员通过分析给出一个的确真实可信的评估，包括影响、成本和得失等评估。开发人员不能由于不想实施变更而随意夸大评估成本。

权利#10：获得满足客户功能和质量要求的系统

每个人都希望项目获得成功。但这不仅要求你要清晰地告知开发人员关于系统“做什么”所需的所有信息，而且还要求开发人员能通过交流了解清楚取舍与限制。一定要明确说明你的假设和潜在的期望。否则，开发人员开发出的产品很可能无法让你满意。

2.2.2 软件客户需求义务书

义务#1：给分析人员讲解你的业务

分析人员要依靠你给他们讲解的业务概念及术语。但你不能指望分析人员会成为该领域的专家，而只能让他们真正明白你的问题和目标。不要期望分析人员能把握你们业务的细微与潜在之处，他们很可能并不知道那些对于你和你的同事来说理所当然的“常识”。

义务#2：抽出时间清楚地说明并完善需求

客户很忙，经常在最忙的时候还得参与需求开发。但无论如何，你有义务抽出时间参与“头脑风暴”会议的讨论，接受采访或其它获取需求的活动。有时分析人员可能先以为明白了你的观点，而过后发现还需要你的讲解。这时，请耐心一些对待需求和需求的精化工作过程中的反复，因为它是人们交流中的很自然的现象，何况这对软件产品的成功极为重要。

义务#3：准确而详细地说明需求

编写一份清晰、准确的需求文档是很困难的。由于处理细节问题不但烦人而且又耗时，故很容易留下模糊不清的需求。但是，在开发过程中，必须得解决这种模糊性和不准确性。而你恰是为解决这些问题作出决定的最佳人选。不然的话，你就只好靠开发人员去正确猜测了。

在需求规格说明中暂时加上待定（to be determined, TBD）的标志是个不错的办法。用该标志可指明了哪些需要进一步探讨、分析或增加信息的地方。不过，有时也可能因为某个特殊需求难以解决或没有人愿意处理它而注上 TBD 标志。尽量将每项需求的内容都阐述清楚，以便分析人员能准确的将其写进软件需求规格说明中。如果你一时不能准确表述，那就得允许获取必要的准确信息这样一个过程。通常使用所谓的原型技术。通过开发的原型，你可以同开发人员一起反复修改，不断完善需求定义。

义务#4：及时地作出决定

正如一位建筑师为你修建房屋，分析人员将要求你做出一些选择和决定。这些决定包括来自多个用户提出的处理方法或在质量特性冲突和信息准确度中选择折衷方案等。有权做出决定的客户必须积极地对待这一切，尽快做处理、做决定。因为开发人员通常只有等你做出了决定才能行动，而这种等待会延误项目的进展。

义务#5：尊重开发人员的需求可行性及成本评估

所有的软件功能都有其成本价格，开发人员最适合预算这些成本（尽管许多开发人员并不擅长评估预测）。你所希望的某些产品特性可能在技术上行不通，或者实现它要付出极为高

昂的代价。而某些需求试图在操作环境中要求不可能达到的性能或试图得到一些根本得不到的数据，开发人员会对此作出负面的评价意见，你应该尊重他们的意见。

有时，你可以重新给出一个在技术上可行、实现上便宜的需求，例如，要求某个行为在“瞬间”发生是不可行的，但换种更具体的时间需求说法（“在50ms以内”），这就可以实现了。

义务#6：划分需求优先级别

绝大多数项目没有足够的时间或资源来实现功能性的每个细节。决定哪些特性是必要的，哪些是重要的，哪些是好的，是需求开发的主要部分。只能由你来负责设定需求优先级，因为开发者并不可能按你的观点决定需求优先级。开发者将为你确定优先级提供有关每个需求的花费和风险的信息。当你设定优先级时，你帮助开发者确保在适当的时间内用最小的开支取得最好的效果。

在时间和资源限制下，关于所需特性能否完成或完成多少应该尊重开发人员的意见。尽管没有人愿意看到自己所希望的需求在项目中未被实现，但毕竟是要面对这种现实的。业务决策有时不得不依据优先级来缩小项目范围或延长工期，或增加资源，或在质量上寻找折衷。

义务#7：评审需求文档和原型

正如我们将在第14章讨论的，无论是正式的还是非正式的方式，对需求文档进行评审都会对软件质量提高有所帮助。让客户参与评审才能真正鉴别需求文档是否的确完整、正确说明了期望的必要特性。评审也给客户代表提供一个机会，给需求分析人员带来反馈信息以改进他们的工作。如果你认为编写的需求文档不够准确，就有义务尽早告诉分析人员并为改进提供建议。

通过阅读需求规格说明，很难想象实际的软件是什么样子的。更好的方法是先为产品开发一个原型。这样你就能提供更有价值的反馈信息给开发人员，帮助他们更好地理解你的需求。必须认识到：原型并非是一个实际产品，但开发人员能将其转变、扩充成功能齐全的系统。

义务#8：需求出现变更要马上联系

不断的需求变更会给在预定计划内完成高质量产品带来严重的负面影响。变更是不可避免的，但在开发周期中变更越在晚期出现，其影响越大。变更不仅会导致代价极高的返工，而且工期也会被迫延误，特别是在大体结构完成后又需要增加新特性时。所以一旦你发现需要变更需求时，请一定立即通知分析人员。

义务#9：应遵照开发组织处理需求变更的过程

为了将变更带来的负面影响减少到最低限度，所有的参与者必须遵照项目的变更控制过程。这要求不放弃所有提出的变更，对每项要求的变更进行分析、综合考虑，最后作出合适的决策以确定将某些变更引入项目中。

义务#10：尊重开发人员采用的需求工程过程

软件开发中最具挑战性的莫过于收集需求并确定其正确性。分析人员采用的方法有其合理性。也许你认为需求过程不太划算，但请相信花在需求开发上的时间是“很有价值”的。如果你理解并支持分析人员为收集、编写需求文档和确保其质量所采用的技术，那么整个过程将会更为顺利。尽管去询问分析人员为什么他们要收集某些信息，或参与与需求有关的活动。

2.3 “签约”意味着什么

为所开发产品的需求签订协议是客户与开发人员关系中的重要部分。许多组织在需求文档中使用“签约”这个概念来作为客户同意需求的标志行为。故要让所有需求参与者都真正明白“签约”的意思。

但存在这样一个问题：客户代表经常把“签约”看作是毫无意义的。“他们要我在一张纸的最后一行文字下面签上名字，于是我就签了，否则这些开发人员不开始编码。”这种态度将来会带来麻烦，譬如客户想更改需求或对产品有不满时。“不错，我是在需求上签署了名字，但我并没有时间去读完所有的内容。我是相信你们的，是你们非要让我签字的。”

同样的问题也会发生在仅把签约看作是完成文档的管理人员身上。一旦有需求变更出现，他便指着软件需求规格说明说道：“但你已经在需求上签约了，所以这些便是我们所要开发的。如果你想要别的什么，你应早些告诉我们。”

这样的态度都是不对的，不可能在项目早期就了解所有需求，而且毫无疑问需求将会上出现变更。在需求上签约是终止需求开发过程的正确方法。然而，参与者必须明白他们的签约意味着什么。

更为重要的是签名是建立在一个需求协议的基线上，因此在需求规格说明上的签约应该这样理解：“我同意这份文档表述了目前我们对项目软件需求的了解。进一步的变更可在此基线上通过项目定义的变更过程来进行。我知道变更可能会使我们要重新协商成本、资源和项目工期任务等”。

关于基线达成一定共识会易于忍受将来的摩擦，这些摩擦来源于项目的改进和需求的误差或市场和业务的新要求等。给初步的需求开发工作画上双方都明确的句号会有助于你形成一个持续良好的客户与开发人员的关系，为项目的成功奠定了基础。

下一步：

- 让客户提供项目的业务需求和用户需求。对权利书和义务书的条目，哪些被客户接受、理解并付诸实践了？哪些没有？
- 与你的主要客户一起讨论权利书和义务书，以达成协议，并付诸实践。这些行为会有助于客户和开发人员更好地互相理解，以形成更融洽的关系。
- 如果你是软件开发项目中的客户参与方，你感到你的需求权利书没有被充分尊重，就可以与软件项目的领导人员或业务分析人员一起讨论权利书的内容。要想建立一种合作的工作关系，就要尽力使对方对你的义务书感到满意。

第3章 需求工程的推荐方法

10年前，我曾热衷于追求软件的开发方法论的研究，将整套整套的模型、技术等用于解决项目难题。但现在我更注重应用“最佳方法”。最佳方法强调将软件工具包拆分成多个子包以分别应用于不同的问题，而并不是去设计或购买一整套的解决方案。即使你采用了一套商业上的方法，你也应当在其中增加那些在业界被认为行之有效的推荐技术。

“最佳方法”这个词值得讨论一下：谁能确定什么是“最佳”的呢？而且，得到这个结论的依据何在？一种方案是把这方面的专家召集起来分析众多不同组织中成功和失败的项目（Brown 1996）。专家们将那些成功项目中提供高效的方法和失败项目中导致低效甚至无效的方法都归纳出来。这样，专家们就能找到公认的能收到实效的关键方法。这些方法即是“最佳方法”，其本质就是有助于项目成功的有效方法。

本章的标题是“需求工程的推荐方法”而非“最佳方法”。下面分七类介绍了四十余种方法，能有助于开发小组做好需求工作，推荐方法如表 3-1 所列。

表3-1 需求工程推荐方法

知 识 技 能	需 求 管 理	项 目 管 理
<ul style="list-style-type: none"> • 培训需求分析人员 • 培训用户代表和管理人员 • 培训应用领域的开发人员 • 汇编术语 	<ul style="list-style-type: none"> • 确定变更控制过程 • 建立变更控制委员会 • 进行变更影响分析 • 跟踪影响工作产品的每项变更 • 编写需求文档的基准版本和控制版本 • 维护变更历史记录 • 跟踪需求状态 • 衡量需求稳定性 • 使用需求管理工具 	<ul style="list-style-type: none"> • 选择合适的生存周期 • 确定需求的基本计划 • 协商约定 • 管理需求风险 • 跟踪需求工作

需求开发

获 取	分 析	编 写 规 格 说 明 书	验 证
<ul style="list-style-type: none"> • 编写项目视图与范围 • 确定需求开发过程 • 用户群分类 • 选择产品代表 • 建立核心队伍 • 确定使用实例 • 召开应用程序开发联系（JAD）会议 • 分析用户工作流程 • 确定质量属性 • 检查问题报告 • 需求重用 	<ul style="list-style-type: none"> • 绘制关联图 • 创建开发原型 • 分析可行性 • 确定需求优先级 • 为需求建立模型 • 编写数据字典 • 应用质量功能调配（QFD） 	<ul style="list-style-type: none"> • 采用软件需求规格说明模版 • 指明需求来源 • 为每项需求注上标号 • 记录业务规范 • 创建需求跟踪能力矩阵 	<ul style="list-style-type: none"> • 审查需求文档 • 依据需求编写测试用例 • 编写用户手册 • 确定合格的标准

并非上面所有的条目都是最佳方法，或许也并非全部经过了系统地评估。但无论如何，我和许多实践者都觉得这些技术是很有效的 (Sommerville and Sawyer 1997)。其中每一条都在本章给予了简要介绍，并在其他章节或其他更详细地讨论技术来源的地方给出了参考文献。

表3-2把表3-1中的方法按实施的优先顺序和实施难度进行了分组。由于所列的方法都是有所裨益的，故最好是循序渐进，先从那些相对容易实施而对项目有很大影响的方法开始。

表3-2 实施需求工程的推荐方法

优先级别	难 度		
	高	中	低
高	<ul style="list-style-type: none"> • 确定需求开发过程 • 确定需求的基本计划 • 协商约定 	<ul style="list-style-type: none"> • 确定使用实例 • 确定质量属性 • 确定需求优先级 • 采用软件规格说明模板 • 确定变更控制过程 • 建立变更控制委员会 • 审查需求文档 	<ul style="list-style-type: none"> • 培训应用领域的开发人员 • 编写项目视图与范围 • 用户群分类 • 绘制关联图 • 指明需求来源 • 为每项需求注上标号 • 编写需求文档的基准版本和控制版本
	<ul style="list-style-type: none"> • 培训用户代表和管理人员 • 为需求建立模型 • 管理需求风险 • 使用需求管理工具 • 创建需求跟踪能力矩阵 	<ul style="list-style-type: none"> • 培训需求分析人员 • 建立核心队伍 • 创建开发原型 • 分析可行性 • 确定合格的标准 • 进行变更影响分析 • 跟踪影响工作产品的每项变更 • 选择合适的生存周期 	<ul style="list-style-type: none"> • 汇编术语 • 选择产品代表 • 编写数据字典 • 记录业务规范 • 依据需求编写测试用例 • 跟踪需求状态
	<ul style="list-style-type: none"> • 召开应用程序开发联系 (JAD) 会议 • 需求重用 • 应用质量功能调配 (QFD) • 衡量需求稳定性 	<ul style="list-style-type: none"> • 分析用户工作流程 • 检查问题报告 • 编写用户手册 • 维护变更历史记录 • 跟踪需求工作 	

不要想着把所有这些方法都用于你的下一个项目。而应该考虑将其中的一些方法推荐到你的需求工具箱中。不管你的项目处在开发的哪个阶段，你都可以马上开始应用某些方法，譬如变更管理的处理。其它如需求获取等可以在你的下一个项目开始时付诸应用。当然其它一些方法也可能并不适合你目前的项目。

第4章介绍一些用于评价需求工程的方法，并可设计一张实施需求方法改进的步骤图。具体的改进方法在此处和第4章中都给予介绍。

3.1 知识技能

绝大部分的软件开发人员都没有接受过高效需求工程所需技能的正规培训。但许多开发人员在职业生涯中的某个阶段总会扮演一个需求分析员的角色，与客户一起工作：收集，分析，编写需求文档。不能过高期望开发人员在需求工程的信息沟通中的“天份”。一定的培训将有助于提高需求分析员的能力和水平。

因为需求对项目成功极为重要，所有的项目风险承担者都应对需求工程的重要性、合理性及其方法有一个基本的了解。把项目风险承担者（例如开发人员，市场人员，客户，测试人员和管理人员）召集起来进行为期一天的需求过程概要学习，这对建立一个合作团队是很有效的。所有参与者都会更好地明白各自所面临的挑战是什么，以及为了整个团队获得成功大家都需要作些什么。同样，开发人员也能对应用领域的术语和一些基本概念有大致的了解。

1) 培训需求分析人员 所有的开发人员都应接受一个基本的需求工程培训。但那些负责收集(capturing)、编写文档和分析用户需求的人员应当进行为期一周或更长时间的培训。把高水平的需求人员组织起来，通过良好的信息交流，了解应用领域并有效地应用需求工程中的成熟技术。

2) 培训软件需求的用户代表和管理人员 参与软件开发的用户代表应接受为期一天左右关于需求工程的培训，开发管理者和客户管理者也应参加。这样的培训将使他们明白强调需求的重要性，以及忽略需求带来的风险。参加过我组织的需求讨论会的一些用户表示，他们在此之后更能理解软件开发人员了。

3) 让开发人员了解应用领域的基本概念 组织一些简短的关于客户业务活动、术语、目标等方面的讨论会以帮助开发人员对应用领域有个基本了解。这能减少误解及工程中的返工。你可能要为每位开发人员安排一个用户伙伴以便在项目过程中解释业务术语和概念。产品代表就应该扮演这样的角色。

4) 编写项目术语汇编 为减少沟通方面的问题，编一部术语汇编将项目应用领域的专用词汇给予定义说明，既要包括那些有多种含义与用法的术语，也要包括那些在专用领域和一般使用中有不同含义的词。

3.2 需求获取

第1章讨论了需求的三个层次：业务，用户和功能。在项目中它们在不同的时间来自不同的来源，也有着不同的目标和对象，并需以不同的方式编写成文档。业务需求（或产品视图和范围）不应包括用户需求（或使用实例），而所有的功能需求都应该源于用户需求。同时你也需要获取非功能需求，如质量属性。你可以在下列章节中找到相关主题的详细内容：

- 第4章——确定需求开发过程。
- 第6章——编写项目视图和范围文档。
- 第7章——将用户群分类并归纳其特点，为每个用户类选择产品代表（product champion）。
- 第8章——让用户代表确定使用实例。
- 第11章——确定质量属性和其它非功能需求。

1) 确定需求开发过程 确定如何组织需求的收集、分析、细化并核实的步骤，并将它编写成文档。对重要的步骤要给予一定指导，这将有助于分析人员的工作，而且也使收集需求活动的安排和进度计划更容易进行。

2) 编写项目视图和范围文档 项目视图和范围文档应该包括高层的产品业务目标，所有的使用实例和功能需求都必须遵从能达到的业务需求。项目视图说明使所有项目参与者对项目的目标能达成共识。而范围则是作为评估需求或潜在特性的参考。

- 3) 将用户群分类并归纳各自特点 为避免出现疏忽某一用户群需求的情况，要将可能使

用产品的客户分成不同组别。他们可能在使用频率、使用特性、优先等级或熟练程度等方面都有所差异。详细描述出它们的个性特点及任务状况，将有助于产品设计。

4) 选择每类用户的产品代表 为每类用户至少选择一位能真正代表他们需求的人作为那一类用户的代表并能作出决策。这对于内部信息系统的开发是最易实现的，因为此时，用户就是身边的职员。而对于商业开发，就得在主要的客户或测试者中建立起良好的合作关系，并确定合适的产品代表。他们必须一直参与项目的开发而且有权作出决策。

5) 建立起典型用户的核心队伍 把同类产品或你的产品的先前版本用户代表召集起来，从他们那里收集目前产品的功能需求和非功能需求。这样的核心队伍对于商业开发尤为有用，因为你拥有一个庞大且多样的客户基础。与产品代表的区别在于，核心队伍成员通常没有决定权。

6) 让用户代表确定使用实例 从用户代表处收集他们使用软件完成所需任务的描述——使用实例，讨论用户与系统间的交互方式和对话要求。在编写使用实例的文档时可采用标准模板，在使用实例基础上可得到功能需求。

7) 召开应用程序开发联系会议 应用程序开发联系（JAD）会议是范围广的、简便的专题讨论会（workshop），也是分析人员与客户代表之间一种很好的合作办法，并能由此拟出需求文档的底稿。该会议通过紧密而集中的讨论得以将客户与开发人员间的合作伙伴关系付诸于实践（Wood and Silver 1995）。

8) 分析用户工作流程 观察用户执行业务任务的过程。画一张简单的示意图（最好用数据流图）来描绘出用户什么时候获得什么数据，并怎样使用这些数据。编制业务过程流程文档将有助于明确产品的使用实例和功能需求。你甚至可能发现客户并不真地需要一个全新的软件系统就能达到他们的业务目标（McGraw and Harbison 1997）。

9) 确定质量属性和其它非功能需求 在功能需求之外再考虑一下非功能的质量特点，这会使你的产品达到并超过客户的期望。这些特点包括性能、有效性、可靠性、可用性等，而在这些质量属性上客户提供的信息相对来说就非常 important了。

10) 通过检查当前系统的问题报告来进一步完善需求 客户的问题报告及补充需求为新产品或新版本提供了大量丰富的改进及增加特性的想法，负责提供用户支持及帮助的人能为收集需求过程提供极有价值的信息。

11) 跨项目重用需求 如果客户要求的功能与已有的产品很相似，则可查看需求是否有足够的灵活性以允许重用一些已有的软件组件。

3.3 需求分析

需求分析（requirement analysis）包括提炼、分析和仔细审查已收集到的需求，以确保所有的风险承担者都明白其含义并找出其中的错误、遗漏或其它不足的地方。分析员通过评价来确定是否所有的需求和软件需求规格说明都达到了第1章中优秀需求说明的要求。分析的目的在于开发出高质量和具体的需求，这样你就能作出实用的项目估算并可以进行设计、构造和测试。

通常，把需求中的一部分用多种形式来描述，如同时用文本和图形来描述。分析这些不同的视图将揭示出一些更深的问题，这是单一视图无法提供的（Davis 1995）。分析还包括与客户的交流以澄清某些易混淆的问题，并明确哪些需求更为重要。其目的是确保所有风险承

担者尽早地对项目达成共识并对将来的产品有个相同而清晰的认识。下面几章对需求分析中的任务进行了详细讨论：

- 第6章——绘制系统关联图。
- 第9章——建立数据字典。
- 第10章——为需求建立模型。
- 第12章——建立用户接口原型。
- 第13章——确定需求优先级。

1) 绘制系统关联图 这种关联图是用于定义系统与系统外部实体间的界限和接口的简单模型。同时它也明确了通过接口的信息流和物质流。

2) 创建用户接口原型 当开发人员或用户不能确定需求时，开发一个用户接口原型——一个可能的局部实现——这样使得许多概念和可能发生的事更为直观明了。用户通过评价原型将使项目参与者能更好地相互理解所要解决的问题。注意要找出需求文档与原型之间所有的冲突之处。

3) 分析需求可行性 在允许的成本、性能要求下，分析每项需求实施的可行性，明确与每项需求实现相联系的风险，包括与其它需求的冲突，对外界因素的依赖和技术障碍。

4) 确定需求的优先级别 应用分析方法来确定使用实例、产品特性或单项需求实现的优先级别。以优先级为基础确定产品版本将包括哪些特性或哪类需求。当允许需求变更时，在特定的版本中加入每一项变更，并在那个版本计划中作出需要的变更。

5) 为需求建立模型 需求的图形分析模型是软件需求规格说明极好的补充说明。它们能提供不同的信息与关系以有助于找到不正确的、不一致的、遗漏的和冗余的需求。这样的模型包括数据流图、实体关系图、状态变换图、对话框图、对象类及交互作用图。

6) 创建数据字典 数据字典是对系统用到的所有数据项和结构的定义，以确保开发人员使用统一的数据定义。在需求阶段，数据字典至少应定义客户数据项以确保客户与开发小组是使用一致的定义和术语。分析和设计工具通常包括数据字典组件。

7) 使用质量功能调配 质量功能调配(QFD)是一种高级系统技术，它将产品特性、属性与对客户的重要性联系起来。该技术提供了一种分析方法以明确那些是客户最为关注的特性。QFD将需求分为三类：期望需求，即客户或许并未提及，但如若缺少会让他们感到不满意；普通需求；兴奋需求，即实现了会给客户带去惊喜，但若未实现也不会受到责备（Zultner 1993;Pardee 1996）。

3.4 需求规格说明

无论你的需求从何而来，也不管你是怎样得到的，你都必须用一种统一的方式来将它们编写成可视文档。业务需求要写成项目视图和范围文档。用户需求要用一种标准使用实例模板编写成文档。而软件需求规格说明（requirement specification）则包含了软件的功能需求和非功能需求。你必须为每项需求明确建立标准的惯例，并确定在 SRS 中采用任何惯例，以确保 SRS 的统一风格，同时读者也会明白怎样解释它。下列章节讨论了关于编写需求文档的几个方面：

- 第8章——记录业务规范。
- 第9章——采用SRS模板；为每项需求注上标号。
- 第18章——指明需求来源；创建需求跟踪能力矩阵。

1) 采用SRS模板 在你的组织中要为编写软件需求文档定义一种标准模板。该模板为记录功能需求和各种其它与需求相关的重要信息提供了统一的结构。注意，其目的并非是创建一种全新的模板，而是采用一种已有的且可满足项目需要并适合项目特点的模板。许多组织一开始都采用IEEE标准830-1998(IEEE 1998)描述的SRS模板。要相信模板是很有用的，但有时要根据项目特点进行适当的改动。

2) 指明需求的来源 为了让所有项目风险承担者明白SRS中为何提供这些功能需求，要都能追溯每项需求的来源，这可能是一种使用实例或其它客户要求，也可能是某项更高层系统需求、业务规范、政府法规、标准或别的外部来源。

3) 为每项需求注上标号 制定一种惯例来为SRS中的每项需求提供一个独立的可识别的标号或记号。这种惯例应当很健全，允许增加、删除和修改。作了标号的需求使得需求能被跟踪，记录需求变更并为需求状态和变更活动建立度量。

4) 记录业务规范 业务规范是指关于产品的操作原则，比如谁能在什么情况下采取什么动作。将这些编写成SRS中的一个独立部分，或一独立的业务规范文档。某些业务规范将引出相应功能需求；当然这些需求也应能追溯相应业务规范。

5) 创建需求跟踪能力矩阵 建立一个矩阵把每项需求与实现、测试它的设计和代码部分联系起来。这样的需求跟踪能力矩阵同时也把功能需求和高层的需求及其它相关需求联系起来了。在开发过程中建立这个矩阵，而不要等到最后才去补建。

3.5 需求验证

验证是为了确保需求说明准确、完整地表达必要的质量特点。当你阅读软件需求规格说明(SRS)时，可能觉得需求是对的，但实现时，却很可能会出现问题。当以需求说明为依据编写测试用例时，你可能会发现说明中的二义性。而所有这些都必须改善，因为需求说明要作为设计和最终系统验证的依据。客户的参与在需求验证(requirement verification)中占有重要的位置，第14章还将进一步讨论它。

1) 审查需求文档 对需求文档进行正式审查是保证软件质量的很有效的方法。组织一个由不同代表(如分析人员，客户，设计人员，测试人员)组成的小组，对SRS及相关模型进行仔细的检查。另外在需求开发期间所做的非正式评审也是有所裨益的。

2) 以需求为依据编写测试用例 根据用户需求所要求的产品特性写出黑盒功能测试用例。客户通过使用测试用例以确认是否达到了期望的要求。还要从测试用例追溯回功能需求以确保没有需求被疏忽，并且确保所有测试结果与测试用例相一致。同时，要使用测试用例来验证需求模型的正确性，如对话框图和原型等。

3) 编写用户手册 在需求开发早期即可起草一份用户手册，用它作为需求规格说明的参考并辅助需求分析。优秀的用户手册要用浅显易懂的语言描述出所有对用户可见的功能。而辅助需求如质量属性、性能需求及对用户不可见的功能则在SRS中予以说明。

4) 确定合格的标准 让用户描述什么样的产品才算满足他们的要求和适合他们使用的。将合格的测试建立在使用情景描述或使用实例的基础之上(Hsia, Kung, and Sell 1997)。

3.6 需求管理

当你完成需求说明之后，不可避免地还会遇到项目需求的变更。有效的变更管理需要对

变更带来的潜在影响及可能的成本费用进行评估。变更控制委员会与关键的项目风险承担者要进行协商，以确定哪些需求可以变更。同时，无论是在开发阶段还是在系统测试阶段，还应跟踪每项需求的状态。

建立起良好的配置管理方法是进行有效需求管理（ requirement management ）的先决条件。许多开发组织使用版本控制和其它管理配置技术来管理代码，所以你也可以采用这些方法来管理你的需求文档，需求管理的改进也是将全新的管理配置方法引入项目的组织中的一种方法。下列章节讨论了需求管理涉及到的各种技术：

- 第16章——建立需求基准版本和需求控制版本文档。
- 第17章——确定需求变更控制过程；建立变更控制委员会。
- 第18章——进行需求变更影响分析；跟踪所有受需求变更影响的工作产品。
- 第19章——使用需求管理工具。

1) 确定需求变更控制过程 确定一个选择、分析和决策需求变更的过程。所有的需求变更都需遵循此过程，商业化的问题跟踪工具都能支持变更控制过程。

2) 建立变更控制委员会 组织一个由项目风险承担者组成的小组作为变更控制委员会，由他们来确定进行哪些需求变更，此变更是否在项目范围内，估价它们，并对此评估作出决策以确定选择哪些，放弃哪些，并设置实现的优先顺序，制定目标版本。

3) 进行需求变更影响分析 应评估每项选择的需求变更，以确定它对项目计划安排和其它需求的影响。明确与变更相关的任务并评估完成这些任务需要的工作量。通过这些分析将有助于变更控制委员会作出更好的决策。

4) 跟踪所有受需求变更影响的工作产品 当进行某项需求变更时，参照需求跟踪能力矩阵找到相关的其它需求、设计模板、源代码和测试用例，这些相关部分可能也需要修改。这样能减少因疏忽而不得不变更产品的机会，这种变更在变更需求的情况下是必须进行的。

5) 建立需求基准版本和需求控制版本文档 确定一个需求基准，这是一致性需求在特定时刻的快照。之后的需求变更就遵循变更控制过程即可。每个版本的需求规格说明都必须是独立说明，以避免将底稿和基准或新旧版本相混淆。最好的办法是使用合适的配置管理工具在版本控制下为需求文档定位。

6) 维护需求变更的历史记录 记录变更需求文档版本的日期以及所做的变更、原因，还包括由谁负责更新和更新的新版本号等。版本控制工具能自动完成这些任务。

7) 跟踪每项需求的状态 建立一个数据库，其中每一条记录保存一项功能需求。保存每项功能需求的重要属性，它包括状态（如已推荐的，已通过的，已实施的，或已验证的），这样在任何时候都能得到每个状态类的需求数量。

8) 衡量需求稳定性 记录基准需求的数量和每周或每月的变更（添加、修改、删除）数量。过多的需求变更“是一个报警信号”，意味着问题并未真正弄清楚，项目范围并未很好地确定下来或是政策变化较大。

9) 使用需求管理工具 商业化的需求管理工具能帮助你在数据库中存储不同类型的需求，为每项需求确定属性，可跟踪其状态，并在需求与其它软件开发工作产品间建立跟踪能力联系链。

3.7 项目管理

软件工程管理方法在本质上与项目的需求过程是紧密相关的。项目计划建立在功能基础

之上，而需求变更会影响这些计划。因此，项目计划应能允许一定程度的需求变更或项目范围的扩展。如果刚开始，需求不能确定，你可以选择一种软件开发方法生存周期以允许这种不确定性，并在弄清后逐渐实施。关于需求工程项目管理（project management）方法的更详细内容将在以下章节讨论：

- 第5章——编写文档和管理与需求相关的风险。
- 第15章——基于需求的项目计划。
- 第16章——记录需求开发和管理中的工作。

1) 选择一种合适的软件开发方法生存周期 经典的瀑布法软件开发生命周期只适用于需求说明在项目早期即可全部完成的情况。你的组织应根据不同类型的项目和需求说明的不同程度选择几种不同的方法（McConnell 1996）。如果需求说明在项目早期无法全部确定，则从最为清晰易懂的需求开始，建立一个健壮的可修改的结构，再逐渐增加补充。实现了部分特性的产品可作为早期版本发布（Gilb 1998）。

2) 基于需求的项目计划 随着需求细节不断变得清晰、完善，项目开发计划的进度安排将会不断改变。一开始可以根据项目视图和范围对开发功能需求所需的工作量作一估算，建立在不甚完善的需求基础之上的成本、进度安排的估计很不可靠，但随着需求说明的完善，估计也会得到不断改善。

3) 发生需求变更时协商项目约定 当在项目中添加新的需求时，估计一下你是否能在目前安排下，利用现有资源保质保量完成。如果不能，将项目的实现与管理联系起来，协商一下新的、切实可行的约定（Humphrey 1997）。如果协商不成功，则将可能的后果和更新项目风险管理计划联系起来，以反映出对项目成功的新的不利因素。

4) 编写文档和管理与需求相关的风险 采用自由讨论的方法并将与需求相关的项目风险编写成文档。利用各种方法来减轻或阻止这些风险，实施这些方法并跟踪其发展及效果。

5) 跟踪需求工程所耗的工作量 记录需求开发和管理活动所花费的工作量。利用这些数据可以评估计划的需求活动是否已达到所期望的要求，并可以为将来项目的需求工程提供更好的所需资源的计划。

下一步：

- 回到第1章的下一步中你已明确了与需求有关的一些问题。从本章中学到的推荐的实践方法，可能会有助于解决这些问题。针对建议的每一种方法，要在你组织中发现那些可能给其实现带来困难的障碍。
- 将在先前步骤中被确认是好的需求方法整理成一张表。针对每个方法，指明你的项目的能力水平：专家，熟练者，生手或新手。如果你的队伍中无一人熟练的话，就得要求项目的某个参与者学习更多的知识，并将他所学的教给队伍中的其他人。

第4章 改进需求过程

第3章介绍了几十种需求工程中的好方法，你应当考虑在实践中应用它们。把理论方法付诸实践是改进软件过程（process）的核心所在。从根本上说，改进过程包括使用更多有效的方法避免使用过去使用过的令人头痛的方法。然而，改进之路却是从失败、错误开始，还要历经诸如受人为抵制的影响及因任务的时间紧迫导致改进被搁置这样的挫折。

软件开发过程的改进有以下两个主要目标：

- 1) 解决在以前项目或目前项目中遇到的问题。
- 2) 防止和避免你可能在将来的项目中要遇到的问题。

如果目前采用的方法好像也挺有效的，你可能觉得没有必要改变你的方法。但是，即便是很成功的软件组织在面临大项目、不同的客户群、紧迫的进度安排或全新的应用领域时也会感到力不从心。因此，至少你应该知道其它一些很有价值也颇有效的需求工程方法，并把它们加入到你的软件工程中。

本章介绍了需求与其它主要的项目过程和风险承担者之间的联系、关于软件开发过程改进的一些基本概念并推荐了一种经改进的生存期。我把一些重要的需求“过程精华”罗列出来以供参考使用。本章还介绍了实施改进需求工程实践的一个流程蓝图。

4.1 需求与其他项目过程的联系

需求是软件项目成功的核心所在，它为其他许多技术、管理活动奠定了基础。变更你的需求开发和管理方法将对其他项目过程产生影响，反之亦然。需求与其他过程的联系见图 4-1。下面简要介绍各过程间的接口。

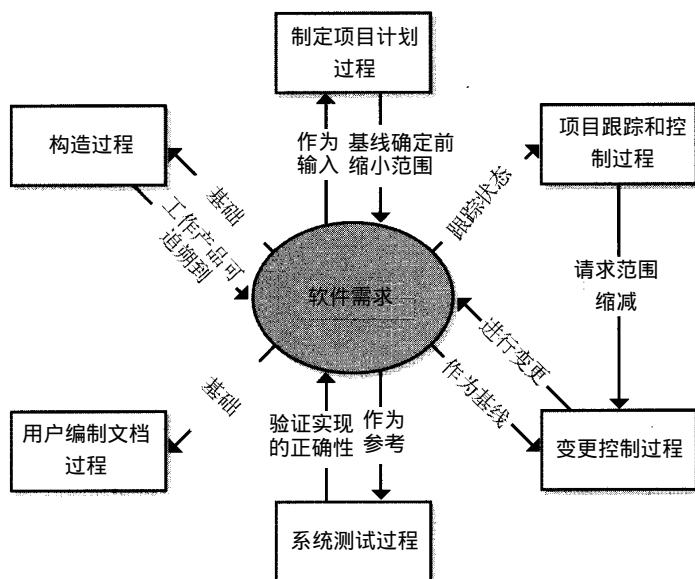


图4-1 需求与其他项目过程的关系

1) 制定项目计划 需求是制定项目计划的基础。因为开发资源和进度安排的估计都要建立在对最终产品的真正理解之上。通常，项目计划指出所有希望的特性不可能在允许的资源和时间内完成，因此，需要缩小项目范围或采用版本计划对功能特性进行选择。

2) 项目跟踪和控制 监控每项需求的状态，以便项目管理者能发现设计和验证是否达到预期的要求。如果没有达到，管理者通常请求变更控制过程来进行范围的缩减。

3) 变更控制 在需求编写成文档并制定基线以后，所有接下来的变更都应通过确定的变更控制过程来进行。变更控制过程能确保：

- 变更的影响是可以接受的。
- 受到变更影响的所有人都接到通知并明白这一点。
- 由合适的人选来作出接受变更的正式决定。
- 资源按需进行调整。
- 保持需求文档是最新版本并是准确的更新文档。

4) 系统测试 用户需求和功能需求是系统测试的重要参考。如果未说明清楚产品在多种多样条件下的期望行为，系统测试者将很难明确正确的测试内容。反过来说，系统测试是一种方法，可以验证计划中所列的功能是否按预期要求实现了。同时，也验证了用户任务是否能正确地执行。

5) 用户编制文档 我曾在一个办公室里工作，办公室里有为商业产品准备用户文档的技术写作人员。我咨询其中一位写作人员为什么他们要工作那么长时间。“我们是食物链的终结者”她回答道，“我们要编写出用户显示界面及性能的最终变更版本”。产品的需求是编写文档的重要参考，低质量和拖延的需求会给编写用户文档带来极大的困难。

6) 构造 软件项目主要产品是交付可执行软件，而不是需求说明文档。但需求文档是所有设计、实现工作的基础。要根据功能要求来确定设计模块，而模块又要作为编写代码的依据。采用设计评审的方法来确保设计正确地反映了所有的需求。而代码的单元测试能确定是否满足了设计规格说明和是否满足了相关的需求。跟踪每项需求与相应的设计和软件代码。

4.2 软件需求对其他项目风险承担者的影响

当软件开发队伍改变他们的需求过程时，与其他项目风险承担者沟通的接口也会发生变化。图4-2说明了一些外部组织功能，这些功能是通过一定的接口与软件开发队伍联系的，这些接口对项目需求活动起着重要作用。

为能顺利进行这些接口操作，要与其他领域的合作者多交流，让他们知道你的改进想法和调整计划。要向他们说明改进后的新过程会带来什么好处。如在改进过程中需要获得合作时，可以从这样的谈话开始：“这些是我们曾经经历过的问题，而我们认为进行这些变更将会有助于问题的解决。这就是为什么我们要这样做的原因，我们需要得到你们的帮助。而我们的这些工作也会给你们帮助的”。反对变更是由于害怕变更带来的影响，因此要指明你进行的过程变更所可能带来的影响，从而减少大家的恐惧感。

向各个功能领域的人说明你从他们那里所需要获取的信息和帮助，从而有助于成功地开发整个产品。在开发过程中要遵从开发组与其他功能领域之间重要交流接口的规范和内容，如系统需求规格说明文档或市场需求文档。通常重要项目的文档从写作者角度是严格规范的，但往往不能给客户提供他们所真正需要的全部信息。

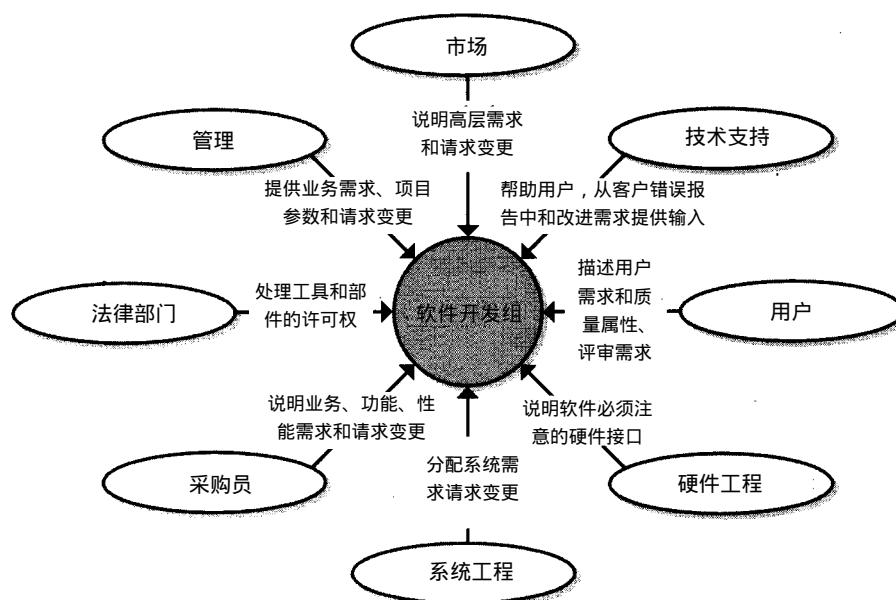


图4-2 软件开发组与其他组织间的重要需求接口

另一方面，询问其他组织需要从开发队伍中获取什么以有助于他们的工作。技术可行性方面哪些能帮助市场部更好地完成产品计划？什么样的需求状态报告能使管理者更充分地看清楚项目的进展情况？与系统工程部之间怎样合作才能确保系统需求在软、硬件间的分配合理？努力在开发组和其他需求过程风险承担者之间建立合作关系以便所有人都能更有效地促进项目成功。

人们都不喜欢被迫离开他们已经习惯的环境，因此，你可能会在需求过程变更中面临抵制与反对。要做好思想准备尽量理解这些反对的缘由，以便你既能尊重他又能化解他。许多反对是由于不了解情况而引起的恐惧所造成的，因此一开始就要给他们说清楚为什么要作这些变更，变更后他们将受到怎样的影响，将会带来什么好处以及为什么你在过程改进一开始就需要他们的参与等等。下面是一些你可能遇到抵制的情况：

- 需求变更控制过程可能被看成变更是很难进行的一个障碍而被丢弃。而实际上，它提供了结构化和有条理的变更过程，并使得知道的人能作出更好的业务决定。你的任务是要确保变更过程真正能起作用。如果新的过程不能带来更好的结果，那大家将会“绕道而行”了。
- 一些开发人员把编写和审查需求文档看作是浪费时间的官僚做法，妨碍他们的“真正工作”——编写代码。如果你能向他们讲清一旦发生重写代码所带来的惨重代价，开发人员和管理人员将更能明白为什么需要做好需求工作。
- 如果客户支持的费用没有和开发过程联系起来，开发小组可能会缺少变更的动力，因为他们并不会因最终产品的低质量而带来损失。
- 如果改进后的请求过程的目标是通过创建高质量产品以减少技术支持费用，那么提供技术支持的管理者可能会感到受到威胁。谁希望看到自己的帝国衰败呢？

4.3 软件过程改进的基础

阅读这本书可能是因为你想改变目前在需求工程中采用的一些方法。在为优秀的需求而

努力工作时，请铭记下面四条改进软件的原则（Wiegers 1996a）：

1) 改进过程应该是革命性的、彻底的、连续的、反复的 不要期望一次就能改进全部的过程，并且要能接受第一次尝试变更时，可能并没做好每一件事。不要奢求完美，要从某些过程的改进、实施开始。当你有一些新技术的经验后，可逐渐调整你的方法。

2) 人们和组织机构都只有在他们获得激励时才愿意变更 而变更引起的最强烈的刺激是痛苦。我的意思并不是要人为地制造痛苦（比如管理者强加的进度压力使开发人员工作异常痛苦），而是你曾在以前项目中经历过的真正艰辛。这些痛苦的激励作用远远超过管理者说：“这本书告诉我们必须做这些新的事情，因此让我们开始吧！”下面是一些历史问题的例子，也许能为需求过程的变更提供驱动力：

- 项目没有时限，因为需求说明变得超想象的复杂。
- 开发人员不得不大量超时工作，因为误解或二义性的需求直到开发后期才发现。
- 系统测试白费了，因为测试者并未明白产品要做什么。
- 功能都实现了，但由于产品的低性能、使用不方便或其它因素用户不满意。
- 维护费用相当高，因为客户的许多增强要求未在需求获取阶段提出。
- 开发组织落得交付一项客户并不想要的产品的名声，声誉受损。

3) 过程变更是面向目标的 在开始运用高级过程之前，先确保你知道变更的目标。是想减少需求问题引起返工的工作量？还是想更好地控制需求变更？或是想在实施中不要遗漏某项需求？有一份明确规定实施蓝图将会有助于你在改进过程中取得成功。

4) 将改进活动看作一些小项目 许多改进活动一开始就失败了。因为缺乏计划或是因为所需资源并未给予。为避免这些问题，把每个改进行为看作一个项目。把改进所需的资源和任务纳入工程项目的总计划中。执行计划、跟踪、衡量和报告那些已在软件开发项目中所做的改进，缩减改进项目的规模。为每个过程改进领域写一份活动计划。跟踪风险承担者们执行计划的情况，看是否获得了预期的资源并知道改进过程实际消耗的费用。

4.4 过程改进周期

图4-3说明了一个软件过程改进的生存期。这种方法我曾经采用过，很有实效。这个周期反映出在活动前知道自己处于哪个阶段的重要性，为改进活动制定计划的必要性以及从自己经验中所学到的持续过程改进的重要性。

4.4.1 评估当前采用的方法

任何过程改进活动的第一步都是评估当前组织中使用的方法。找出其优势和缺陷所在。评估本身不能带来任何改进，但能提供信息，评估为你正确选择变更奠定了基础。

可以用不同的方法来评估当前过程。如果你已在尝试前面章节末尾的“下一步”，那你已经开始对你的需求方法及其结果在进行非正式的评估了。设计自我评价问卷是一种系统方法，它能以较低费用对当前过程进行评估。

一种更彻底的方法是让来自外部的顾问客观地评估你目前的软件开发方法。这种正式过程的评估方法要以一种已建立的过程改进框架工作为基础，如软件工程研究所（CMU/SEI 1995）开发的软件功能成熟度模型（CMM）。评估者将会检查软件开发和管理过程，而不限于需求活动。要根据你想通过的过程改进取得的业务目标来选择评估方法，不要过多担心是

否满足CMM或其它专用模型的需求。

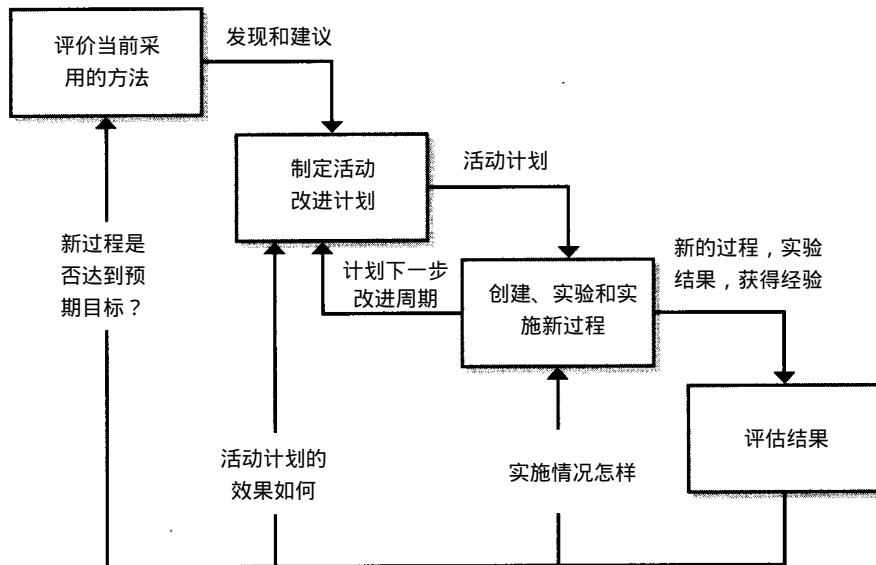


图4-3 软件开发过程改进的周期

本书附录包含了一个用于对目前需求方法的自我评估问卷表，可用这张问卷表来评估组织目前的需求工程方法。这样的自我评估将帮助你决定需求过程中的哪一个环节最需要进行改进。但仅仅根据在某项问题上的低分还不足以说明需要立即改进。要将注意力更多地放在对将来项目的成功带来最大风险和困难的领域。自我评估中的每一个问题都是本书中某一章节的主要论题。Motorola公司也开发了一套相似的“软件需求质量模型”用于软件需求过程评估（Smith 1998）。

正式的评估将获得一个列表——关于目前方法的长处和短处的说明——和关于改进机会的说明与推荐。不正式的评估，比如自我评估问卷，能使你了解并有助于你选择改进领域。在本书的有关章节将发现许多针对自我评估问题的相应推荐方法。分析所考虑的每一项改进活动以确保它能在允许费用下实施。选择那些可能给投资带来相当回报的改进活动。

4.4.2 制定改进活动计划

遵从将过程改进活动看作是项目这一“哲学”，在评估后制定一个活动计划。考虑制定出描述组织整个软件过程改进初始工作的战略计划和在各个特定改进领域的战术行动计划，正如你收集需求时所采用的方法。每项战术行动计划应该指明改进行动的目标、风险承担者和一些必须完成的活动条目。如没有计划，则更容易疏忽比较重要的任务。计划也提供了跟踪过程的方法，使你能监控各活动条目完成情况。

图4-4举例说明了我经常使用的过程改进活动计划模板。在每一个活动计划中不要超过10个条目，这样使得计划简单易于取得早期成功。例如，我看到一个需求管理改进的计划包括如下活动条目：

- 1) 起草一个需求变更控制过程草案。
- 2) 评审并修改变更控制过程。

- 3) 以一个项目 A 来实验(pilot)变更控制过程。
- 4) 以实验反馈为基础修改变更控制过程。
- 5) 评估问题跟踪工具并选择其一来支持变更控制过程。
- 6) 定制并购买问题跟踪工具以支持变更控制过程。
- 7) 在组织中使用新的变更控制过程和工具。

将每项活动条目交给专门的人来负责完成。不要让整个小组作为活动条目的主人，小组不会做工作，个人才会做。

如果你需要的活动条目多于 10 条，则先把注意力放在最重要的条目上，然后再处理其它条目。记住，变更是周期性反复的。之后介绍的过程改进蓝图将说明怎样才能把多个改进活动组成一个完整的软件开发过程改进计划。

需求过程改进的活动计划

项目：	日期：
<项目名称>	<编写计划的日期>
目标：	
<成功执行这份计划后希望达到的一些目标。说明业务方面的目标，而不是过程变更方面的。>	
成功度量：	
<描述怎样确定过程变更是否达到了预期要求。>	
组织受影响的范围：	
<说明在本计划中所描述的过程变更带来影响的广度。>	
人员和风险承担者：	
<明确谁实施该计划，每个人的角色，及投入时间承诺（按小时 /周或百分比为基础计算）。>	
跟踪和报告过程：	
<说明怎样跟踪计划中的活动条目进展情况，以及报告其状态结果等。>	
依赖、风险和限制：	
<明确对计划成功有帮助或有阻碍的各种外部因素。>	
估计所有活动的完成日期：	
<希望该计划什么时候完成？>	
活动条目：	
<为每个活动计划写出了 3-10 个活动条目。>	

活动条目	负责人	截止日期	目标	活动描述	结果	所需资源
<顺序号>	<负责人>	<目标日期>	<本活动 条目标>	<实施活动条目 要采取的行为>	<建立规程、模 板或其它过程 评估方法>	<各种所需的外部 资源；包括物质 材料、工具、文 档或其他人员>

图4-4 软件开发过程改进的活动计划模板

4.4.3 建立、实验和实施新的过程

到目前为止，已经对需求方法进行了评估并起草了一份活动计划，指出了很可能带来收获的过程领域。现在进入较困难的一步：实施计划。许多过程改进在试图由计划付诸实践时，一开始便夭折了。

实施一项活动计划意味着开发新的、更好的方法，并且相信它能提供一个比目前过程更好的结果。然而，并非第一次就能使新过程完美无缺。许多看起来很不错的办法付诸实施后会变得既不实用又低效。因此，要为你建立的新过程或文档模板计划一个“实验”。运用在实验中获取的经验来调整新技术，这样将它运用于整个目标群体时，改进活动会更有效果。请铭记下面这些关于引导实验的建议：

- 选择实验参与者（participant），他们将尝试新方法并提供反馈信息，这些参与者可以是生手也可以是老手，但他们不应该对过程改进持有强烈的反对意向。
- 确定用于评估实验的标准，使得到的结果易于解释。
- 通知那些需要知道实验是什么以及为什么要实施的工程风险承担者。
- 考虑在不同的项目中实验新过程的不同部分。用这种方式可使更多的人尝试新方法，因此能提高认知水平，增加反馈信息。
- 作为评估的一部分工作，询问实验参与者，如果他们不得不回头采用他们原有的工作方法，他们会觉得怎样。

即便是很有激励与善于理解的队伍，他们接纳变更的能力也是很有限的。所以不要一次给予项目或队伍太多的期望。编写出一整套实施计划，明确你将怎样把新方法运用于整个项目队伍以及你能提供的训练和支持；同时也要考虑管理者怎样阐明他们对新过程的期望。一种正式的关于需求工程和需求管理的文件通常要阐述清楚管理人员的任务和期望（CMU /SEI 1995）。

4.4.4 评估结果

过程改进周期的最后一步就是评估已实施的活动及取得的成果。这样的评估有助你在将来的改进活动中做得更好。评估实验工作进行得如何，采用新过程解决问题是否很有效。下一次在管理过程实验工作时是否需要稍作变更。

同时也要考虑整个新过程在群体中执行的情况。是否能使每个人都明白新过程或模板的好处？参与者是否理解并成功地应用了新过程？是否在下次工作中需要有所变更？

其中关键的一步是评估新实施的过程是否带来了期望的结果。尽管有一些新技术和管理办法都带来明显的改进，但更多的却需要时间来证明其全部的价值。例如，如果你实施一种新过程来处理需求变更，你就能很快看到项目变更以一种更规范的方式在进行。然而，一个新的软件需求规格说明 SRS 模板需要一段时间来证明其价值，因为分析人员和客户已习惯了

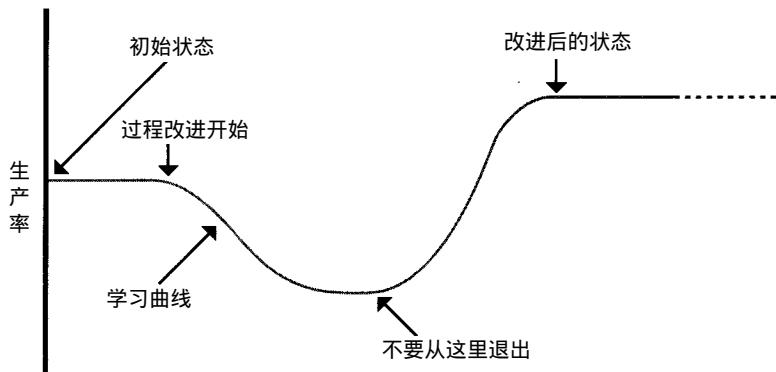


图4-5 过程改进学习曲线

一种需求文档的格式。给予新方法以足够的运行时间，选定能说明每项过程变更成功与否的衡量标准。

要接受学习曲线的事实。当从业者（practitioner）花费时间去吸收新方法时，生产率会降低，如图4-5所示。这种短期的生产率降低是组织进行过程改进的一部分投入。如果你不理解这一点，可能在得到回报之前就半途而废了，白白损失了投入而没有回报。对你的管理人员和同事进行有关学习曲线的教育，并使其明白：采用高级的需求过程，将会获得更广泛的项目和业务回报。

4.5 需求过程的积累材料

如果想要项目不断取得满意的结果，你需要有效地执行需求工程的各个过程：信息获取、分析、编写规格说明、验证以及管理。为了执行这些步骤，你应当把过程中积累的材料收集起来。过程包含已完成的活动和可交付的产品。过程中积累的材料有助于小组成员一致而有效地执行过程，还有助于大家理解他们遵从的步骤及要开发的产品。积累的材料包括下面几种类型的文档：

- | | |
|------|---|
| 检查清单 | 清单列出各项活动，交付的结果和其它应注意或验证的条目。检查清单是用来提示记忆的，有助于确保处于忙碌中的工作人员不要忽略重要细节。 |
| 实例 | 一种特定类型工作产品的代表，积累起能在你组织中运用的更好的实例。 |
| 计划 | 概括说明怎样完成目标与完成时需要什么样的文档。 |
| 方针 | 确立活动期望、产品期望和交付产品期望的指导原则。过程都应遵从的方针。 |
| 过程 | 描述完成某个活动的任务顺序或步骤，说明要执行的任务及其在项目中所扮演的角色。不要包括示范信息。 |
| 过程描述 | 一组完成某些目的活动文档的定义。过程描述应包括过程目标、里程碑、参与者和执行任务的适合时间、交流步骤，期望结果以及与过程相关的输入和输出数据（Caputo 1998）。 |
| 模板 | 一种完成整个工作产品的指导方式。重要工程文档的模板提醒你检查是否遗漏了什么。一个结构很好的模板提供了许多捕获和组织信息的栏目（slot）。模板中包含的指导信息将帮助文档作者有效地使用它。 |

图4-6指出一些过程中应积累的材料，使需求开发和需求管理能在项目中更有效地进行，没有哪个软件过程规则书会说你必须拥有所有这些条目，但它们对你在整个需求开发和管理过程会有所帮助。

需求开发过程的积累材料	需求管理过程的积累材料
<ul style="list-style-type: none">• 项目视图与范围模板• 需求开发过程• 需求分配过程• 使用实例模板• 软件需求规格说明模板• 需求优先级确定过程• SRS 和使用实例审查清单	<ul style="list-style-type: none">• 变更控制过程• 变更控制委员会过程• 需求变更影响分析检查清单和模板• 需求状态跟踪过程• 需求跟踪能力矩阵模板

图4-6 需求开发和管理的重要过程的积累材料

图4-6列出的过程并不需要写在独立的文档中。例如，一个完整需求管理过程的描述可以包括变更控制过程、状态跟踪过程和影响分析清单。参见附录JCMM实施向导(Caputo1998)。

下面是对图4-6中所列条目的简要说明，在相关的章节有详细介绍。请记住，每项工程都应调整组织的过程来满足其需要。

4.5.1 需求开发过程的积累材料

1) 项目视图与范围模板 项目的视图与范围文档明确了项目的概念性功能，并提供了确定需求优先级和变更的参考。需求视图与范围文档是简明扼要的、高度概括的新项目业务需求说明。用统一的方式编写项目视图与范围文档能确保在项目进行过程中作决定时能考虑到所有应考虑的情况。第6章推荐了一个需求视图与范围文档的模板。

2) 需求开发过程 该过程介绍了怎样确定客户及从客户那里获取需求的技术。也描述了项目。需要创建的各种需求文档和分析模型。这个过程还指明了每项需求包含的信息种类，比如：优先级、预计的稳定性或计划发行版本号。同时还应指明需求分析及需求文档检验需要执行的步骤以及确认软件需求规格说明和建立需求基线的步骤。

3) 需求分配过程 把高层的产品需求分成若干特定子系统是非常重要的，尤其是当开发的系统既含有软件又含有硬件或是包括多个子系统的软件产品时尤为重要(Nelsen 1990)。需求分配是在系统级需求完成和系统体系结构确定后才进行的，这个过程包含的信息是怎样执行分配以确保功能分配到合适的系统组件中，同时也说明分配的需求怎样才能追溯回它们的上两级系统需求以及在其它子系统中的相关需求。

4) 使用实例模板 使用实例模板提供了一种把每项用户希望使用软件系统完成的任务编写成文档的标准方法。使用实例定义包括一个简要的任务介绍，必须处理的异常情况的说明和描述用户任务特点的附加信息。使用实例可作为软件需求规格说明中一条独立的功能需求。另外，你也可将使用实例与SRS模板合并为一个文档，既包括产品的使用实例，又包括软件功能需求，第8章介绍了一种使用实例模板。

5) 软件需求规格说明模板 软件需求规格说明模板提供了一种组织功能需求和非功能需求的结构化方法。采用标准的SRS模板将有助于创建统一且高质量的需求文档。可能要采用多个模板以适应组织承担的不同类型和规模的项目。这样可减少因一种“万能”模板并不适合你的项目所带来的障碍。第9章介绍了一种SRS模板样例。

6) 需求优先级确定过程 我的一个朋友Matt把软件项目的最后阶段称作“快速缩减范围阶段(rapid descoping phase)”，此时为满足进度时限要求，计划的功能不得不放弃掉。我们需要知道哪些性能、使用实例或功能需求的优先级最低，以便在任何阶段，我们都可适当缩减范围。第13章介绍了一种优先级确定过程及一种工具，它能综合考虑需求对客户的价值、相应的技术风险及实施成本费用。

7) SRS和使用实例审查清单 对需求文档的正式审查是保证软件质量的一项重要措施。审查清单指出在需求文档中发现的一些错误。在审查会议的准备中运用清单将使你的注意力集中到通常存在问题的地方。第14章包含了SRS和使用实例审查清单样例。

4.5.2 需求管理过程的积累材料

1) 变更控制过程 变更控制过程能够减少因无休止、失控的需求变更引起的混乱。它明

确了一种方法来提出、协商、评估一个新的需求或在已有需求上的一项变更。变更控制通常需要问题跟踪工具的支持，但请铭记工具并不能替代过程。第 17 章详细介绍了变更控制过程。

2) 变更控制委员会过程 变更控制委员会(CCB)是由风险承担者的主要成员组成的，对提出的需求变更决定执行哪一项，拒绝哪一项，以及在各产品发行版本中包括哪些变更。CCB 过程描述了变更控制委员会的组成及操作过程。CCB 的主要活动是对提出的变更进行影响分析，为每项变更作出决定，并且告知那些将受到影响的人。第 17 章进一步讨论了 CCB 的组成及其功能。

3) 需求变更影响分析清单和模板 估计提出的需求变更的成本费用和影响是决定是否执行变更的重要步骤。影响分析能帮助 CCB 作出正确的决定。如在第 18 章中说明的，影响分析清单包括许多自问自答型的问题，如：要考虑到可能的任务、边界影响、实施所确定的变更引起的相关的潜在风险。一张参与人员工作表可以作为估计任务工作量的简单方法，从这里就能明白确认变更的复杂性。第 18 章还提供了一个用于展示执行需求变更影响分析结果的模板样例。

4) 需求状态跟踪过程 需求管理包括监控和报告每项功能需求的状态和状态改变的条件。你需要一个数据库或一种商业需求管理工具来跟踪一个复杂系统中大量的需求状态。此过程也描述了当你随时查看收集到的需求状态时输出的报告格式。如要获得关于需求状态跟踪方面更多的内容请参见第 16 章。

5) 需求跟踪能力矩阵模板 需求跟踪能力矩阵列出了 SRS 中的所有功能需求及相应的设计模块，源文件和实施需求的过程，还有验证需求实施正确性的测试用例。跟踪能力矩阵应该也可以指出对应的上一层用户需求或系统需求。第 18 章将具体介绍需求跟踪能力。

4.6 需求过程改进路标

要知道改进你们组织的整个需求工程过程可不是件小事。毫无计划地进行改进很容易失败。所以，应当为实施改进需求过程开发一个路标。该路标应是软件开发过程改进战略计划的一部分内容。如果你已尝试进行前面介绍的评估方法，那么你已对采用的技术过程的优点及存在的缺陷有一定的了解。所以现在需要把这些改进活动排序以便能用最小的投资得到最多的收益。过程改进流程图描述了改进活动的一种前后次序。

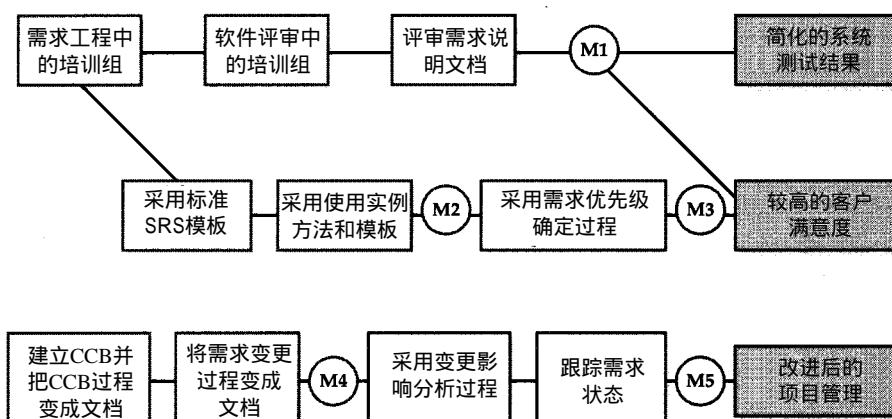


图4-7 需求过程改进路标样例

因为有各种不同的情况，我不可能提供一种“万能”的路标。公式化的方法不能替代思考和常识，图4-7说明一个组织改进它的需求过程的路标。期望的业务结果写在图的右边方框里。主要的改进活动在其它方框里，一些中间的里程碑（圆圈）指明了取得的期望业务目标。从左到右实施每组改进活动。一旦已经建立了一个类似的路标，让一个人负责一个里程碑活动，他得为获得里程碑制定活动计划，然后把计划付诸实践！

下一步：

- 完成附录中的目前需求实践自我评估。以你目前实践缺陷的影响严重程度为基础，确定需求过程的三个最佳改进机会。
- 确定图4-6中列出的哪项需求工程的积累材料在你组织中还没有，但你认为会很有用。
- 在前面两步的基础上，建立一个需求过程改进路标。说服你组织的某个人来负责某项里程碑活动。让每位负责人写一份活动计划，用于实施活动，采用图4-4中介绍的活动计划模板。当实施计划时，跟踪各活动条目的进展情况。

第5章 软件需求与风险管理

负责Contoso制药公司“化学制品跟踪系统”的项目管理人员 Dave会见他的首席程序员Helen和首席测试员Ramesh。他们对新项目都很有兴趣，但他们也记得在以前一个称作“药品仿真”的项目中遇到的问题。

“还记得我们直到进入测试时才发现用户对仿真程序的用户界面极为不满意吗？”Helen问道。“我们花了五周时间重新实现，重新测试，我可再不愿玩这样的死亡游戏了。”

“的确是烦人，”Dave附和道。“同样麻烦的是那些用户提出一大堆没人用过的特性，这样的交互导致编码花费了预计时间的三倍，我们是不管好歹，编完了事，简直是废品！”

“我们太匆忙了，以至没有时间写详细的需求说明”Ramesh回忆道。“测试人员有一半的时间都在问程序员怎样才能判断他们的程序工作正常，以便能测试它。可是程序员设计的一些功能根本就不是用户所要求的。”

“特别麻烦的是，要求开发药品仿真的管理者根本没有看需求规格说明就在上面签字确认了。”Dave补充道：“于是我们不断遇到要求新的特性及各种变更，所以工程超期四个月，成本费用超出预算的一倍这也就不足为怪了。若再发生这样的事，我肯定会被解雇了。”

Ramesh建议道：“也许我们应该把在仿真项目中遇到的问题一一列出来以便我们能在化学制品跟踪系统中避免重蹈覆辙。我看了篇关于软件风险管理的文章，上面介绍说我们应指出各种风险并说明了怎样才能避免它们”。

“我可不那样想”Dave坚持道：“我们已从仿真项目学到了不少，我们不会再有那些问题了。这个项目还没有达到需要用风险管理的地步。如果要把我们可能犯的错误都写下来，好像我连怎样做软件项目都不知道似的。我不想要任何消极想法影响项目。我们必须为成功而制定计划。”

正如Dave的最后一句话所反映的那样，软件工程师都是绝对的乐观主义者。我们总是希望我们的下一个项目进行顺利，而忽略以前项目发生的问题。事实却是许多潜在威胁阻碍项目按计划进行。与Dave的想法恰恰相反的是，软件项目管理者必须要明确和控制他们的项目风险，并且要从需求工程的风险开始进行。

所谓风险是可能给项目的成功带来威胁或损失的情况。这种情况还没有发生，也没有带来问题，而你希望它永远不会发生。但这些潜在的问题可能会给项目成本费用、进度安排、技术方面、产品质量及团队工作效率等带来较大的负面影响。而风险管理——一种软件工业的最佳方法——就是在风险给项目带来损失之前，就指明、评估并对风险加以控制。如果不希望发生的事已经发生了，那就不再是风险，而是事实了。只好通过项目事务(ongoing)状态跟踪和校正过程来处理当前的问题。

正如没有人能确切地预测未来，风险管理也仅是让你采取一些措施尽可能减少潜在问题

发生的可能性或减少其带来的影响。风险管理的意思是在一种担忧转变为危机或实际困难之前处理它。这将提高项目成功的可能性且可减少不可避免的风险造成的损失。对处于个人控制领域之外的风险应由相应层次的管理者来负责。

由于需求说明在软件项目中扮演着一个核心的角色，故精明的项目管理者会在初期就指明与需求相关的风险并积极地控制它们。典型的需求风险包括对需求的误解、不恰当的用户参与、不确定或随意变更项目的范围和目标以及持续变更需求。项目管理者只能通过与客户、或客户代表（如市场人员）的合作来控制需求风险。合作编写需求风险文档，共同制定减轻风险的措施，增强客户与开发人员之间的合作伙伴关系，这在第2章中已作介绍了。

不仔细研究是不能把风险撵走的，因此本章对需求风险管理进行简略的介绍。本章后面还会提到需求工程中常出现的一些风险因素。运用这些信息可以使你在风险攻击项目前处理风险。

5.1 软件风险管理基础

除了与项目范围和需求有关的风险外，项目还面临着许多风险。依赖于外界实体，例如一个转包承揽者或生产重用部件的另一个项目就是一种常见的风险来源。项目管理一直面临各种风险挑战：不准确的估计、对准确估计的否决、对项目状态不清楚及资金的周转的困难。技术风险威胁着高度复杂或很前沿(leading-edge)的开发项目，缺乏知识是另一种风险源，以及参与者对所用的技术和应用领域很陌生等等。强制的或总是变更的政府规范会使一个很好的计划彻底作废。

很可怕吧？这就是为什么所有的项目都应该认真地进行风险管理的原因，风险管理是不断察看水平线上是否出现了冰山，而不是以充足的自信认为船不会沉就以全速挺进。注意同其他过程一样，让你的风险管理活动与工程规模相适应。小规模的工程可以只列出一张简单的风险清单，但对于一个大规模项目的成功，正式的风险管理计划则显得非常重要。

5.1.1 风险管理的要素

风险管理就是使用某些工具和步骤把项目风险限制在一个可接受的范围内。风险管理提供了一种标准的方法来指出风险并把风险因素编成文档，评估其潜在的威胁，以及确定减少这些风险的战略(Williams, Walker, and Dorofee 1997)。风险管理包括的活动如图5-1所示。

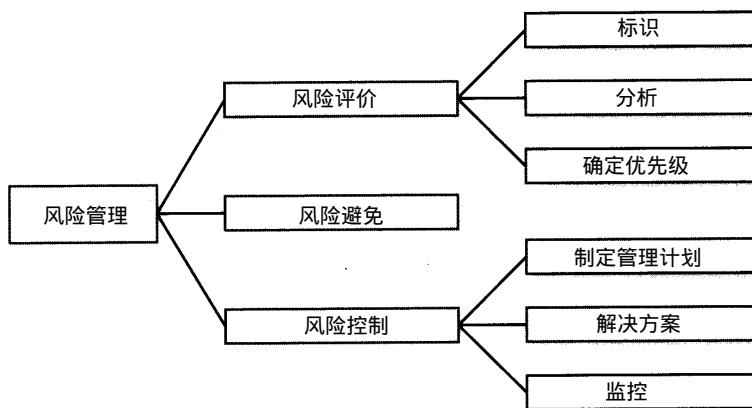


图5-1 风险管理要素

风险评价(risk assessment)是一个检查工程项目并识别潜在风险区域的过程。可以通过

列举通常的软件项目风险因素，如需求风险因素的办法来使风险识别（risk identification）更加方便容易。本章后面描述了一些需求风险因素（Carr et al 1993； McConnell 1996）。在风险分析中，应检查一些特定风险对项目可能造成的潜在后果。风险分级（risk prioritization）有助于你通过评价每项风险的潜在危害值，优先处理最严重的风险。风险危害值（risk exposure）包括带来损失的可能性大小和潜在损失的规模。

风险避免（risk avoidance）是处理风险的一种方法：尽量别作冒险的事。如果你不承担任何项目，采用成熟而并非处于研究阶段的技术，或者将难以实现的特性都排除在项目之外你就可以避开风险。

但更常见的是，需要采取风险控制（risk control）的方法来管理那些已被发现为高优先级的风险。制定风险管理计划是一项处理具有一旦发生，影响较大的风险的计划，包括降低风险的方法、应急计划、负责人和截止日期。应尽量避免让风险成为真正的问题，或即便问题发生了，也应尽量让其影响降低到最小。风险不能够自我控制，所以风险解决方案就包括了降低、减少每项风险的执行计划。最后，通过风险监控（risk monitoring）来跟踪风险解决过程的进展情况。这也是例外的项目状态跟踪的一部分内容。监控可以很好了解降低风险工作的进展情况，可以定期地修订先前风险清单的内容和划分的优先级。

5.1.2 编写项目风险文档

仅仅认识到项目面临的风险是远远不够的。应该将其编写成文档并妥善进行管理，这样在整个项目开发过程中有利于风险承担者了解风险情况和状态。图 5-2 提供了一个编写一个条目（item）风险说明的模板。你可能觉得以表格形式存放在电子表格中更加方便，因为那样更容易把各项风险进行排序（列表）。但将风险列表（清单）组成一独立文档以便在整个项目中进行升级和维护。

风险条目跟踪模板

序列号：	<顺序号>
确定日期：	<风险被识别出的日期>
撤销日期：	<撤销风险确定日期>
描述：	<以“条件-结果”的形式描述风险>
可能性：	<风险转变为问题的可能性>
影响：	<如果风险变成了事实将造成的损失>
危害值：	<可能性 × 影响>
降低风险计划：	<一种或多种用来控制、避免、最小化及降低风险的方法>
负责人：	<解决风险的责任承担者>
截止日期：	<完成降低风险措施的截止日期>

图5-2 风险条目跟踪模板

在编写风险说明时，最好采用条件——结果的形式。也就是，先说明你关心的条件，接着是潜在的有害结果（如果风险成为事实）。有时，人们只说明了风险条件（如“客户不同意产品的需求说明”）或者只说明了结果（“我们只能满足某些主要的客户”）。最好将这样的说明句子合并成条件——结果形式的结构：“如果有些客户不赞同产品的需求说明，那我们只能满足某些主要客户的意见。”而一个条件下可能有多个结果，同时也可能出现多个条件下导致同一个结果。

模板能记录风险变为事实的可能性及对项目的消极影响，还有整个的风险危害值（可能性×影响）。我用0.1（极不可能）到1.0（肯定发生）来描述可能性，用1（无甚么影响）到10（有很深、很大的影响）来表示影响。将这两个因素相乘即可作为评估风险危害值的依据。

不要试图精确量化风险。你的目标是将最有威胁的风险和那些不急需处理的风险区别开来。大家可能更愿意用高、中和低来估计可能性及影响。但风险条目中至少应有一个为高的风险。

制定降低风险计划来明确控制风险要采取的活动，其中一些策略是尽量降低风险发生的可能性；而另一些则是减少风险发生后带来的影响。做计划时要考虑降低风险所耗费用，千万别花费20 000美元来控制一项仅会损失10 000美元的风险。为每项风险安排一个负责人，并确定完成活动的截止日期。长期或复杂的风险可能需要具有多个阶段性成果的多步骤降低风险策略计划。

图5-3说明了本章开始部分介绍的“化学制品跟踪系统”小组领导者讨论的一个风险。小组凭他们以前的经验估计了风险的可能性及其影响。除非他们把其它风险因素也估计出来，否则他们并不明白风险危害值4.2究竟有多严重。降低风险措施的前两条是通过更多的用户参与项目来减少风险发生的可能性。而采用原型法则可以利用用户关于界面的早期反馈来减少风险的潜在影响。

5.1.3 制定风险管理计划

一张风险列表还不等于一个风险管理计划。对于一个小项目，你可以把控制风险的计划放在软件项目管理计划里。但一个大项目则需要一份独立的风险管理计划，包括用于识别、评估、编写、跟踪风险的各种方法与途径。这份计划还应包括风险管理活动的角色和责任。你可能希望专门让一个项目风险管理人员负责可能引起麻烦的事。

通常，项目小组为他们的关键活动制定了计划，却在项目中没有按计划去实施或者未能按实际情况进行及时的调整。要坚持按照所采取的风险管理活动计划去执行。项目的进度安排上也应给风险管理留出足够时间来确保项目并未浪费早期投资在风险计划制定上。工程项目的工作分类细目结构中包括降低风险的活动、状态报告，以及更新风险清单。

和其它项目管理活动一样，你需要建立起周期性的监控措施。保持对十来个有最大危害的风险的高度重视，并追踪降低风险措施的有效性。当完成一项活动后，重新评估那项风险的可能性和影响，更新风险清单和其它相关的计划。当控制住原本有很高优先级的风险后，必然有新条目会进入前十条。请记住，不要仅仅因为完成了一项降低风险的活动而人为增加一条风险来进行控制。应当想想你降低风险的方法是否的确减少了风险的危害，使其减少到了一个可以接受的水平。

化学制品跟踪系统的风险条目样例

序列号：

1

确定日期：

5/4/99

撤消日期：

描述：

需求获取中无合适用户参与，导致测试之后用户界面的返工。.

可能性：

0.6

影响：

7

危害值：

4.2

降低风险计划：

1. 在第一阶段早期就要收集易学、易用的需求。
2. 与产品代表一起召开JAD会议以开发需求。
3. 通过与产品代表和顾问的交流，开发一个包含核心功能的用户界面原型。让产品代表和其他用户来评估此原型。

负责人：

Helen

截止日期：

在6/16/99前完成JAD会议。

图5-3 化学制品跟踪系统的风险条目样例

5.2 与需求有关的风险

下面介绍的风险因素是按需求工程中获取、分析、编写规格说明、验证和管理汇总起来的，并推荐了一些方法用于降低风险发生的可能性或减轻风险发生给项目带来的影响。这张清单仅仅是一个起点，在你做项目逐渐积累经验过程中，加入你的风险因素清单和减轻风险的策略。使用这里提供的条目来帮助你识别需求风险并采用条件——结果的格式来书写风险说明。

5.2.1 需求获取

1) 产品视图与范围 如果团队成员没有对他们要做的产品功能达成一个清晰的共识，则很可能导致项目范围的逐渐扩大。因此最好在项目早期写一份项目视图与范围将业务需求涵盖在内，并将其作为新的需求及修改需求的指导。

2) 需求开发所需时间 紧张的工程进度安排给管理者造成很大的压力，使他们觉得不赶紧开始编码将无法按时完成项目，因而对需求一带而过。项目因其规模和应用种类不同（如信息系统，系统软件，商业的或军事的应用）而有着很大的不同。粗略的统计表明：需求开发工作应占全部工作量的 15% (Rubin 1999)。记录你参与的每个项目中实际需求开发的工作量，这样就能知道所花的时间是否合适并改进将来项目的工作计划。

3) 需求规格说明的完整性和正确性 为确保需求是客户真正需要的，要以用户的任务为中心，应用使用实例技术获取需求。根据不同的使用情景编写需求测试用例，建立原型，使需求对用户来说更加直观，同时获取用户的反馈信息。让客户代表对需求规格说明和分析模型进行正式的评审。

4) 对革新产品的请求 有时容易忽略市场对产品的反馈信息。故要强调市场调查研究，建立原型，并运用客户核心小组来获得革新产品的任务的反馈信息。

5) 明确非功能需求 由于一般强调产品的功能性要求，非常容易忽略产品的非功能性的需求。询问客户关于产品性能、使用性、完整性、可靠性等质量特性，编写非功能需求文档和验收标准，(像在SRS中一样)作为可接受的标准。

6) 客户赞同产品需求 如果不同的客户对产品有不同的意见，那最后必将有些客户会不满意。确定出主要的客户，并采用产品代表的方法来确保客户代表的积极参与，确保在需求决定权上有正确的人选。

7) 未加说明的需求 客户可能会有一些隐含的期望要求，但并未说明。要尽量识别并记录这些假设。提出大量的问题来提示客户以充分表达他们的想法、主意和应关注的一切。

8) 把已有的产品作为需求基线 在升级或重做的项目中需求开发可能显得不很重要。开发人员有时被迫把已有的产品作为需求说明的来源。“只是修改一些错误和增加一些新特性”，这时的开发人员不得不通过现有产品的逆向工程(reverse engineering)来获取需求。可是，逆向工程对收集需求是一种既不充分也不完整的方法。因此新系统很可能会有一些与现有系统同样的缺陷。将在逆向工程中收集的需求编写成文档，并让客户评审以确保其正确性。

9) 给出期望的解决办法 用户推荐的解决方法往往掩盖了用户的需求，导致业务处理的低效，或者给开发人员带来压力以至做出很差的设计方案。因此分析人员应尽力从客户叙说的解决方法中提炼出其本质核心。

5.2.2 需求分析

1) 划分需求优先级 划分出每项需求、特性或使用实例的优先级并安排在特定的产品版本或实现步骤中。评估每项新需求的优先级并与已有的工作主体相对比以做出相应的决策。

2) 带来技术困难的特性 分析每项需求的可行性以确定是否能按计划实现。成功好象总是悬于一线的，于是运用项目状态跟踪的办法管理那些落后于计划安排的需求，并尽早采取措施纠正。

3) 不熟悉的技术、方法、语言、工具或硬件平台 不要低估了学习曲线中表明的满足某项需求所需要的新技术的速度跟进情况。明确那些高风险的需求并留出一段充裕时间从错误中学习、实验及测试原型。

5.2.3 需求规格说明

1) 需求理解 开发人员和客户对需求的不同理解会带来彼此间的期望差异，将导致最终产品无法满足客户的要求。对需求文档进行正式评审的团队应包括开发人员，测试人员和客户。训练有素且颇有经验的需求分析人员能通过询问客户一些合适的问题，从而写出更好的规格说明。模型和原型能从不同角度说明需求，这样可使一些模糊的需求变得清晰。

2) 时间压力对TBD的影响 将SRS中需要将来进一步解决的需求注上TBD记号，但如果这

些TBD并未解决，则将给结构设计与项目的继续进行带来很大风险。因此应记录解决每项TBD的负责人的名字，如何解决的以及解决的截止日期。

3) 具有二义性的术语 建立一本术语和数据字典，用于定义所有的业务和技术词汇，以防止它被不同的读者理解为不同的意思。特别是要说明清楚那些既有普通含义又有专用领域含义的词语。对SRS的评审能够帮助参与者对关键术语、概念等达成一致的共识。

4) 需求说明中包括了设计 包含在SRS中的设计方法将对开发人员造成不必要的限制并妨碍他们发挥创造性设计出最佳的方案。仔细评审需求说明以确保它是在强调解决业务问题需要做什么，而不是在说怎么做。

5.2.4 需求验证

1) 未经验证的需求 审查相当篇幅的SRS是有些令人沮丧，正如要在开发过程早期编写测试用例一样。但如果在构造设计开始之前通过验证基于需求的测试计划和原型测试来验证需求的正确性及其质量，就能大大减少项目后期的返工现象。在项目计划中应为这些保证质量的活动预留时间并提供资源。从客户代表方获得参与需求评审的赞同（承诺），并尽早且以尽可能低的成本通过非正式的评审逐渐到正式评审来找出其存在的问题。

2) 审查的有效性 如果评审人员不懂得怎样正确地评审需求文档和怎样做到有效评审，那么很可能会遗留一些严重的问题。故要对参与需求文档评审的所有团队成员进行培训，请组织内部有经验的评审专家或外界的咨询顾问来讲课、授教以使评审工作更加有效。

5.2.5 需求管理

1) 变更需求 将项目视图与范围文档作为变更的参照可以减少项目范围的延伸。用户积极参与的具有良好合作精神的需求获取过程可把需求变更减少近一半（Jones 1996a）。能在早期发现需求错误的质量控制方法可以减少以后发生变更的可能。而为了减少需求变更的影响，将那些易于变更的需求用多种方案实现，并在设计时更要注重其可修改性。

2) 需求变更过程 需求变更的风险来源于未曾明确的变更过程或采用的变动机制无效或不按计划的过程来做出变更。应当在开发的各阶层都建立变更管理的纪律和氛围，当然这需要时间。需求变更过程包括对变更的影响评估，提供决策的变更控制委员会，以及支持确定重要起点步骤的工具。

3) 未实现的需求 需求跟踪能力矩阵有助于避免在设计、结构建立及测试期间遗漏的任何需求。也有助于确保不会因为交流不充分而导致多个开发人员都未实现某项需求。

4) 扩充项目范围 如果开始未很好定义需求，那么很可能隔段时间就要扩充项目的范围。产品中未说明白的地方将耗费比预料中更多的工作量，而且按最初需求所分配好的项目资源也可能不按实际更改后用户的需求而调整。为减少这些风险，要对阶段递增式的生存期制定计划，在早期版本中实现核心功能，并在以后的阶段中逐步增加实现需求。

5.3 风险管理是你的好助手

项目管理人员可以运用风险管理来提高对造成项目损失的条件的警惕，在需求获取阶段要有用户的积极参与。精明的管理者不仅能认识到它能带来风险的条件，而且将它编入风险清单，并依据以往项目的经验估计其可能性和影响。如果用户一直没有参与，风险危害值将

会扩大以至危害项目的成功。我曾说服管理人员把项目延期是由于缺少用户的积极参与，我告诉他们不能把公司的资金投入一项注定要失败的项目。

周期性的风险跟踪能使管理人员保持对风险危害变化的了解，对那些并未得到完全控制的风险能得到高层管理人员的注意。他们要么开始采取一些修正措施，要么不管风险，依旧按原业务决策思路进行。即使不能控制项目可能遇到的所有风险，风险管理也能使你看清形势，做出的决策是有所依据。

下一步：

- 明确你当前项目面临的一些与需求有关的风险，不要把当前的问题当作风险，一定要是那些还未发生的事情。将风险因素用条件——结果形式编写成文档，正如图5-2模板所示的那样。为每项风险推荐至少一种可能的降低风险的方法。
- 召集代表开发、市场、客户和管理各方面的风险承担者召开风险“集体研讨”会议。尽力找出更多与需求有关的风险因素。估计每项风险发生的可能性及其影响，两者乘积就是风险危害值。通过按风险危害值降序排列找到最高的五项风险。为每项风险安排一个负责人负责实施降低风险的活动。

第二部分 软件需求工程

第6章 建立项目视图与范围

我的同事 Karen 已经在她的公司里成功地引入软件需求文档的正式评审。她已经注意到在评审会议上所提出的许多问题都与项目所设定的范围有关。参与评审的专家经常难以理解项目所设定的范围，并且在项目的最终目标上所持的看法各不相同。因此，他们发现在哪—个功能需求应该列入软件需求规格说明的问题上很难达成一致的意见。

正如我们在第1章所叙述的那样，业务需求代表了需求链中最高层的抽象：他们为软件系统定义了项目视图（vision）和范围（scope）。软件功能需求必须根据用户的需求来考虑，且要与业务需求所设定的目标相一致。对不利于实现项目业务目标的需求应该排除在外。一个项目可能包括一些与软件没有直接关系的需求，例如：硬件的购买、产品的安装、维护或广告。但在此，我们只关心与软件产品有关系的业务需求。

如果一个项目缺乏明确的规划和良好的信息交流途径，那将是十分糟糕的。如果项目的参与者持有不同的目标和优先权，那么他们只能各抒己见，无心工作。如果项目的风险承担者在产品所能满足的业务需要和产品所能提供的利益问题上不能达成一致的意见，那么需求决不会稳定。一个清晰的项目视图和范围过于分散在多个地方开发，在这样的项目中，地理位置上的分离使项目开发组成员必须天天进行相互沟通才能保证他们之间能进行更有效的合作。

业务需求中某些特性最初被列入规格说明，而后又被删除，最后又加入，则说明此业务需求未完全定义好。在确定详细的功能需求之前，必须很好地解决项目的视图和范围问题。对范围和局限性的明确说明将在很大程度上有助于对所建议特性的探讨和最终产品的发行。一个明确定义了项目视图和范围的文档也可以为所建议的需求变更的决策提供参考。

6.1 通过业务需求确定项目视图

项目视图可以把项目参与者定位到一个共同和明确的方向上。项目视图描述了产品所涉及的各个方面和在一个完美环境中最终所具有的功能。相反的，范围描述了产品应包括的部分和不应包括的部分。范围的说明在包括与不包括之间划清了界线，当然，它还确定了项目的局限性。

项目的业务需求在视图上和范围上形成文档，这些必须在创建项目之前起草。开发商业软件的公司经常编写市场需求文档，其实这种文档也是为了类似的目的，但这种文档较为详细地涉及关于目标市场部分的内容，这是为适应商业的需要。视图和范围的文档为项目的主办者或具有同等地位的人所拥有。业务需求是从各个不同的人那里收集来的，这些人对于为什么要从事该项目和该项目最终能为业务和客户提供哪些价值有较清楚的了解。它们包括主办者(sponsor)、客户、开发公司的高级管理人员及项目的幻想者(visionary)，例如产品的代表和市场部门人员。

来自各个渠道的业务需求可能会发生冲突。比如，考虑具有嵌入软件的售货亭管理系统，它将卖给零售店并由零售客户使用。售货亭管理系统的开发者有如下的业务目标：

- 向零售商发行并销售售货亭产品。
- 通过售货亭软件向客户销售消费品。
- 吸引客户对商品的兴趣。
- 改变原有的开发者—客户的关系。

零售亭业务对如下方面感兴趣：

- 通过客户使用售货亭软件而获利。
- 吸引更多的客户来商店购买。
- 如果售货亭软件替代了人工操作，就可节省钱。

开发者可能要为客户建立高科技系统，并且引导客户紧跟新的发展方向。而零售商则需要一个简易、方便使用的系统，客户需要便利和良好的性能。这三者在目标、限制和费用因素上的不同将导致业务需求的冲突，这必须在售货亭管理系统的软件需求说明制订之前予以解决。

也可以利用业务需求对使用实例及与它们相关的功能需求设置实现优先级。例如，业务需求的确定可以从售货亭软件产生最大收益考虑，这意味着软件性能的最初实现是与销售更多的产品或对客户服务有直接关系，而不是去强调只吸引少量客户的软件性能。

业务需求不仅决定了应用程序所能实现的业务任务（使用实例）的设置（所谓的应用宽度），还决定了对使用实例所支持的等级和深度。支持的深度可以从一个很小的实现细节到具有许多辅助功能的完全自动化的操作。对于每个使用实例都必须决定其宽度和深度，并编写出文档。如果业务需求帮助你确定一个在应用范围之外特殊的使用实例，那么此时，你正在确定产品的应用宽度。业务需求还可以帮助你确定哪一个使用实例需要健壮的、综合的功能实现，哪一个仅需要一般实现，至少需要初始实现。

6.2 项目视图和范围的文档

项目视图和范围的文档（vision and scope document）把业务需求集中在一个简单、紧凑的文档里，这个文档为以后的开发工作奠定了基础。项目视图和范围文档包括了业务机遇的描述、项目的视图和目标、产品适用范围和局限性的陈述、客户的特点、项目优先级别和项目成功因素的描述。这必须是一个相对简短的文档，也许只有3~8页纸，这取决于项目的性质和大小。

- | | |
|--------------|-------------|
| a. 业务需求 | c. 范围和局限性 |
| a.1 背景 | c.1 首次发行的范围 |
| a.2 业务机遇 | c.2 随后发行的范围 |
| a.3 业务目标 | c.3 局限性和专用性 |
| a.4 客户或市场需求 | d. 业务环境 |
| a.5 提供给客户的价值 | d.1 客户概貌 |
| a.6 业务风险 | d.2 项目优先级 |
| b. 项目视图的解决方案 | e. 产品成功的因素 |
| b.1 项目视图陈述 | |
| b.2 主要特性 | |
| b.3 假设和依赖环境 | |

图6-1 项目视图和范围文档的模板

图6-1描述了一个对项目视图和范围文档的推荐模板，文档模板对公司项目所创建的文档结构进行了标准化。就像其它模板一样，必须对图 6-1所示的模板图进行改写来满足你项目的需求。

下面介绍这个模板的每一个部分。

a. 业务需求

业务需求说明了提供给客户和产品开发商的新系统的最初利益。不同的产品，例如信息管理系统，商业软件包，系统捆绑软件将有不同的侧重点。然而，项目开发的投入是由于人们坚信：有了新产品，世界将变得更加美好。本部分描述了你为什么要从事此项项目的开发，以及它将给开发者和购买者带来的利益。

a.1 背景

在这一部分，总结新产品的理论基础，并提供关于产品开发的历史背景或形势的一般性描述。

a.2 业务机遇

描述现存的市场机遇或正在解决的业务问题。描述商品竞争的市场和信息系统将运用的环境。包括对现存产品的一个简要的相对评价和解决方案，并指出所建议的产品为什么具有吸引力和它们所能带来的竞争优势。认识到目前只能使用该产品才能解决的一些问题，并描述产品是怎样顺应市场趋势和战略目标的。

a.3 业务目标

用一个定量和可测量的合理方法总结产品所带来的商业利润。关于给客户带来的价值在本模板 a.5 的项目视图和范围文档中阐述，这里仅把重点放在给业务的价值上。这些目标与收入预算或节省开支有关，并影响到投资分析和最终产品的交付日期。如果这些信息在其它地方已叙述，就请参考有关文档，在此就不再重复了。

a.4 客户或市场需求

描述一些典型客户的需求，包括不满足现有市场上的产品或信息系统的需求。提出客户目前所遇到的问题在新产品中将可能（或不可能）出现的阐述，提供客户怎样使用产品的例子。确定了产品所能运行的软、硬件平台。定义了较高层次的关键接口或性能要求，但避免设计或实现细节。把这些要求写在列表中，可以反过来跟踪调查特殊用户和功能需求。

a.5 提供给客户的价值

确定产品给客户带来的价值，并指明产品怎样满足客户的需要。可以用下列言辞表达产品带给客户的价值：

- 提高生产效率，减少返工。
- 节省开支。
- 业务过程的流水线化。
- 先前人工劳动的自动化。
- 符合相关标准和规则。
- 与目前的应用产品相比较，提高了可用性或减少了失效程度。

a.6 业务风险

总结开发（或不开发）该产品有关的主要业务风险，例如市场竞争、时间问题、用户的接受能力、实现的问题或对业务可能带来的消极影响。预测风险的严重性，指明你所能采取

的减轻风险的措施。

b. 项目视图的解决方案

文档中的这一部分为系统建立了一个长远的项目视图，它将指明业务目标。这一项目视图为在软件开发生存期中作出决策提供了相关环境背景。这部分不应包括详细的功能需求和项目计划信息。

b.1 项目视图陈述

编写一个总结长远目标和有关开发新产品目的的简要项目视图陈述。项目视图陈述将考虑权衡有不同需求客户的看法。它可能有点理想化，但必须以现有的或所期待的客户市场、企业框架、组织的战略方向和资源局限性为基础。

这里是曾在前面章节讨论过的“化学制品跟踪系统”的简单项目视图陈述的一个实例。

“化学制品跟踪系统”可使科学家查询到化学制品仓库或供应商将提供的化学制品容器。系统可随时了解公司中每一个化学制品容器所处的位置，容器中所剩余的药品剂量，任何时候每个容器所处的位置和用法的历史记录。通过充分利用公司内部的可用化学制品，废弃极少量已使用或过期失效的化学制品，使用标准的化学制品的购买过程等将在化学制品上节省25%开支。“化学制品跟踪系统”还能产生符合政府部门规定所要求的全部报表，包括化学制品的使用、存储和废弃等报表。

b.2 主要特性

包括新产品将提供的主要特性和用户性能的列表。强调的是区别于以往产品和竞争产品的特性。可以从用户需求和功能需求中得到这些特性。

b.3 假设和依赖环境

在构思项目和编写项目视图和范围文档时，要记录所作出的任何假设。通常一方所持的假设应与另一方不同。如果你把它们都记录下来，并加以评论，就能对项目内部隐含的基本假设达成共识。比如，“化学制品跟踪系统”的开发者假设：该系统可以替代现有的仓库存货系统，并能与有关采购部门的应用相连接。把这些都记录下来以防止将来可能的混淆和冲突。还有，记录项目所依赖的主要环境，比如：所使用的特殊的技术、第三方供应商、开发伙伴或其它业务关系。

c. 范围和局限性

当一个化学家发明了可以把一种化学制品转变为另一种化学制品的新的化学变化时，它所发表的论文中包含了“范围和局限性”部分，这一部分描述了这一化学变化所能作和不能作的一种限定。类似地，一个软件项目也必须定义它的范围和局限性，并作为业务需求的一部分。

项目范围定义了所提出的解决方案的概念和适用领域，而局限性则指出产品所不包括的某些性能。澄清范围和局限性这两个概念有助于建立各风险承担者所企盼的目标。有时客户所要求的性能太奢华或者与产品所制定的范围不一致。一般客户所提出的需求超出项目的范围时就应当拒绝它，除非这些需求是很有益的。这时，可适当扩大项目范围来适应这些需求（在预算、计划、人员方面也要相应进行变化）。记录这些需求以及拒绝它们的原因，以备日后重新遇到时，有记录可查。

c.1 首次发行的范围

总结首次发行的产品所具有的性能。描述了产品的质量特性，这些特性使产品可以为不

同的客户群 (customer community) 提供预期的成果。如果你的目标集中在开发成果和维持一个可行的项目规划上，应当避免一种倾向，那就是把一些潜在的客户所能想到的每一特性都包括到 1.0 版本的产品中。这一倾向所带来的普遍恶果是产生软件规划的动荡性和错误性。开发者应把重点放在能提供最大价值、花费最合理的开发费用及普及率最高的产品上。

例如，我的同事 Scott 的上一个项目开发小组决定，用户可以用首发版的软件进行包裹传递业务。1.0 版本并不要求快速、结构紧凑或易于使用，但该软件必须稳定运行；整个开发小组始终以这一目标为准。首发版的软件完成了基本的系统目标，而随后的版本则包含了附加的特性、选项和使用帮助。

c.2 随后发行的范围

如果你想象一个周期性的产品演变过程，就要指明哪一个主要特性的开发将被延期，并期待随后版本发行的日期。

c.3 局限性和专用性

明确定义包括和不包括的特性和功能的界线是处理范围设定和客户期望的一个途径。列出风险承担者们期望的而你却不打算把它包括到产品中的特性和功能。

d. 业务环境

这一部分总结了一些项目的业务问题，包括主要的客户分类概述和项目的管理优先级。

d.1 客户概貌

客户概述明确了这一产品的不同类型客户的一些本质的特点，以及目标市场部门和在这些部门中的不同客户的特征。对于每一种客户类型，概述要包括以下信息：

- 各种客户类型将从产品中获得的主要益处。
- 它们对产品所持的态度。
- 感兴趣的关键产品的特性。
- 哪一类型客户能成功使用。
- 必须适应任何客户的限制。

d.2 项目的优先级

一旦明确建立项目的优先级，风险承担者和项目的参与者就能把精力集中在一系列共同的目标上。达到这一目的的一个途径是考虑软件项目的五个方面：性能、质量、计划、成本和人员 (Wiegers 1996a)。在所给的项目中，其每一方面应与下面三个因素之一相适应。

- 一个驱动 (driver) ——一个最高级别的目标。
- 一个约束 (constraint) ——项目管理者必须操纵一个对象的限制因素。
- 一个自由度 (degree of freedom) ——项目管理者能权衡其它方面，进而在约束限制的范围内完成目标的一个因素。

未必所有的因素都能成为驱动，或所有的因素都能成为约束因素。在项目开始时记录和分析哪一个因素适用于哪一类型，将有助于使每一个人的努力和期望与普遍认可的优先级相一致。

e. 产品成功的因素

明确产品的成功是如何定义和测量的，并指明对产品的成功有巨大影响的几个因素。不仅要包括组织直接控制的范围内的事务，还要包括外部因素。如果可能，可建立测量的标准，用于评价是否达到业务目标，这些标准的实例有：市场股票、销售量或收入、客户满意程度

的测量、交易处理量和准确度。

6.3 关联图

软件项目范围的描述为我们正在开发的系统和宇宙万物之间划清了界线。关联图 (Context diagram)通过正在开发的系统或正在讨论的问题和外部世界之间的联系来描述这一界线。关联图确定了通过某一接口与系统相连的外部实体 (称为“端点”或“外部实体”)，同时也确定了外部实体和系统之间的数据流和物流。我们把关联图作为按照结构化分析所形成的数据流图的最高抽象层 (Robertson and Robertson 1994)。可以把关联图写入项目视图和范围文档或软件需求规格说明中，或者作为系统数据流模型的一部分。

“化学制品跟踪系统”的简单的关联图如图 6-2 所示。整个系统被描述成一个简单的循环，所以关联图并不明确提供系统的内部过程和数据。关联图中的流可以用信息(“化学制品请求”)或物理项(“化学制品容器”)来表示。以矩形图示的端点可以表示用户类(“药剂师”)、组织(“采购部门”)或其它计算机系统(“培训用数据库”)。

你可能希望把化学制品的供应商作为一个端点放入关联图中。别忘了，公司总是发购买化学制品的订单给化学制品供应商，并从供应商那里得到装有化学制品的容器和发票，而供应商则得到支票。然而，这些过程发生在“化学制品跟踪系统”范围之外，并作为购买和进货部门日常事务的一部分。关联图明确告诉我们，系统并不直接与供应商订货、进货、或付账。

虽然某些端点与所规划的系统没有直接联系，但有时关联图将给出与项目的问题域有关的端点之间的联系 (Jackson 1995)。不是教条地去追求如何绘制“正确”的关联图，而是使用这样的图来确定项目风险承担者之间清晰而精确的关系。

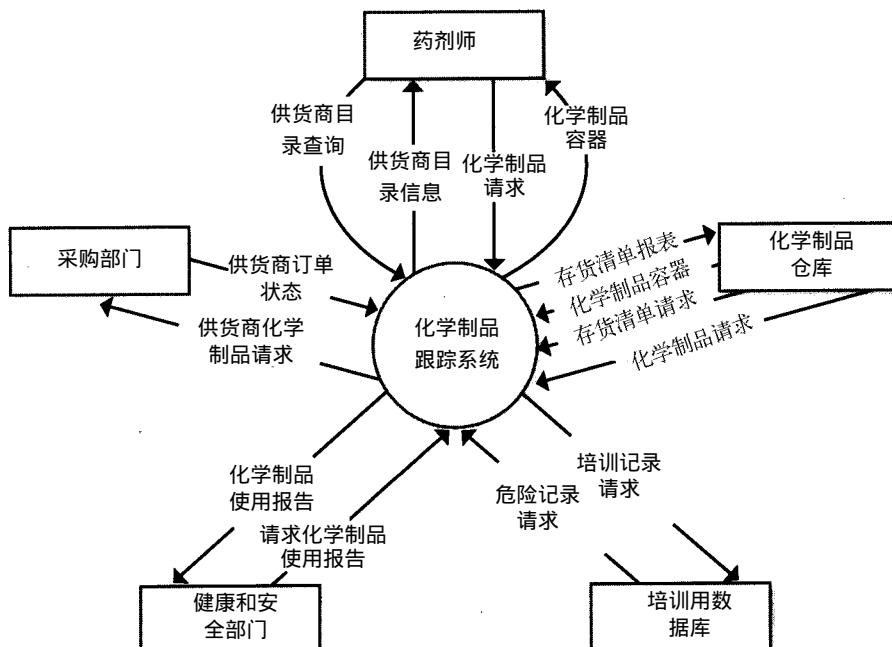


图6-2 “化学制品跟踪系统”关联图

6.4 把注意力始终集中在项目的范围上

在项目视图和范围文档中记录业务需求为防止开发过程范围的扩展（creep）提供了有利的手段。项目视图和范围文档可以使你判断所提出的特性和需求放进项目是否合适。当某些人提出新的需求或改变需求或特性时，你必须问的第一个问题是：“这是否包含在项目范围内？”

所提出的一些建议有时完全在项目范围之外。它们可能是一个好的方案，但这个方案适用于其它项目或将来要发行的产品。另外一些建议很明显是在项目范围定义之内。如果这种建议与已经为某一确定产品所制定的需求相比，具有更高的优先级，则这些新的并符合要求的需求就可以加入到项目中。不过此时你必须作出权衡（trade off），决定是延迟还是取消其它已经确定的需求或特性。

第三种可能性是“所提出的新需求在项目范围之外，但这是一个很有价值的方案，此时可以改变项目的范围来适应这一需求。但要求你要修改项目视图和范围文档。当改变项目的范围时，你必须重新商议计划预算、资源及进度安排，也许还有开发人员（特别是在需要新的技术和技巧时）。理想的情况是原先的安排和资源能合理地适应需求变更。然而，你必须在需求变更得到赞同后才重新进行计划安排，除非你原先对需求的预算留有余地。

范围扩展存在固有的两个主要问题：1) 全部的工作必须重新进行以适应变化。2) 当项目的范围增大时，如果没有调整原先所分配的资源和时间，则属性会遭到破坏。一组确定的业务需求可以使项目依照计划正常进行，在市场或业务需要变更时可以合理地调整项目范围的大小；当一些有影响的人企图往有许多约束的项目中添加更多的特性时，可以合理地拒绝这些要求。

下一步：

- 无论你是正要开发一项项目或正在开发一项项目，请用图 6-1 的模板编写一份项目视图和范围文档。如果你的开发组对项目范围各抒己见，则这一任务可能比较困难。现在开始就要解决这一问题，而不是让它模棱两可；如果你对这个问题置之不理，则这个问题就会越来越难解决。这一活动也可通过修改模板来更好地满足你公司组织项目的需要。

第7章 寻找客户的需求

如果你赞成客户的参与是发布一个优秀软件的关键因素，在项目的开始阶段就会努力致力于为你的项目征求各个客户的意见。软件需求的成功，和软件开发的成功都取决于开发者是否尽可能地采纳客户的意见。在第6章讨论了一种类型的客户——项目主持者(sponsor)、项目的幻想者(visionary)或市场部门——他们提供了项目的商业需求。本章将集中介绍需求的第二级——用户需求。为了征求客户的意见，必须采取以下几步：

- 明确项目用户需求的来源。
- 明确使用该产品的不同类型的用户。
- 与产品不同用户类的代表进行沟通。
- 遵从项目的最终决策者的意见。

客户参与是避免期望差异(expectation gap)的唯一途径，这一期望差异表现在客户期望得到的产品与开发者所设计的产品之间不相符。然而，在项目的开始阶段仅仅简单地问一两个客户的需求，然后就开始编码，这样做是不够的。如果开发者仅仅为了客户的最初需求去开发软件，那么，他们可能要重新进行开发，因为，客户常常不知道他们的真正需要，而开发者也不知道。

用户提出“需要”的特性并不总是与用户利用新产品来处理他们的任务(task)时所需的功能相等价。因此，当你收集到用户的意見后，必须分析、整理这些需求意见，直到你理解它为止，并把你的理解写成文档，然后与用户一起探讨，这是一个反复的过程，并且需要花费时间。如果你不在这一方面花时间，对预期产品一致的看法未达成共识——最终的后果可能是返工，并且产品不尽人意。

7.1 需求的来源

软件需求可以来自方方面面，这取决于所开发产品的性质和开发环境。需从不同用户代表和来源收集需求，这说明了需求工程是以相互交流为核心的性质。下面是几个软件需求的典型来源。

1. 访问并与有潜力的用户探讨

为找出新软件产品的用户需求，最直截了当的方法是询问他们。本章讨论如何寻找合适的用户代表，而在第8章讲述从这些代表中获取需求的技巧。

2. 把对目前的或竞争产品的描述写成文档

文档可以描述一种所必须遵循的标准或产品所必须遵循的政府或工业规则。

3. 系统需求规格说明

一个包含软、硬件的产品需要一个高档次的系统需求规格说明以介绍整个产品。系统需求的子集被分配到每个软件子系统中(Nelsen 1990)。附加的详细软件功能需求将从有关软件的系统需求里获得。

4. 对当前系统的问题报告和增强要求

指导用户和提供技术支持的工作人员是最有价值的需求来源。他们收集了用户在使用现有系统过程中所遇到问题的信息，还接受了用户关于系统改进的想法。

5. 市场调查和用户问卷调查

调查有助于从广大有潜力的用户那里获得大量定量的数据，务必调查相关的用户并询问一些能产生反响的好问题。

6. 观察正在工作的用户

对当前系统的用户和将来系统的有潜力的用户，分析员观察“日常工作”以获得经验，这些经验能提供很有价值的信息。分析员可通过观察用户与所关联的任务环境的工作流程来预见用户在使用当前系统时所遇到的问题，并能分析新的系统可有效支持工作流程的方面（McGraw and Harbison 1997; Beyer and Holtzblatt 1998）。比起仅仅简单地询问用户，并记下用户在处理任务时的步骤来说，直接观察用户的工作流程可以对他们的活动有更正确的理解。分析员必须抽象和总结用户的直接活动，以确保所获得的需求具有普遍性，而不仅仅代表单个用户。一个富有技巧的分析员还可以为改进用户的当前事务处理过程提出一些见解。

7. 用户任务的内容分析

通常通过开发具体的情节（scenario）或活动顺序（有时称作“情节”），可以确定用户利用系统需要完成的任务，分析员由此可以获得用户用于处理任务的必要的功能需求。这是使用实例方法的精髓（参看第8章）。

7.2 用户类

产品的用户在很多方面存在着差异，例如：用户使用的频度、他们的应用领域和计算机系统知识、他们所使用的产品特性、他们所进行的业务过程、他们在地理上的布局以及他们的访问优先级。根据这些差异，你可以把这些不同的用户分成小组。比起其他用户，其中某些用户类的需求可能对你更为重要（Gause and Lawrence 1999）。

每一个用户类都将有自己的一系列功能和非功能要求。比如：一个没有经验或偶尔使用电脑的用户关心系统是否简单易用，因此，菜单、提示符和向导是很重要的。然而，对于那些一天使用几小时产品的用户，他们更关心产品的易用性和高效性，所以他们喜欢使用快捷键、宏以及脚本功能(scripting facility)。

有一些受产品影响的人并不一定是产品的直接使用者，而是通过报表或其它应用程序访问你产品的数据和服务。这些非直接的或次级(secondary)的用户也有他们的需求，于是他们组成了附加的用户类。就像我的同事Kathy所说的：“只要一日是你的用户，则终身都是你的用户。”

用户类不一定都指人，你可以把其它应用程序或系统接口所用的硬件组件也看成是附加用户类的成员。以这种方式来看待应用程序接口，可以帮助你确定产品中那些与外部应用程序或组件有关的需求。

在项目中，要尽早为产品确定并描述出不同的用户类，这样，就能从每一个重要的用户类代表中获取不同的需求。我听说过一个给65个团体用户开发一个专门的业务产品的公司，当他们意识到可以把用户分成6个截然不同的用户类时，这些用户对未来发行的产品的需求就被简化了。在软件需求规格说明中，把这些用户类和他们的特征编写成文档。现举一例，在前面所讨论的“化学制品跟踪系统”中，项目的管理者所确定的用户类和他们的特征如表7-1

所示。

表7-1 “化学制品跟踪系统”的用户类

药剂师	药剂师将使用系统请求来自供应商和仓库的化学制品。药剂师每天多次使用系统，主要用于跟踪进出实验室的化学制品容器。药剂师需要在供应商目录中查找指定化学制品
采购者	采购者在采购部门处理其他用户所提交的化学制品请求，他们与外部的供应商建立联系，制定并发出订单。采购者对化学制品几乎不了解，因此将需要简单的查询机制来查找供应商目录。采购者不使用系统中容器跟踪这一特性。每个采购者平均每天使用系统 10 次
化学制品仓库人员	化学制品仓库人员包括三个技师，管理着多达 500 000 种化学制品容器。他们将处理来自药剂师的请求并提供可用的容器，向供应商请求新的化学制品以及跟踪进出仓库的所有容器的流向，他们是货存清单和化学制品使用报告特性的唯一使用者。由于交易量大，化学制品仓库人员所使用的系统功能必须是自动化并且高效
卫生和安全人员	卫生和安全人员使用系统是为了生成符合官方关于化学制品使用和处理规则的季度报表。这些报表必须提前定义，并不需要特别查询能力，当官方的规则改变时，卫生和安全管理人员可能每年多次要求变化报表中的内容。报表变更优先级最高

7.3 寻找用户代表

每种类型项目——包括企业信息系统、商业应用程序、集成系统、软件嵌入式产品、Web 开发程序和签约合同的软件——在获取 (elicitation) 用户需求时需要挑选合适的用户代表来反映各类用户的需求。用户代表必须参加整个软件开发生命周期，而不仅仅是只参加开始的需求阶段。虽然你必须把重点放在最重要的用户代表身上，但是你还是需要广泛的用户参与者来代表不同的用户类和不同的专业层次。

当公司开发应用程序时，最容易接近真正的用户。然而，如果你正在开发商业软件，可能要在开发过程的早期阶段，从目前的 beta 测试版或先前版本产品的使用者中收集需求意见。建立现存的长期客户关系或组成一个核心用户组，它是由目前产品的用户组或竞争产品的用户组组成的。如果建立起一个用户组，必须明确，这些参与者是否真正代表了各个方面的用户，而这些用户的需求是否可以促进产品的开发。核心用户组必须包含各种用户类型，不仅包括知识渊博的用户，还应包括缺乏经验的用户。如果核心用户组仅代表早期的需求提供者或空想家，那么将会遇到一系列复杂和技术上困难的需求，这些需求与目标市场没有太大的关系。

图7-1描述了用户的需求和开发者之间的一些典型的信息关联。当开发者能直接与有关的用户对话时，就产生了最直接的交流；非直接联系一般是无效的 (Kiel and carmel 1995)。就象孩子们“打电话”的游戏一样，在用户和开发者之间插入一些中间层，会增加他们之间交流信息时产生错误的可能性。例如：如果开发者只向最终用户的管理者获取意见，那么这些需求就不太可能正确反映用户的需求。

你必须确信，插入的这些层次是具有价值的，就像一个有经验的需求分析员可以与用户或其他参与者一起工作，为开发者收集、评价并组织整理需求信息。必须确信你明白你所承担的风险，这个风险就是使用市场人员或其他人员作为用户真正需求的代理人，你还要判断这些风险是否值得承担。虽然要获得最优的用户代表是困难的，且可能要花很多钱，然而，

如果你不与能提供最佳信息的人交流，那么你的产品和用户将蒙受损失。

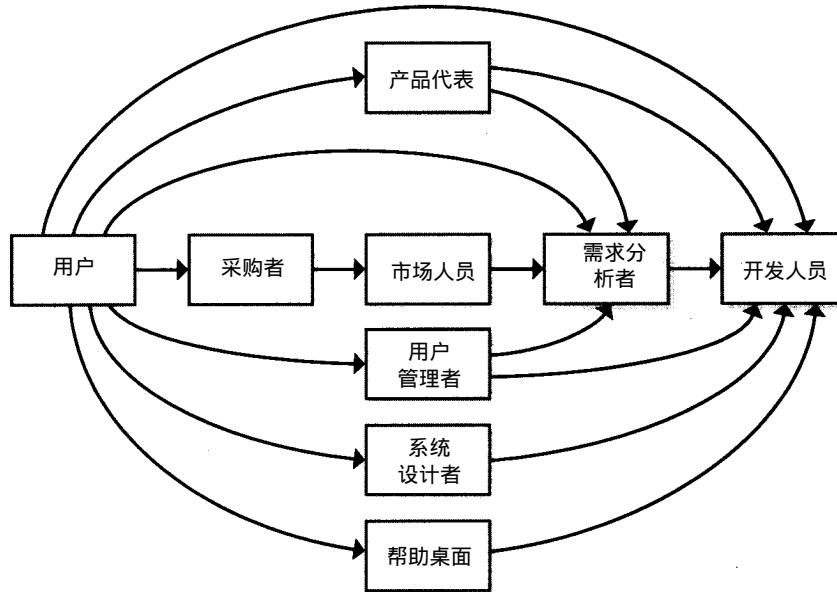


图7-1 用户和开发者之间的可能通讯路径

7.4 产品的代表者

许多年前，我曾经是一个软件开发小组的成员，这个开发小组支持着一个大公司的科学的研究活动。当我们组建开发组时，决定我们的每一个工程项目都包括为数不多的核心参与者，这些参与者来自相关的用户团体，并提供客户的需求。我们称这些人为产品的代表者（project champion）或项目的代表者，（Wedgers 1996a）。通过产品代表者这一途径，可以提供一个有效的方法来使客户——开发者之间的伙伴关系结构化和形式化。

每一个产品代表者代表了一个特定的用户类，并在那个用户类和开发者之间充当主要的接口。产品代表者必须是真正的用户，而并不是用户的代理人，如主办者、产品客户、市场人员或者软件组成员充当用户。产品代表者从他们所代表的用户类中收集需求信息。每个产品代表者都负责协调他们所代表的用户在需求表达上的不一致性和不兼容性。目的就是由每个产品代表者与分析员合作，为那个用户类整理出统一的需求意见。虽然分析员通常起草需求文档，但需求的收集与整理是分析员和一些核心客户的共同责任。

一个优秀的产品代表者对新系统有明确的认识并有极大的热情，因为他们知道新产品将使他们以及他们的伙伴受益。产品代表者必须是有力的交流者，且是同组成员的代表者。他们必须对应用领域有彻底的了解，并在软件方面具有足够的经验，他们知道在当前技术背景下，什么是可行的；在运行环境中，什么是可实现的。通常，许多产品代表者在工作中又去承担其它的任务，因此，你必须有令人信服的观点，阐明一些特殊个人的参与对项目的成功具有重要意义。

如果产品代表者有足够的权利为他们所代表的用户作出共同的决定，那么他们将会很好地发挥作用。如果产品代表者的决策总是受到管理者或软件开发小组的否决，那么他们所花的时间和心思(goodwill)就白白浪费了。然而，产品代表者必须牢记，他们不是唯一的用户。我曾经见过产品代表者工作失误，因为充当核心联络角色的产品代理者没有与他们的同行进

行充分的交流，而仅仅代表了他们自己的需求和对产品的概念。

7.4.1 寻求产品代表者

如果你正在开发业务软件而不是内部软件，就很难在你的公司之外寻找可以充当产品代表者的人。但是，如果你与一些大公司的用户有着紧密的工作伙伴关系，那么他们会很乐意或（要求）参与用户需求获取。此时，你将面临着一个挑战：如何避免片面地听取某些产品代表者的需求而忽视了其他代表者的需求。如果有一个广泛的客户基础，那么你就有可能先确定代表所有客户的核心需求，而后确定对于特定个体客户和用户类的附加需求。

只有在产品代表者通过商业展示会或其它专业上的交流相互联系之后，他们才可能与其它公司的产品代表者相互交流。不过，这样存在着风险，因为这种讨论交流可能泄漏内部业务过程的细节，潜在地影响了每个公司的竞争力。如果一个知道本公司未来产品计划的产品代表者把这一内部信息告诉那些不应该知道这一信息的人，应该怎么办？双方签署不泄密协定有助于解决这一问题，但这种协定对私人秘密没有保证。另一种可能性是产品代表者所在的公司可能决定不购买早期版本的产品，因为他们知道更好的版本即将发行。此时，可能需要给你的外部产品代表者一些经济上的鼓励以获得他们的贡献，例如提供产品打折或者聘请他们与你一起进行用户需求的讨论工作。

另一种方法是真正聘请一个具有丰富阅历的合适的产品代表者。一个为某一特殊产业开发零售销售点（point-of-sale POS）和后台办公系统（back-office）的公司曾经聘请了三位零售店的经理来充当全天候的产品代表者的角色。还有一个例子，我的长期家庭医生 Art，放弃了她的行医生涯，现在在一个医学软件公司工作，作为一个产品代理者为公司提供医生对产品的需求。Art的老板清楚地认识到，聘请一个医生来帮助他们开发软件是值得的，因为这个软件最终将被其他医生欣然接受。另一个公司从大多数客户中聘请了许多以前的雇员，这些雇员可以提供有价值的关于本部门的专门知识，还可以提供对客户组织中策略的认识。

7.4.2 产品代表者的期望

把产品代表者的期望编写成文档，这样有助于通过产品代表者这一途径获得成功。当然，并不是每一个产品代表者都能提供你所喜爱的服务，此时，你可用书写期望建立实例的方法，让特定的用户来填写这一关键的任务，并作为探讨每个产品代表者确定责任的起点。表 7-2 确定了产品代表者的活动。并不是在每种情况下都要有这些活动，但是在不同的项目中，产品代表者执行其中不同的活动。

表7-2 产品代表者的可能活动

分 类	活 动
计划 (planning)	<ul style="list-style-type: none"> • 转化产品的适用范围和局限性 • 定义与其它系统的外部接口 • 定义从目前用户应用程序过渡到新系统的过渡途径
需求 (requirement)	<ul style="list-style-type: none"> • 访问其它用户，收集他们的需求描述 • 提出使用脚本和使用实例 • 解决所建议的需求之间的冲突 • 定义实现优先级 • 确定质量属性和其它非功能需求 • 评估用户接口原型

(续)

分 类	活 动
验证和确认 (verification and validation)	<ul style="list-style-type: none"> • 审查需求文档 • 确定用户接受标准 • 根据使用脚本编写测试用例 • 进行beta测试 • 提供测试数据集
用户帮助 (user aid)	<ul style="list-style-type: none"> • 编写用户手册和在线帮助 • 准备教学用的培训材料 • 为同行者提供产品演示
变更管理 (change management)	<ul style="list-style-type: none"> • 对缺陷的改正进行评估并设置其优先级 • 对增强 (enhancement) 的请求进行评估和设置其优先级 • 评估需求变更给用户带来的影响 • 参加变更控制委员会并参与变更决策

7.4.3 多个产品代表者

“化学制品跟踪系统”有四个用户类，因此在 Contoso Pharmaceutical 需要从内部用户群中选择多个产品代表者。图 7-2 描述了项目总经理如何建立产品代表者小组以从各个渠道收集有效的需求。这些产品代表者并不是全职，但他们每个星期都要花数个小时在项目研究上。三个分析员与四个产品代表者一起协作进行需求获取、分析，并把它们的需求编写成文档（采购人员和卫生与安全用户类很小，他们的需求也不多）。然后由一名分析员综合这些信息，并编写到软件需求规格说明中。

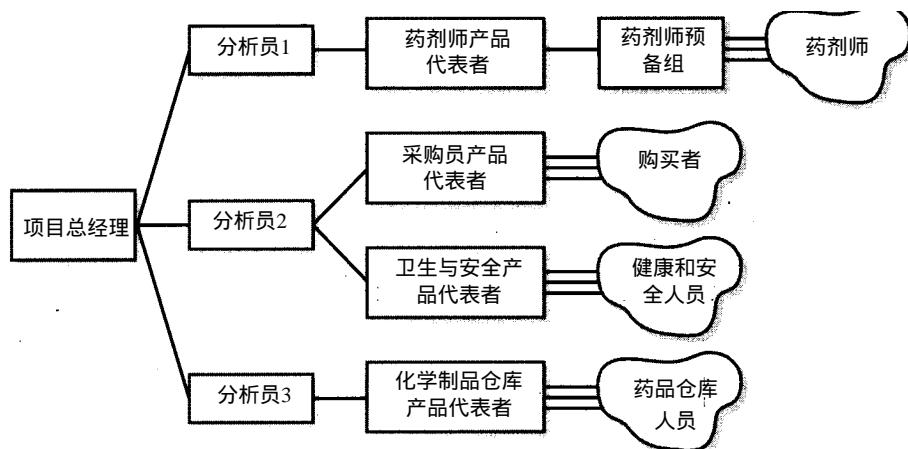


图 7-2 “化学制品跟踪系统”中的产品代表者模型

指望由一个人来提供一个大型用户类的所有需求是不现实的，这样的用户类正如在 Contoso Pharmaceutical 中由几百名药剂师组成。因此，代表药剂师用户类的产品代表者组织了一个由五个药剂师组成的预备组，这五个药剂师分别来自公司的不同部门。五个药剂师从他们各自部门的同行中收集信息，分析他们对“化学制品跟踪系统”所提出的问题，并为他们的同行提供项目状况的修改。这种分层方法增加了获取需求的用户数，但避免了大量的开销，这些开销主要用于整理和协调大量收集需求的专题讨论会或者长时间连续个人访问。化

学药剂师产品代表者总是力求达成一致意见，但是当他们意见不一致时，他们总是乐于作出必要的决策，因此，项目总能向前推进，而不是陷入僵局。

7.5 谁作出决策

我曾经在卫星发射中心见到一位在大型开发公司工作的项目经理。当我问他为谁工作时，他给我四个名字：当地的开发经理、总部的开发经理、还有两个来自其他客户群的业务单位经理。这个项目经理感到十分沮丧，因为他总是收到令人迷惑的信息，总是被四个领导的不一致决策甚至相反的决策所误导。

当需求从四面八方收集来后，人们难以解决冲突，澄清模糊之处以及协调不一致之处。某些人还要对不可避免要发生的范围问题单独作出决定。在每个项目的早期阶段，你必须决定谁是需求问题的决策者。如果不清楚谁有权并且有责任来作出决策，或者授权的个人不愿意或不能作出决策，那么决策者的角色将自然而然地落在开发者身上。这是一个非常糟糕的选择，因为开发者通常没有足够多的信息和观点来作出业务上的决策。

在软件项目中，谁将对需求作出决策的问题并没有统一的正确答案。分析员有时听从呼声高的或来自最高层人物的最大的需求。即便使用产品代表者这一手段，必须解决来自不同用户类的相冲突的需求。通常，应尽可能由处于公司底层的人作出决策，因为他们与问题密切相关，并能得到关于这些问题的广泛信息。我倾向在大家一致决策之上参与决策。达成一致意见固然是理想的，但在风险承担者对每个问题作出决策的同时，你不能停止项目的进展。

下面是一些在项目中可能发生的决策问题，并带有建议性的解决方案。项目中的领导者需要决定：当这些问题发生时，谁可以决策该如何做，并且在不能达成一致意见时，由谁来协调这一冲突。

- 如果个别用户不能在需求方面达成一致的意见，那么必须由产品代表者作出决策。产品代表者这种方法的实质是授权给产品代表者，由其解决他们所代表的需求冲突问题。
- 如果不同的用户类有不一致的需求，那么必须决策出满足哪一类用户的需求更为重要。了解可能使用产品的客户种类的信息和他们的用法与产品的业务目标的关系如何，将有助于你决定哪一个用户类所占份额最大。
- 不同公司的客户可能都要求产品按照他们各自的喜好来设计。还是老办法，运用项目的业务目标来决定那些是你最关心的客户。一个核心的客户有助于主要特性的开发，然而，其它你认为获利较少的客户需求可能要拖延到下一个版本中去。
- 有时，客户经理所提出的需求与他所在部门的真正用户提出的需求相冲突。用户需求必须与业务需求一致，因此，那些没有亲自使用过产品的经理必须服从代表他们用户的的产品代表者提出的详细的用户需求和功能性规格说明。应避免强迫开发人员对客户不同意的需求作出公断，因为这是一个十分失策的观点。
- 当开发者想象中的产品与客户需求冲突时，通常应该由客户作出决策。然而，不要陷到“客户总是对的”的陷阱中去，对他们百依百顺。现实中，客户并不总是对的。客户总是持有自己的观点，开发者必须理解并尊重这一观点。
- 如果市场部门所提出的需求与开发者所想要开发的系统发生冲突时，类似的情况就发生了。作为客户的代理人，市场需求具有更重的分量。但是，我常见的情况是：市场总是迁就客户需求，而不管这些需求的合理性和费用如何。我还见过其它的情况：市场提供

极少的信息，而必须由开发者去定义产品并且由他们自己编写需求文档。

没有简单的正确答案。在遇到这些问题之前，你决定谁将对项目需求作出决策，以免由于优柔寡断和对先前决策的回溯导致你的项目停工。

下一步：

- 在你所处的环境中，把听取客户需求的方法与图 7-1联系起来。在你当前的交流联系中是否遇到一些问题？确定最短的并且最有效的交流途径，在将来可根据这一途径收集用户需求。
- 为项目确定不同的用户类型并决定谁将成为每个用户类的产品代表者。利用表 7-2作为一个起点来定义你希望产品代表者所具备的功能。与产品代表者和他们的经理商讨，有助于你的项目开发。
- 决定谁是项目中需求问题和冲突解决方案的决策者。你目前的决策者业绩如何？失误在什么地方？这些人适合作决策者吗？如果回答是否定的，那么在开发者必须亲自解决这些问题之前，决定谁将参与需求决策工作，并提出协调建议。

第8章 聆听客户的需求

需求获取(requirement elicitation)是需求工程的主体。对于所建议的软件产品，获取需求是一个确定和理解不同用户类的需要和限制的过程。获取用户需求位于软件需求三层结构的中间一层。来自项目视图和范围文档的业务需求决定用户需求，它描述了用户利用系统需要完成的任务。从这些任务中，分析者能获得用于描述系统活动的特定的软件功能需求，这些系统活动有助于用户执行他们的任务。本章将介绍需求获取原理，并把重点放在“使用实例”方法的应用，这种方法是用于获取产品的用户需求。

需求获取是在问题及其最终解决方案之间架设桥梁的第一步。获取需求的一个必不可少的结果是对项目中描述的客户需求的普遍理解。一旦理解了需求，分析者、开发者和客户就能探索出描述这些需求的多种解决方案。参与需求获取者只有在他们理解了问题之后才能开始设计系统，否则，对需求定义的任何改进，设计上都必须大量的返工。把需求获取集中在用户任务上——而不是集中在用户接口上——有助于防止开发组由于草率处理设计问题而造成的失误。

需求获取、分析、编写需求规格说明和验证并不遵循线性的顺序，这些活动是相互隔开、增量和反复的。当你和客户合作时，你就将会问一些问题，并且取得他们所提供的信息（需求获取）。同时，你将处理这些信息以理解它们，并把它们分成不同的类别，还要把客户需求同可能的软件需求相联系（分析）。然后，你可以使客户信息结构化，并编写成文档和示意图（说明）。下一步，就可以让客户代表评审文档并纠正存在的错误（验证）。这四个过程贯穿着需求开发的整个阶段。

由于软件开发项目和组织文化的不同，对于需求开发没有一个简单的、公式化的途径。下面列出了14个步骤，你可以利用它们指导你的需求开发活动。对于需求的任何子集，一旦你完成了第十三步，那么你就可以很有信心地继续进行系统的每一部分的设计、构造，因为你将开发出一个好的产品。

8.1 需求获取的指导方针

需求获取可能是软件开发中最困难、最关键、最易出错及最需要交流的方面。需求获取只有通过有效的客户——开发者的合作才能成功。分析者必须建立一个对问题进行彻底探讨的环境，而这些问题与产品有关。为了方便清晰地进行交流，就要列出重要的小组，而不是假想所有的参与者都持有相同的看法。对需求问题的全面考察需要一种技术，利用这种技术不但考虑了问题的功能需求方面，还可讨论项目的非功能需求。确定用户已经理解：对于某些功能的讨论并不意味着即将在产品中实现它。对于想到的需求必须集中处理并设定优先级，以避免一个不能带来任何益处的无限大的项目。

需求获取是一个需要高度合作的活动，而并不是客户所说的需求的简单誊本。作为一个分析者，你必须透过客户所提出的表面需求理解他们的真正需求。询问一个可扩充（open-ended）的问题有助于你更好地理解用户目前的业务过程并且知道新系统如何帮助或改进他们

的工作。调查用户任务可能遇到的变更，或者用户需要使用系统其它可能的方式。想像你自己在学习用户的工作，你需要完成什么任务？你有什么问题？从这一角度来指导需求的开发和利用。

还有，探讨例外的情况：什么会妨碍用户顺利完成任务？对系统错误情况的反映，用户是如何想的？询问问题时，以“还有什么能……”，“当……时，将会发生什么”“你有没有曾经想过……”，“有没有人曾经……”为开头。记下每一个需求的来源，这样向下跟踪直到发现特定的客户。

建议需求开发过程

1. 定义项目的视图和范围
2. 确定用户类
3. 在每个用户类中确定适当的代表
4. 确定需求决策者和他们的决策过程
5. 选择你所用的需求获取技术
6. 运用需求获取技术对作为系统一部分的使用实例进行开发并设置优先级
7. 从用户那里收集质量属性的信息和其它非功能需求
8. 详细拟订使用实例使其融合到必要的功能需求中
9. 评审使用实例的描述和功能需求
10. 如果有必要，就要开发分析模型用以澄清需求获取的参与者对需求的理解
11. 开发并评估用户界面原型以助想像还未理解的需求
12. 从使用实例中开发出概念测试用例
13. 用测试用例来论证使用实例、功能需求、分析模型和原型
14. 在继续进行设计和构造系统每一部分之前，重复 6~13步

尽量把客户所持的假设解释清楚，特别是那些发生冲突的部分。从字里行间去理解以明确客户没有表达清楚但又想加入的特性或特征。Gause 和Weinberg (1989) 提出使用“上下文无关问题”——这是一个高层次的问题，它可以获取业务问题和可能的解决方案的全部信息。客户对这些问题的回答诸如“产品要求怎样的精确度”或“你能帮我解释一下你为什么不同意某人的回答吗？”这些回答可以更直接地认识问题，而这是封闭（close-end）问题所不能做到的。

需求获取利用了所有可用的信息来源，这些信息描述了问题域或在软件解决方案中合理的特性。第7章曾列出了软件需求的来源。一个研究表明：比起不成功的项目，一个成功的项目在开发者和客户之间采用了更多的交流方式（Kiel and Carmel 1995）。与单个客户或潜在的用户组一起座谈，对于业务软件包或信息管理系统（MIS）的应用来说是一种传统的需求来源。（如何与用户座谈方面的指导，见Beyer and Holtzblatt [1998]，Wood and Silver[1995]，McGraw and Harbison[1997]）。直接聘请用户进行获取需求的过程是为项目获得支持和买入（buy-in）的一种方式。

在每一次座谈讨论之后，记下所讨论的条目(item)，并请参与讨论的用户评论并更正。及早并经常进行座谈讨论是需求获取成功的一个关键途径，因为只有提供需求的人才能确定是否真正获取需求。进行深入收集和分析以消除任何冲突或不一致性。

尽量理解用户用于表述他们需求的思维过程。充分研究用户执行任务时作出决策的过程，并提取出潜在的逻辑关系。流程图和决策树是描述这些逻辑决策途径的好方法。

当进行需求获取时，应避免受不成熟的细节的影响。在对切合的客户任务取得共识之前，用户能很容易地在一个报表或对话框中列出每一项的精确设计。如果这些细节都作为需求记录下来，他们会给随后的设计过程带来不必要的限制。你可能要周期性地检查需求获取，以确保用户参与者将注意力集中在与今天所讨论的话题适合的抽象层上。向他们保证在开发过程中，将会详尽阐述他们的需求。

在一个逐次详细描述过程中，重复地详述需求，以确定用户目标和任务，并作为使用实例。然后，把任务描述成功能需求，这些功能需求可以使用户完成其任务，也可以把它们描述成非功能需求，这些非功能需求描述了系统的限制和用户对质量的期望。虽然最初的屏幕构思有助于描述你对需求的理解，但是你必须细化用户界面设计。

8.2 基于使用实例的方法

多年来，分析者总是利用情节或经历来描述用户和软件系统的交互方式，从而获取需求 (McGraw and Harbison 1997)。最近，Ivar Jacobson (1992) 和其他人把这种看法系统地阐述成用使用实例(use-case)的方法进行需求获取和建模。虽然使用实例来源于面向对象的开发环境，但是它也能应用在具有许多开发方法的项目中，因为用户并不关心你是怎样开发你的软件。一些实际方法包括使用实例模型概念 (Regnell, Kimbler and Wesslen 1995; Booth, Rumbaugh and Jacobson 1999)。然而，使用实例的观点和思维过程带给需求开发的改变比起是否画正式的使用实例图显得更为重要。注意用户要利用系统做什么远远强于询问用户希望系统为他们做什么这一传统方法。

一个使用实例描述了系统和一个外部“执行者”(actor)的交互顺序，这体现执行者完成一项任务并给某人带来益处。执行者是指一个人，或另一个软件应用，或一个硬件，或其它一些与系统交互以实现某些目标的实体 (Cockburn 1997a, b)。执行者可以映像到一个或多个可以操作的用户类的角色。例如，“化学制品跟踪系统”中的“请求一种化学制品”使用实例包括一个名为“请求者”的执行者。在“化学制品跟踪系统”中设有一个客户类叫请求者，但是药剂师和化学制品仓库的人员均可充当这一角色。

使用实例为表达用户需求提供了一种方法，而这一方法必须与系统的业务需求相一致。分析者和用户必须检查每一个使用实例，在把它们纳入需求之前决定其是否在项目所定义的范围内。基于“使用实例”方法进行需求获取的目的在于：描述用户需要使用系统完成的所有任务。在理论上，使用实例的结果集将包括所有合理的系统功能。在现实中，你不可能获得完全包容，但是比起我所知道的其它获取方法，基于使用实例的方法可以为你带来更好的效果。

8.2.1 使用实例和用法说明

一个单一的使用实例可能包括完成某项任务的许多逻辑相关任务和交互顺序。因此，一个使用实例是相关的用法说明的集合，并且一个说明 (scenario) 是使用实例的例子。在使用实例中，一个说明被视为事件的普通过程 (normal course)，也叫作主过程、基本过程，普通流，或“满意之路”(happy path)。在描述普通过程时列出执行者和系统之间相互交互或对话的顺序。当这种对话结束时，执行者也达到了预期的目的。“请求一种化学制品”的使用实例的普通过程导致一个用户请求，要求从外界供应商订购化学制品。

图8-1描述了“请求化学制品”的使用实例图，并运用了统一建模语言 (UML) 概念

(Booth, Rumbaugh and Jacobson 1999^②)。执行者(请求者)用一个简单的分叉状图形表示，并在执行者和他所需要的每个使用实例(以椭圆形所示)之间画一条线。在这幅图中，主要的使用实例是“请求一种化学制品”。其它的使用实例：“查看仓库中可用的化学制品容器”和“输入货物编号”，描述了两种可能的使用实例中《extend》和《include》的关系，这将在这一章的后面部分介绍。使用实例图提供了一个高级别的用户需求的可视化表示。

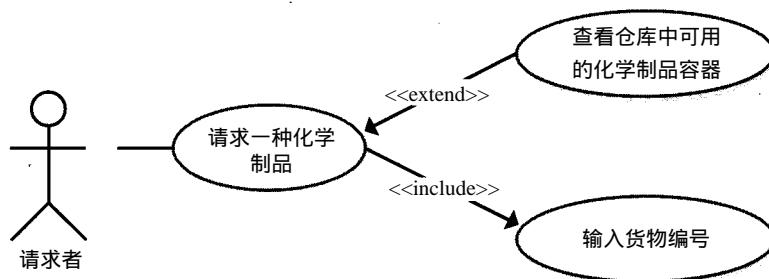


图8-1 来自“化学制品跟踪系统”的“请求一种化学制品”使用实例

在使用实例中的其它说明可以描述为可选过程(alternative coruse)。可选过程也可促进成功地完成任务，但它们代表了任务的细节或用于完成任务的途径的变化部分。在对话序列中，普通过程可以在一些决策点上分解成可选过程，然后再重新汇成一个普通过程。在“请求一种化学制品”的使用实例中，“向化学制品仓库请求一种化学制品”是一个可选过程。虽然用户在普通过程和可选过程中请求一种化学制品。但是用户和系统之间的详细交互在许多方面存在差异。在可选过程中，用户仍必须确定合适的化学制品，但也许可以从化学制品仓库里现存的多种容器中选择。

可选过程中的一些步骤与在普通过程中的步骤相同，但是需要一些唯一的动作来完成可选路径(见图8-2)。有时，通过在流中插入一个定义可选过程的分离的使用实例可以很方便地扩充(extend)普通过程。被扩充的使用实例必须是一个完整的使用实例，它可以独立地运行。图8-1表示：使用实例“查看仓库中可用的化学制品容器”扩充了“请求一种化学制品”使用实例。这一扩充形成了“从化学制品仓库中请求一种化学制品”的可选过程。

许多使用实例可能共享一些公共函数。为了避免重复，你可以定义一个独立的使用实例，这一实例包含这个公共函数，并指定其它使用实例必须包括这个公共使用实例。这与在编程语言中的所谓“公共子过程”在逻辑上是等价的。被包括的使用实例对于任务的完成是必不可少的，但一个扩充其它实例的使用实例则是可选的，因为普通过程可以完全替代它(Rumbaugh 1994)。举一个例子：在图8-1中，“请求一种化学制品”实例是那些包含一个称为“输入货物编号”独立实例的许多使用实例中的一种。

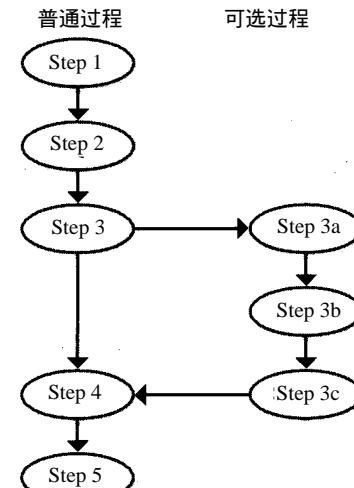


图8-2 使用实例中普通过程和可选过程的对话流

^② 发展中的软件设计语言的规则或术语，如UML在不断改变，因此这里所用的表示方法在将来可能会过时。本章所介绍的使用实例的概念比起用于表示它们的形式更为重要。

引起任务不能顺序完成的情况称为例外 (exception)，在某些时候它可以视为可选过程。

在定义使用实例时，描述例外路径是很重要的，因为它们描述了在特定条件下用户对系统如何工作的看法。“请求一种化学制品”使用实例中的一个例外是不存在业务上可用的化学制品。如果你没有将例外记录在文档上，那么开发者可能在设计和构造阶段忽视这些可能性。此时，当系统遇到一个例外条件时，就会发生系统崩溃。

8.2.2 确定使用实例并编写使用实例文档

可以使用多种方法来确定使用实例 (Ham1998; Larman 1998)：

- 首先明确执行者和他们的角色，然后确定业务过程，在这一过程中每一个参与者都在为确定使用实例而努力。
- 确定系统所能反映的外部事件，然后把这些事件与参与的执行者和特定的使用实例联系起来。
- 以特定的说明形式表达业务过程或日常行为，从这些说明中获得使用实例，并确定参与到使用实例中的执行者，
- 有可能从现在的功能需求说明中获得使用实例。如果有些需求与使用实例不一致，就应考虑是否真的需要它们。

由于使用实例代表了用户的需求，在你的系统中，应该直接从不同用户类的代表或至少应从代理那里收集需求。在化学制品跟踪系统项目中，分析者召集一系列 2到3小时的使用实例收集工作专题研习会 (Workshop)，会议隔天举行一次。在客户和开发者的交流方面，召集与组织小组是一种最有效的技术之一 (kiel and Carmel 1995)。每一个研习会的参与者包括代表特定用户类的产品代表者，其它选出的用户代表，还有一个或多个开发者。当用户提出不可行的需求时，开发者总是慎重考虑这些需求的实现问题，参与需求获取的工作有助于使他们能及早认识所要开发的产品。“化学制品跟踪系统”中的不同用户类成员参加到不同的研习会，这一切顺利进展是由于对于多个客户类只有少数几个使用实例是通用的。

在举行研习会内的讨论之前，每一分析者总是要求用户考虑他们使用新系统需要完成的任务或业务过程。每一个任务成为一个候选的使用实例，分析者用任务的简明说明对这些使用实例进行编号和命名，例如“请求一种化学制品”或“为化学制品打印安全数据单”。当小组在研习会中探讨使用实例时，他们发现，一些实例代表了相关的说明，这些说明可以合并到一个简单的抽象使用实例中。抛弃一些建议的使用实例，因为它们超出了项目的范围。小组成员还发现了他们所定义的最初集合以外的附加使用实例。

用户总是趋向于首先确定最重要的使用实例，所以可以根据发现的顺序来确定优先级。另一种确定优先级的方法是当提出候选的使用实例时，给每个实例写 2到3句说明。确定候选的使用实例的优先级，首先为最高优先级的实例填写细节部分，然后为余下的候选使用实例重新评估其优先级。

需求获取的每一次讨论会都可以探索出多个使用实例，并根据标准化模板把它们编写成文档。图8-3描述了简化的“请求一种化学制品”使用实例的模板。首先，参与者将第一个从使用实例中获益的人定为操作者；接下来，他们定义任何满足使用实例运行的先决条件及在完成使用实例后描述系统的后续条件。预先估计的使用频度为并行使用和性能需求提供了一个早期指示。下一步，分析者询问参与者想怎样与系统交互来完成任务。操作者动作的最终

对话顺序和系统响应构成一个流，这个流可视为事件的普通过程。虽然每个参与者对真正的用户接口和交互机制有不同的想法，但是在执行者——系统对话中，开发组可以在这些步骤上达成共识。

分析者总是在注释中获取单个执行者动作和系统响应，这些注释位于每一个使用实例的流程单上。另一种指导工作的方法是在计算机中生成一个使用实例模板，并在讨论过程中完善模板内容。由于对话的顺序包含了复杂的逻辑或符合式决策，用诸如流程图等图形表达比用一系列标明数字的步骤表达更有启发性。

使用实例标识号	CU——5										
使用实例名称	请求一种化学制品										
创建者	Tim	最后一次更新者	Janice								
创建时间	10/4	最后一次更新时间	10/27								
执行者	请求者										
说明	求者通过输入化学制品的 ID或输入化学制品的结构来指定对化学制品的请求。系统则提供给请求者一个来自化学制品仓库的新的或已用过的化学制品容器，或者让请求者向外界供货商订货										
先决条件	1. 用户的身份被证实 2. 具有在线的化学制品存货清单数据库										
请求结果	1. 将完成的请求存入“化学制品跟踪系统” 2. 通过电子邮件把请求发往化学制品仓库或完成采购										
优先级	高										
使用频度	对于每个药剂师大约每周 5 次，对于化学制品仓库的每个成员每周 100 次										
普通过程	5.0 从供应商那里请求一种化学制品 <table> <tr> <td style="text-align: center;"><u>执行者行为</u></td> <td style="text-align: center;"><u>系统响应</u></td> </tr> <tr> <td>1. 输入化学制品的 ID 或者包含化学制品结构的文件名</td> <td>2. 验证化学制品的 ID 是否合法</td> </tr> <tr> <td></td> <td>3. 询问请求者需要一个新的供应商订单或一个来自化学制品仓库的容器</td> </tr> <tr> <td>4. 确定供应商（待续）或化学制品仓库（可选过程 5.1）</td> <td>5.<继续对话，直到请求完成></td> </tr> </table>			<u>执行者行为</u>	<u>系统响应</u>	1. 输入化学制品的 ID 或者包含化学制品结构的文件名	2. 验证化学制品的 ID 是否合法		3. 询问请求者需要一个新的供应商订单或一个来自化学制品仓库的容器	4. 确定供应商（待续）或化学制品仓库（可选过程 5.1）	5.<继续对话，直到请求完成>
<u>执行者行为</u>	<u>系统响应</u>										
1. 输入化学制品的 ID 或者包含化学制品结构的文件名	2. 验证化学制品的 ID 是否合法										
	3. 询问请求者需要一个新的供应商订单或一个来自化学制品仓库的容器										
4. 确定供应商（待续）或化学制品仓库（可选过程 5.1）	5.<继续对话，直到请求完成>										
可选过程	5.1 从化学制品仓库中请求一种化学制品（5.0.4之后的分枝） <table> <tr> <td style="text-align: center;"><u>执行者行为</u></td> <td style="text-align: center;"><u>系统响应</u></td> </tr> <tr> <td>2. 可选择地查看任何容器的历史</td> <td>1. 显示出化学制品仓库中现存的所要求的化学容器的列表</td> </tr> <tr> <td>3. 选择一个特定的容器或发出供应商订单</td> <td></td> </tr> </table>			<u>执行者行为</u>	<u>系统响应</u>	2. 可选择地查看任何容器的历史	1. 显示出化学制品仓库中现存的所要求的化学容器的列表	3. 选择一个特定的容器或发出供应商订单			
<u>执行者行为</u>	<u>系统响应</u>										
2. 可选择地查看任何容器的历史	1. 显示出化学制品仓库中现存的所要求的化学容器的列表										
3. 选择一个特定的容器或发出供应商订单											
例外	5.E.1 化学制品在业务上不可用 <table> <tr> <td style="text-align: center;"><u>执行者行为</u></td> <td style="text-align: center;"><u>系统响应</u></td> </tr> <tr> <td>3. 请求另一种化学制品</td> <td> 1. 显示消息：不存在供应商 2. 询问请求者是否要请求另一种化学制品或退出 4. 普通常程结束 </td> </tr> </table>			<u>执行者行为</u>	<u>系统响应</u>	3. 请求另一种化学制品	1. 显示消息：不存在供应商 2. 询问请求者是否要请求另一种化学制品或退出 4. 普通常程结束				
<u>执行者行为</u>	<u>系统响应</u>										
3. 请求另一种化学制品	1. 显示消息：不存在供应商 2. 询问请求者是否要请求另一种化学制品或退出 4. 普通常程结束										
包括	UC-12 输入货物编号										
特定需求	系统必须可以按标准的编码形式输入化学制品的结构，这一标准来自对不同化学制品图形包的支持										
假设	输入的化学制品结构被认为是合法的										
注释和问题	Tim将查明：在危险列表的第一级上请求一种化学制品是否需要管理层的批准。预期时间1/4										

图8-3 “化学制品跟踪系统”中“请求一种化学制品”使用实例的部分描述

需求获取小组为可选的过程确定出相似的对话并确定例外情况。当分析者问一些诸如“那时，倘若数据库没有联机，将会发生什么情况？”或“当化学制品在业务上不可用时该怎么办？”这类问题时，将会发现许多例外条件。在每个使用实例完全研究清楚后，并且没有提出附加的变化、例外或细节时，研习会的参与者就可以转入到另一个使用实例中去。他们并不尽力在马拉松式的会议中挖掘所有的使用实例。相反，他们计划渐增地挖掘使用实例，然后就不断评价和提炼他们，详尽阐述功能需求的细节和可能的用户接口。

图8-4展示了与使用实例获取研习会及后继活动相关的事件的顺序。在每一个研习会之后，分析者得到使用实例的说明并开始从说明中获取功能需求。这其中有一些是显而易见的，例如“使用一个唯一的、系统赋予的序列号来标识每一个已提交的请求”。其它的则比较微妙，代表了开发者将要开发出的功能，这些功能可以使用户在与系统的交互过程中执行那些步骤。分析者以分级结构化形式编写这些功能需求文档，这种结构化形式适合于写入软件需求规格说明中。

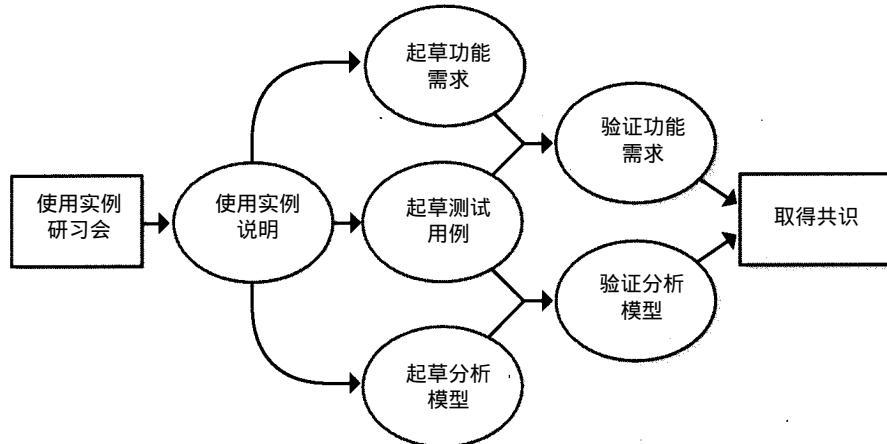


图8-4 使用实例获取方法

分析者发现对于一些复杂的使用实例，画出图形分析模型是有益的，这些模型包括数据流程图、实体关系图、状态转化图、对象类和联系图（interaction diagram），还有可能的用户接口机制的对话映像。这些模型描述了需求的不同观点，这样可以经常发现在文档中不易被发现的遗漏、模糊和不一致性的地方。第10章将为“化学制品跟踪系统”图示多种这样的分析模型。

一天之中，在每一个专题研习会之后，分析者给出使用实例的说明和与研习会参与者的有关的功能需求，这些参与者在把他们的工作成果送至下一个研习会之前都要检查这些功能需求。这些经常性的、不正式的检查可以发现许多错误，例如先前没有发现的可选过程和例外、不正确的功能需求及在执行者—系统（actor-system）对话中遗失的步骤。使用实例方法通过这样渐增式的检查为提高需求质量提供了有力手段。

对于主持高效的获取使用实例的研习会的频繁程度有一个限制。那些每天都主持研习会的分析者发现：参与者难于在他们所检查的文档中发现错误，因为在他们的思维中这些信息太生疏了。从紧张的思维行为中脱离出来，让头脑休息一两天，可以使你从一个新的角度来看待早先的工作。如果你要从检查文档中得益，那么一周最好只主持二或三个研习会。

早在需求开发阶段，测试者就可以从使用实例中为“化学制品跟踪系统”开发出概念性测试用例。这些测试用例有助于开发组编写清晰的文档，并且在针对特定的使用说明、系统如何运作方面享有共同的见解。测试用例可以使分析者验证他们是否获得了在测试用例中规定的用于产生系统行为的所有功能需求。他们也可以用测试用例来判断分析方法是否完善、正确，并且与功能需求是否一致。第14章更详细地讨论了从需求中生成测试用例。

大系统可以有成百上千个使用实例，因此需要花费相当的时间来确定、详述、编写文档并审核他们。然而，通过使用实例研习会来定义系统预期的功能，对于建立一个高质量软件系统，并且可以满足不同用户需要方面是一个极好的投资(investment)。

8.2.3 使用实例和功能需求

使用实例的描述并不向开发者提供他们所要开发的功能的细节。如果你在用户需求阶段停止了需求开发，你将会发现在软件的构造阶段，开发者必须询问许多问题来弥补他们的信息空白。为了减少这种不确定性，你需要把每一个使用实例叙述成详细的功能需求(Arlow 1998)。

每一个使用实例可引伸出多个功能需求，这将使执行者可以执行相关的任务；并且多个使用实例可能需要相同的功能需求。例如，如果有5个使用实例需要进行用户身份的验证，你不必因此编写5个不同的代码块。可以用多种方法来编写与一个使用实例相联系的功能需求文档。采用何种方法取决于你是否希望开发组从使用实例文档、软件需求规模说明，或二者相结合来设计、构造并测试。这些方法都不完善，因此选择那些最适合你编写文档和管理项目的软件需求方法。避免在不同地方产生信息冗余，因为冗余将使需求管理变得更加困难。

1. 仅利用使用实例的方法

虽然你仍可能需要用一个独立的软件需求规格说明来记录与特定的使用实例无关的需求，但是一种可能的方法是把功能需求包括在每一个使用实例的说明之中。你必须交叉引用在多个使用实例中重复的那些功能需求。一种方法是可以通过先前讨论的“包括”的关系来阐述，在这一关系中一些公共功能(如用户身份验证)被分割到一个独立的，可重用的使用实例中。

2. 利用使用实例和SRS相结合的方法

另一种方法是把使用实例说明限制在抽象的用户需求级上，并且把从使用实例中获得的功能需求编入软件需求规格说明中。在这种方法中，你将需要在使用实例和与之相关的功能需求之间建立可跟踪性。最好的方法是把所有使用实例和功能需求存入数据库或者业务需求管理工具中，这将允许你定义这些跟踪性联系(见第19章)。

3. 仅利用软件需求规格说明的方法

第三种方法是通过使用实例来组织软件需求规格说明，并且包括使用实例和功能需求的说明。采用这种方法，你无需独立编写详细的使用实例文档；然而，你需要确定冗余的功能需求，或者对每个功能需求仅陈述一次，并且无论需求是否重复出现在其它使用实例中，都要参考它的原始说明。

8.2.4 使用实例的益处

使用实例方法给需求获取带来的好处来自于该方法是以任务为中心和以用户为中心的观点。比起使用以功能为中心的方法，使用实例方法可以使用户更清楚地认识到新系统允许他

们做什么。在许多 Web 开发工程中，用户代表发现，使用实例的方法真正有助于他们弄清 Web 站点的访问者可以做什么。使用实例有助于分析者和开发者理解用户的业务和应用领域。认真思考执行者——系统对话的顺序，使其可以在开发过程早期发现模糊性，也有助于从使用实例中生成测试用例。

如果开发者所编写的代码从未被使用过，这将是令人沮丧的。有了使用实例，所得到的功能需求明确规定了用户执行的特定任务。使用实例技术防止了“孤立”的功能——这些功能在需求获取阶段似乎是一个好的见解，但没有人使用它们，因为它们并没有与用户任务真正地联系在一起。

在技术方面，使用实例的方法也带来了好处。使用实例观点揭示了域对象以及它们之间的责任。开发者运用面向对象的设计方法可以把使用实例转化为对象模型。进而，当业务过程随时间而变时，内嵌在特定的使用实例中的任务也会相应改变。如果你跟踪功能需求、设计、编码和测试以至到它们父类的使用实例——用户意见——那么这就很容易看出整个系统中业务过程的级联变化。

8.2.5 避免使用实例陷阱

在使用实例的方法中应注意如下的陷阱：

- 太多的使用实例 如果你发现自己陷入使用实例的爆炸之中，你就可能不能在一个合适的抽象级上为之编写文档。不要为每一个可能的说明编写单独的使用实例。而是把普通过程、可选过程以及例外集成起来写入一个简单的使用实例。也不要将交互顺序中的每个步骤看成一个单独的使用实例。每一个使用实例都必须描述一个单独的任务。你将获得比业务需求多得多的使用实例，但你的功能需求将比使用实例还多。每一个使用实例都描述了一个方法，用户可以利用这个方法与系统进行交互，从而达到特定的目标。
- 使用实例的冗余（duplication across） 如果相同的函数出现在多个使用实例中，那么就有可能多次重写函数的实现部分。为了避免重复，可以使用“包括”关系，在这一关系中，将公共函数分离出来并写到一个单独的使用实例中，当其它使用实例需要该函数时，可以请求调用它。
- 使用实例中的用户界面设计 使用实例应该把重点放在用户使用系统做什么，而不是关心屏幕上是怎么显示的。强调在执行者和系统之间概念性的对话的理解，而把用户界面的细节推迟到设计阶段。任何屏幕上的画面和用户界面机制的影响只是使执行者——系统交互的可视化，而并不作为主要的设计说明。
- 使用实例中包括数据定义 我见过一些包括数据项和数据结构定义的使用实例，可以在该使用实例中操纵这些数据项和数据结构，包括数据类型、长度、格式和合法值。这个方法使项目的参与者难于找到他们所需要的定义，因为使用实例中说明它所包含的每一条数据定义是不明显的。这也可能导致冗余的定义，当一个使用实例改变时，其它的没有改变，但破坏了同步性。应该把数据定义集中在适用于整个项目范围的数据字典中（第 9 章讨论），以便在使用这些数据时减少不一致性。
- 试图把每一个需求与一个使用实例相联系 使用实例可有效地捕捉大多数所期望的系统行为，但是你可能有一些需求，这些需求与用户任务或其他执行者之间的交互没有特定的关系。这时你就需要一个独立的需求规格说明，在这个规格说明中可以编写非功能需

求文档、外部接口需求文档以及一些不能由使用实例得到的功能需求支持。

8.3 对客户输入进行分类

不要指望你的客户会给需求分析者提供一个简洁、完整、组织良好的需求清单。分析者必须把代表客户需求的许多信息分成不同的类型，这样他们就能合理地编写信息文档并把它们用于最合理的方式上。图 8-5 描述了许多这样的类型。那些不属于这些类型的信息可能代表一个非软件项目的需求，例如，为使用新系统而进行的用户培训或者它仅仅是不重要的信息。下面讨论在听取客户需求过程中的一些建议，这将有助于对信息进行分类处理。



图8-5 用户需求分类

1) 业务需求 描述客户或开发公司可以从产品中得到的资金、市场或其它业务利润的需求就是最可能的业务需求。再听听直接或间接的软件用户得到好处，例如“市场股票价格上升x%”，“每年节约\$Y”，或“替代了维护费用高的老一代系统Z”。

2) 使用实例或说明 有关利用系统执行的业务任务或达到用户目标的总的陈述可能就是使用实例，而特定的任务描述表示了用法说明。与客户一起商讨，把特定的任务概括成更广泛的使用实例。可以通过让客户描述他们的业务工作流活动来获取使用实例。

3) 业务规则 当一个客户说，一些活动只能在特定的条件下，由一些特定的人来完成时，该用户可能在描述一个业务规则，例如，“如果一个药剂师在危险化学制品培训方面是可靠的，那么他就可以在一级危险药品清单上订购化学制品”。业务规则是有关业务过程的操作原则。你可以用一些软件功能需求来加强规则，例如，让“化学制品跟踪系统”可以访问培训记录数据库。正如上面所说的那样，这里业务规则不是功能需求。

4) 功能需求 客户所说的诸如“用户应该能<执行某些功能>”或者“系统应该<具备某些行为>”，这是最可能的功能需求。功能需求描述了系统所展示的可观察的行为，并且大多数是处于执行者——系统响应顺序的环境中。功能需求定义了系统应该做什么，它们组成了软件需求规格说明的一部分。分析者应该明确，每个人应理解系统为什么“必须”执行某一功能。所提出的功能需求有时反映了过时的或无效的业务过程，而这些过程不能加入到新系统中。

5) 质量属性 对系统如何能很好地执行某些行为或让用户采取某一措施的陈述就是质量属性，这是一种非功能需求。听取那些描述合理特性的意见：快捷、简易、直觉性、用户友

好、健壮性、可靠性、安全性和高效性。你将要和用户一起商讨精确定义他们模糊的和主观言辞的真正含义，这将在第 11 章描述。

6) 外部接口需求 这类需求描述了系统与外部的联系。软件需求规格说明必须包括用户接口和通信机制、硬件和其它软件系统需求部分。客户描述外部接口需求包括如下习惯用语：

- “从<某些设备>读取信号”
- “给<一些其它系统>发送消息”
- “以<某种格式>读取文件”
- “能控制<一些硬件>”

7) 限制 限制是指一些合理限制设计者和程序员选择的条件。它们代表了另一种类型的非功能需求，你必须把这些需求写入软件需求规格说明。尽量防止客户施加不必要的限制，因为这将妨碍提出一个好的解决方案。不必要的限制将会降低利用现有商业化软件集成解决方案的能力。一定的限制有助于提高产品质量属性。只利用程序语言的标准命令而不允许使用供应商的扩展，可以提高可移植性。下面是客户描述限制的一些习惯用语：

- “必须使用<一个特定的数据库产品或语言>”
- “不能申请多于<一定数量的内存>”
- “操作必须与<其它系统>相同”
- “必须与<其它应用程序>一致”

8) 数据定义 当客户描述一个数据项或一个复杂的业务数据结构的格式、允许值或缺省值时，他们正在进行数据定义。例如，“邮政编码由 5 个数字组成，后跟一个可选的短划线或一个可选的四位数字，缺省为 0000”就是一个数据定义。把这些集中在一个数据字典中，作为项目的参与者在整个项目的开发和维护中的主要参考文档。

9) 解决思想 如果一个客户描述了用户与系统交互的特定方法，以使系统产生一系列活动（例如：用户从下载清单中选择一个所需要的项），这时你正在听取建议性的解决方案，而不是需求。所建议的解决方案使获取需求小组成员在潜在的真正需求上分散精力。在获取需求时，应该把重点放在需要作什么而不是新系统应该如何设计和构造。探讨客户为什么提出一个特定的实现方法，因为这可以帮助你理解真正的需求和用户对如何构造系统的隐含的期望。

8.4 需求获取中的注意事项

在需求获取的过程中，你可能会发现对产品范围的定义存在误差，不是太大就是太小（Christer and Kang 1992）。如果范围太大，你将要收集比真正需要更多的需求，以传递足够的业务和客户的值，此时获取过程将会拖延。如果项目范围太小，那么客户将会提出很重要的但又在当前产品范围之外的需求。当前的范围太小，以致不能提供一个令人满意的产品。需求的获取将导致修改项目的范围和任务，但作出这样具有深远影响的改变，一定要小心谨慎。

正如经常所说的，需求主要是关于系统做什么，而解决方案如何实现是属于设计的范围。这样说虽然很简洁，但似乎过于简单化。需求的获取应该把重点放在“做什么”上，但在分析和设计之间还是存在一定的距离。你可以使用假设“怎么做”来分类并改善你对用户需求的理解。在需求的获取过程中，分析模型、屏幕图形和原型可以使概念表达得更加清楚，然后提供一个寻找错误和遗漏的办法。把你需求开发阶段所形成的模型和屏幕效果看成是方便高效交流的概念性建议，而不应该看成是对设计者选择的一种限制。

需求获取研习会中如果参与者过多，就会减慢进度。我的同事 Debbie 在她从事过 Web 开发项目中，由于使用实例研习会进展缓慢，曾一度感到灰心丧气。12位参与者对不必要的细节进行激烈的讨论，并且在每个使用实例如何工作的问题上难以达成一致意见。当她把工作人员减少到只有 6 个人时，进度马上加快了，而这 6 个人代表了客户、系统设计者、开发者和可视化设计者等主要工程角色。

相反地，从极少的代表那里收集信息或者只听到呼声最高、最有舆论影响的用户的声音，也会造成问题。这将导致忽视特定用户类的重要的需求，或者其需求不能代表绝大多数用户的需要。最好的权衡在于选择一些授权为他们的用户类发言的产品代表者，他们也被同组用户类的其它代表所支持。

8.5 如何知道你何时完成需求的获取

没有一个简单、清楚的信号暗示你什么时候已完成需求获取。当客户和开发者与他们的同事聊天、阅读工业和商业上的文献及在早上沐浴时思考时，他们都将对潜在产品产生新的构思。你不可能全面收集需求，但是下列的提示将会暗示你在需求获取的过程中的返回点。

- 如果用户不能想出更多的使用实例，也许你就完成了收集需求的工作。用户总是按其重要性的顺序来确定使用实例的。
- 如果用户提出新的使用实例，但你可以从其它使用实例的相关功能需求中获得这些新的使用实例，这时也许你就完成了收集需求的工作。这些新的使用实例可能是你已获取的其它使用实例的可选过程。
- 如果用户开始重复原先讨论过的问题，此时，也许你就完成了收集需求的工作。
- 如果所提出的新需求比你已确定的需求的优先级都低时，也许你就完成了收集需求的工作。
- 如果用户提出对将来产品的要求，而不是现在我们讨论的特定产品，也许你就完成了收集需求的工作。

在项目的早期阶段，列出系统可能包括的功能区。在考虑多个项目的情况下，你可能要列出公共函数的清单，例如：写错误日志、备份与恢复、报表、打印、预览功能、格式和用户最喜爱的特性。然后在需求开发阶段，把你所确定的功能与原始列表进行比较。如果没有发现不一致性，则你可能完成了收集需求的工作。

下一步：

- 在你的项目或软件需求规格说明中选择一部分已经写好的客户意见输入。如图 8-5 所示，把包括在这一部分中的每一项进行分类。如果你发现有些项并不是功能需求中所要求的，就把它们放到软件需求规格说明中的正确位置上，或其它合适的文档中。
- 运用图 8-3 所示的使用实例模板为你的项目编写一个使用实例。包括所有可选过程和例外。定义功能需求，以使用户可以成功地完成这个使用实例。检查你目前的软件需求规格说明中是否包含了全部的功能需求。
- 列出在你当前项目中所用的所有需求获取的方法。那一个最有效？为什么？哪一个不可行？为什么？确定你认为最好的需求获取技术，并决定下次怎样应用它们。记住你在进行这些工作时遇到的障碍和在克服这些障碍时机智有效的办法。

第9章 编写需求文档

需求开发的最终成果是：客户和开发小组对将要开发的产品达成一致协议。这一协议综合了业务需求、用户需求和软件功能需求。就像我们早先所看到的，项目视图和范围文档包含了业务需求，而使用实例文档则包含了用户需求。你必须编写从使用实例派生出的功能需求文档，还要编写产品的非功能需求文档，包括质量属性和外部接口需求。只有以结构化和可读性方式编写这些文档，并由项目的风险承担者评审通过后，各方面人员才能确信他们所赞同的需求是可靠的。

你可以用三种方法编写软件需求规格说明：

- 用好的结构化和自然语言编写文本型文档。
- 建立图形化模型，这些模型可以描绘转换过程、系统状态和它们之间的变化、数据关系、逻辑流或对象类和它们的关系。
- 编写形式化规格说明，这可以通过使用数学上精确的形式化逻辑语言来定义需求。

由于形式化规格说明具有很强的严密性和精确度，因此，所使用的形式化语言只有极少数软件开发人员才熟悉，更不用说客户了。虽然结构化的自然语言具有许多缺点，但在大多数软件工程中，它仍是编写需求文档最现实的方法。包含了功能和非功能需求的基于文本的软件需求规格说明已经为大多数项目所接受。图形化分析模型通过提供另一种需求视图，增强了软件需求规格说明。

本章介绍软件需求规格说明的目的和结构，包括一个建议性的文档模板。同时还提供编写功能需求规格说明的原则并附带讲述几个不完善的需求陈述以及改进建议的例子。在第 10 章将介绍利用图形化技术表示需求。本书并不深入介绍形式化需求方法；若要深入讨论形式化需求方法，可参考 Alan Davis 编著的《软件需求：对象、功能和说明》(1993)。

9.1 软件需求规格说明

软件需求规格说明，也称为功能规格说明、需求协议以及系统规格说明。它精确地阐述一个软件系统必须提供的功能和性能以及它所要考虑的限制条件。软件需求规格说明不仅是系统测试和用户文档的基础，也是所有子系列项目规划、设计和编码的基础。它应该尽可能完整地描述系统预期的外部行为和用户可视化行为。除了设计和实现上的限制，软件需求规格说明不应该包括设计、构造、测试或工程管理的细节。许多读者使用软件需求规格说明来达到不同的目的：

- 客户和营销部门依赖它来了解他们所能提供的产品。
- 项目经理根据包含在软件需求规格说明中描述的产品来制定规划并预测进度安排、工作量和资源。
- 软件开发小组依赖它来理解他们将要开发的产品。
- 测试小组使用软件需求规格说明中对产品行为的描述制定测试计划、测试用例和试过程。

- 软件维护和支持人员根据 SRS 了解产品的某部分是做什么的。
- 产品发布组在 SRS 和用户界面设计的基础上编写客户文档，如用户手册和帮助屏幕等。
- 培训人员根据 SRS 和用户文档编写培训材料。

软件需求规格说明作为产品需求的最终成果必须具有综合性：必须包括所有的需求。开发者和客户不能作任何假设。如果任何所期望的功能或非功能需求未写入软件需求规格说明，那么它将不能作为协议的一部分并且不能在产品中出现。

毫无疑问，你必须在开始设计和构造之前编写出整个产品的软件需求规格说明。你可以反复地或以渐增方式编写需求规格说明，这取决于以下几个因素：是否可以一开始就确定所有的需求，编写软件需求规格说明的人是否将参与开发系统，计划发行的版本数量等等。然而，每个项目针对要实现的每个需求集合必须有一个基准协议，基准（baseline）是指正在开发的软件需求规格说明向已通过评审的软件需求规格说明的过渡过程。必须通过项目中所定义的变更控制过程来更改基准软件需求规格说明。

所有的参与者必须根据已通过评审的需求来安排工作以避免不必要的返工和误解。我见过有一个项目突然接到测试人员发出的错误灾难的报告。结果是他们测试的是老版本的软件需求规格说明，而他们觉得错误的地方正是产品所独有的特性。他们的测试工作是徒劳的，因为他们一直在老版本的软件需求规格说明中寻找错误的系统行为。

第1章提出了高质量需求文档所具有的特征：完整性、一致性、可更改性和可跟踪性。构造并编写软件需求规格说明，并使用户和其它读者能理解它（Sommerville and Sawyer 1997）。牢记以下可读性的建议：

- 对节、小节和单个需求的号码编排必须一致。
- 在右边部分留下文本注释区。
- 允许不加限制地使用空格。
- 正确使用各种可视化强调标志（例如，黑体、下划线、斜体和其它不同字体）。
- 创建目录表和索引表有助于读者寻找所需的信息。
- 对所有图和表指定号码和标识号，并且可按号码进行查阅。
- 使用字处理程序中交叉引用的功能来查阅文档中其它项或位置，而不是通过页码或章节号。

9.1.1 标识需求

为了满足软件需求规格说明的可跟踪性和可修改性的质量标准，必须唯一确定每个软件需求。这可以使你在变更请求、修改历史记录、交叉引用或需求的可跟踪矩阵中查阅特定的需求。由于要达到这一目的，用单一的项目列表是不够的，因此，我将描述几个不同的需求标识方法，并阐明它们的优点与缺点。可以选择最适合你的方法。

1) 序列号 最简单的方法是赋予每个需求一个唯一的序列号，例如 UR-2 或 SRS13。当一个新的需求加入到商业需求管理工具的数据库之后，这些管理工具就会为其分配一个序列号（许多这样的工具也支持层次化编号）。序列号的前缀代表了需求类型，例如 UR 代表“用户需求”。由于序列号不能重用，所以把需求从数据库中删除时，并不释放其所占据的序列号，而新的需求只能得到下一个可用的序列号。这种简单的编号方法并不能提供任何相关需求在逻辑上或层次上的区别，而且需求的标识不能提供任何有关每个需求内容的信息。

2) 层次化编码 这也许是最常用的方法。如果功能需求出现在软件需求规格说明中第 3.2 部分，那么它们将具有诸如 3.2.4.3 这样的标识号。标识号中的数字越多则表示该需求越详细，属于较低层次上的需求。这种方法简单且紧凑，你使用的字处理程序可能可以自动编排序号。然而，即使在一个中型的软件需求规格说明中，这些标识号也会扩展到许多位数字，并且这些标识也不提供任何有关每个需求目的的信息。如果你要插入一个新的需求，那么该需求所在部分其后所有需求的序号将要增加。删除或移去一个需求，那么该需求所在部分其后所有需求的序号将要减少。这些变化将破坏系统其它地方需求的引用。

对于这种简单的层次化编号的一种改进方法是对需求中主要的部分进行层次化编号，然后对于每个部分中的单一功能需求用一个简短文字代码加上一个序列号来识别。例如，软件需求规格说明可能包含“第 3.2.5 部分——编辑功能”，那么这一部分需求的编号可以写成 ED-1、ED-2 等等。这种方法具有层次性和组织性，同时使标识号简短和具有一定意义并与位置无关等特点。如果要在 ED-1 和 ED-2 之间插入新的需求 ED-9，则不必对该部分中余下的需求重新编号。

3) 层次化文本标签 Tom Gilb 顾问提出基于文本的层次化标签方案来标识单个需求 (Gilb 1988)。考虑这样一个需求：“当用户请求打印超过 10 个副本时，系统必须让用户进行确认判断。”这一需求可能被标识为 PRINT.COPIES.CONFIRM，这意味着这个需求是打印功能的一部分，并且与要打印的副本数量的设置问题有关。层次化文本标签是结构化的，具有语义上的含义，并且不受增加、删除或移动其它需求的影响。它们的主要缺点是文本标签比层次化数字标识码复杂。

9.1.2 处理不完整性

有时，你觉得缺少特定需求的某些信息。在解决这个不确定性之前，可能必须与客户商议、检查与另一个系统的接口或者定义另一个需求。使用“待确定”(to be determined, TBD) 符号作为标准指示器来强调软件需求规格说明中这些需求的缺陷 (gap)。通过这种方法，你可以在软件需求规格说明中查找所要澄清需求的部分。记录谁将解决哪个问题、怎样解决及什么时候解决。把每个 TBD 编号并创建一个 TBD 列表，这有助于方便地跟踪每个项目。

在继续进行构造需求集合之前，必须解决所有的 TBD 问题，因为任何遗留下来的不确定问题将会增加出错的风险和需求返工。当开发人员遇到一个 TBD 问题或其它模糊之处时，他可能不会返回到原始需求来解决问题。多半开发者对它进行猜测，但并不总是正确的。如果有 TBD 问题尚未解决，而你又要继续进行开发工作，那么尽可能推迟实现这些需求，或者解决这些需求的开放式问题，把产品的这部分设计得易于更改。

9.1.3 用户界面和软件需求规格说明

把用户界面的设计编入软件需求规格说明既有好处也有坏处。消极方面，屏幕映像和用户界面机制是解决方案（设计）的描述，而不是需求。如果你在完成了用户界面的设计之后才能确定软件需求规格说明，那么需求开发的过程将会花费很长的时间。这将会使那些只关心需求开发时间的经理、客户或开发人员失去耐心。用户界面的布局不能替代定义功能需求。不要指望开发人员可以从屏幕中推断出潜在的功能和数据关系。把用户界面的设计加入软件需求规格说明还意味着开发人员在更改一个用户界面的元素时必须相应更改需求的过程。

积极方面，探索潜在的用户界面有助于你精化需求并使用户—系统的交互对用户和开发人

员更具有实在性。用户界面的演示也有助于项目计划的制定和预测。根据以往类似开发活动的经验，你可以数清图形用户界面（GUI）的元素数目或者计算与每个屏幕有关的功能点[⊖]数目，然后估计实现这些屏幕功能所需的工作量。

一个合理的权衡点是：在软件需求规格说明中加入所选择的用户界面组件的概念映像草图，而在实现时并不一定要精确地遵循这些方法。通过使用另一种方式来表示需求，从而增强相互交流的能力，但并不增加对开发人员的限制，也不增加变更管理过程的负担。例如，一个复杂对话框的最初草案将描述支持需求部分的意图，但是一个有经验的设计者可以把它转化为一个带有标签组件的对话框或使用其它方法以提高其可用性。

9.2 软件需求规格说明模板

每个软件开发组织都应该在它们的项目中采用一种标准的软件需求规格说明的模板。有许多推荐的软件需求规格说明模板可以使用（Davis 1993；Robertson and Robertson 1999），Dorfman和Thayer（1990）从美国国家标准局、美国国防部、美国宇航局以及许多英国和加拿大的有关部门中收集了20几个需求标准和许多实例。许多人使用来自IEEE标准830-1998的模板，“IEEE推荐的软件需求规格说明的方法”（IEEE 1998）。这是一个结构好并适用于许多种软件项目的灵活的模板。

图9-1描绘了从IEEE 830标准改写并扩充的软件需求规格说明的模板。可以根据项目的需要来修改这个模板。如果模板中某一特定部分不适合你的项目，那么就在原处保留标题，并注明该项不适用。这将防止读者认为是否不小心遗漏了一些重要的部分。与其它任何软件项目文档一样，该模板包括一个内容列表和一个修正的历史记录，该记录包括对软件需求规格说明所作的修改、修改日期、修改人员和修改原因。

- | | |
|----------------|--------------|
| a. 引言 | d. 系统特性 |
| a.1 目的 | d.1 说明和优先级 |
| a.2 文档约定 | d.2 激励/响应序列 |
| a.3 预期的读者和阅读建议 | d.3 功能需求 |
| a.4 产品的范围 | e. 其它非功能需求 |
| a.5 参考文献 | e.1 性能需求 |
| b. 综合描述 | e.2 安全设施需求 |
| b.1 产品的前景 | e.3 安全性需求 |
| b.2 产品的功能 | e.4 软件质量属性 |
| b.3 用户类和特征 | e.5 业务规则 |
| b.4 运行环境 | e.6 用户文档 |
| b.5 设计和实现上的限制 | f. 其它需求 |
| b.6 假设和依赖 | 附录A：词汇表 |
| c. 外部接口需求 | 附录B：分析模型 |
| c.1 用户界面 | 附录C：待确定问题的列表 |
| c.2 硬件接口 | |
| c.3 软件接口 | |
| c.4 通信接口 | |

图9-1 软件需求规格说明模板

[⊖] 功能点是对一个应用程序中用户可见功能的数量的测量，而与如何构造功能无关。你可以根据内部逻辑文件、外部接口文件以及外部输入、输出和请求的数量，从对用户需求的理解中估计功能点（IFPUG 1999）。

本章的剩余部分将描述图 9-1 所示的模板中每一部分应包含的信息。你可以通过参考其它已编写好的项目文档（例如项目视图和范围文档或接口规格说明）来将每一部分内容具体化，而不是复制信息或者把所有的内容组成一个单一的文档。不要生搬硬套这个模板，应该把这个模板转换为你所需要的文档。

a. 引言

引言提出了对软件需求规格说明的纵览，这有助于读者理解文档如何编写并且如何阅读和解释。

a.1 目的

对产品进行定义，在该文档中详尽说明了这个产品的软件需求，包括修正或发行版本号。如果这个软件需求规格说明只与整个系统的一部分有关系，那么就只定义文档中说明的部分或子系统。

a.2 文档约定

描述编写文档时所采用的标准或排版约定，包括正文风格、提示区或重要符号。例如，说明了高层需求的优先级是否可以被其所有细化的需求所继承，或者每个需求陈述是否都有其自身的优先级。

a.3 预期的读者和阅读建议

列举了软件需求规格说明所针对的不同读者，例如开发人员、项目经理、营销人员、用户、测试人员或文档的编写人员。描述了文档中剩余部分的内容及其组织结构。提出了最适合于每一类型读者阅读文档的建议。

a.4 产品的范围

提供了对指定的软件及其目的的简短描述，包括利益和目标。把软件与企业目标或业务策略相联系。可以参考项目视图和范围文档而不是将其内容复制到这里。

a.5 参考文献

列举了编写软件需求规格说明时所参考的资料或其它资源。这可能包括用户界面风格指导、合同、标准、系统需求规格说明、使用实例文档，或相关产品的软件需求规格说明。在这里应该给出详细的信息，包括标题名称、作者、版本号、日期、出版单位或资料来源，以方便读者查阅这些文献。

b. 综合描述

这一部分概述了正在定义的产品以及它所运行的环境、使用产品的用户和已知的限制、假设和依赖。

b.1 产品的前景

描述了软件需求规格说明中所定义的产品的背景和起源。说明了该产品是否是产品系列中的下一成员，是否是成熟产品所改进的下一代产品、是否是现有应用程序的替代品，或者是否是一个新型的、自含型产品。如果软件需求规格说明定义了大系统的一个组成部分，那么就要说明这部分软件是怎样与整个系统相关联的，并且要定义出两者之间的接口。

b.2 产品的功能

概述了产品所具有的主要功能。其详细内容将在 d 中描述，所以在此只需要概略地总结，例如用列表的方法给出。很好地组织产品的功能，使每个读者都易于理解。用图形表示主要的需求分组以及它们之间的联系，例如数据流程图的顶层图或类图，都是有用的。

b.3 用户类和特征

确定你觉得可能使用该产品的不同用户类并描述它们相关的特征（见第 7 章）。有一些需求可能只与特定的用户类相关。将该产品的重要用户类与那些不太重要的用户类区分开。

b.4 运行环境

描述了软件的运行环境，包括硬件平台、操作系统和版本，还有其它的软件组件或与其共存的应用程序。

b.5 设计和实现上的限制

确定影响开发人员自由选择的问题，并说明这些问题为什么成为一种限制。可能的限制包括如下内容：

- 必须使用或者避免的特定技术、工具、编程语言和数据库。
- 所要求的开发规范或标准（例如，如果由客户的公司负责软件维护，就必须定义转包者所使用的设计符号表示和编码标准。
- 企业策略、政府法规或工业标准。
- 硬件限制，例如定时需求或存储器限制。
- 数据转换格式标准。

b.6 假设和依赖

列举出在对软件需求规格说明中影响需求陈述的假设因素（与已知因素相对立）。这可能包括你打算要用的商业组件或有关开发或运行环境的问题。你可能认为产品将符合一个特殊的用户界面设计约定，但是另一个 SRS 读者却可能不这样认为。如果这些假设不正确、不一致或被更改，就会使项目受到影响。

此外，确定项目对外部因素存在的依赖。例如，如果你打算把其它项目开发的组件集成到系统中，那么你就要依赖那个项目按时提供正确的操作组件。如果这些依赖已经记录到其它文档（例如项目计划）中了，那么在此就可以参考其它文档。

c. 外部接口需求

利用本节来确定可以保证新产品与外部组件正确连接的需求。关联图表示了高层抽象的外部接口。需要把对接口数据和控制组件的详细描述写入数据字典中。如果产品的不同部分有不同的外部接口，那么应把这些外部接口的详细需求并入到这一部分的实例中。

c.1 用户界面

陈述所需要的用户界面的软件组件。描述每个用户界面的逻辑特征。以下是可能要包括的一些特征：

- 将要采用的图形用户界面（GUI）标准或产品系列的风格。
- 屏幕布局或解决方案的限制。
- 将出现在每个屏幕的标准按钮、功能或导航链接（例如一个帮助按钮）。
- 快捷键。
- 错误信息显示标准。

对于用户界面的细节，例如特定对话框的布局，应该写入一个独立的用户界面规格说明中，而不能写入软件需求规格说明中。

c.2 硬件接口

描述系统中软件和硬件每一接口的特征。这种描述可能包括支持的硬件类型、软硬件之

间交流的数据和控制信息的性质以及所使用的通信协议。

c.3 软件接口

描述该产品与其它外部组件（由名字和版本识别）的连接，包括数据库、操作系统、工具、库和集成的商业组件。明确并描述在软件组件之间交换数据或消息的目的。描述所需要的服务以及内部组件通信的性质。确定将在组件之间共享的数据。如果必须用一种特殊的方法来实现数据共享机制，例如在多任务操作系统中的一个全局数据区，那么就必须把它定义为一种实现上的限制。

c.4 通信接口

描述与产品所使用的通信功能相关的需求，包括电子邮件、Web浏览器、网络通信标准或协议及电子表格等等。定义了相关的消息格式。规定通信安全或加密问题、数据传输速率和同步通信机制。

d. 系统特性

在图9-1所示的模板中，功能需求是根据系统特性即产品所提供的主要服务来组织的。你可能更喜欢通过使用实例、运行模式、用户类、对象类或功能等级来组织这部分内容（IEEE 1998）。你还可以使用这些元素的组合。总而言之，你必须选择一种使读者易于理解预期产品的组织方案。

仅用简短的语句说明特性的名称，例如“4.1拼写检查和拼写字典管理”。无论你想说明何种特性，阐述每种特性时都将重述从d.1～d.3这三步系统特性。

d.1 说明和优先级

提出了对该系统特性的简短说明并指出该特性的优先级是高、中，还是低。或者你还可以包括对特定优先级部分的评价，例如利益、损失、费用和风险，其相对优先等级可以从1（低）到9（高）（见第13章）。

d.2 激励/响应序列

列出输入激励（用户动作、来自外部设备的信号或其它触发器）和定义这一特性行为的系统响应序列。就像在第8章讨论的那样，这些序列将与使用实例相关的对话元素相对应。

d.3 功能需求

详列出与该特性相关的详细功能需求。这些是必须提交给用户的软件功能，使用户可以使用所提供的特性执行服务或者使用所指定的使用实例执行任务。描述产品如何响应可预知的出错条件或者非法输入或动作。就像本章开头所描述的那样，你必须唯一地标识每个需求。

e. 其它非功能需求

这部分列举出了所有非功能需求，而不是外部接口需求和限制。

e.1 性能需求

阐述了不同的应用领域对产品性能的需求，并解释它们的原理以帮助开发人员作出合理的设计选择。确定相互合作的用户数或者所支持的操作、响应时间以及与实时系统的时间关系。你还可以在这里定义容量需求，例如存储器和磁盘空间的需求或者存储在数据库中表的最大行数。尽可能详细地确定性能需求。可能需要针对每个功能需求或特性分别陈述其性能需求，而不是把它们都集中在一起陈述。例如，“在运行微软Window 2000的450 MHzPentium的计算机上，当系统至少有50%的空闲资源时，95%的目录数据库查询必须在两秒内完成”。

e.2 安全设施需求

详尽陈述与产品使用过程中可能发生的损失、破坏或危害相关的需求。定义必须采取的安全保护或动作，还有那些预防的潜在的危险动作。明确产品必须遵从的安全标准、策略或规则。一个安全设施需求的范例如下：“如果油箱的压力超过了规定的最大压力的 95%，那么必须在1秒钟内终止操作”。

e.3 安全性需求

详尽陈述与系统安全性、完整性或与私人问题相关的需求，这些问题将会影响到产品的使用和产品所创建或使用的数据的保护。定义用户身份确认或授权需求。明确产品必须满足的安全性或保密性策略。你可能更喜欢通过称为完整性的质量属性来阐述这些需求，完整性将在第11章介绍。一个软件系统的安全需求的范例如下：“每个用户在第一次登录后，必须更改他的最初登录密码。最初的登录密码不能重用。”

e.4 软件质量属性

详尽陈述与客户或开发人员至关重要的其它产品质量特性（见第 11 章）。这些特性必须是确定、定量的并在可能时是可验证的。至少应指明不同属性的相对侧重点，例如易用程度优于易学程度，或者可移植性优于有效性。

e.5 业务规则

列举出有关产品的所有操作规则，例如什么人在特定环境下可以进行何种操作。这些本身不是功能需求，但它们可以暗示某些功能需求执行这些规则。一个业务规则的范例如下：“只有持有管理员密码的用户才能执行 \$100.00 或更大额的退款操作。”

e.6 用户文档

列举出将与软件一同发行的用户文档部分，例如，用户手册、在线帮助和教程。明确所有已知的用户文档的交付格式或标准。

f. 其它需求

定义在软件需求规格说明的其它部分未出现的需求，例如国际化需求或法律上的需求。你还可以增加有关操作、管理和维护部分来完善产品安装、配置、启动和关闭、修复和容错，以及登录和监控操作等方面的需求。在模板中加入与你的项目相关的新部分。如果你不需要增加其它需求，就省略这一部分。

附录A：词汇表

定义所有必要的术语，以便读者可以正确地解释软件需求规格说明，包括词头和缩写。你可能希望为整个公司创建一张跨越多项项目的词汇表，并且只包括特定于单一项目的软件需求规格说明中的术语。

附录B：分析模型

这个可选部分包括或涉及到相关的分析模型的位置，例如数据流程图、类图、状态转换图或实体-关系图（见第10章）。

附录C：待确定问题的列表

编辑一张在软件需求规格说明中待确定问题的列表，其中每一项都是编上号的，以便于跟踪调查。

9.3 编写需求文档的原则

编写优秀的需求文档没有现成固定的方法，最好是根据经验进行。从过去所遇到的问题

中可使你受益匪浅。许多需求文档可以通过使用有效的技术编写风格和使用用户术语而不是计算机专业术语的方式得以改进 (Kovitz 1999)。你在编写软件需求文档时，应牢记以下几点建议：

- 保持语句和段落的简短。
- 采用主动语态的表达方式。
- 编写具有正确的语法、拼写和标点的完整句子。
- 使用的术语与词汇表中所定义的应该一致。
- 需求陈述应该具有一致的样式，例如“系统必须……”或者“用户必须……”，并紧跟一个行为动作和可观察的结果。例如，“仓库管理子系统必须显示一张所请求的仓库中有存货的化学药品容器清单。”
- 为了减少不确定性，必须避免模糊的、主观的术语，例如，用户友好、容易、简单、迅速、有效、支持、许多、最新技术、优越的、可接受的和健壮的。当用客说“用户友好”或者“快”或者“健壮”时，你应该明确它们的真正含义并且在需求中阐明用户的意图。
- 避免使用比较性的词汇，例如：提高、最大化、最小化和最佳化。定量地说明所需要提高的程度或者说清一些参数可接受的最大值和最小值。当客户说明系统应该“处理”、“支持”或“管理”某些事情时，你应该能理解客户的意图。含糊的语句表达将引起需求的不可验证。

由于需求的编写是层次化的，因此，可以把顶层不明确的需求向低层详细分解，直到消除不明确性为止。编写详细的需求文档，所带来的益处是如果需求得到满足，那么客户的目的也就达到了，但是不要让过于详细的需求影响了设计。如果你能用不同的方法来满足需求且这种方法都是可接受的，那么需求的详细程度也就足够了。然而，如果评审软件需求规格说明的设计人员对客户的意图还不甚了解，那么就需要增加额外的说明，以减少由于误解而产生返工的风险。

需求文档的编写人员总是力求寻找到恰如其分的需求详细程度。一个有益的原则就是编写单个的可测试需求文档。如果你想出一些相关的测试用例可以验证这个需求能够正确地实现，那么就达到了合理的详细程度。如果你预想的测试很多并且很分散，那么可能就要将一些集合在一起的需求分离开。已经建议将可测试的需求作为衡量软件产品规模大小的尺度 (Wilson 1995)。

文档的编写人员必须以相同的详细程度编写每个需求文档。我曾见过在同一份软件需求规格说明中，对需求的说明五花八门。例如，“组合键Control-S代表保存文件”和“组合键Control-P代表打印文件”被当成两个独立的需求。然而，“产品必须响应以语音方式输入的编辑指令”则被作为一个子系统，而并不作为一个简单的功能需求。

文档的编写人员不应该把多个需求集中在一个冗长的叙述段落中。在需求中诸如“和”，“或”之类的连词就表明了该部分集中了多个需求。务必记住，不要在需求说明中使用“和 / 或”，“等等”之类的连词。

文档的编写人员在编写软件需求规格说明时不应该出现需求冗余。虽然在不同的地方出现相同的需求可能会使文档更易读，但这也造成了维护上的困难。需求的多个实例都需要同时更新，以免造成需求各实例之间的不一致。在软件需求规格说明中交叉引用相关的各项，

在进行更改时有助于保持它们之间的同步。让独立性强的需求在需求管理工具或数据库中只出现一次，这样可以缓和冗余问题。

表9-1 适合某种模式的需求编号列表的表格化示例

管 区	带标记格式	无标记格式	ASCII格式
联邦	ED-13.1	ED-13.2	ED-13.3
州	ED-13.4	ED-13.5	ED-13.6
地区	ED-13.7	ED-13.8	ED-13.9
国际	ED-13.10	ED-13.11	ED-13.12

文档的编写人员应考虑用最有效的方法表达每个需求。考虑符合如下句型的一系列需求：“文本编辑器应该能分析定义有<管区>法律的<格式>文档。”对于12种相似的需求中，<格式>所取的可能值有3种，<管区>所取的可能值有4种。当你评审与此类似的需求时，很难发现遗漏了一个需求，例如“文本编辑器应该能分析定义了国际法的无标记文档。”就像表9-1所示那样，可以用表中的这种格式表示需求，以确保你没有遗漏掉任何一个需求。更高层的需求可以这样描述：“ED-13文本编辑器应该能分析定义有不同管区法律的多种格式的文档，如表<xx>所示。”

9.4 需求示例的改进前后

第1章曾经提出了高质量需求陈述的许多特性：完整性、正确性、可行性、必要性、设定优先级、明确性和可验证性。由于不具有这些特征的需求将会引起混淆，导致将来的返工，因此，你必须尽快找出并纠正这些问题。下面是从真实项目中改编一些存在问题的需求。根据这些质量特性来检查每一个需求陈述，看一看你能否发现问题所在。针对每个需求陈述，将给出我的看法和建议性的改进方案。我确信，体验每个改进方案将使你受益匪浅，但你的目标不是编写完美的需求文档。你只需在一个可接受的风险程度上编写需求文档，使之有助于你的设计和软件构造。

例1

“产品必须在固定的时间间隔内提供状态消息，并且每次时间间隔不得小于60秒”。

这个需求看起来是不完整的：什么是状态消息，并且在什么情况下向用户提供这些消息？显示时间多长？我们所说的是产品的哪一部分？时间间隔也会导致混淆。假定显示状态消息之间的时间间隔只要求不少于60秒，按这样推理，是否可以每隔一年显示一次状态信息？如果意图是消息之间的时间间隔不多于60秒，那么1毫秒会不会太短？消息显示的时间间隔怎样才能一致？“每次”这个词混淆了这一问题。由于这些问题的存在，导致了需求是不可验证的。

这里提出一个方法使我们能重写需求文档来表述这些缺点（从我们与客户一致的目标出发作出一些猜测）即：后台任务管理器（BTM）应该在用户界面的指定区域显示状态消息。

- 在后台任务进程启动之后，消息必须每隔60（±10）秒更新一次，并且保持连续的可见性。
- 如果正在正常处理后台任务进程，那么后台任务管理器（BTM）必须显示后台任务进程已完成的百分比。
- 当完成后台任务时，后台任务管理器（BTM）必须显示一个“已完成”的消息。

d. 如果后台任务中止执行，那么后台任务管理器（BTM）必须显示一个出错消息。

我把原先的需求分割成多个需求，因为每个需求都需要独立的测试用例并且使各个需求都具有可跟踪性。如果把多个需求都集中在一个段落中，那么在构造软件和测试时就很容易忽略其中某个需求。注意，修改之后的需求并没有精确地说明是怎样显示状态信息的。这是一个设计问题，如果你在这个地方详述该问题，那么就会给开发人员带来设计上的一些限制。过早地限制设计上的可选方案将会给编程人员带来不利因素，并可导致产品设计的失败。

例2

“产品必须在显示和隐藏非打印字符之间进行瞬间切换”。

在瞬间这一时间概念上，计算机不能完成任何工作，因此，这个需求是不可行的。该需求也是不完整的，因为它没有说清状态切换的原因。在特定的条件下，软件产品是否可以进行自动切换或者可否由用户采取某些措施来激发这样转变？还有，在文档中显示转变的范围是什么？是所选的文本、整个文档或其它内容？这个需求也存在一个不确定性问题。“非打印”字符是否指隐藏文本、属性标记或者其它的控制字符？由于存在这些问题，该需求是不可验证的。

用如下的语句描述这个需求可能会更好一些：“用户在编辑文档时，通过激活特定的触发机制，可以在显示和隐藏所有HTML标记之间进行切换”。现在，指代关系就清楚了，非打印字符指的是HTML标记。修改过的需求指明了是用户触发了显示状态的转换，但是它并没有对设计造成限制，因为它并没有精确定义所使用的机制。当设计人员选择好一种触发机制（例如热键、菜单命令或语音输入）时，你就可以编写详细的测试用例来验证这种转换操作是否正确。

例3

“分析程序应该能生成HTML标记出错的报告，这样就可以使HTML的初学者使用它来迅速排错。”

“迅速”这个词具有模糊性。缺乏对出错报告内容的定义表明该需求是不完整的。我不知道你是如何验证这个需求的。找一些HTML的初学者，看他们利用这个报告是否可以迅速排错？还有一点不清楚的是：HTML初学者使用的是分析程序还是出错报告。并且何时生成这样的报告？

让我们使用另一种方式表述这个需求：

a. 在HTML分析程序完全分析完一个文件后，该分析程序必须生成一个出错报告，这个报告中包含了在分析文件过程中所发现错误的HTML所在的行号以及文本内容，还包含了对每个错误的描述。

b. 如果在分析过程中未发现任何错误，就不必生成出错报告。

现在我们知道了任何生成出错报告及其所包含的内容，但是我们已经把该需求提交给设计人员，让他们来决定报告的形式。我们还指明了一种例外情况：如果没有任何错误，就不生成出错报告。

例4

“如果可能的话，应当根据主货物编号列表在线确认所输入的货物编号。”

我在想：“如果可能的话”这句话意味着什么？该需求是否在技术上可行？是否可以在线访问主货物编号列表？如果你不能确信是否可以递交一个请求，那么就使用“待确定”（TBD）

来表示未解决的问题。这个需求是不完整的，因为它并没有指明如果确认通过或失败，将会发生什么情况。应该尽量避免使用不精确的词汇，例如“应当”。客户可能需要这个功能或者不需要这个功能。一些需求规格说明利用关键字之间微妙的差别如“应当”，“必须”和“可能”来指明重要性。我更喜欢使用“必须”或“将要”来明确说明需求的目的并且明确指定其优先级。

改进后的该需求描述如下：

“系统必须根据在线的主货物编号列表确认所输入的货物编号。如果在主列表中查不到该货物的编号，系统必须显示一个出错消息并且拒绝订货。”

第二种相关需求可能记录了一种异常情况：当进行货物编号确认时，主货物编号列表不可访问。

例5

“产品不应该提供将带来灾难性后果的查询和替换选择。”

“灾难性后果”的含义是解释的中心词。在编辑文档时，毫无目的地作出全局性变化而用户又不能检测出错误或没有任何办法来纠正它，此时就可能带来灾难性后果。你也要合理地使用反面需求，因为这些需求描述了系统所不能做的事情。潜在的关注焦点在于当发生意外损坏时，能保护文件的内容。也许，真正的需求是针对多级撤销(undo)能力、全局变化或其它可导致数据丢失行为确定的。

9.5 数据字典

在很久以前，我曾经参与一项项目，在此期间，由三位编程人员对于同一数据项使用不同的变量名称、长度和有效性验证。这将导致在真正的数据定义上的混淆，当数据存入占据空间太小的变量中时将要截短数据，并且造成维护上的困难。我们所缺少的是一个数据字典——一个定义应用程序中使用的所有数据元素和结构的含义、类型、数据大小、格式、度量单位、精度以及允许取值范围的共享仓库。

数据字典可以把不同的需求文档和分析模型紧密结合起来。如果所有的开发人员在数据字典上取得一致意见，那么就可以缓和集成性问题。为了避免冗余和不一致性，应该为你的项目创建一个独立的数据字典，而并不是在每个需求出现的地方定义每一个数据项。数据字典的维护独立于软件需求规格说明，并且在产品的开发和维护的任何阶段，各个风险承担者都可以访问数据字典。

在数据字典中，可以使用简单的符号表示数据项(Robertson and Robertson 1994)。项写在等号的左边，而其定义写在右边。这种符号定义了原数据元素、组成结构体的复杂数据元素、重复的数据项、一个数据项的枚举值以及可选的数据项。以下所示的例子来自“化学制品跟踪系统”。

1) 原数据元素 一个原数据元素是不可分解的。可以给它赋予一个数量值。原数据的定义必须确定其数据类型、大小、允许取值的范围等等。典型的原数据元素的定义是一行注释文本，并以星号作为界限：

请求标识号= *6位系统生成的顺序整数，以1开头，并能唯一标识每个请求*

2) 组合项 一个数据结构或记录包含了多个数据项。如果数据结构中的项是可选的，就把它用括弧括起来：

请求的化学制品=化学制品标识号

+数量

+数量单位

+ (供应商名称)

这个结构确定了与请求一种特定化学制品相关的所有信息。供应商名称是可选的，因为提出请求的人并不关心化学制品是从哪个供应商处购买来的。每个出现在结构中的数据项都必须写入数据字典。结构中还可以包含其它结构。

3) 重复项 如果一个项的多个实例将出现在数据结构中，就把该项用花括弧括起来。如果你知道可能允许的重复次数，就用“最小值：最大值”这种形式写在括号之前：

请求=请求标识号

+产品编号

+ 1 :10{ 请求的化学制品 }

这个例子表明，一个化学制品的请求至少应包含一种化学制品，但不能多于 10种。每个请求也包括一个单一的请求标识号和一个产品编号，它们的格式将在数据字典的其它地方定义。

4) 选择项 如果一个原数据项元素可以取得有限的离散值，就把这些值列举出来：

数量单位 = [“克” | “千克” | “个”]

* 文本串表示了与所请求的化学制品的量相关的单位 *

表明了数量单位的文本串只允许 3 种取值。注释提供了数据项定义的信息。

你在创建数据字典和词汇表上所花费的时间可以大大减少由于项目的参与者对一些关键信息的理解不一致所带来的浪费。如果你保持词汇表和数据字典的正确性，那么在系统的整个维护期间和相关的及以后产品的开发中，它们将是很有价值的工具。

下一步：

- 从你的软件需求规格说明中取出一页功能需求说明。检查每个语句，看它是否与好的需求特性相符，重写那些不符合的需求。
- 如果公司对需求文档没有标准的格式，那么就召集一个小的工作组讨论决定采纳的一个标准的软件需求规格说明模板。从图 9-1 的模板开始并改编这个模板，使其最好地适合你的项目和产品。在标识需求方面也要一致。
- 召集一个由 3 到 7 个项目的风险承担者组成的小组来正式评审项目中的软件需求规格说明。确保每个需求规格说明都是清晰的、可行的、可验证的及无二义性的等等。寻找规格说明中不同需求间的冲突，以及软件需求规格说明中遗漏的部分。通过检查软件需求规格说明，确保纠正了在软件需求规格说明中及其后续产品里出现的错误。

第10章 需求的图形化分析

“化学制品跟踪系统”的项目开发组正在进行第一次软件需求规格说明的评审。参加者有Dave(项目经理), Lori(需求分析者), Helen(高级程序员), Ramesh(测试专家), Tim(化学制品的产品代表者), 还有Roxanne(化学制品仓库的产品代表者)。Tim开始说：“我阅读过整个软件需求规格说明。大部分都符合我的需求，但是有几个部分我很难同意。我不能确信在化学制品请求过程中，我们是否确定了这些步骤。”

Ramesh又补充说：“当一个请求通过系统时，我很难想象用于覆盖该请求状态变化的所有测试用例。我发现许多关于状态变化的需求散布在整个软件需求规格说明中，但我无法确定是否有一些需求遗漏了或存在不一致性。”

Roxanne有一个类似的问题。“当我阅读了如何真正请求一种化学药品时，我感到困惑”，她说，“单个需求是能感觉到的，但我难以想像我所要完成的步骤顺序。”

在各评审员提出其它相关的问题后，Lori做出了总结：“看来软件需求规格说明似乎没有完全告诉我们对于理解系统所需的各个方面，也不能确保我们没有错过一个需求或不犯任何错误。我将画一些图来帮助我们想像这些需求，并看一下能否澄清这些问题域。谢谢你们的反馈意见。”

根据在需求方面的权威Alan Davis的见解，仅仅单一来看需求并不能提供对需求的完全理解(Davis 1995)，你需要把用文本表示的需求和用图形表示的需求结合起来，绘制出对预期系统的完整描述，并可帮助你检测不一致性、模糊性、错误和遗漏。这些图形化的表示或者分析模型可以增强你对系统需求的理解。在项目的参与者之间，对于某些类型的信息，图形化交互比文本交互更高效，并且可以在不同的开发组成员之间扫清语言和词汇上的障碍。本章将提供对需求建模技术的简要概述，在我看来，这些技术有助于理解用户的业务问题和软件需求。

10.1 需求建模

许多年前，当我开始绘制分析模型时，我希望找到一种技术，可以把所有的内容都包容进一个完整的需求描述中。最终我得出一个结论：不存在一个包罗万象的图。早期的结构化系统分析的目标是用比叙述文本更正式的图形表示来替换整个分类功能规格说明(DeMarco 1979)。然而，经验告诉我们：分析模型应该增强自然语言的需求规格说明，而不是替换之(Davis 1995)。

需求的图形化表示的模型包括数据流图(DFD)、实体关系图(ERD)、状态转化图(STD)、对话图和类图。还有一些非常规的建模方法也是有价值的。一个项目开发组利用项目规划工具为嵌入式软件产品成功地画出时间需求，其工作在毫秒级，而不是以天或星期计算。这些模型有助于解决设计软件的问题，而且对详述和探索需求也是有益的。作为需求分析工具，你可以用这些图对问题域进行建模，或者创建新系统的概念表示法。图形有助于分析者和客户在需求方面形成一致的、综合的理解，并且还可以发现需求的错误。

在需求分析方面或设计方面是否使用模型取决于建模的定时和目的(timing and intent of the modeling)。在需求开发中通过建立模型来确信你理解了需求。模型描述了问题域的逻辑方面，如数据组成、事务和转换、现实世界对象和允许的状态。或者可以从文本需求出发来

画模型，从不同的角度来表示这些需求，或者可以从所画的基于用户输入的模型来获得功能需求。在设计阶段，要从物理上而不是从逻辑上画出模型来明确说明将如何实现该系统：规划建立的数据库，将举例说明的对象类，还有你将开发的编码模块。

本章所叙述的分析建模技术是由各种商业计算机辅助软件工程或 CASE 工具支持的。CASE 工具提供了普通画图工具所没有的许多性能。首先，这些工具通过交互画图使你易于对模型进行改进。决不可能第一次就画出一个正确的模型，因此，在系统建模中提供交互功能是成功的一个关键 (Wiegers 1996a)。第二，CASE 工具知道它们所支持的每一种建模方法的规则。它们可以验证模型，并且识别人们在评审图形时没有发现的语法或逻辑错误。该工具还可以把多系统图形一起连接到数据字典中以共享数据定义。CASE 工具有助于你保持模型之间的一致性并使模型与软件需求规格说明中的功能需求保持一致。

分析模型方便了项目参与者在系统的某些方面的交流。可能不需要整个系统的模型集，只需关注建模中系统最复杂和最关键的部分，因为这部分最容易产生模糊性和不确定性。这里所表示的符号为项目的参与者提供了统一的语言，但是也可以使用非正式的图来增强口头和书面的方案交流。

10.2 从客户需求到分析模型

通过认真听取客户如何陈述它们的需求，分析者可以挑选出关键字，这些关键字可以翻译成特定的分析模型元素。表 10-1 建议了一些可能的映射，根据客户输入，把重要的名词和动词映射成特定的模型组件，这将在本章的后面部分介绍。当把客户输入转变为书面的需求或模型时，还可以根据模型的每个组件回溯到需求部分。

表10-1 把客户的需求关联到分析模型的组件

单词类型	例 子	分析模型组件
名词	• 人、组织、软件系统、 数据项或者存在对象	• 端点或数据存储 (DFD) • 实体或它们的属性 (ERD) • 类或它们的属性 (类图)
动词	• 行为、用户可做的事 或可发生的事件	• 过程 (DFD) • 关系 (ERD) • 转化 (STD) • 类操作 (类图)

在整本书中，我已经使用“化学制品跟踪系统”作为一个学习的例子。基于此例子，考虑如下用户需求部分，这些需求是由代表化学制品用户类的产品代表者提供的。由于用图示例的缘故，一些模型所示的信息可能会超出这部分所包含的信息，而另一些模型可能只描述部分信息。

“一位化学家或化学制品仓库人员可以提出对一种或多种化学制品的请求。对该请求的执行可以有两种途径：一是传送一个存在于化学制品仓库清单上的化学制品容器，二是向外界供应商提交一份订购新的化学制品的订单。提出请求的人在准备他/她的请求时应该可以通过在线查找供应商目录表找到特定的化学制品。系统需要从请求准备直到请求执行或取消这一阶段跟踪每一个化学制品的状态。系统还必须保持跟踪每个化学制品容器的历史记录，从化学制品容器到达公司到它完全被消耗或废弃为止。”

10.3 数据流图

数据流图 (data flow diagram , DFD) 是结构化系统分析的基本工具 (DeMarco 1979 ; Robertson and Robertson 1994)。一个数据流图确定了系统的转化过程、系统所操纵的数据或物质的收集 (存储) , 还有过程、存储、外部世界之间的数据流或物质流。数据流模型把层次分解方法运用到系统分析上 , 这种方法很适用于事务处理系统和其它功能密集型应用程序。通过加入控制流元素后 , 数据流图技术就可以扩充到允许实时系统的建模。

数据流图是当前业务过程或新系统操作步骤的一种表示方法。数据流图可以在一个抽象的广泛范围内表示系统。在一个多步骤的活动中 , 高层数据流图对数据和处理部分提供一个整体的统览 , 这是对包含在软件需求规格说明中的精确、详细叙述的补充。数据流图描述了软件需求规格说明中的功能需求怎样结合在一起使用户可以执行指定的任务 , 例如请求一种化学制品。在与用户一起讨论业务过程时我经常绘制数据流图。从图中迅速反馈的信息有助于对所探讨的任务流的理解进行提炼加工。

图6-2所示的关联图是数据流图最高层的抽象。关联图把整个系统表示成一个简单的黑匣子的过程 , 并用一个圆圈表示。关联图还表示出外部实体或与系统有关的端点 , 以及在系统与端点之间的数据和物质流。在关联图中 , 元素之间的流往往代表了复杂的数据结构 , 这些数据结构在数据字典中定义。

你可以把关联图详述成 0 层数据流图 , 这时将系统划分为主要部分或过程。图 10-1展示了“化学制品跟踪系统”的 0 层数据流图 (略作简化)。在关联图中代表整个“化学制品跟踪系统”的单一过程圆圈被细分成 7 个主要过程 (圆圈)。在关联图中 , 端点用矩形框表示。关联图中所有的数据流也出现在 0 层数据流图上。此外 , 0 层数据流图包含了许多数据存储 (data store) , 它是用一对水平的平行线表示 , 由于数据存储在系统内部 , 因此它们并不出现在关联图中。从圆圈到数据存储的流表示数据放入数据存储器 , 从数据存储器出来的流表示一个读操作 , 而数据存储器和圆圈之间的双向箭头则表示一个更新操作。

在 0 层图中 , 每个独立的圆圈所代表的过程可以进一步扩展成一个独立的数据流图 , 以揭示系统中程序的细节部分。这种循序渐进的细化过程可以继续进行 , 直到最低层的图仅描述原子过程操作为止 , 这些原子操作可以清楚地表示所叙述文本、伪码、流程图或程序代码。软件需求规格说明中的功能需求将精确地定义每个原子过程的行为。每一层数据流图必须与它上一层数据流图保持平衡和一致 , 因此 , 子图的所有输入输出流要与其父图相匹配。高层图中复杂的数据流可以分解到低层数据流图中 , 并把这些数据结构写入数据字典中。

初看起来 , 图 10-1 可能有点混乱。然而 , 如果你仔细观察每一个过程的周围环境 , 你就会看到该过程的输入和输出数据项 , 还有它们的源和目的地。与数据存储相连的流可以引起建立或消耗数据存储内容的过程。为了看清一个过程如何使用数据项 , 你需要画出更详细的 DFD 子图或者参考系统有关部分的功能需求。

以下是绘制数据流图的一些规则。并不是每个人都要遵循这些规则 , 但是我发现这些规则很有用。利用模型以增进项目参与者之间的交流比生搬硬套这些规则更为重要。

- 把数据存储放在 0 层数据流图或更低层子图上 , 不要放在关联图上。
- 过程是通过数据存储进行通讯 , 而不是从一个过程直接流到另一过程。类似地 , 数据不能直接由一个数据存储直接流到另一个数据存储 , 它必须通过一个过程圆圈。
- 使用数据流图时 , 不要试图让数据流图反映处理的顺序。

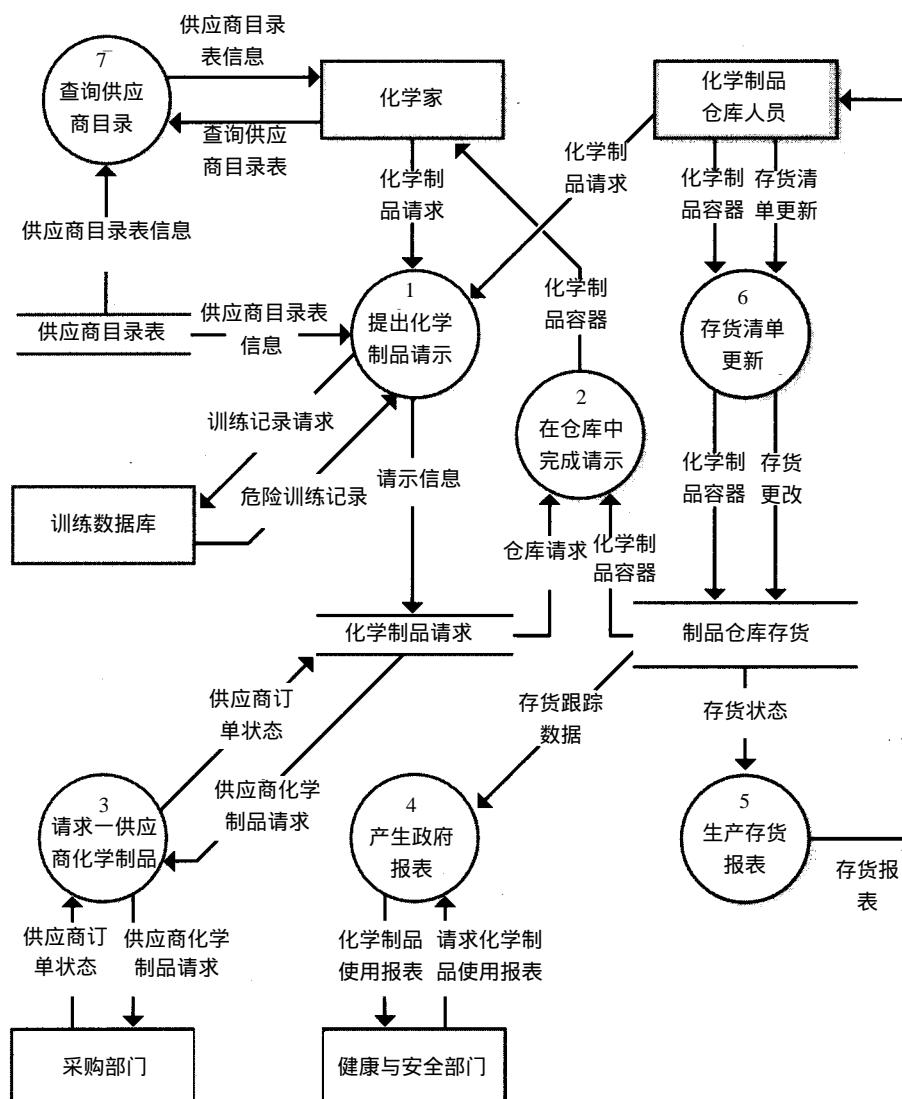


图10-1 “化学品跟踪系统”的0层数据流图

- 用一个简明的动作命名过程：动词 + 对象。数据流图中所用的名字应对客户有意义，并且与业务或问题域相关。
- 对过程的编号要唯一且具有层次性。在 0 层图上，每个过程的编号用整数表示。如果你为过程 3 创建子图，则子图中的过程编号应表示为 3.1, 3.2 等等。
- 不要在一个图中绘制多达 7~10 个以上的过程，否则就很难绘制、更改和理解。
- 不要使某些圆圈只有输入或只有输出。数据流图中圆圈所代表的处理过程通常要求既有输入又有输出。

10.4 实体联系图

与数据流图描绘了系统中发生的过程一样，实体联系图（entity-relationship diagram, ERD）描绘了系统的数据关系（Wieringa 1996）。如果你的实体联系图表示来自于问题域及其

联系的逻辑信息组，那么你正在利用实体联系图作为需求分析的工具。分析实体联系图有助于对业务或系统数据组成的理解和交互，并暗示产品将有必要包含一个数据库。相反，当你在系统设计阶段建立实体联系图时，通常要定义系统数据库的物理结构。

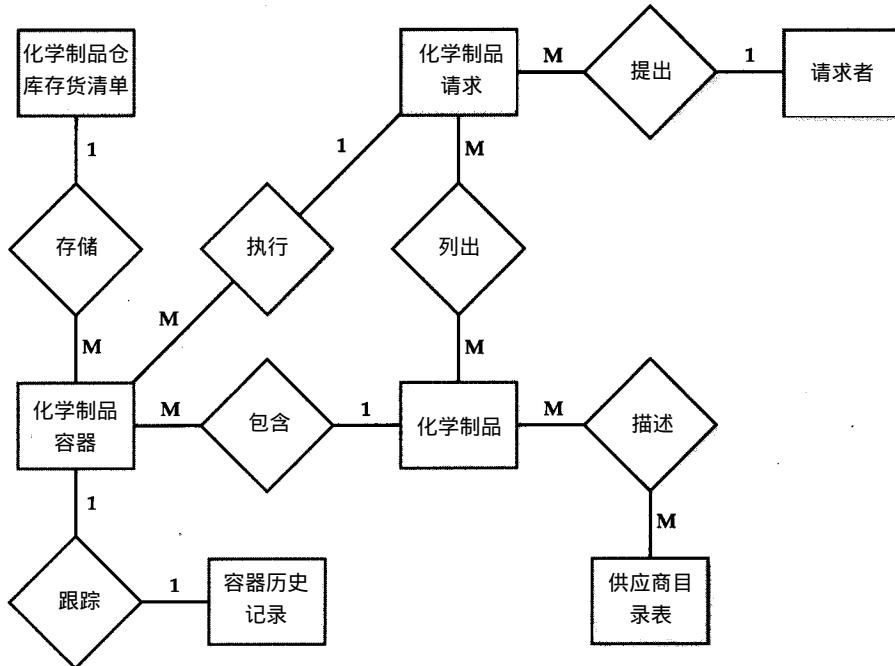


图10-2 “化学制品跟踪系统”的实体联系图

实体 (entity) 是物理数据项 (包括人) 或者数据项的集合，这对所分析的业务或所要构造的系统是很重要的 (Robertson and Robertson 1994)。实体用单数名词命名并在矩形框中表示。图 10-2 描绘了“化学制品跟踪系统”实体联系图的一部分。注意：被命名为化学制品请求、供应商目录表和化学制品仓库存货清单的实体在图 10-1 的数据流图中是作为数据存储出现的。其它一些实体代表与系统交互的操作员 (请求者)，业务运转中一部分物理项 (化学制品容器)、数据块等，并不出现在 0 层数据流图中，但将出现在一个更低层的数据流图中 (容器的历史记录，化学制品)。

每一实体用几个属性 (attribute) 来描述；每一实体的单个实例将具有不同的属性值。例如每一个化学制品的属性包括一个唯一的化学制品标识号、它的正式化学制品名称和它的化学结构的图形表示。数据字典中包含这些属性的详细定义，保证了实体联系图中的实体和在数据流图中相应的数据存储定义一致。

实体联系图中的菱形框代表关系 (relationship)，它确定了实体对之间逻辑上和数量上的联系。关系按照关联的属性来命名。例如，请求者和化学制品请求之间的关系被称为“提出”(placing)，因为一个请求者提出一个化学制品的请求，或者，一个化学制品请求被一个请求者提出。

在实体和关系的连线上用一个数字或字母表示实体的单联系和多联系。不同的实体联系图符号用不同的规则表示联系；本例子描述了一个普遍的方法。因为每一个请求者可以提出多个请求，所以在请求者和化学制品请求之间存在一对多的联系。单联系在请求者和“提出”

关系的连线上标明“1”，多联系在化学制品请求和“提出”关系的连线上标明“M”(代表多)。其它可能的联系如下所示：

- 一对一（每一个容器历史记录跟踪一个化学制品容器）。
- 多对多（每一个供应商目录中描述了许多种化学制品，并且每种化学制品可以被多个供应商目录表所描述）。

如果你知道存在精确的关系，而不仅仅是“很多”关系，就可以用特定的数字或一个数字范围来表示，而不是用一般的“M”表示。

在需求分析时绘制实体联系图是一种组织你所知道的业务或新系统的重要数据元素的好方法。当我在一个业务过程再建工程小组担任信息技术代表时，我就使用过这种技术。当开发组提出一个新的过程流时，我总是要问负责每一个新过程步骤的开发组成员：执行这些步骤需要什么数据项。我还问他们由这些步骤所创建的哪些数据项要存储起来以备后用。

在与这些过程步骤的拥有者商讨之后，我们定位调整所有步骤所需要的数据和所产生的数据。利用这种关联关系可以发现不能在系统中产生的所需要的数据项，还可以发现只被存储而未使用过的数据。最终，我用一个实体联系图和一个数据字典来记录这些数据关系，这可以为新的业务过程提供一个数据组成的概念性框架。这些分析工具可以增强我们对问题的理解，即使我们从未构造过软件系统来支持新的业务过程，这也是值得的。

10.5 状态转换图

实时系统和过程控制应用程序可以在任何给定的时间内以有限的状态存在。当满足所定义的标准时，状态就会发生改变，例如在特定条件下，接收到一个特定的输入激励。这样的系统是有限状态机的例子。此外，许多业务对象（如销售订单、发票，或存货清单项）的信息系统处理是贯穿着复杂生存周期的；此生存周期也可以看成有限状态机。大多数软件系统需要一些状态建模或分析，就像大多数系统涉及到转换过程、数据实体和业务对象。

用自然语言描述一个复杂的有限状态机可能会忽略一个允许的状态改变或者引起一个不允许的改变。与状态机的行为有关的需求可能将多次出现在软件需求规格说明中，它取决于软件需求规格说明是如何组织的。这对综合理解系统行为造成困难。

状态转换图(state-transition diagram, STD)为有限状态机提供了一个简洁、完整、无二义性的表示(Davis 1993)。

状态转换图包括如下面三种元素：

- 用矩形框表示可能的系统状态。
- 用箭头连接一对矩形框表示允许的状态改变或转换。
- 用每个转换箭头上的文本标签表示引起每个转换的条件。标签经常既指示条件也指示相应的系统输出。

状态转换图没有表示出系统所执行的处理，只表示了处理结果可能的状态转换。对于软件系统中只能存在于特定状态的那一部分，你可以使用状态转换图来建模。特定状态或者指诸如汽车巡航控制系统等实时世界实体的行为，或者是这系统操纵的个体项状态。

状态转换图有助于开发者理解系统的预期行为，它对于检查所要求的状态和转换是否已全部正确地写入功能需求中也是一种好方法。测试者可以从覆盖所有转换路径的状态转换图中获得测试用例。用户只要稍微学一些符号就可以读懂状态转换图。

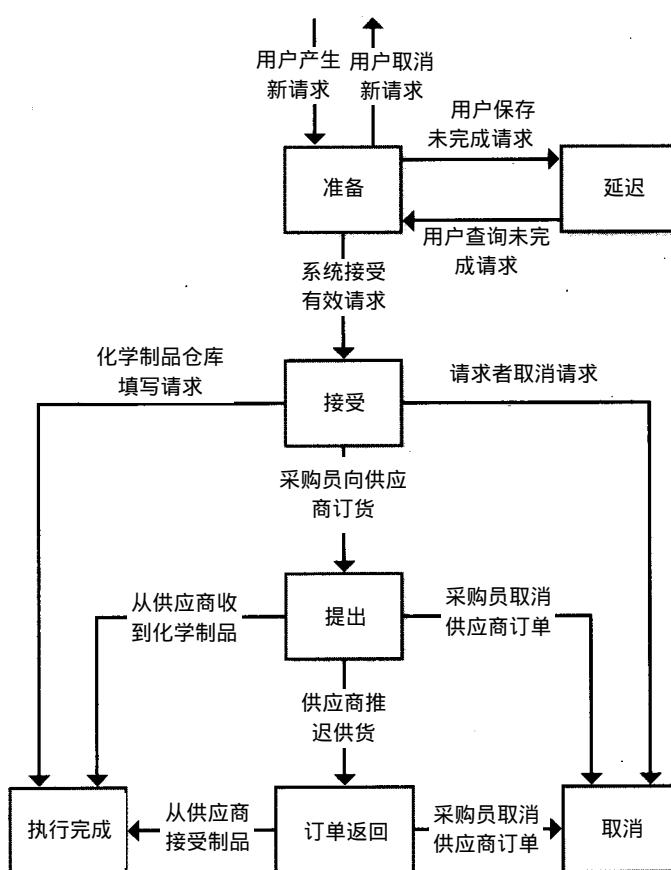


图10-3 “化学制品跟踪系统”中化学制品请求的状态转换图

回想一下，在化学制品跟踪系统中有一个主要功能是允许称为请求者的操作员提出对化学制品的请求，这一请求可以由化学制品仓库中的存货清单来执行完成，也可以通过向外界供应商发出订单来执行完成。每一个请求从创建到完成或取消这一时间段内将经历一系列可能的状态。于是，我们就可以把化学制品请求的生存周期看成一个有限状态机，其建模如图10-3所示。这个状态转换图说明了一个请求可取下列七种可能状态中的一种：

- 准备(in preparation)——请求者正在创建一个新的请求，已经从系统的其它部分进入那个功能。
- 延迟(postponed)——请求者存储他/她的工作以备后用，而并不向系统提交请求或者取消请求。
- 接受(accepted)——用户提交一个完整的化学制品请求，系统判断该请求顺序是否有效，如果有效就接受该请求进行处理。
- 提出(placed)——采购员向供应商订货，并且外部供应商必须满足请求。
- 执行完成(fulfilled)——通过从化学制品仓库交付一个化学制品容器给请求者或者收到供应商的化学制品收据时才能使请求得以满足。
- 订单返回(back-ordered)——供应商没有可用的化学制品，他通知采购员货物已订购以后再交付。

- 取消(canceled)——在请求执行完成之前，请求者取消一个已被接受的请求，或者采购员在订单执行完成或返回订单之前，取消供应商订单。

当“化学制品跟踪系统”的用户代表评审最初的化学制品请求的状态转换图时，他们发现有一个不必要的状态，另外有一个必不可少的状态但分析者却没有记录，还有两个不正确的转换。这些错误在他们评审功能需求时，却没有一个人发现。

状态转换图并没有提供使开发者如何构造系统的详细细节。因此，软件需求规格说明必须包括与处理化学制品请求和它的可能状态变化相关的功能需求。然而，状态转换图对可能的请求状态和它们是如何允许变化提供了一个简洁的可视化表示。

实时系统的状态转换图除了包括一个空闲状态外，与图 10-3 所示的相类似，在这种系统中，当系统不再执行其它处理时就返回空闲状态。相反，对于一个贯穿整个定义的生存期的对象，例如一个化学制品请求，其状态转换图将有一个或多个终结状态；在图 10-3 中则表现为执行完成和取消状态。

10.6 对话图

在许多应用程序中，用户界面可以看作是一个有限状态机。在任何情况下仅有一个对话元素（例如一个菜单，工作区，行提示符或对话框）对用户输入是可用的。在激活的输入区中，用户根据他所采取的活动，可以导航到有限个其它对话元素。在一个复杂的图形用户界面中，可能的导航路径可以有许多种，但其数目是有限的，并且其选择通常是可知的。因此，许多用户界面可以用状态转换图中的一种称为对话图(dialog map)来建模。

对话图代表了一个高层抽象的用户界面体系结构。对话图描绘了系统中的对话元素和它们之间的导航连接，但它没有揭示具体的屏幕设计。对话图可以使你在对需求的理解上探索假设的用户界面概念。用户和开发者可以通过对话图在用户如何利用系统执行任务上达成共同的视觉界面。可视化 Web 站点的构建对话图也很有用，在 Web 站点上，它们有时被称作“站点图”。你在 Web 站点中所建立的导航连接在对话图中则表现为转换。对话图与系统情节叙述相关联，这些叙述还包括对每一个屏幕意图的简短说明。

对话图抓住了用户—系统交互作用和任务流的本质，而不会使你太快陷入到屏幕布局和数据元素的特定细节中。用户可以通过跟踪对话图寻找遗漏、错误或多余的需求。你可以把在需求分析过程中形成的对话图用作详细用户界面设计时的指南，最终形成一个执行的对话图，该对话图记录了产品的真正用户界面的体系结构。

就像在普通的状态转换图中一样，在对话图中，对话元素作为一个状态（矩形框），每一个允许的导航选择作为转换（箭头）。触发用户界面导航的条件用文本标签写在转换箭头上。下面列出几种触发条件的类型：

- 一个用户动作，例如按下一个功能键或点击一个超链接或对话框的按钮。
- 一个数据值，例如一个无效的用户输入触发显示一个错误信息。
- 一个系统条件，例如检测到打印机无纸。
- 这些情况的一些组合，例如输入一个菜单项数字并按下回车键。

为了简化对话图，可以省略全局功能，例如按下 F1 键显示帮助。软件需求规格说明中用户界面部分必须确定这个功能可用，但把它写入对话图中作为交互工具的模型，其意义不大。类似地，在为 Web 站点建模时，你不必包括站点中每一页都出现的标准导航链接。你还可以

省略那些 Web 页导航顺序的反向流转换，因为 Web 浏览器上的退格键 (back button) 可以处理这个导航。

对话图是表示在使用实例中所描述的操作员和系统交互的好方法。对话图可以把可选过程叙述成普通过程流的分支。我发现在使用实例获取讨论会期间，在白板上绘制对话图是有益的，在这一阶段开发组成员正在探索导向任务完成的操作员动作顺序和系统响应的顺序。

第8章提出了在“化学制品跟踪系统”中称为请求一种化学制品的使用实例。这个使用实例的正常过程包括了请求一种化学制品，并由向外部供应商订货来满足该请求。可选过程将供给来自化学制品仓库存货清单的化学制品容器。提出请求的用户在进行选择之前，需要浏览仓库中可用的化学制品的历史。图 10-4 显示了这个使用实例的对话图。

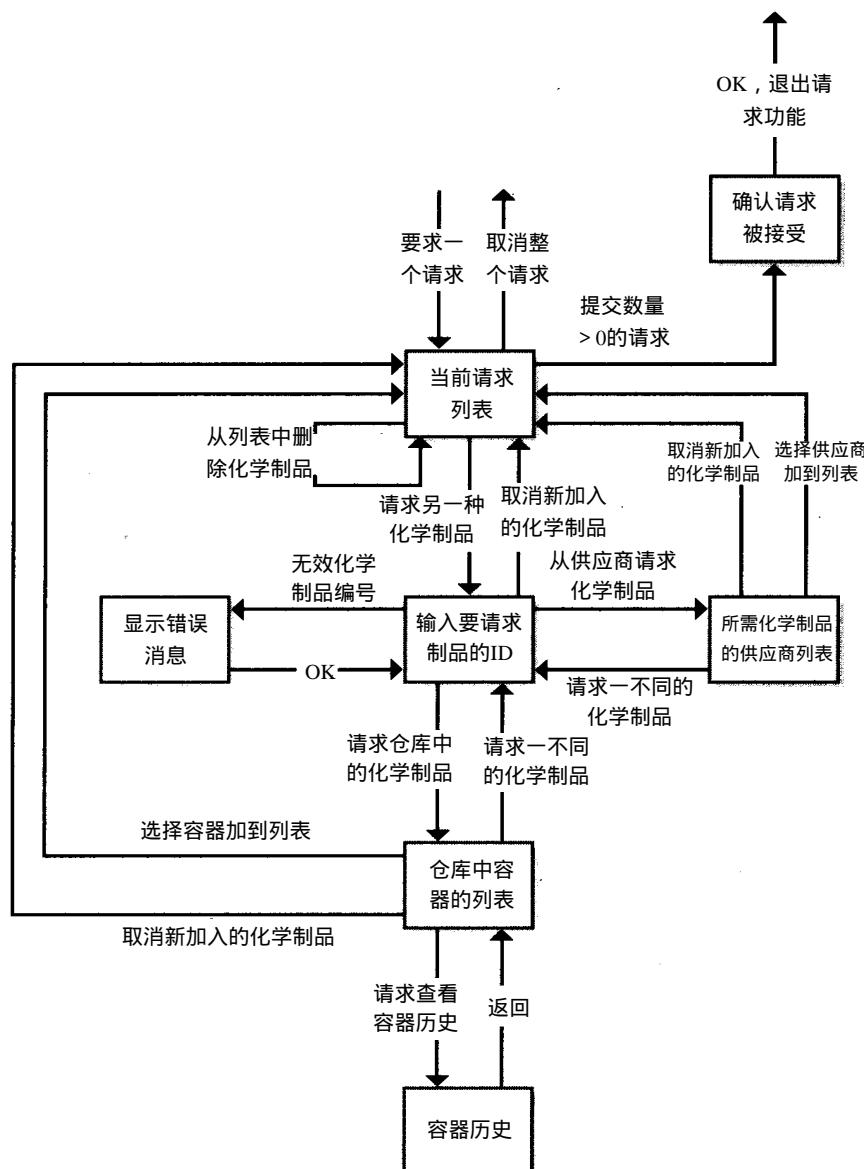


图 10-4 “化学制品跟踪系统”中一种化学制品使用实例的对话图

这种对话图最初可能看起来比较复杂，但如果通过一次一条线和一个框来跟踪，也是不难理解的。请记住，在需求分析阶段，对话图代表了用户和系统在概念级上可能的交互通信作用；但真正的实现可能是有所不同的。在“化学制品跟踪系统”中，用户从菜单中选择“请求一种化学制品”启动使用实例。这个使用实例的主工作区间是在这一请求中的一个化学制品列表，并在一个称为当前请求列表中显示出来。从矩形框出来的箭头显示了所有导航选择，于是用户可从那些联系中获得可用的功能：

- 取消整个请求。
- 如果请求包含了至少一种化学制品，则提交该请求。
- 在请求列表中加入一个新的化学制品。
- 从列表中删除一种化学制品。

注意：最后一个操作，删除一种化学制品，并不涉及其它的对话元素；而仅仅是在用户做出更改之后，刷新当前请求列表。

当你遍历这张对话图时，你将会看到反映其余的“请求一种化学制品”使用实例的元素：

- 向供应商请购化学制品的一条流路径。
- 来自化学制品仓库的执行请求的另一条路径。
- 查看特定化学仓库容器的历史记录的一条可选路径。
- 对处理无效化学制品标识号输入的一条错误信息显示。

对话图中的一些转换允许用户退出操作。对于查看是否忽略任何可用性需求对话图是一种很好的方法，比如向导中遗漏了一个退格键，此时就会强迫用户完成一个不需要的动作。

当用户检查这个对话图时，他可能发现一个遗漏需求。例如，一个谨慎的客户可能认为确认取消整个请求的操作是个好主意，因它可以预防不小心丢失数据。在分析阶段，增加这一新功能只需要极少的代价，但是在已发布的产品中增加这一功能，则代价甚大。由于对话图仅表示了在用户和系统交互中包含的可能元素的概念视图，所以不要试图在需求阶段去攻克所有用户界面的设计细节。而是利用这些模型使项目的风险承担者在系统预期的功能上达成共识。

10.7 类图

面向对象的软件开发优于结构化分析和设计，并且它运用于许多项目的设计中，从而产生了面向对象分析、设计和编程的域。在业务或问题域中，对象 (object)通常与现实世界中的项相类似。对象代表了从称为类的普通模板获得的单个实例。类描述包含了属性（数据）和在属性上执行的操作。类图 (class diagram)是用图形方式叙述面向对象分析所确定的类以及它们之间的关系。

利用面向对象方法开发的产品并不需要特殊的需求开发方法。这是因为需求开发强调用户需要系统做什么以及系统所应包含的功能，而并不关心系统如何做。用户并不关心你如何构造系统，也不关心对象和类。然而，如果你要用面向对象的技术来构造系统，这将有助于你在需求分析阶段确定类和它们的属性及行为。当你考虑如何将问题域对象映射到系统对象，并进一步细化每个类的属性和操作时，面向对象技术可以方便需求开发到设计阶段的转换。

许多不同的面向对象的方法和符号几年来已取得很大的进展。最近，这些方法及符号中的许多部分都将它纳入“统一建模语言”(UML)中(Booth, Rumbaugh and Jacobson 1999)。

在适合于需求分析的抽象层上，你可以像图 10-5 所示那样，用统一建模语言的符号为化学制品跟踪系统的一部分（你所假设的）绘制类图。你可以容易地把这些不包含实现细节的概念性类图详叙成在面向对象设计和实现中所需的更详细的类图。使用顺序图（sequence diagram）和联系图（collaboration diagram）可以表示类之间的交互以及它们所交换的信息，本书未对它作深入的研究（见 Booch, Rumbaugh 和 Jacobson 1999）。

图10-5显示了四个类，每个类都放在一个大的矩形框中：请求者、供应商目录表、化学制品请求和请求行项目。这个类图和其它分析模型所表示的信息有相似之处。出现在图 10-2 实体联系图中的请求者，代表了可以由化学家或化学制品仓库用户类扮演的操作员角色。图 10-2 数据流图也表示这两个用户类可以提出对化学制品的请求。不要把用户类和对象类相混淆，虽然它们名字相似，但并不存在特定的联系。

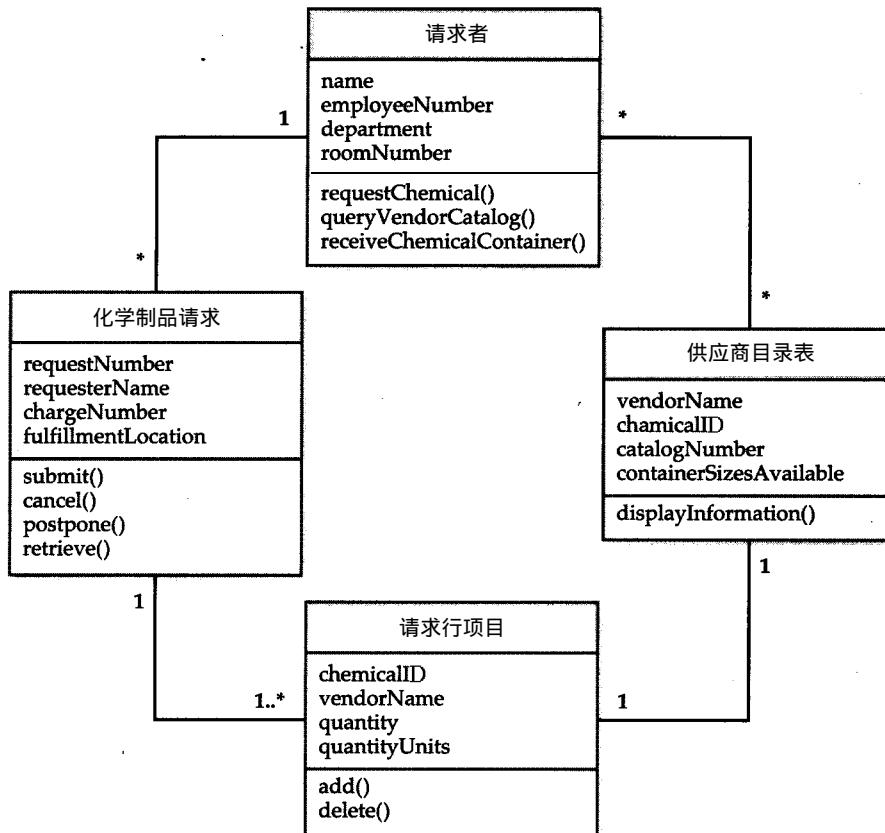


图10-5 化学制品跟踪系统中的部分类图

与请求者类相关的属性列在类方框的中部：姓名，雇员号，部门和房间号（大写是统一建模语言的一个规则）。这些数据项是与请求者类的每一对象相关的。在数据流图中，类似的属性出现在存储定义中。

请求者类的对象可以执行的操作列在类方框的底部，并且后面都跟着空括号。在表示设计的类图中，这些操作将和类的函数或方法相对应，分析类模型仅能看出：请求者可以请求化学制品，查询供应商目录和得到化学制品容器。在类图中所列出的操作与出现在底层数据流图的圆圈中的过程是相对应的。

在图10-5中连接类方框的连线代表了类之间的联系。在线上所示的数字表示了联系的复杂度，就像在数据流图中，连线上的数字代表了实体之间联系的复杂度一样。在图 10-5中，星号代表了在请求者和一个化学制品请求之间一对多的关系：一个请求者可以提出多个请求，但每个请求只属于一个请求者。

10.8 最后的提醒

本章所述的每一种建模都有其优点和局限性。牢记，你创建的分析模型可以提供对需求理解和通信的一个等级，而这却是文本方式下的软件需求规格说明和其它单一的表示法所不能提供的。应该避免陷入在软件开发方法和模型中发生的教条的思维模式和派系斗争。

下一步：

- 使用一个编写完整的软件设计文档来实践本章所描述的建模技术。
- 找出软件需求规格说明中读者难于理解的部分或者发现缺陷的部分。选择一个本章所描述的适于表示部分需求的分析模型。绘制模型并评估如果你在需求开发的早期阶段创建它，是否对你有帮助。
- 接着，需要把一些需求编写成文档，选择一种可以补充文本模型的建模技术。在纸上勾画模型一两次，以确信你的方法是正确的，然后使用支持你所使用的建模符号的商业CASE工具。逐渐地，把对这部分的不断建模融合进你的标准需求开发过程。

第11章 软件的质量属性

许多年前，我参加了一项工程，在该项目中用新的应用程序替换许多已有的主机（main frame）应用程序。根据用户的要求，开发组设计了一个基于窗口的用户界面并定义了新的数据文件，其容量是旧文件的两倍。虽然新系统满足了技术上的规范，但并没有达到客户可接受的程度。用户总是抱怨用户界面运行缓慢，并且新的数据文件所占用的磁盘空间太大。

用户没有陈述对新产品的一些特性的期望，这就不能在他们所提出的功能需求中体现出来。糟糕的是，开发者和用户没有详细地讨论新技术方法所牵涉到可能的性能，从而导致了用户期望与产品实际性能之间的期望差异。比起仅仅满足客户所要求的功能，软件的成功似乎更为重要。

11.1 非功能需求

用户总是强调确定他们的功能、行为或需求——软件让他们做的事情。除此之外，用户对产品如何良好地运转抱有许多期望。这些特性包括：产品的易用程度如何，执行速度如何，可靠性如何，当发生异常情况时，系统如何处理。这些被称为软件质量属性（或质量因素）的特性是系统非功能（也叫非行为）部分的需求。

质量属性是很难定义的，并且他们经常造成开发者设计的产品和客户满意的产品之间的差异。就像 Robert Charette(1990)指出的那样：“真正的现实系统中，在决定系统的成功或失败的因素中，满足非功能需求往往比满足功能需求更为重要”。优秀的软件产品反映了这些竞争性质量特性的优化平衡。如果你在需求的获取阶段不去探索客户对质量的期望，那么产品满足了他们的要求，这只能说你很幸运。但更多的可能是客户失望和开发者沮丧。

虽然，在需求获取阶段客户所提出的信息中包含提供了一些关于重要质量特性的线索，但客户通常不能主动提出他们的非功能期望。用户说软件必须“健壮”，“可靠”或“高效”时，这是很技巧地指出他们所想要的东西。从多方面考虑，质量必须由客户和那些构造测试和维护软件的人员来定义。探索用户隐含期望的问题可以导致对质量目标的描述，并且制定可以帮助开发者创建完美产品的标准。

11.2 质量属性

虽然有许多产品特性可以称为质量属性（Quality Attribute），但是在许多系统中需要认真考虑的仅是其中的一小部分。如果开发者知道哪些特性对项目的成功至关重要，那么他们就能选择软件工程方法来达到特定的质量目标（Glass 1992; DeGrace and Stahl 1993）。根据不同的设计可以把质量属性分类（Boehm 1976; DeGrace and Stahl 1993）。一种属性分类的方法是把在运行时可识别的特性与那些不可识别的特性区分开（Bass, Clements and Kazman 1998）。另一种方法是把对用户很重要的可见特性与对开发者和维护者很重要的不可见特性区分开。那些对开发者具有重要意义的属性使产品易于更改、验证，并易于移植到新的平台上，从而可以间接地满足客户的需要。

在表11-1中，分两类来描述每个项目都要考虑的质量属性；还有其它许多属性（Charette 1990）。一些属性对于嵌入式系统是很重要的（高效性和可靠性），而其它的属性则用于主机应用程序（有效性和可维护性）或桌面系统（互操作性和可用性）。在一个理想的范围内，每一个系统总是最大限度地展示所有这些属性的可能价值。系统将随时可用，决不会崩溃，可立即提供结果，并且易于使用。因为理想环境是不可得到的，因此，你必须知道表 11-1中那些属性的子集对项目的成功至关重要。然后，根据这些基本属性来定义用户和开发者的目 标，从而产品的设计者可以作出合适的选择。

表11-1 软件质量属性

对用户最重要的属性	对开发者最重要的属性
有效性 (availability)	可维护性 (maintainability)
高效性 (efficiency)	可移植性 (portability)
灵活性 (flexibility)	可重用性 (reusability)
完整性 (integrity)	可测试性 (testability)
互操作性 (interoperability)	
可靠性 (reliability)	
健壮性 (robustness)	
可用性 (usability)	

产品的不同部分与所期望的质量特性有着不同的组合。高效性可能对某些部分是很重要的，而可用性对其他部分则很重要。把应用于整个产品的质量特性与特定某些部分、某些用户类或特殊使用环境的质量属性要区分开。把任何全局属性需求记录到在第 9 章所示的软件需求规格说明的第 e.4 部分中，并把特定的目标和列在软件需求规格说明第 d 部分的特性、使用实例或功能需求相联系起来。

11.3 定义质量属性

你必须根据用户对系统的期望来确定质量属性。定量地确定重要属性提供了对用户期望的清晰理解，这将有助于设计者提出最合理的解决方案（Gilb 1988）。然而，大多数用户并不知道如何回答诸如“互操作性对你的重要性如何？”或者“软件应该具有怎样的可靠性？”等问题。在一个项目中，分析员想出了对于不同的用户类可能很重要的属性，并根据每一个属性设计出许多问题。他们利用这些问题询问每一个用户类的代表，可以把每个属性分成一级（不必多加考虑的属性）到五级（极其重要的属性）。这些问题的回答有助于分析员决定哪些质量特性用作设计标准是最重要的。

然后，分析员与用户一起为每一属性确定特定的、可测量的和可验证的需求（Robertson and Robertson 1997）。如果质量目标不可验证，那么就说不清你是否达到这些目标。在合适的地方为每一个属性或目标指定级别或测量单位，以及最大和最小值。如果你不能定量地确定某些对你的项目很重要的属性，那么至少应该确定其优先级。IEEE关于软件质量度量方法的标准提出了一个在综合质量度量基准体系下定义软件质量需求的方法（IEEE 1992）。

另一个定义属性的方法是确定任何与质量期望相冲突的系统行为（Voas 1999）。通过定义不悦人意行为——一种反向需求——你可以设计出强制系统表现出那些行为的测试用例。如果你不能强制系统，那么你可能达到了你的属性目标。这种方法最适用于要求安全性能很高的

应用程序，在这些应用程序中，系统的差错可能会导致生命危险。

这一节的剩余部分将简要地介绍列在表 11-1中的每一个质量属性，并提出一些有助于用户陈述他们对质量属性期望的问题。虽然这些例子有些简单，但提供了由“化学制品跟踪系统”或其他项目总结出的样本目标的陈述。你将需要选择最好的方法来表达每一个质量属性需求，这样可以指导开发者进行设计选择。

11.3.1 对用户重要的属性

1) 有效性 有效性指的是在预定的启动时间中，系统真正可用并且完全运行时间所占的百分比。更正式地说，有效性等于系统的平均故障时间（MTTF）除以平均故障时间与故障修复时间之和。有些任务比起其它任务具有更严格的时间要求，此时，当用户要执行一个任务但系统在那一时刻不可用时，用户会感到很沮丧。询问用户需要多高的有效性，并且是否在任何时间，对满足业务或安全目标有效性都是必须的。一个有效性需求可能这样说明：“工作日期间，在当地时间早上 6点到午夜，系统的有效性至少达到 99.5%，在下午 4点到6点，系统的有效性至少可达到 99.95%。”

2) 效率 效率是用来衡量系统如何优化处理器、磁盘空间或通信带宽的（Davis 1993）。如果系统用完了所有可用的资源，那么用户遇到的将是性能的下降，这是效率降低的一个表现。拙劣的系统性能可激怒等待数据库查询结果的用户，或者可能对系统安全性造成威胁，就像一个实时处理系统超负荷一样。为了在不可预料的条件下允许安全缓冲，你可以这样定义：“在预计的高峰负载条件下，10% 处理器能力和15% 系统可用内存必须留出备用。”在定义性能、能力和效率目标时，考虑硬件的最小配置是很重要的。

3) 灵活性 就像我们所知道的可扩充性、增加性、可延伸性和可扩展性一样，灵活性表明了在产品中增加新功能时所需工作量的大小。如果开发者预料到系统的扩展性，那么他们可以选择合适的方法来最大限度地增大系统的灵活性。灵活性对于通过一系列连续的发行版本，并采用渐增型和重复型方式开发的产品是很重要的。在我曾经参与的一个图形工程中，灵活性目标是如下设定的：“一个至少具有 6个月产品支持经验的软件维护程序员可以在一个小时之内为系统添加一个新的可支持硬拷贝的输出设备。”

4) 完整性 完整性（或安全性）主要涉及：防止非法访问系统功能、防止数据丢失、防止病毒入侵并防止私人数据进入系统。完整性对于通过 WWW 执行的软件已成为一个重要的议题。电子商务系统的用户关心的是保护信用卡信息，Web 的浏览者不愿意那些私人信息或他们所访问过的站点记录被非法使用。完整性的需求不能犯任何错误，即数据和访问必须通过特定的方法完全保护起来。用明确的术语陈述完整性的需求，如身份验证、用户特权级别、访问约束或者需要保护的精确数据。一个完整性的需求样本可以这样描述：“只有拥有查账员访问特权的用户才可以查看客户交易历史。”

5) 互操作性 互操作性表明了产品与其它系统交换数据和服务的难易程度。为了评估互操作性是否达到要求的程度，你必须知道用户使用其它哪一种应用程序与你的产品相连接，还要知道他们要交换什么数据。“化学制品跟踪系统”的用户习惯于使用一些商业工具绘制化学制品的结构图，所以他们提出如下的互操作性需求：“化学制品跟踪系统应该能够从 ChemiDraw 和 Chem-Struct 工具中导入任何有效化学制品结构图。”

6) 可靠性 可靠性是软件无故障执行一段时间的概率（Musa, Iannino and Okumoto 1987）。

健壮性和有效性有时可看成是可靠性的一部分。衡量软件可靠性的方法包括正确执行操作所占的比例，在发现新缺陷之前系统运行的时间长度和缺陷出现的密度。根据如果发生故障对系统有多大影响和对于最大的可靠性的费用是否合理，来定量地确定可靠性需求。如果软件满足了它的可靠性需求，那么即使该软件还存在缺陷，也可认为达到其可靠性目标。要求高可靠性的系统也是为高可测试性系统设计的。

我的开发组曾经开发过一个用于控制实验室设备的软件，这些设备全天工作并且使用稀有的、昂贵的化学制品。用户要求真正与实验相关的那部分软件要高可靠性，而其它系统功能，例如周期性地记录温度数据，则对可靠性要求不高。对于该系统的一个可靠性需求说明如下：“由于软件失效引起实验失败的概率应不超过 5‰”。

7) 健壮性 健壮性指的是当系统或其组成部分遇到非法输入数据、相关软件或硬件组成部分的缺陷或异常的操作情况时，能继续正确运行功能的程度。健壮的软件可以从发生问题的环境中完好地恢复并且可容忍用户的错误。当从用户那里获取健壮性的目标时，询问系统可能遇到的错误条件并且要了解用户想让系统如何响应。

我曾经主持过一个叫作图形引擎的可重用软件组件的开发，该图形引擎具有描述图形规划的数据文件，并且把这一规划传送到指定的输出设备上（Wiegers 1996b）。许多需要产生规划的应用程序就要请求调用图形引擎。由于在图形引擎中，我们将无法控制这些应用程序的数据，所以此时健壮性就成为必不可少的质量属性。我们的一个健壮性需求是这样说明的：“所有的规划参数都要指定一个缺省值，当输入数据丢失或无效时，就使用缺省值数据。”这个例子反映了对一个“用户”是另一个软件应用程序的产品，其健壮性设计的方法。

8) 可用性 可用性也称为“易用性”和“人类工程”，它所描述的是许多组成“用户友好”的因素。可用性衡量准备输入、操作和理解产品输出所花费的努力。你必须权衡易用性和学习如何操纵产品的简易性。“化学制品跟踪系统”的分析员询问用户这样的问题：“你能快速、简单地请求化学制品并浏览其它信息，这对你有多重要？”和“你请求一种化学制品大概需花多少时间？”对于定义使软件易于使用的许多特性而言，这只是一个简单的起点。对于可用性的讨论可以得出可测量的目标，例如“一个培训过的用户应该可以在平均 3分钟或最多 5分钟时间以内，完成从供应商目录表中请求一种化学制品的操作。”

同样，调查新系统是否一定要与任何用户界面标准或常规的相符合，或者其用户界面是否一定要与其它常用系统的用户界面相一致。这里有一个可用性需求的例子：“在文件菜单中的所有功能都必须定义快捷键，该快捷键是由 Ctrl键和其它键组合实现的。出现在 Microsoft Word 2000 中的菜单命令必须与 Word 使用相同的快捷键”。

可用性还包括对于新用户或不常使用产品的用户在学习使用产品时的简易程度。易学程度的目标可以经常定量地测量，例如，“一个新用户用不到 30分钟时间适应环境后，就应该可以对一个化学制品提出请求”，或者“新的操作员在一天的培训学习之后，就应该可以正确执行他们所要求的任务的 95%”。当你定义可用性或可学性的需求时，应考虑到在判断产品是否达到需求而对产品进行测试的费用。

11.3.2 对开发者重要的属性

1) 可维护性 可维护性表明了在软件中纠正一个缺陷或做一次更改的简易程度。可维护性取决于理解软件、更改软件和测试软件的简易程度，可维护性与灵活性密切相关。高可维

护性对于那些经历周期性更改的产品或快速开发的产品很重要。你可以根据修复 (fix) 一个问题所花的平均时间和修复正确的百分比来衡量可维护性。

“化学制品跟踪系统”包括如下的可维护性需求：“在接到来自联邦政府修订的化学制品报告的规定后，对于现有报表的更改操作必须在一周内完成。”在图形引擎工程中，我们知道，我们必须不断更新软件以满足用户日益发展的需要，因此，我们确定了设计标准以增强系统总的可维护性：“函数调用不能超过两层深度”，并且“每一个软件模块中，注释与源代码语句的比例至少为 1 : 2。”认真并精确地描述设计目标，以防止开发者做出与预定目标不相符的愚蠢行为。

2) 可移植性 可移植性是度量把一个软件从一种运行环境转移到另一种运行环境中所花费的工作量。软件可移植的设计方法与软件可重用的设计方法相似 (Glass 1992)。可移植性对于工程的成功是不重要的，对工程的结果也无关紧要。可以移植的目标必须陈述产品中可以移植到其它环境的那一部分，并确定相应的目标环境。于是，开发者就能选择设计和编码方法以适当提高产品的可移植性。

3) 可重用性 从软件开发的长远目标上看，可重用性表明了一个软件组件除了在最初开发的系统中使用之外，还可以在其它应用程序中使用的程度。比起创建一个你打算只在一个应用程序中使用的组件，开发可重用软件的费用会更大些。可重用软件必须标准化、资料齐全、不依赖于特定的应用程序和运行环境，并具有一般性 (DeGrace and Stahl 1993)。确定新系统中哪些元素需要用方便于代码重用的方法设计，或者规定作为项目副产品的可重用性组件库。

4) 可测试性 可测试性指的是测试软件组件或集成产品时查找缺陷的简易程度。如果产品中包含复杂的算法和逻辑，或如果具有复杂的功能性的相互关系，那么对于可测试性的设计就很重要。如果经常更改产品，那么可测试性也是很重要的，因为将经常对产品进行回归测试来判断更改是否破坏了现有的功能。

因为我们知道随着图形引擎功能的不断增强，我们需要对它进行多次测试，所以作出了如下的设计目标：“一个模块的最大循环复杂度不能超过 20。”循环复杂度度量一个模块源代码中逻辑分支数目 (McCabe 1982)。在一个模块中加入过多的分支和循环将使该模块难于测试、理解和维护。如果一些模块的循环复杂度大于 20，这并不会导致整个项目的失败，但指定这样的设计标准有助于开发者达到一个令人满意的质量目标。

11.4 属性的取舍

有时，不可避免地要对一些特定的属性对进行取舍。用户和开发者必须确定哪些属性比其它属性更为重要，并定出优先级。在他们作决策时，要始终遵照那些优先级。图 11-1描述了来自表 11-1 的质量属性之间一些典型的相互关系，当然你也可能会遇到一些例外 (Charette 1990 ; IEEE 1992; Glass 1993)。一个单元格中的加号表明单元格所在行的属性增加了对其所在列的属性的积极影响。例如，增强软件可重用性的设计方法也可以使软件变得灵活、更易于与其它软件组件相连接、更易于维护、更易于移植并且更易于测试。

一个单元格中的减号表明单元格所在行的属性增加了对其所在列的属性的不利影响。高效性对其他许多属性具有消极影响。如果你编写最紧凑，最快的代码，并使用一种特殊的预编译器和操作系统，那么这将不易移植到其它环境，而且还难于维护和改进软件。类似地，一些优化操作者易用性的系统或企图具有灵活性、可用性并且可以与其它软硬件相互操作的

系统将付出性能方面的代价。例如，比起使用具有完整的制定图形代码的旧应用系统，使用外部的通用图形引擎工具生成图形规划将大大降低性能。你必须在性能代价和你所提出的解决方案的预期利益之间作出权衡，以确保作出合理的取舍。

	Availability	Efficiency	Flexibility	Integrity	Interoperability	Maintainability	Portability	Reliability	Reusability	Robustness	Testability	Usability
Availability							+	+				
Efficiency		-		-	-	-	-	-	-	-	-	
Flexibility	-		-		+	+	+			+		
Integrity	-			-				-		-	-	
Interoperability	-	+	-			+						
Maintainability	+	-	+				+			+		
Portability	-	+		+	-			+		+	-	
Reliability	+	-	+		+				+	+	+	
Reusability	-	+	-	+	+	+	-			+		
Robustness	+	-					+				+	
Testability	+	-	+			+	+				+	
Usability	-							+	-			

图11-1 选择的质量属性之间的正负关系

为了达到产品特性的最佳平衡，你必须在需求获取阶段识别、确定相关的质量属性，并且为之确定优先级。当你为项目定义重要的质量属性时，利用图 11-1可以防止发生与目标冲突的行为。以下是一些例子：

- 如果软件要在多平台下运行（可移植性），那么就不要对可用性抱有乐观态度。
- 可重用软件能普遍适用于多种环境中，因此，不能达到特定的容错（可靠性）或完整性目标。
- 对于高安全的系统，很难完全测试其完整性需求；可重用的类组件或与其它应用程序的互操作可能会破坏其安全机制。

在软件中，其自身不能实现质量特性的合理平衡。在需求获取的过程中，加入对质量属性期望的讨论，并把你所了解的写入软件需求规格说明中。这样，才有可能提供使所有项目风险承担者满意的产品。

下一步：

- 从表11-1中确定一些对于你目前项目的用户至关重要的质量属性。系统地陈述每个属性的一两个问题，这将有助于用户说清他们的期望。基于用户的回答，为每一个重要属性写出一两个明确的目标。

第12章 通过原型法减少项目风险

我最近遇到一个软件开发组，他们有一个令人不悦的经历：用户以不适合为理由拒绝了他们开发的整个产品。在产品发布之前，用户并没有见过用户界面，他们发现界面和潜在的需求都存在问题。软件原型是一种技术，你可以利用这种技术减少客户对产品不满意的风险。来自用户的早期反馈可以使开发小组正确理解需求，并知道如何最好地实现这些需求。

即使你完成了在前几章所叙述的需求获取、分析和编写规格说明，你的需求中还有一部分对客户或开发者仍然不明确或不清晰。如果不解决这些问题，那么必然在用户产品视图和开发者对于开发什么产品的理解之间存在期望差距。通过阅读文本需求或研究分析模型，很难想象软件产品在特定的环境中如何运行。原型可以使新产品实在化，为使用实例带来生机，并消除你在需求理解上的差异。比起阅读一份冗长无味的软件需求规格说明，用户通常更愿意尝试建立有趣的原型。

原型有多种含义，并且参与建立原型的人可以有不同的期望。例如，一个飞机原型实际上可以飞翔——它是真实飞机的雏形。相反，一个软件原型通常仅仅是真实系统的一部分或一个模型，并且它可能根本不能完成任何有用的事。本章将研究各种类型的软件原型、它们在需求开发中的应用以及如何使原型成为软件开发过程中有效的组成部分（Wood and Kang 1992）。

12.1 原型是“什么”和“为什么”要原型

一个软件原型是所提出的新产品的部分实现。使用原型有三个主要目的：

- 明确并完善需求 原型作为一种需求工具，它初步实现所理解的系统的一部分。用户对原型的评价可以指出需求中的许多问题，在你开发真正产品之前，可以最低的费用来解决这些问题。
- 探索设计选择方案 原型作为一种设计工具，用它可以探索不同的用户界面技术，使系统达到最佳的可用性，并且可以评价可能的技术方案。
- 发展为最终的产品原型 作为一种构造工具，是产品最初子集的完整功能实现，通过一系列小规模的开发循环，你可以完成整个产品的开发。

建立原型的主要原因是为了解决在产品开发的早期阶段不确定的问题。利用这些不确定性来判断系统中哪一部分需要建立原型和希望从用户对原型的评价中获得什么。对于发现和解决需求中的二义性，原型也是一种很好的方法。二义性和不完整性使开发者对所开发的产品产生困惑，建立一个原型有助于说明和纠正这些不确定性。用户、经理和其他非技术项目风险承担者发现在确定和开发产品时，原型可以使他们的想象更具体化。原型比开发者常用的技术术语更易于理解。

12.2 水平和垂直的原型

当人们谈到“软件原型”时，他们所想到的通常是可能的用户界面的“水平原型”

(horizontal prototype)。水平原型也叫做“行为原型”(behavioral prototype)或“模型”(mock-up)。它可以让你探索预期系统的一些特定行为，并达到细化需求的目的。当用户在考虑原型中所提出的功能可否使他们完成各自的业务任务时，原型使用户所探讨的问题更加具体化。需要注意的是，这种原型中所提出的功能经常并没有真正地实现。

一个水平原型就像一个电影集。它在屏幕上显示出用户界面的正面像，可能允许这些界面之间的一些导航，但是它仅包含少量的功能并没有真正实现所有的功能。想像你心目中的美国西部：牛仔走进酒店，而后从马房中走出来，然而，他并没有喝酒，也没见到一匹马，因为在虚假的建筑物之后什么也不存在。

一个模型(mock-up)展示给用户的是在原型化屏幕上可用的功能和导航选择。有一些导航可能起作用，但是用户可能仅看到描述在那一点将真正显示的内容的信息。数据库查询所响应的信息是假的或者只是一个固定不变的信息，并且报表内容也是固定不变的。虽然原型看起来似乎可以执行一些有意义的工作，但其实不然。这种模拟足以使用户判断是否有遗漏、错误或不必要的功能。原型代表了开发者对于如何实现一个特定的使用实例的一种观念。用户对原型的评价可以指出使用实例的可选过程，遗漏的过程步骤，或原先没有发现的异常情况。

当你在相当抽象的级别上建立原型时，用户可以把注意力集中在需求和工作流问题上，而不会被精细的外形或屏幕上元素的位置所干扰(Constantine 1998)。在澄清了需求并确定了界面中的框架之后，你可以建立更详细的原型来探索用户界面的设计。还可以使用不同的屏幕设计工具或甚至使用纸和铅笔来建立水平原型，这将在以后讨论。

垂直原型(vertical prototype)，也叫做结构化原型或概念的证明，实现了一部分应用功能。当你不能确信所提出的构造软件的方法是否完善或者当你需要优化算法，评价一个数据库的图表或测试临界时间需求时，你就要开发一个垂直原型。垂直原型通常用在生产运行环境中的生产工具构造，使其结果一目了然(更有意义)。比起在软件的需求开发阶段，垂直原型更常用于软件的设计阶段以减少风险。

我曾经参与一项需要实现一个特殊的客户/服务器体系结构的项目，并作为从以主机为中心的环境到基于网络化的 Unix 服务器和工作站的应用环境的转换策略的一部分(Thompson and Wiegers 1995)。一个垂直原型只实现客户一部分用户界面和相应的服务器功能，这可以使我们评估所提出体系结构的通信组件、性能和可靠性。实验是成功的，我们基于那一体系结构的实施也是成功的。

12.3 抛弃型原型或进化型原型

在构造一个原型以前，需要充分与客户交流，并作出一个明确的判断，在评价原型以后，是抛弃掉原型还是把该原型进化为最终产品的一部分。你可以建立一个抛弃型原型(throwaway prototype)或探索型原型(exploratory prototype)来回答这些问题，解决不可测性并提高需求质量(Davis 1993)。因为你打算在原型达到预期目的以后将它抛弃，所以可以花最小的代价尽快地建立该原型。如果你在原型上付出的努力越多，那么项目的参与者就越不愿意将它抛弃。

当你建立抛弃型原型时，你忽略了很多具体的软件构造技术。而强调在健壮性、可靠性、性能和长期的可维护原则下迅速实现软件并易于维护。基于这一原因，你不能将抛弃型原型

中的代码移植到你的产品系统中，除非它达到产品质量代码的标准，否则，你和用户将在软件生存期中遭遇种种麻烦。

当你遇到需求中的不确定性、二义性、不完整性或含糊性时，最合适的方法是建立抛弃式模型。你需要解决这些问题以减少在继续开发时存在的风险。原型可帮助用户和开发者想象如何实现需求和可以发现需求中的漏洞。它还可以使用户判断出这些需求是否可以完成必要的业务过程。

图12-1描述了在抛弃型原型的帮助下，从用户任务到详细用户界面设计的开发活动序列。每一个使用实例的描述包括了一系列操作和系统响应，这些可以用对话图来建立模型以描述一种可能的用户界面机制（见第10章）。抛弃型原型把对话元素细化为特定的屏幕显示、菜单和对话框。当用户评价原型时，他们的反馈可能会引起使用实例描述的改变（例如发现一个新的可选过程时）并且也会引起相对对话图的改变。一旦确定了需求并勾画出屏幕的大体布局，你就可以从最佳使用的角度设计每一个用户界面元素的细节。比起直接从使用实例的描述跳跃到完整的用户界面的实现，然后在需求中发现重大错误，利用逐步求精的方法所花费的努力将会更小。

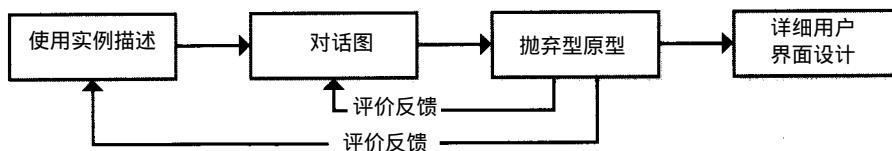


图12-1 利用抛弃型原型从用户任务到界面设计的活动序列

与抛弃型原型相对应的是进化型原型 (evolutionary prototype)，在已经清楚地定义了需求的情况下，进化型原型为开发渐增式产品提供了坚实的构造基础。进化型原型是螺旋式软件开发生命周期模型的一部分 (Boehm 1988)，也是一些面向对象软件开发过程的一部分 (Kruchten 1996)。与抛弃型原型的快速、粗略的特点相比，进化型模型一开始就必须具有健壮性和产品质量级的代码。因此，对于描述相同的功能，建立进化型原型比建立抛弃型原型所花的时间要多。一个进化型原型必须设计为易于升级和优化的，因此，你必须重视软件系统性和完整性的设计原则。要达到进化型原型的质量要求并没有捷径。

我们应该考虑进化型原型的第一次演变，因为它将作为实现需求中易于理解和稳定部分的试验性版本。从测试和首次使用中获得的信息将引起下一次软件原型的更新，正是这样不断增长并更新，使软件才能从一系列进化型原型发展为实现最终完整的产品。这种原型提供了可以使用户快速获得有用功能的方法。

演化式模型适用于Web开发项目。在我曾经主持的一个Web项目中，我们根据从使用实例分析中得出的需求，建立了四个原型序列。许多用户对每一个原型进行了评价，根据他们对我们提出的问题的回答，我们对原型进行了修正。对第四个原型修改之后，产生了我们的Web站点。

图12-2描述了在软件开发过程中，可以综合使用多种原型的许多方法。例如，可以利用从一系列抛弃型原型中获得的知识来精化需求，然后，通过一个进化型原型序列，可以渐增式地实现需求。贯穿图12-2的一条可选路径在最终设计用户界面之前，将使用抛弃式水平原型澄清需求，而与之对应的垂直原型则使核心应用程序算法有效。你所不能实现的是：把一

一个抛弃型原型固有的低劣性转化为产品系统所要求的可维护性和健壮性。表 12-1 总结了抛弃式、演化式、水平和垂直原型的一些典型应用。

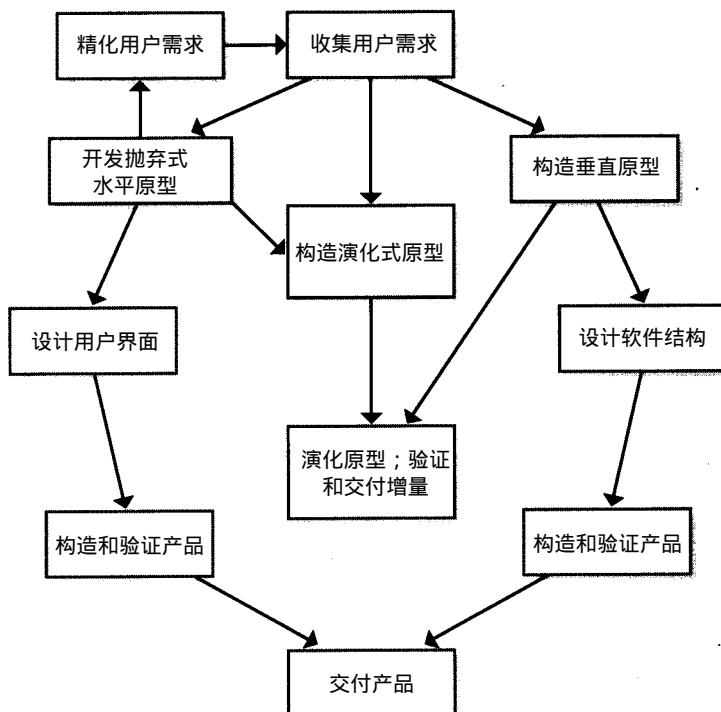


图12-2 在软件开发过程中使用原型法的一些可能的方法

表12-1 软件原型的典型应用

	抛 弃 型	演 化 型
水平	<ul style="list-style-type: none"> 澄清并精化使用实例和功能需求 查明遗漏的功能 探索用户界面方法 	<ul style="list-style-type: none"> 实现核心的使用实例 根据优先级，实现附加的使用实例 开发并精化 Web 站点
垂直	<ul style="list-style-type: none"> 证明技术的可行性 	<ul style="list-style-type: none"> 实现并发展核心的客户/服务器功能层和通信层 实现并优化核心算法

12.4 书面原型和电子原型

在许多情况下，一个可执行的原型未必可以获取所需的用于解决关于需求不确定性的信息。书面原型 (paper prototype) (有时也叫低保真原型) 是一种廉价、快速并且不涉及高技术的方法，它可以把一个系统某部分是如何实现的呈现在用户面前 (Rettig 1994; Hohmann 1997)。书面原型有助于判断用户和开发者在需求上是否达成共识。他们可以使你在开发产品代码前，对各种可能的解决方案进行试验性的并且低风险的尝试。

书面原型所包括的工具仅仅是纸张、索引卡、粘贴纸、塑料板、白板和标记器。你可以对屏幕布局进行构思，而并不必关心那些按钮和小装饰物应该出现在什么位置上。用户愿意提供反馈，这将引起多页书面原型极充分的改变。有时，他们并不急于评论一个基于计算机的可爱的原型，因为该原型凝结了开发者的许多辛劳。开发者也是经常不愿意对精心制作的

电子原型（ electronic prototype ）做重大更改。

有了“低保真”原型，当用户评价原型时，一个人可以充当计算机的角色。用户最初的动作是高呼他想在特定的屏幕上做什么：“我需要在 File 菜单中选择 Print Preview 选项。”当用户进行这一“操作”时，模仿计算机的人就会把关于显示方面的纸张和索引卡给用户看。用户就可以判断这些界面是否是所期望的响应，并且还可以判断所显示的项是否正确。如果有错误，只要用一张新纸或索引卡，重画一张就可以了。

不管你建立原型的工具多么高效，在纸张上勾画界面是最快的。书面原型方便了原型的快速反复性，而在需求开发中反复性是一个关键的成功因素。在运用自动化工具建立详细用户界面原型，构造一个进化型原型或者从事传统设计和构造活动之前，书面原型对于精化需求是一种优秀的技术，它还提供了一个管理客户期望的有用工具。

如果你决定建立一个电子抛弃型原型，那么就有许多工具可以使用（ Andriole 1996 ）。这些工具包括：

- 编程语言，例如： Microsoft Visual Basic, IBM VisualAge Smalltalk 和 Inprise Delphi.
- 脚本语言，例如： Perl, Python 和 Rexx
- 商品化的建立原型的工具包、屏幕绘图器和图形用户界面构筑师。

基于 Web 使用可以快速修改的 HTML 页（超文本标注语言），它对于建立澄清需求而不去探索特定的用户界面设计的原型是很有用的。合适的工具可以使你迅速地实现并更改用户界面组件，而不管在界面后面的代码效率的高低。当然，如果你正在建立一个演化式模型，那么你必须一开始就使用产品开发工具。

12.5 原型评价

通过建立脚本使用户遵从一系列步骤并且回答一些特定的问题以获取所需要的信息，这样你可以提高原型评价的有效性。这些活动是对一般的询问“告诉我，你对这个原型的看法如何”的有价值的补充。你可以从使用实例和原型描述的功能中获得评价脚本。这一脚本可以让用户执行特定的任务并且指导他们评价你觉得最不确定的原型部分。在每个任务之后，脚本将为评价者提供特定的与任务有关的问题。此外，你可以询问以下一般性的问题：

- 这个原型所实现的功能与你所期望的一致吗？
- 有遗漏的功能吗？
- 你能考虑一下这个原型所没涉及的一些出错情况吗？
- 有多余的功能吗？
- 这些导航对于你意味着怎样的逻辑性和完整性？
- 有更简单的方法来完成这一任务吗？

务必让一些合适的人从恰当的角度评价原型。原型的评价者必须是所期望的用户群的代表。评价组必须从使用原型中功能的用户类里挑选出具有经验和经验不足的用户。在把原型呈递给评价者时，应注意原型不包括要在以后真正产品开发中实现的所有的业务逻辑。

比起只是简单地让用户自己评价原型然后把他们的想法告诉你，亲自观察用户使用原型将获得更多信息，用户界面原型可用性的正式测试是很庞大的，但是你可以通过观察获得很多信息。要注意用户所指出那些原型部分。善于发现与原型的方法相冲突的用户所习惯的应用程序的操作规范。寻找那些有疑惑的用户，他们不知道该如何做并且并不知道如何才能达

到满意的程度。当用户在评价原型时，让用户尽量把自己的想法大胆地说出来，这样你才能理解他们想什么，并且能够发现原型表示的不合理的需求部分。努力创造一个公平的环境，这样评价者可以畅所欲言，表达他们的想法和所关心的事物。在用户评价原型时，你要避免诱导用户用设计好的特定方法执行一些功能。

把从原型评价中获得的信息编写成文档。对于一个水平原型，用所收集的信息精化软件需求规格说明中的需求。如果原型评价得出一些用户界面设计的决策或者特定交互技术的选择，那么把这些结论和你如何实现都记录下来。没有用户想法参与的决策，就必须不断地回溯，将造成不必要的时间浪费。对于一个垂直原型，记录好所实施的评价以及评价结果，从而做出关于所探索的不同技术方法可行性的决策。

12.6 原型法的最大风险

原型法是一种减少软件项目失败风险的技术。然而，原型法又引入了自身的风险。最大的风险是用户或者经理看到一个正在运行的原型从而以为产品即将完成。“哦，这看起来好像差不多了！”充满热情的原型评价者说：“这看起来真的很好，你能把它完成后交给我吗？”

一句话：不行！如果你正在演示或评价一个抛弃型原型，无论它与真正的产品是如何相像，它决不会达到产品的使用程度。它仅是一个模型，一种模拟或一次实验。处理风险承担者的期望是成功原型法的一个关键因素，因此要保证那些见到原型的人理解为什么要建立原型并且怎样建立原型。决不能把抛弃型原型当作可交付的产品。交付原型可导致项目的延期完成，因为那些设计和编码并没有考虑到软件质量和容错性。

不要因为害怕提交不成熟产品的压力而阻碍你建立原型，但是你必须让见到原型的人明白你不会交付原型，甚至不会将它称之为软件。控制这种风险的一种方法是利用书面原型而不是电子原型。评价书面原型的人决不会误认为产品已经完成开发并可以交付了。另一种可能的方法是使用不同于在真正开发时所用的原型法工具，这将有助于你抵抗“已完成”原型开发并可把它当作产品交付的压力。

在原型评价期间，继续处理那些期望。如果评价者看到原型可以对一个模拟的数据库查询响应甚快，那么他们可能期望在最终的软件产品中也具有同样惊人的性能。在对最终产品的行为进行模拟时，要考虑现实中的时间延迟（这可以使原型不易被看作可即将交付的产品）。

12.7 原型法成功的因素

软件原型法提供了一套强有力的技术，它可以缩短开发进度，增加用户的满意程度，生产出高质量的产品并且可以减少需求错误和用户界面的缺陷。为了帮助你在需求开发过程中建立有效的原型，请遵循如下原则：

- 你的项目计划中应包括原型风险。安排好开发、评价和可能的修改原型的时间。
- 计划开发多个原型，因为你很少能一次成功。
- 尽快并且廉价地建立抛弃型原型。用最少的投资开发那些用于回答问题和解决需求的不确定性的原型。不要努力去完善一个抛弃型原型的用户界面。
- 在抛弃型原型中不应含有代码注释、输入数据有效性检查、保护性编码技术，或者错误处理的代码。

- 对于已经理解的需求不要建立原型。
- 不能随意地增加功能。当一个简单的抛弃型原型达到原型目的时，就不应该随便扩充它的功能。
- 不要从水平原型的性能推测最终产品的性能。原型可能没有运行在最终产品所处的特定环境中，并且你开发原型的工具与开发产品的工具在效率上是存在差异的。
- 在原型屏幕显示和报表中使用合理的模拟数据。那些评价原型的用户会受不现实数据的影响而不能把原型看成真正产品的模型。
- 不要期望原型可以代替需求文档。原型只是暗示了许多后台功能，因此必须把这些功能写入软件需求规格说明，使之完善、详细并且可以有案可稽。

下一步：

- 查明你的项目中引起需求混乱的部分（如使用实例）。用书面原型勾画出代表你对需求的理解和如何实现的可能用户界面。让一些用户利用你的原型模拟执行一个任务或使用实例。确定对需求的最初理解中不完整和不正确的部分。相应地更改原型并重新检验。
- 给你的原型评价者递交一份本章信息的总结。这将有助于他们理解原型背后的原理，并且使之期望原型法结果的现实。

第13章 设定需求优先级

关于“化学制品跟踪系统”的大部分用户需求编写成文档以后，项目经理 Dave 和需求分析员 Lori 接见了两个产品代表。Tim 代表了药剂师群体，而 Roxanne 则代表了化学制品仓库人员。

“就像你们所知道的那样”，Dave 开始说：“产品代表为化学制品跟踪系统收集了许多需求，但我们不能在产品的首发版中包含你们所需的全部功能。由于大部分需求来自药剂师和化学制品仓库，所以我想与你们谈一谈关于设定需求优先级的问题。”

Tim 感到很困惑。“你为什么要设定需求优先级？它们全都很重要，否则我们不会向你们提出这些需求。”

Dave 解释说：“我知道它们都很重要，但我们不能做到同时交付一个包罗万象并且具有高质量的产品。由于没有更多的可用资源，所以我们需要为下一季度末就要交付的产品确定最重要的需求。我们希望你们帮助我们把首发版中必须包括的需求与可以放入以后版本的需求区分开。”

“我知道卫生和安全办公室已向政府提交的化学制品使用和销毁的报表必须在这个季度末完成”Roxanne 指出，“如果有必要的话，我们可以多使用几个月化学制品仓库现行的存货清单系统。但是条形码标签和扫描功能是必须的，这比药剂师所需的可查找的供应商目录更为重要。”

Tim 提出抗议：“我已向药剂师保证，为他们提供在线的目录查询功能，以节省他们的时间。所以目录查询从项目刚开始就必须考虑”。

分析员 Lori 说：“当我与药剂师共同探讨使用实例时，有一些使用实例似乎经常执行而其它的则很少有人使用。我们可以分析全部的使用实例，并确定那些你们不会马上就用到的使用实例吗？如果我们可以这样做，那么可以推迟决定那些高优先级的使用实例。

对于必须等待系统部分功能的实现，Tim 和 Roxanne 并没有感到很惊讶。然而他们意识到如果开发组不能在发行 1.0 版本时实现全部需求，那么最好每个人都赞成首先实现需求的子集。

每一个具有有限资源的软件项目必须理解所要求的特性、使用实例和功能需求的相对优先级。设定优先级有助于项目经理解决冲突、安排阶段性交付，并且做出必要的取舍。本章将讨论设定需求优先级的重要性，并且提出一个基于价值、费用和风险的设定优先级方案。

13.1 为什么要设定需求的优先级

当客户的期望很高、开发时间短并且资源有限时，你必须尽早确定出所交付的产品应具备的最重要的功能。建立每个功能的相对重要性有助于你规划软件的构造，以最少的费用提供产品的最大功能。如果你正在做时间盒图或者进行渐增式开发，那么设定优先级就特别重要，因为在这些开发中，交付进度安排很紧迫并且不可改变日期，你需要排除或推迟一些不

重要的功能。

一个项目经理必须权衡合理的项目范围和进度安排、预算、人力资源以及质量目标的约束。一个实现这种权衡的方法是：当接受一个新的高优先级的需求或者其它项目环境变化时，删除低优先级的需求，或者把它们推迟到下一版本中去实现。如果客户没有以重要性和紧迫性来区分它们的需求，那么项目经理就必须自己作出决策。由于客户可能不赞成项目经理所设定的优先级，所以客户必须指明哪些需求必须包括在首发版中，而哪些需求可以延期实现。当你有很多选择可以完成一个成功的产品时，应尽早在项目中设定其优先级。

让每一个客户都来决定他们的需求中哪一些是最重要的，这是很难做到的；要在众多具有不同期望的客户之间达成一致意见就更难了。人们心中都存在个人的利益，并且他们并不总能与其它群体的利益相妥协。然而，就像第2章所讨论的那样，在客户——开发者的合作关系中，设定需求优先级是客户的责任之一。

客户和开发者都必须为设定需求的优先级提供信息。客户总是让可以给他们带来最大利益的需求享有最高优先级。然而，一旦开发者指出费用、难度、技术风险，或其它与特定需求相关的权衡时，客户可能会觉得他们最初所想的需求似乎变得不必要了。开发者也可能认为在早期阶段必须先实现那些优先级较低的功能，因为它们会影响系统的体系结构。设定优先级意味着权衡每个需求的业务利益和它的费用，以及它所牵涉到的结构基础和产品的未来评价。

13.2 不同角色的人处理优先级

对客户请求的“膝跳”(knee-jerk)响应设定优先级，“我需要所有的特性，只要以某种方式使它发生即可。”如果用户知道低优先级需求可能不会实现，那么就很难说服用户设定需求的优先级。一个开发者曾经告诉我，优先级是不必要的，因为如果他把需求写入软件需求规格说明中，那么他就会不遗余力地去实现这些需求。然而，这并没有考虑到每个功能何时实现的问题。开发者更喜欢避开设定优先级，因为他们觉得建立优先级与它们要向客户和经理表示的“我们可以全部完成产品”的态度相冲突。

现实中，一些特性比其它特性更重要。项目接近尾声时，在极其简单的“快速开发阶段”，当开发者抛弃掉一些不必要的功能以保证按时完工的时候，这表现得尤为明显。在项目的早期阶段设定优先级有助于你逐步作出相互协调的决策，而不是在最后阶段匆忙决定。在你判断出需求的低优先级之前，如果你已经实现了将近一半的特性，那这将是一种浪费。

如果让客户自己设计，那么他们将会把85%的需求设定为高优先级，10%的需求设定为中等优先级，5%的需求设定为低优先级。这没有给项目经理很多灵活性。通过废除那些不必要的需求并且简化那些不必要的复杂部分，这被视为是快速软件开发的最佳实践(McConnell 1996)。

在一个大的项目中，以管理为指导的开发组对系统分析员设定需求优先级的意见表现出极大的厌烦。经理指出，他们可以不需要一些特殊的特征，但另外的特征需要用于弥补遗漏的需求。如果他们拖延实现太多的需求，那么目标系统将达不到业务计划中所反映的情况。当你评价优先级时，应该看到不同需求之间的内在联系，以及它们与项目业务需求的一致性。

13.3 设定优先级的规模

设定优先级的一般方法是：把需求分成三类。表13-1描述了实现的三层规模的方法。这

些是主观上的并且是不精确的，因此，所涉及到的每个人必须在他们所使用的规模中每一层的含义上达成一致意见。如果人们混淆了高、中、低这样的术语，那么就要更多地使用如提交、允许时间、和将来发行版本等确定的词语。

表13-1 多种设定需求优先级的规模

命 名	意 义	参 考
高	一个关键任务的需求；下一版本所需求的	—
中	支持必要的系统操作；最终所要求的，但如果有必要的话，可以延迟到下一个版本	
低	功能或质量上的增强；如果资源允许的话，实现这些需求总有一天使产品更完美	
基本的 条件的	只有在这些需求上达成一致意见，软件才会被接受 实现这些需求将增强产品的性能，但如果忽略这些需求，产品也是可以被接受的	(IEEE 1998)
可选的	一个功能类，实现或不实现均可	
3	必须完美地实现	(Kovitz 1999)
2	需要付出努力，但不必做得太完美	
1	可以包含缺陷	

每一个需求的优先级必须写入软件需求规格说明或使用实例的说明中。为你的软件需求规格说明建立一个规则，这样读者就可以知道分配给一个高层需求的优先级是否被其所有下层需求所继承，或者每个用户需求是否应该有它自己的优先级属性。

即使是一个中等大小的项目也会有成千上万个功能需求，以至于不能从分析和一致性角度对这些需求进行分类。为了使需求易于管理，你必须为设定优先级选择一个合适的抽象层次——使用实例、特性或详细功能需求。在一个单一的使用实例中，某些特定的可选过程可能比其它过程具有更高的优先级。你可能决定在特性层上进行最初的优先级设定，然后在特定的特性中分别设定功能需求的优先级。这有助于你从可以延期实现的精化需求中识别核心功能。文档则同等对待所有低优先级的需求，因为它们的优先级可能后来还要被改变，并且知道关于这些需求的信息有助于开发者提前规划将来软件的升级版本。

13.4 基于价值、费用和风险的优先级设定

在一个小项目中，风险承担者们可以随意赞成需求的优先级。对于大的、有争议的项目则需要一种更加结构化的方法，采用这种方法可以消除一些情感、政策以及处理过程中的推测。人们提出许多分析上和数学上的技术用于辅助需求优先级的确定，这些方法包括建立每个需求的相对价值和相对费用。优先级最高的需求是那些以最小的费用比例产生出最大产品价值比例的需求 (Karlsson and Ryan 1997; Jung 1998)。当需求多于 24 个时，通过两两比较来主观地估计费用和价值就变得不合实际了。

另一种方法是质量功能开发 (QFD)，它是能够为产品提供用户价值与性能相联系的一种综合方法。第三种方法来自完全质量管理 (TQM)，它以多个重大项目成功的标准来评价每个需求，并且计算出一个分值用于编排需求的优先级。然而，尽管 QFD 与 TQM 具有精确性，却很少有公司愿意使用它。

表13-2 “化学制品跟踪系统”优先级设计的矩阵范例

相对权值：	2	1		1		0.5			
特 性	相对利润	相对损失	总价值	价值%	相对费用	费用%	相对风险	风险%	优先级
1.查询供应商订单的状态	5	3	13	8.4	2	4.8	1	3.0	1.345
2.建立化学制品仓库存货清单报表	9	7	25	16.2	5	11.9	3	9.1	0.987
3.查看一个特定化学制品容器的历史记录	5	5	15	9.7	3	7.1	2	6.1	0.957
4.打印化学制品安全数据表	2	1	5	3.2	1	2.4	1	3.0	0.833
5.维护危险化学品列表	4	9	17	11.0	4	9.5	4	12.1	0.708
6.更改未定的化学制品请求	4	3	11	7.1	3	7.1	2	6.1	0.702
7.建立个人实验室存货清单报表	6	2	14	9.1	4	9.5	3	9.1	0.646
8.在供应商目录中查询一个特定的化学制品	9	8	26	16.9	7	16.7	8	24.2	0.586
9.在训练数据库中查询一种危险化学制品训练记录	3	4	10	6.5	4	9.5	2	6.1	0.517
10.从结构绘图工具导入化学制品的结构	7	4	18	11.7	9	21.4	7	21.2	0.365
总计	54	46	154	100	42	100	33	100	-----

表13-2描述了一个简单的数据表，该数据表有助于估计使用实例、产品特性或个人功能需求集合的相对优先级。这个例子描述了“化学制品跟踪系统”的许多特性。这个图解来自于QFD关于客户价值的概念，客户价值取决于两个方面：一方面，如果实现了特定的产品特性，那么将为客户提供利益；另一方面，如果不能实现产品特性，就要受到损失（Pardee 1996）。特性的诱人之处是与它所提供的价值成正比，而与实现该特性时的费用和技术风险成反比。一切都是平等的，只有那些具有最高的价值/费用比率的特性才应当具有最高的优先级。这个方法在连续的区间上分配估计的优先级，而不只是把它们分成几个不同的优先级层次上。

你只能把这个设定优先级的图解应用于非最高优先级的可变动的特性上。例如，不能把实现产品核心业务功能的特性和被视为产品的独特之处或者那些为符合政府规定所要求的特性作为该优先级图解的分析项。无论如何，必须首先实现这些特性。一旦分清对于产品交付必不可少的特性，就可以对其他的特性采用模型来确定其相对优先级。

在设定优先级的过程中典型的参与者有：

- 项目经理、他指导全过程，解决冲突，并且在必要的时候调整其它参与者的方案。

- 重要的客户代表，例如产品的代表者，他可以提供受益和损失程度。
- 开发者代表，例如开发组的技术指导者，他提供了费用和风险程度。

你必须遵循如下步骤来使用这个优先级设定工具：

1) 在一个平面中列出你要设定优先级的所有需求、特性或使用实例；在这个例子中，我们将使用特性。所有项都必须在同一抽象级别上；不要把个人需求与产品特性混合在一起。如果某些特性有逻辑上的联系——例如，只有包括特性 A的情况下才能实现特性 B——那么在分析中只要列出驱动特性就可以了。这种模型在其有效范围内可以容纳几十种特性。如果你有更多的项，那么就把相关的特性归成一类，并建立一个可管理的初始化列表。如果你需要的话，可以在更详细的级别上进行第二轮分析。

2) 估计每一个特性提供给客户或业务的相关利益，并用 1~9 划分等级，1 代表可忽略的利益，9 代表最大的价值。这些利益等级表明了与产品的业务需求的一致性。客户代表是判断这些利益的最佳人选。

3) 估计出如果没有把应该实现的特性包括到产品中，将会给客户或业务上带来的损失。使用 1~9 划分等级，这里 1 代表基本无损失，9 代表严重损失。虽然不服从工业标准与客户关系不大，但可能蒙受巨大损失，这将会遗漏客户提出的一些合理的特性。对于具有低利润低损失的需求只会增加费用，而不会增加价值；它们可能只是作为修饰的实例。

4) 总价值栏是相对利润和相对损失的总和。在缺省情况下，利润和损失的权值是相等的。作为一种精化，你可以更改这两个因素的相对权值。在表 13-2 的例子中，所有利润估价的权值都是损失估价权值的两倍。平面图算出了特性价值的总和并计算出每个特性价值占总价值的百分比（价值百分比栏）。

5) 估计实现每个特性的相对费用，使用 1（低）~9（高）划分等级。平面图将计算出由每一个特性所构成的总费用的百分比。根据需求的复杂度，所需求的用户界面的实现情况、重用当前代码的潜在能力、所需要的测试量和文档等等，开发者可以估算出费用。

6) 类似地，开发者应该要估计出与每个特性相关的技术或风险相对程度，并利用 1~9 划分等级。1 级表示你可以轻而易举地实现编程，而 9 级表示需要极大地关注其可行性、缺乏具有专门知识的人员，或者使用不成熟或不熟悉的工具和技术。平面图将计算出每个特性所产生的风险百分比。在缺省情况下，利润损失，费用和风险的权值是相等的，但是你可以在平面图中调整其权值。在表 13-2 中，风险权值是费用权值的一半，而费用权值与损失权值相等。如果你无需在模型中考虑风险，就把风险的权值设为 0。

7) 一旦把所有的估算写入平面图，你就可以利用如下公式计算出每一特性的优先级：

$$\text{优先级} = \frac{\text{价值 \%}}{(\text{费用 \%} \times \text{费用权值}) + (\text{风险 \%} \times \text{风险权值})}$$

8) 按计算出的优先级的降序排列表中的特性。处于列表最顶端的特性是价值、费用和风险之间的最佳平衡，因此必须具有最高优先级。

这种半定量方法从数学上讲并不严密，并且其准确程度受到对每个项目的利润、损失、费用和风险的估算能力的影响。因此，只能把计算出来的优先级序列作为一种指导策略。客户和开发者代表应该讨论整个平面图，从而在评价和优先级排序结果上达成共识。利用先前项目中一系列完整的需求，根据你自己的使用情况来校正这个模型。你可以适当调整每一因素的权值，直到所计算出的优先级序列与后来对测试集中需求的重要性评估相吻合为止。

当你评估所提出的需求时，这个模型有助于你作出合理的决策。评估这些需求的优先级以指明它们与现存的需求基础之间的一致性，这样，你就可以选择一个合理的实现序列。在把需求优先级的设定由政治竞技场上的妄加评论转向以客观和分析为基础的评估过程，在项目中你所采取的措施将可以提高以最合理的序列实现最重要功能的能力。

下一步：

- 将本章所描述的设定优先级模型应用于你当前项目中十个或十五个特性或使用实例上。把计算出来的优先级与用其它方法所设定的优先级进行比较，其结果如何？把它们与你凭直觉判断的优先级相比，其结果又如何？
- 如果模型所描述的优先级与所设想的有不符合之处，分析模型中哪一部分的结果不合理。尝试着对利润、损失、费用和风险应用不同的优先级。调整模型，直到它所提供的结果与你所期望的一致为止。
- 一旦你校准并调整好设定需求优先级的模型，把它应用于新的项目。把计算出的优先级结合到决策过程之中，并与使用其它设定需求优先级的方法相比较，看是否所得到的结果更能使风险承担者满意。

第14章 需求质量验证

大多数软件开发者都经历过在开发阶段后期或在交付产品之后才发现需求的问题。当以原来需求为基础的工作完成以后，要修补（fix）需求错误就需要作大量的工作。有研究表明：比起在需求开发阶段由客户发现的一个错误，然后更正这一错误需要多花 68~110倍的时间。另外一个研究发现，在需求开发阶段发现的一个错误，平均仅需要花 30分钟修复，但是在系统测试时发现的错误需要花 5~17个小时来修复（Kelly, Sherif and Hops 1992）。检测需求规格说明中的错误所采取的任何措施都将为你节省相当多的时间和金钱。

在许多项目中，包括使用典型的瀑布型生存周期法的项目，测试是一项后期的开发活动。与需求相关的问题总是依附在产品之中，直到通过昂贵并且耗时的系统测试或由客户才可最终发现它们。如果你在开发过程的早期阶段就开始制订测试计划和进行测试用例的开发，就可以在发生错误时立即检测到并纠正它。这样可以防止这些错误进一步产生危害，并且可以减少你的测试和维护费用。

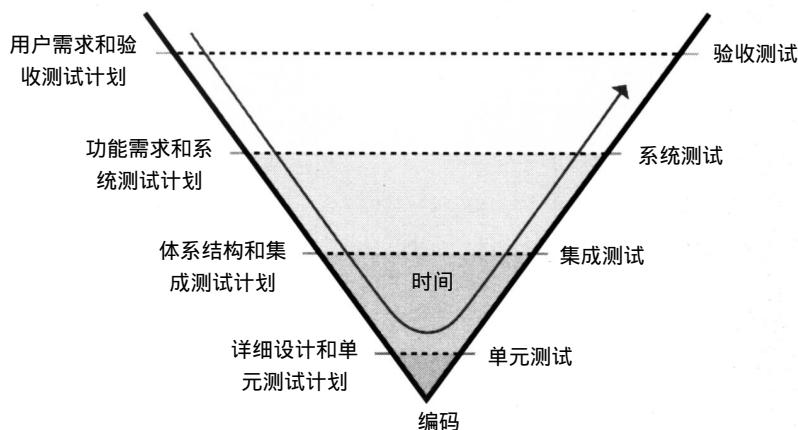


图14-1 软件开发的V字模型

图14-1描绘了软件开发的 V字模型，它表明了测试活动总是与开发活动平行发展的（Davis 1993）。这个模型指明了验收测试是以用户需求为基础的，系统测试是以功能需求为基础的，而集成测试是以系统的体系结构为基础的。在相应的开发阶段，必须规划测试活动并为每一种测试设计测试用例。不可能在需求开发阶段真正进行任何测试，因为还没有可执行的软件。然而，你可以在开发组编写代码之前，以需求为基础建立概念性测试用例，并使用它们发现软件需求规格说明中的错误、二义性和遗漏，还可以进行模型分析。

需求验证是需求开发的第四部分（其余三个为获取、分析和编写规格说明）[⊖]。需求验证

⊖ 一些作者在需求工程这阶段中使用“确认”这一术语（Thayer和Dorfman 1997）。验证决定了开发成的产品是否能满足开始时所确定的需求（即正确完成任务）。确认只评估了过渡产品或最终产品是否能真正满足最高层次的特定需求（即完成特定任务）。对于软件需求，这两个术语的差别是微妙的并且是有争议的，所以需求工程阶段，我采用IEEE的术语“验证”。

所包括的活动是为了确定以下几方面的内容：

- 软件需求规格说明正确描述了预期的系统行为和特征。
- 从系统需求或其它来源中得到软件需求。
- 需求是完整的和高质量的。
- 所有对需求的看法是一致的。
- 需求为继续进行产品设计、构造和测试提供了足够的基础。

需求验证确保了需求符合需求陈述（requirement statement）的良好特征（完整的、正确的、灵活的、必要的、具有优先级的、无二义行及可验证的）并且符合需求规格说明的良好特性（完整的、一致的、易修改的、可跟踪的）。当然，你只能验证那些已编写成文档的需求，而那些存在于用户或开发者思维中的没有表露的、含蓄的需求则不予验证。

在收集需求并编写成需求文档后，你所进行的需求验证并不仅仅是一个独立的阶段。一些验证活动，例如对渐增型软件需求规格说明的反复评审，将贯穿于反复获取需求、分析和编写规格说明的整个过程。其它的验证步骤，例如软件需求规格说明的正式审查，是在正式确定软件需求规格说明基线之前对需求分析质量进行的最后一次有用的质量过滤。当你的项目计划或实际工作中的独立任务破坏了结构性时，就要结合进行需求验证活动，并且为随后出现的返工预先安排一段时间，这通常会在质量控制活动之后进行。

有时，项目的参与者不愿意在评审和测试软件需求规格说明上花费时间。虽然在计划安排中插入一段时间来提高需求质量似乎相应地把交付日期延迟了一段时间，但是这种想法是建立在假设验证需求上的投资将不产生效果的基础上的。实际上，这种投资可以减少返工并加快系统测试，从而真正缩短了开发时间。Capers Jones提出：为防止错误而花费1美元将可以为你修补错误节省3~10美元（Jones 1994）。更好的需求将会带来更好的产品质量和客户更大的满意程度，这可以降低产品生存期中的维护、增强和客户支持的费用。在需求质量上的投资可以使你节省更多的钱。

使用不同的技术有助于你验证需求的正确性及其质量（Wallace and Ippolito 1997）。本章将重点介绍两种最重要的验证技术：正式和非正式的需求评审，还有从使用实例和功能需求中开发出来的概念性测试用例。

14.1 需求评审

通常，总是由一些非软件开发人员进行产品检查以发现产品存在的问题，这就是技术评审。需求文档的评审是一项精益求精的技术，它可以发现那些二义性的或不确定的需求、那些由于定义不清而不能作为设计基础的需求，还有那些实际上是设计规格说明的所谓的“需求”。

需求评审也为风险承担者们提供了在特定问题上达成共识的方法。我的同事 Barry 曾经主持了一个包括来自四个用户代表的软件需求规格说明的评审工作。一个用户提出了一个灾难性的问题：它将使需求做重大更改。会后，需求分析员和项目经理很恼火，因为在前两个月的定义需求会议上，该用户也在场，但她却没有提出这一问题。经过一些调查之后才发现该用户已经反复提出了这个问题，但都被忽略了。在评审过程中，当许多用户一致认为这是一个严重的问题时，分析员和项目经理意识到，他们再也不能忽略这一问题了。

不同种类的技术评审具有不同的称谓。非正式评审的方法包括把工作产品分发给许多其

它的开发人员粗略看一看和走过场似地检查一遍 (walkthrough)。同时执行者描述产品，且征求意见。非正式评审对于培养其他人对产品的认识并获得非结构化的反馈是有利的，但非正式评审是非系统化的，不彻底的，或者在实施过程中具有不一致性。非正式评审不需要记录备案。

非正式评审可以根据个人爱好的方式进行评审，而正式评审则遵循预先定义好的一系列步骤过程。正式评审内容需要记录在案，它包括确定材料、评审员、评审小组对产品是否完整或是否需要进一步工作的判定，以及对所发现的错误和所提出的问题的总结。正式评审小组的成员对评审的质量负责，而开发者则最终对他们所开发的产品的质量负责 (Freedman and Weinberg 1990)

正式技术评审的最好类型叫作审查 (inspection) (Ebenau and Strauss 1994; Gilb and Graham 1993)。对需求文档的审查是可利用的最高级软件质量技术。一些公司已经认识到：在审查需求文档或其它软件产品上花费一个小时，可节省十个小时的工作时间 (Grady 1994)。我尚不知道有哪些其它的软件开发或质量评估可以产生十倍的回收投资比。

如果你对提高软件的质量持有认真的态度，那么就审查所编写需求文档的每一行。虽然对大型的需求文档进行详细审查很无聊并且也很费时，但是我所知道的采用需求审查的人都一致认为他们所花的每一分钟都是值得的。如果你认为没有时间详细审查各个方面，那么就使用简单的风险分析模型来区分需求文档哪些部分是需要详细审查的和那些不重要部分只要用非正式评审就能满足质量要求。

在“化学制品跟踪系统”中，在每次获取需求的专题讨论会之后代表不同用户类的小组对渐增性软件需求规格说明进行非正式评审，立刻就发现了许多错误。在需求获取完成以后，由一个系统分析员把来自不同用户类的软件需求规格说明归纳在一起组成一份大约有 50 页的文档，并加上许多附录。两个分析员、一个开发人员、三个产品代表者、项目经理以及一个测试人员一起在三次长达两个小时的审查会上对软件需求规格说明进行审查。审查小组发现了 223 个错误，其中包括几十个重大缺陷。所有的审查员一致认为在审查会上，他们在软件需求规格说明上所花的时间（一次一个需求），从长远目标来看，节省了项目开发小组大量的时间。

14.1.1 审查过程

在十九世纪七十年代中叶，Michael Fagan 在 IBM 制定出了审查的过程 (Fagan 1976)，该过程被认为是软件业最好的实践 (Brown 1996)。人们可以审查任何一种软件工作产品，包括需求和设计文档、源代码、测试文档及项目计划等等。审查定义为多阶段过程，涉及到由受过培训的参与者组成的小组，他们把重点放在查找工作产品缺陷上。审查提供了一个质量关卡 (quality gate)，文档在最终确定以前，必须通过该关卡的检查。虽然，对于 Fagan 的方法是否最有效并且是不是最有效的审查的形式还存在争议 (Glass, 1999)，但是审查是强有力的质量技术，这是毫无疑问的。

1. 参与者

审查参与者必须代表三个方面的观点：

- a. 产品的开发者及其可能的同组成员——编写需求文档的分析员提供这方面观点。
- b. 先前产品的开发者或正在评审的项目的规格说明编写者——这可能是一个系统工程师或系统构造师，他们可以检查软件需求规格说明，以获得系统说明中每个需求的正确可跟踪

能力。由于没有高层次需求文档，审查工作必须包括真正的客户，以保证软件需求规格说明能正确并完整地描述了他们的需求。

c. 要根据正在审查的文档来开展工作的人们——对于一个软件需求规格说明，你可能需要包括一个开发人员、一个测试人员、一个项目经理和一个用户文档编写人员，他们的工作基础都是软件需求规格说明。这些审查人员将会发现不同类型的问题。一个测试人员很可能会发现一个不明确的需求，而一个开发人员将会发现一个技术上不可实现的需求。

审查组中的审查人员应限制在 7个人左右或者更少。如果审查人员太多则很容易在边际讨论、解决问题和关于某些事是对还是错的争论上造成混乱，从而在审查过程中降低分析问题的速度并且会增加发现每个错误所花的费用。

2. 审查中每个成员扮演的角色

一些审查组中的成员在审查期间扮演着特定的角色；这些角色随着不同的审查过程而不同，但其所起的作用是相似的。

作者 作者创建或维护正在被审查的产品。软件需求规格说明的作者通常是收集用客需求并编写文档的分析员。在诸如“一包到底”的非正式审查中，作者经常主持讨论。然而，作者在审查中却起着被动的作用，不应该充当下列任一角色：调解者、读者或记录员。由于作者在审查中不起积极作用，因此，他只能听取其它审查员的评论，思考回答他们所提出的问题，但他并不参与讨论。作者经常可以发现其它审查员没有觉察到的错误。

调解者 调解者 (moderator) 或者审查主持者所做的是：与作者一起为审查制订计划，协调各种活动，并且推进审查会的进行。调解者在审查会开始前几天就把待审查的材料分发到各个审查员，按时召开会议，从审查员那获得审查结果，并且使会议集中在发现错误上，而不是解决提出的问题。向经理或者那些收集审查数据的工作人员汇报审查结果也是调解者的责任。调解者最后的角色是督促作者对规格说明做出建议性的更改，以保证向执行者明确说明在审查过程中提出的问题和缺陷。

读者 读者的角色由审查员扮演。在审查会进行期间，读者一次审查规格说明中的一块内容，并做出解释，而且允许其它审查员在审查时提出问题。对于一份需求规格说明，审查员每次必须对需求给出注解或一个简短评论。通过用自己的话来陈述，读者可能做出与其它审查员不同的解释，这将有利于发现二义性或可能的错误。

记录员 记录员，或书记员，用标准化的形式记录在审查会中提出的问题和缺陷。记录员必须仔细审查所写的材料以确保记录的正确性。其它的审查员必须用有说服力的方式帮助记录员抓住每个问题的本质，这一方法也使作者清楚地认识到问题的所在和本质。

3. 审查阶段

正如图14-2所示，审查是一个多步骤事件（从 Ebenau and Strauss 1994 改编而来）。每一个审查过程阶段的目的简要地总结如下：

规划 (planning) 作者和调解者协同对审查进行规划，以决定谁该参加审查，审查员在召开审查会之前应收到什么材料并且需要召开几次审查会。审查速度对能发现多少错误影响甚大 (Gilb and Graham 1993)。如图14-3所示，审查软件需求规格说明越慢，就能发现越多的错误（对这一现象的另一种解释是如果遇到太多的错误，就会引起审查速度的下降。）由于不能把太多的时间用于需求审查，所以应根据忽略重大缺陷的风险来选择一种合理的速度。每小时审查4~6页是合理的，但应根据如下因素调整审查速度：

- 一页中的文字数量
- 规格说明的复杂性
- 待审查的材料对项目成功的重要程度
- 以前的审查数据
- 软件需求规格说明作者的经验水平

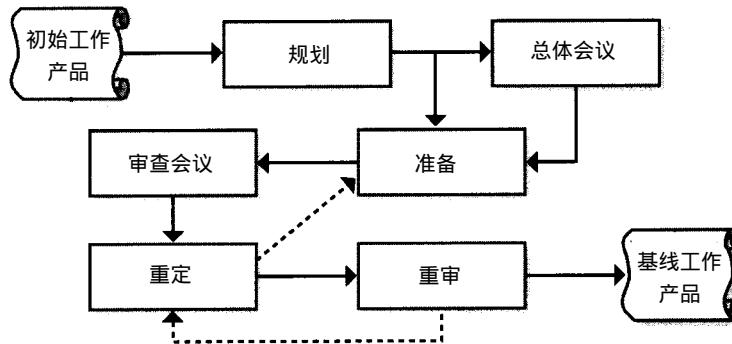


图14-2 审查过程阶段

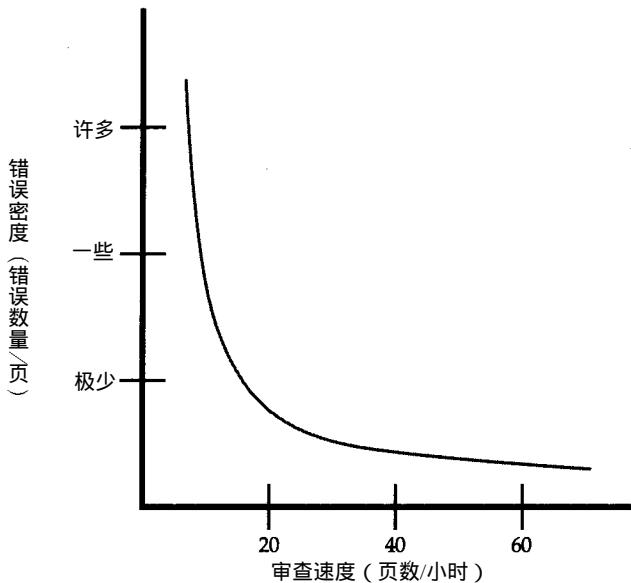


图14-3 审查速度与发现的错误数量之间的关系

总体会议 (overview meeting) 总体会议可以为审查员提供了解会议的信息，包括他们要审查的材料的背景，作者所作的假设和作者的特定审查目标。如果所有的审查员对要审查的项目都很熟悉，那么你就可以省略本次会议。

准备 (preparation) 在正式审查的准备阶段，每个审查员以典型缺陷 (defect) 清单 (在本章的后面部分介绍) 为指导，检查产品可能出现的错误，并提出问题。审查员所发现的错误中有高达 75% 的错误是在准备阶段发现的，所以这一步骤不能省略。如果审查员准备不充分，将会使审查会变得低效，并可能作出错误的结论，此时，审查就是一种时间的浪费。记住，你花在检查同僚工作的时间是对提高整体产品质量的一种投资，并且其他组员也会以

同样的方式帮助你提高产品。

审查会议 (inspection meeting) 在审查会进行过程中，读者通过软件需求规格说明指导审查小组，一次解释一个需求。当审查员提出可能的错误或其它问题时，记录员就记录这些内容，其形式可以成为需求作者的工作项列表。会议的目的是尽可能多地发现需求规格说明中的重大缺陷。审查员很容易提出肤浅的和表面的问题，或者偏离到讨论一个问题是否是一个错误，讨论项目范围的问题，并且集体研讨提出问题的解决方案。这些活动是有益的，但是他们偏离了寻找重要错误以及提高发现错误几率的中心目标。审查会不应该超过两个小时；如果你需要更多的时间，就另外再安排一次会次。在会议的总结中，审查小组将决定是否可以接受需求文档、经过少量的修改后可接受或者由于需要大量的修改重审而不被接受。

一些研究者认为审查会议是劳动密集型的，以至于很难说清它们的价值。然而，我发现审查能发现审查员在进行个人准备时没有发现的错误。即使有这些审查质量的活动，在继续进行设计和构造软件时，应该根据风险，对为了提高需求质量需要投入多少精力作出决策。

重写 (rework) 我所观察到的几乎每一个质量控制活动都可能发现一些需求缺陷。因此，作者必须在审查会之后，安排一段时间用于重写文档。如果把不正确的需求拖延到以后修改，那将十分费时，所以现在正是解决二义性、消除模糊性，并且为成功开发一个项目打下坚实的基础。如果你不打算纠正缺陷，那么进行需求审查将是无意义的。

重审 (follow-up) 这是审查工作的最后一步，调解者或指派人单独重审由作者重写的需求规格说明。重审确保了所有提出的问题都能得到解决，并且正确修改了需求的错误。重审结束了审查的全过程并且可以使调解者做出判断：是否已满足审查的退出标准。

4. 进入和退出审查的标准

当软件需求文档满足特定的前提条件时，你就可以进行需求审查了。在审查的准备阶段，进入审查的标准为作者设定了前进的方向。这些标准还可以使审查小组避免把时间浪费在审查之前就应该解决的问题上。调解者在决定进行审查之前，可以把进入审查的标准作为一种清单，并以此作为判断的标准。下面是一些关于需求文档的进入审查的标准：

- 文档符合标准模板。
- 文档已经做过拼写检查和语法检查。
- 作者已经检查了文档在版面安排上所存在的错误。
- 已经获得了审查员所需要的先前或参考文档，例如系统需求规格说明。
- 在文档中打印了行序号以方便在审查中对特定位置的查阅。
- 所有未解决的问题都被标记为 TBD (待确定)。
- 包括了文档中使用到的术语词汇表。

相似地，在调解者宣布审查结束之前，你应该定义所满足的退出审查的标准。这里有一些关于需求文档的退出标准：

- 已经明确阐述了审查员提出的所有问题。
- 已经正确修改了文档。
- 修订过的文档已经进行了拼写检查和语法检查。
- 所有TBD的问题已经全部解决，或者已经记录下每个待确定问题的解决过程，目标日期和提出问题的人。
- 文档已经登记入项目的配置管理系统。

- 检查是否已将审查过的资料送到有关收集处。

5. 需求审查清单

为了使审查员警惕他们所审查的产品中的习惯性错误，对你的公司所创建的每一类型的需求文档建立一份清单。这些清单可以提醒审查员以前经常发生的需求问题。图 14-4描述了审查软件需求规格说明时的清单，图 14-5包含了一个使用实例的审查清单。

没有人可以记住冗长清单中的所有项目。削减每一份清单以满足公司的需要，并修改这些清单使其能反映需求中最常发现的错误。可以让不同的审查员使用完整清单的不同子集来查找错误。一个人可以检查出所有文档内部的交叉引用是正确的，而另一个人可以判断这些需求是否可以作为设计的基础，并且第三个人可以专门评价可验证性。一些研究结果表明：赋予审查员特定的查错责任，向他们提供结构化思维过程或情节以帮助他们寻找特定类型的错误，这比仅仅向审查员发放一份清单所产生的效果要好得多（Porter, Votta and Basili 1995）。

14.1.2 需求评审的困难

那些审查需求文档的公司面临着许多困难（challenge）。下面是一些普遍的问题，并附有解决的方案。

1) 大型的需求文档 审查一份几百页的软件需求规格说明是令人畏惧的。你很可能完全忽略整个审查过程，并继续进行软件的构造开发——这不是一个好的选择。即使是一份中型的软件需求规格说明，审查员们可能会认真地检查第一部分，一些意志坚定的人可以检查到中间部分，但没有一个人可能检查到最后一部分。为了避免使审查小组感到不安，只在把软件需求规格说明作为基线时才进行审查，在审查全部的文档之前，在你开发软件需求规格说明时，可以采用非正式的、渐增式的审查。让一些审查员从文档的不同位置开始检查，以确保认真地检查其中的每一页。如果你有足够的审查员，可以分成几个小组分别审查材料的不同部分。

2) 庞大的审查小组 许多项目参与者和客户都与需求有关系，所以你可能要为需求审查的参与者制作一张冗长的名单列表。然而，庞大的审查小组将导致难于安排会议，并且在审查会上经常引发题外话，在许多问题上也难于达成一致意见。我曾参加过一个有 14 名审查员的审查会。14个人对是否离开一个燃烧的房子意见不一，更不用说在判断一个特定的需求是否正确上达成一致意见了。尝试用以下的方法处理庞大的审查小组：

- 确保每个参与者都是为了寻找错误，而不是为了解软件需求规格说明中的内容或者为了维护行政上的位置。如果一些参与者只是想大概了解审查的内容，那么就邀请他们去参加总体会议，而不是参加审查会。
- 理解审查员所代表的观点（例如客户、开发者或测试者），并且委婉地拒绝以相同的观点看待问题的参与者。在准备阶段，你可能要收集持有同样观点的反馈人的信息，但只要派其中的一个作为代表参加会议。
- 把审查组分成若干小组并行地审查软件需求规格说明，并把他们发现的错误集中起来，剔除重复的部分。研究表明：多个审查小组比起单一的大组而言，可以发现需求文档中更多的错误（Martin and Tsai 1990; Schneider, Martin and Tsai 1992; Kosman 1997）。审查小组总是发现错误的不同子集，所以并行审查的结果是追加的，而不是冗余的。

3) 审查员在地域上的分散 越来越多的公司正通过地域上分散的开发组进行合作开发产

品。地域上的分散性使需求审查更加困难。视频会议是一种有效的解决方案，但在电话会议中，你无法知道对方赋予形体的语言以及脸部的表情，其效果比面对面的会议要差。比起面对面的会议，这两种远程会议更难于进行调节（控制）。

组织和完整性

- 所有对其它需求的内部交叉引用是否正确？
- 所有需求的编写在细节上是否都一致或者合适？
- 需求是否能为设计提供足够的基础？
- 是否包括了每个需求的实现优先级？
- 是否定义了所有外部硬件、软件和通信接口？
- 是否定义了功能需求内在的算法？
- 软件需求规格说明中是否包括了所有客户代表或系统的需求？
- 是否在需求中遗漏了必要的信息？如果有的话，就把它们标记为待确定的问题。
- 是否记录了所有可能的错误条件所产生的系统行为？

正确性

- 是否有需求与其它需求相冲突或重复？
- 是否简明、简洁、无二义性地表达每个需求的？
- 是否每个需求都能通过测试、演示、审查得以验证或分析？
- 是否每个需求都在项目的范围内？
- 是否每个需求都没有内容上和语法上的错误？
- 在现有的资源限制内，是否能实现所有的需求？
- 是否任一个特定的错误信息都具有唯一性和明确的意义？

质量属性

- 是否合理地确定了性能目标？
- 是否合理地确定了安全与保密方面的考虑？
- 在确定了合理的折衷情况下，是否详实地记录了其它相关的质量属性？

可跟踪性

- 是否每个需求都具有唯一性并且可以正确地识别它？
- 是否可以根据高层需求（如系统需求或使用实例）跟踪到软件功能需求？

特殊的问题

- 是否所有的需求都是名副其实的需求而不是设计或实现方案？
- 是否确定了对时间要求很高的功能并且定义了它们的时间标准？
- 是否已经明确地阐述了国际化问题？

图14-4 软件需求规格说明的审查清单

- 使用实例是否是独立的分散任务？
- 使用实例的目标或价值度量是否明确？
- 使用实例给操作者带来的益处是否明确？
- 使用实例是否处于抽象级别上，而不具有详细的情节？
- 使用实例中是否不包含设计和实现的细节？
- 是否记录了所有可能的可选过程？
- 是否记录了所有可能的例外条件？
- 是否存在一些普通的动作序列可以分解成独立的使用实例？
- 是否简明书写、无二义性和完整地记录了每个过程的对话？
- 使用实例中的每个操作和步骤是否都与所执行的任务相关？
- 使用实例中定义的每个过程是否都可行？
- 使用实例中定义的每个过程是否都可验证？

图14-5 使用实例文档审查清单

在共享网络文件夹中的电子文件进行文档评审改变了传统的评审会议。在这一方法中，评审员利用字处理软件的特性，在他所审查的文档中插入评论。每个审查员的评论都做上初始标记，这样每个审查员就能看见先前审查员所写的评论。基于 Web 的聊天工具可以进行实时的远程讨论，但他们只提供了很窄的通信带宽。基于 Web 的嵌入式协作软件工具，就像软件开发技术公司所提供的 Review Pro(<http://www.sdtcorp.com>)也有助于进行远程讨论。如果你不想通过审查会进行审查，那么必须认识到审查效率将下降约 25%。

14.2 测试需求

通过阅读软件需求规格说明，通常很难想像在特定环境下的系统行为。以功能需求为基础或者从使用实例派生出来的测试用例可以使项目参与者看清系统的行为。虽然没有在运行系统上执行测试用例，但是设计测试用例的简单动作可以解释需求的许多问题（Beizer 1990），如果你在部分需求稳定时就开始开发测试用例，那么就可以及早发现问题并以较少的费用解决这些问题。

编写关于黑盒子或功能上的测试用例可以明确在特定条件下系统运行的任务。因为你无法描述可能的系统响应，在你面前将会出现一些模糊的和二义性的需求。当分析员、开发人员和客户通过测试用例进行研究时，他们将对产品如何运行的问题有更清晰的认识。

在开发过程的早期阶段，可以从使用实例中获得概念上的功能测试用例（Ambler 1995; Collard 1999）。然后，你就可以利用测试用例来验证文本需求规格说明和分析模型（例如对话图）并评价原型。这些基于模仿使用的测试用例可以作为客户验收测试的基础。在正式的系统测试中，可以把它们详述成测试用例和过程（Hsia, Kung and Sell 1997）。在客户定义他们验收的标准时，你询问客户的基本问题是：“如果开发出你们所期望的软件，你是怎么来判断开发出的软件是你真正所需要的？”如果他们不能回答关于每个特性或使用实例的这种问题，他们就必须澄清需求。

在前面的章节中，我提到过一种情况：我让开发组中的 UNIX 脚本专家 Charlie 为我们正在使用的“商业错误跟踪系统”编写一个简单的电子邮件接口扩展。我写了许多需求，Charlie 觉得这对他很有帮助；因为他以前编写脚本时，别人从未给他提出需求。不幸的是，在我为电子邮件功能编写测试用例之前却延误了两个星期。后来，我找到了错误，那是因为二十多个测试用例中代表的我对电子邮件功能的认识与我提出的需求格格不入。在 Charlie 完成他的脚本之前我纠正了错误的需求，所以在完成脚本的编写时，这些脚本是正确无误的。

需求测试的含义最初对你来说可能看起来比较抽象。可以用一个例子把这个概念描述得更清楚，所以让我们看一下“化学制品跟踪系统”的开发组是如何把需求规格说明、分析模型和早期创建的测试用例结合在一起。下面列出了与请求化学制品这一任务相关的一个业务需求、使用实例、功能需求、部分对话图和一个测试用例。

1) 业务需求 该系统的一个主要业务动机可以用如下的需求来描述：

“化学制品跟踪系统”通过鼓励重复使用公司中可用的那些化学制品容器以降低购买费用。

2) 使用实例 与这个业务需求相一致的一个使用实例是“请求一种化学制品”，它包括允许用户请求化学制品仓库中已有的化学制品的路径。以下是这个使用实例的描述（详见图 8-3）：

请求者通过输入化学制品的 ID 号或从化学制品绘图工具导入（import）化学结构来

下载

请求一种化学制品。系统则通过向请求者提供来自化学制品仓库的一个新的或已用过的化学制品容器或者让请求者向外部供应商发送订单，从而满足请求者的要求。

3) 功能需求 以下是关于让用户选择可用的化学制品的一些功能，而不是向外部供应商发送订单：

如果请求化学制品仓库中的容器，系统将显示可用容器的列表，用户就可以选择一个容器或要求向外部供应商订购一个新容器。

4) 对话图 (dialog map) 图14-6描述了“请求一种化学制品”使用实例中关于这一功能的部分对话图。对话图中的矩形框表示了用户和系统之间概念上的对话元素，箭头则代表了对话元素之间可能的导航路径。

5) 测试用例 (test case) 由于这个使用实例有许多可能的执行路径，所以你可以想出许多测试用例来阐明普通过程、可选过程和例外。以下只是一个测试用例，该测试用例是以向用户显示化学制品仓库中可用容器的列表为基础的。这个测试用例是从该用户任务的使用实例说明和图14-6描述的对话图中派生出来的：

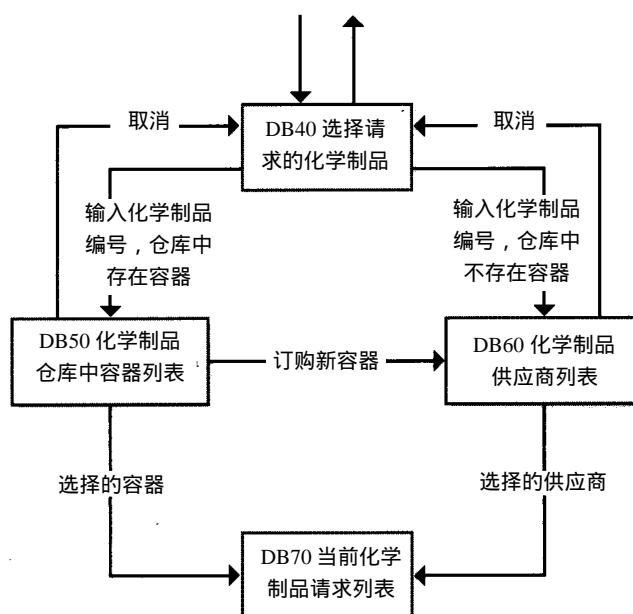


图14-6 在“请求化学制品”使用实例的部分对话图

在DB40对话框中，输入一个合法的化学制品ID号；化学制品仓库中有两个这种化学制品的容器。此时出现了DB50对话框，并带有两个容器号码。选择第二种容器。关闭DB50，此时容器2被加入DB70对话框中当前化学制品请求列表的底部。

Ramesh是“化学制品跟踪系统”的测试主持人，根据他对用户如何与系统交互来请求一种化学制品的理解，编写了许多这样的测试用例。他把每个测试用例映射到相应功能需求上，以保证现有的需求集合可以“执行”每个测试用例，并且至少使每个测试用例覆盖每个功能需求。下一步，Ramesh用高亮度的笔跟踪对话图中每个测试用例的执行路径。图14-7中的阴影线描绘了上面的测试用例样本是如何跟踪进入对话图的。

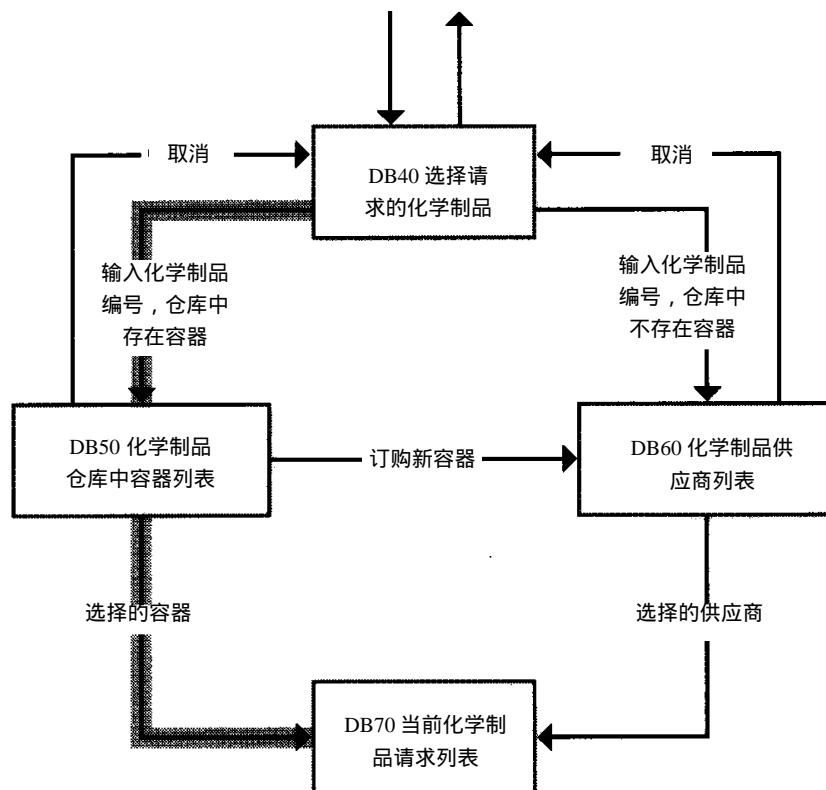


图14-7 在“请求化学制品”使用实例中跟踪一个测试用例进入对话图

通过跟踪每个测试用例的可执行路径，你可以发现不正确和遗漏的需求，并在对话图中纠正错误，精化测试用例。例如，假设以这种方式执行完所有测试用例后，对话图中从 DB50 到 DB60 之间标有“订购新容器”的导航线未被加亮。可能有两种解释：

1) 该导航是一个非法的系统行为。这条线必须从对话图中移去，并且如果软件需求规格说明中包含有这样过渡的需求，那么也应该移去这一需求。

2) 该导航是合法的系统行为，但是遗漏了验证这一系统行为的测试用例。

类似地，假设一个测试用例说明了用户可以采取一些措施从 DB40 直接移到 DB70。然而，图14-6对话图中没有包含这样的导航线，所以测试用例不能以现有的需求来执行。因此，又存在两种解释，要判断哪一个是对的：

1) 从 DB40 到 DB70 的导航是一个非法的系统行为，所以测试用例是错误的。

2) 从 DB40 到 DB70 的导航是一个合法的系统行为，则可能是对话图或可能是软件需求规格说明遗漏了用于执行测试用例的需求。

在这些例子中，分析员和测试人员在编写代码以前把需求、分析模型和测试用例结合在一起检测遗漏、错误和不必要的需求。软件需求在概念上的测试是通过在开发早期的阶段寻找需求错误，从而成为一种在控制项目费用和进度上的强有力的技术。

收集需求并编写需求文档是软件项目设计成功的很好起点。但还需要保证需求的正确性，使需求能体现出良好需求说明的全部特性。如果你能把早期的黑盒子测试设计、非正式需求评审、软件需求规格说明审查和其它需求验证技术相结合，你将花比以前更少的时间、更低的费用来构造质量更高的系统。

下一步：

- 从你的项目软件需求规格说明中任意选择一页功能需求。召集代表不同风险承担者观点的一个小组并仔细审查一页需求以寻找与好的需求说明特性相偏离的那些需求。
- 如果从随机评审中发现太多问题，以至使评审员对整个需求质量感到担忧，那么就要让用户和开发人员代表审查整个软件需求规格说明。在审查过程中要培训审查小组中的成员，使之发挥最高效率。
- 为使用实例或软件需求规格说明中未编码的部分，定义概念上的测试用例。判断风险承担者们是否对测试用例所反映的预期的系统行为持有一致的意见。确保你已经定义了允许测试用例“执行”的所有功能需求，并且不存在多余的需求。

第15章 需求开发向设计规划的转化

虽然在开发的开始阶段困难重重，但是“化学制品跟踪系统”的开发工作总算进展顺利。项目的主办者Gerhard和化学制品仓库的产品代表者Roxanne怀疑是否有必要花费这么多时间来收集需求。然而，他们很愿意同开发小组和其它产品代表一起参加为期一天的软件需求方面的培训。这次培训着重讲述在编写软件程序之前，使项目所有风险承担者在对需求的理解上达成共识的重要性。在培训课中，所有的成员了解了需求的术语、概念和他们要用到的方法，这可以激发他们在具体需求实施过程中采用一些改进需求的技术。

在项目开发过程中，Gerhard收到了用户代表关于如何进行需求开发的一些良好反馈。后来，他请开发组成员和产品代表者共进午餐，以庆祝他们在“化学制品跟踪系统”确定需求基线方面达到了一个重要的里程碑。在就餐时，Gerhard满怀热情地感谢了参与获取需求的人员所做出的贡献以及他们的合作，然后他继续说，“现在你们有了完整正规的需求，我盼望开发小组很快能写出程序代码。”

“我们并不准备马上编写程序代码”，项目经理提醒Gerhard。“我们打算分阶段发行产品，所以我们需要想出一个最好的方法来设计系统，使其能适应将来系统的扩展。我们可以从产品原型中获得关于有效技术途径的一些思想，并且产品原型有助于我们理解用户喜欢的界面特性。如果现在我们在软件的设计上多花上一点时间，那么到明年我们扩充产品的更多功能时就不会遇到较大的问题。”

这时Gerhard有一点沮丧。因为他觉得开发人员现在就可以开始编制程序了。但是Gerhard是否有点行动过早？

有经验的项目经理和开发人员知道把软件需求转化为健壮的设计和合理的项目规划的重要性。本章通过需求和项目规划、设计、编码和测试之间的联系来探讨需求开发和一个成功发行的产品之间的转换方法。

15.1 从需求到项目规划

由于需求定义了项目预期的成果（outcome），所以你的项目规划、预测和进度安排都必须以软件需求为基础。

15.1.1 需求和进度安排

许多软件工程实行“从右到左的进度安排”，此时，规定了发行产品的具体日期而后定义产品的需求。当开发者要实现预期质量标准下所有要求的功能时，他们常常不能按时完成项目。在做出详细的规划和约定之前定义软件需求是更现实的。然而，如果你在需求的哪些部分能适应进度安排的限制哪些部分不能适应进度安排的限制这一问题上还有商量余地的话，那么“从设计到进度安排”的策略是可以起作用的。

对于复杂的系统，软件仅是最终产品的一部分时，只有在产品级（系统）需求产生以后，才能建立高层的进度安排。然后，将系统需求分解并分配到各个不同的软硬件子系统中。从

这一点上看，就可以以不同来源（包括市场、销售、客户服务以及开发）的输入为基础建立起一致的产品发行日期。如果存在进度安排的约束条件，那么具有交叉功能的开发小组必须在功能、质量和费用上作出合理（trade-off）的决策。

你可能考虑按阶段规划和提供项目资金。需求探索作为第一阶段将提供足够的信息，使你能为一个或更多的构造阶段进行现实的规划和预测。具有不确定需求的项目也可以从反复或渐增的软件开发生存期中得以改善。定义需求的优先级可以使你判断出哪些功能应包括在首发版中，哪些功能放到随后发行的版本中。

软件项目可能经常不能达到预定的目标，这是因为开发人员和其他项目参与者是拙劣的规划者，倒不是因为他们是拙劣的软件工程师。主要的规划失误包括：忽略公共（用）的项目任务，低估了要花费的工作量和时间，没有考虑项目风险，并且没有考虑返工所需的时间。正确的项目规划需要以下元素：

- 根据对需求的清楚理解来估计产品规模的大小。
- 根据历史记录了解开发小组的工作效率。
- 需要一张综合的任务列表以完整实现和验证每一特性或使用实例。
- 有效的预测和规划过程。
- 经验。

15.1.2 需求和预估

项目估计（预估）的第一步就是要把需求和软件产品规模的大小相联系。你可以根据文本需求、图形分析模型、原型或用户界面设计来预估产品的大小。虽然对于软件大小没有完善的度量标准，但以下给出了一些常用的度量标准：

- 功能点和特性点的多少（Jones 1996b），或者3-D功能点的数量(Whitmire 1995)。
- 图形用户界面（GUI）元素的数量、类型和复杂度。
- 用于实现特定需求所需的源代码行数。
- 对象类的数量或者其它面向对象系统的衡量标准（Whitmire 1997）。
- 单个可测试需求的数量（Wilson 1995）。

所有这些方法都可用于预估软件的大小，但不管采用什么方法你都必须根据经验进行选择。如果你没有记录当前项目所完成的真正结果，并和你所预估的进行比较，利用那些知识来提高你的预估能力，那么你所预测的将永远是一种猜测。积累数据是需要时间的，所以你可以把度量软件大小的标准与实际的开发工作量相联系。你的目标是建立可以使你从需求文档中预估整个软件大小的方程（等式）或者借助归纳等解决问题的方法。另一种方法是从需求中预测代码行、功能点或图形用户界面元素的数量，并使用商业预估软件作出人员水平与开发进度的合理安排。你可以从 <http://www.method-tools.com> 获得一些可用的预估工具的信息。

含糊的和不确定的需求必定会引起你在软件大小预估中的不确定性，从而导致你的工作量和进度安排预估的不确定。因为在项目的早期阶段，需求的不确定性是不可避免的，所以在进度安排中应包括临时的事件并要合理预算资金以适应一些需求的增加和可能的超限。一个合理的住房改造项目包括了对不可预料的偶然事件，在你的项目中是否也要如法炮制呢？

花一点时间考虑一下，哪些标准最适合你所主持的项目。应该意识到开发时间与产品大

小或开发小组的大小没有线性关系。在产品大小、工作量、开发时间、生产率和人员技术积累时间之间是存在着复杂的关系 (Putnam and Myers 1997)。理解这种关系可以防止你陷入进度安排或人员任务安排承诺的误区 (以前没有类似项目成功完成过)，否则，项目开发将不会成功完成。

15.2 从需求到设计和编码

需求和设计之间存在差别，但尽量使你的规格说明的具体实现无倾向性。理想情况是：在设计上的考虑不应该歪曲对预期系统的描述 (Jackson 1995)。需求开发和规格说明应该强调对预期系统外部行为的理解和描述。让设计者和开发者参与需求审查以判断需求是否可以作为设计的基础。

不同的软件设计方法常常都会满足最终需求，而设计方法会随着性能、有效性、健壮性以及所采用的技术上的不同而变化。如果你直接从需求规格说明跳到编码阶段，你所设计的软件将会是空中阁楼，其可能的结果只能是结构性很差的一个软件。在构造软件之前，你应该仔细考虑构造系统的最有效的方法。考虑一下其它的设计方案将有助于确保开发人员遵从所提出的设计约束或遵从与设计有关的质量属性规格说明。

我曾经参与一项项目，进行了完整的需求分析，建立了详细描述模拟摄像系统行为的 8 个变换过程的数据流程图。经过大量的需求分析后，我们并没有直接进行源代码的编写工作。而是以数据流程图为表示方法，创建了一个设计模型。我们立刻意识到模型中有三个步骤使用了相同的计算算法，另外三个使用不同的方程集，而剩下的两个步骤共享三分之一集合。

分析模型代表了用户和开发小组对我们正在解决的问题的理解，而设计模型则描绘了我们应该如何构造系统。通过设计期间的仔细思考，我们把核心问题简化了 60%，把 8 个复杂的计算集合减少到 3 个。如果我们在需求分析之后立刻进行编码，那么在构造阶段必定会出现代码重复。但是，由于及早发现了可简化问题，我们节省了许多时间和金钱。设计上的返工比编码返工可能要效率高一些。

以需求为基础，反复设计将产生优良成果。当你得到更多的信息或额外的思想时，用不同的方法进行设计可以精细化你最初的概念。设计上的失误将导致软件系统难以维护和扩充，最终会导致不能满足客户在性能和可靠性上的目标。在把需求转化为设计时你所花的时间将是对建立高质量、健壮性产品的关键的投资。

在设计产品时，产品的需求和质量属性决定了所采用的合适的构造方法 (Bass, Clements and Kazman 1998)。研究和评审所提出的体系结构是另一种解释需求的方法且会使需求更加明确。与原型法类似，这是一种自下而上的需求分析方法。两种方法都围绕着这样一种思维过程：“如果我正确理解需求，那么这种方法可以满足这种需求。既然我手中有一个最初的体系结构 (或原型)，它是否有助于我更好地理解需求呢？”

在你可以开始实现各个部分需求前，不必为整个产品进行完整、详细的设计。然而，在你进行编码前，必须设计好每个部分。设计规划将有益于大难度项目 (有许多内部组件接口和交互作用的系统和开发人员无经验的项目) (McConnell 1998)。然而，下面介绍的步骤将有益于所有的项目：

- 应该为在维护过程中起支撑作用的子系统和软件组件建立一个坚固的体系结构。
- 明确需要创建的对象类或功能模块，定义他们的接口、功能范围以及与其它代码单元的

协作。

- 根据强内聚、松耦合和信息隐藏的良好设计原则定义每个代码单元的预期功能。
- 确保你的设计满足了所有的功能需求并且不包括任何不必要的功能。

当开发者把需求转化为设计和代码时，他们将会遇到不确定和混淆的地方。理想情况下，开发者可沿着发生的问题回溯至客户并获得解决方案。如果不能马上解决问题，那么开发者所做出的任何假设，猜想或解释都要编写成文档记录下来，并由客户代表评审。如果遇到许多诸如此类的问题，那么就说明开发者在实现需求之前，这些需求还不十分清晰或具体。在这种情况下，最好安排一两个开发人员对剩余的需求进行评审后才能使开发工作继续进行。

15.3 从需求到测试

详尽的需求是系统测试的基础，反过来只能通过测试来判断软件是否满足了需求。你必须针对软件需求规格说明中所记录的产品的预期行为来测试整个软件，而不是针对设计或编码。基于代码的系统测试可以变成“自满足的预见（self-fulfilling prophecy）”。产品可以正确呈现基于代码的测试用例所描述的所有行为，但这并不意味着产品正确地实现了用户的需求。如果你没有文档形式的需求，你应该重新获得需求以开发合适的测试用例，这将是一个低效的和不准确的方法。让测试人员参与需求审查以确保需求是明确的，通过验证的需求才可以作为系统测试的基础。

在需求开发中，当每个需求都稳定之后，项目的系统测试人员应该编写文档，以记录他们如何验证需求——通过测试、审查，演示或分析。对如何验证每一需求的思考过程本身就是一种很有用的质量审查实践。根据需求中的逻辑描述，利用诸如因果图等分析技术来获得测试用例，这将会揭示需求的二义性、遗漏或隐含的其它条件和其它问题。在你的系统测试方案中，每个需求应至少由一个测试用例来测试，这样就会验证所有的系统行为。你可以由跟踪通过测试的需求所占的比例来衡量测试进度。有经验的测试人员可以根据他们对产品的预期功能、用法、质量特性和特有行为的理解，概括出纯粹基于需求的测试。

基于规格说明的测试适用于许多测试设计策略：动作驱动、数据驱动（包括边界值分析和等价类的划分）、逻辑驱动、事件驱动和状态驱动（Poston 1996）。从正式的规格说明中很容易自动生成测试用例，但是对于更多的由自然语言描述的需求规格说明，你必须手工开发测试用例。比起结构化分析图，对象模型更易于自动生成测试用例。

在开发的进展过程中，你将通过详细的软件功能需求仔细推敲来自使用实例高层抽象的需求，并最终转化成单个代码模块的规格说明。测试方面的权威专家 Boris Beizer (1999)指出针对需求的测试必须在软件结构的每一层进行，而不只是在用户层进行。在一个应用程序中有许多代码不会被用户所访问，但这些代码却是产品基础操作所需要的。即使有些模块功能在整个软件产品中对用户都不可见，但是每个模块功能必须满足其自身的需求或规格说明要求。因此，针对用户需求来测试系统是系统测试的必要但非充分条件。

15.4 从需求到成功

我最近遇到一个项目，在该项目中，一个新的开发小组将针对早先的开发小组所开发的需求实现一个应用程序。这个新的开发小组一看到由3英寸活页纸订成的软件需求规格说明就感到十分恐惧，于是立刻编写代码。在构造软件的过程中，他们没有参考软件需求规格说明，

而是根据他们对预期系统的不完整且不正确的理解，按他们自己的想像进行编码设计。因此，该项目遇到许多问题便不足为奇了。试图理解这个庞大的需求规格说明肯定会令人恐惧的，况且需求规格说明有些部分可能还不完善，但忽略了需求规格说明必定会导致失败。

我知道一个十分成功的项目，在开发项目的过程中，开发人员列出了与特定代码版本相对应的需求。项目的质量保证组通过对照需求来执行测试用例从而评价其对应的代码版本。根据测试标准，如果不满足需求的话就算是一个错误。如果不满足的需求数量超过预先给定的一个数字，或者特定的有重大影响的需求没有被满足，那么这个代码块就被拒绝了。这个项目的成功之处在于把需求文档作为何时发行产品的基础。

一个软件开发项目最终可发行的是满足客户需求和期望的软件系统。需求是从产品概念通向用户满意之路的最本质的一步。如果你不以高质量的需求作为项目规划、软件设计和系统测试的基础，那么在试图开发优秀产品的过程中将浪费大量的人力和物力。

然而，也不要成为需求过程的奴隶。避免陷入畸形分析的陷阱，此时，开发小组将花费大量的时间创建不必要的文档，并举行各种形式上的会议和评审，而并不编写任何软件代码，这将会导致项目被取消。努力在精确的规格说明与可将产品失败的风险降至可接受程度的编码之间做出明智的选择。

下一步：

- 检查你的软件需求规格说明中所有需求是否与软件设计的各元素相对应。这些可能在数据流模型、实体联系图中的表、对象类或方法，或者其它设计元素中处理。如果开发人员在编写代码前，没有进行软件设计，那么他们可能要进行软件设计方面的培训。
- 记录实现每一个产品特性或使用实例的代码行、功能点、对象类或图形用户界面（GUI）元素的数量。并且还要记录对完全实现并验证每个特性或使用实例所估计的工作量和实际的工作量。尽力获取产品大小与所花费工作量的关系，这将有助于你对将来的需求规格说明作出更精确的预估。

第三部分 软件需求管理

第16章 需求管理的原则与实现

在第一章中，我们将需求工程分为需求开发和需求管理。需求开发包括对一个软件项目需求的获取、分析、规格说明及验证。典型需求开发的结果应该有项目视图和范围文档、使用实例文档、软件需求规格说明及相关分析模型。经评审批准，这些文档就定义了开发工作的需求基线（baseline）。这个基线在客户和开发人员之间就构筑了计划产品功能需求和非功能需求的一个约定（agreement）。工程项目可能会有其它的约定，例如可交付性、约束条件、进度安排、预算及合同约定等。但这些均超出了本书范围。

需求约定是需求开发和需求管理之间的桥梁，需求管理包括在工程进展过程中维持需求约定集成性和精确性的所有活动，如图 16-1 所示。需求管理强调：

- 控制对需求基线的变动。
- 保持项目计划与需求一致。
- 控制单个需求和需求文档的版本情况。
- 管理需求和联系链之间的联系或管理单个需求和其它项目可交付品之间的依赖关系。
- 跟踪基线中需求的状态。

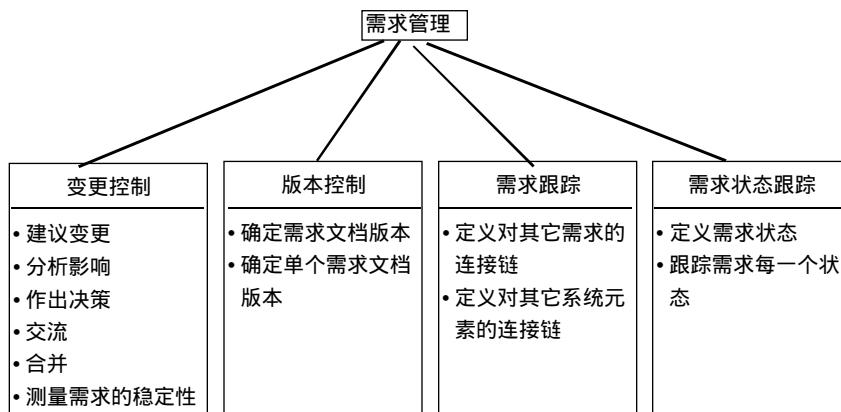


图 16-1 需求管理的主要活动

本章给出了需求管理的基本原则。第三部分的其它内容更详细地评述了专门的需求管理的策略：包括变更控制（第 17 章），需求跟踪（第 18 章）和变更的影响分析（第 18 章）。第三部分的最后是关于帮助管理项目需求的商业工具的讨论（第 19 章）。

16.1 需求管理和过程能力成熟度模型

过程能力成熟度模型（Capability Maturity Model，CMM）对需求管理是一个有用的指导

(CMU/SEI 1995)。位于宾夕法尼亚洲匹兹堡市的卡内基梅隆大学所属的软件工程研究所提出了软件过程能力成熟度模型的概念。CMM是在软件开发机构中被广泛地用来指导过程改进工作的模型。该方法描述了软件处理能力的五个成熟级别。处于一级的组织典型地以非正式的方式管理项目进度，要获得成功，主要依靠天才从业者和管理者的英雄史诗般的奋斗。处于更高成熟度级别的组织把具有创造性、训练有素的员工同软件工程和项目管理过程结合起来，将持续不断地获得成功。

为达到软件过程能力成熟度模型的第二级，组织必须具有在软件开发与管理的六个关键过程域 (key process areas , KPA) 以展示达到目标的能力。需求管理是其中之一，它的目标如下：

- 1) 把软件需求建立一个基线供软件工程和管理使用。
- 2) 软件计划，产品和活动同软件需求保持一致。

无论是否知道或关心过程成熟度模型，大多数软件开发组织将会从达成这两个目标中获益。过程成熟度模型确定若干先决条件和技术策略，使组织能持续地达到这两个目标，但并不指定组织必须遵循的需求管理过程。

需求管理的关键过程领域不涉及收集和分析项目需求。而是假定已收集了软件需求或已由更高一级的系统给定了需求。一旦需求到手且文档化了，软件开发团队和有关的团队（例如质量保证和测试）需要评审文档。发现问题应与客户或其它需求源协商解决，软件开发计划是基于已确认的需求。

开发团队在向客户、市场部或经理们作出承诺 (commitment) 之前，应该确认需求和确认约束条件、风险、偶然因素、假定条件。也许不得不面对由于技术因素或进度原因而不现实的需求作出承诺。但是，决不要承诺任何无法实现的事。

关键处理领域同样建议通过版本控制和变更控制来管理需求文档。版本控制确保随时能知道在开发和计划中正在使用的需求的版本情况。变更控制提供了支配下的规范的方式来统一需求变更，并且基于业务和技术的因素来同意或反对建议的变更。当在开发中修改、增加、减少需求时，软件开发计划应该随时更新以与新的需求保持一致。不反映现实的计划于事无补。

当接受了所建议的变更时，你可能在进程调度或质量上不能满足这项变更。在这种情况下，必须就约定的变更与所涉及的经理、开发者以及其它相关组织进行协商。通过如下方法能使项目反映最新的或变更过的需求。

- 暂时搁置次要需求。
- 得到一定数量的后备人员。
- 短期内带薪加班处理。
- 将新的功能排入进度安排。
- 为了保证按时交工使质量受些必要的影响（通常，这是缺省反应）。

由于项目在特性、进度、人员、预算、质量各个方面的要求不同，所以不存在一个放之四海皆准的模式 (Wiegers 1996a)。根据早期计划阶段中项目风险承担者确定的优先级顺序挑选各项选择。不管你对变更需求或项目情况（如全体职工工作完成量）采取何种措施，必要时调整一些约定仍是需要养成的一个好习惯。这总比不现实地期待所有新要求在原定交付日期前魔术般地实现且其他方面（例如预算或员工工作强度）不受什么影响要好。

即使你现在没用CMM来指导你的软件过程改进，以上所述关于需求管理关键过程领域内的原则和策略（practice）总是有用的。每个开发组织都会从这些方法应用中获益。

16.2 需求管理步骤

开发组织应该定义项目组执行管理他们需求的步骤。文档化编写这些步骤能使组织成员持续有效地进行必要的项目活动。请考虑选择以下主题：

- 用于控制各种需求文档和单个需求版本的工具、技术和习惯做法。
- 建议、处理、协商、通告新的需求和变更给有关的功能域的方法。
- 如何制定需求基线。
- 将使用的需求状态，并且是谁允许作出的变更。
- 需求状态跟踪和报告过程。
- 分析已建议变动的影响应遵循的步骤。
- 在何种情况下需求变更将会怎样影响项目计划和约定。

你可以在一个文档中包含上面所有的信息。或者，你可能喜欢专题分述，例如分成变更控制过程，影响分析过程，状态跟踪过程。这些过程可能在多个项目中都有用，因为他们反映每个项目所应遵循的公共功能。

16.3 需求规格说明的版本控制

“我终于实现了库存报告中重排序的功能。”Shari在项目的每周例会上说。

“噢，用户在两周前就取消这个功能了。”项目管理者说，“你没看改过的软件需求规格说明吗？”

如果以前曾听过这样的谈话，你一定知道浪费时间为已废弃的需求工作会使员工多么地沮丧。我了解到曾有一个开发组在把改进的软件版本交去测试后，收到一大堆错误发现报告。其实是测试者使用了一个已过时软件需求规格说明，结果导致了一大堆错误。开发组花了大量的时间试着确定问题，甚至花了相当多时间对照正确版本的软件需求规格说明进行重复测试。

版本控制是管理需求的一个必要方面。需求文档的每一个版本必须被统一确定。组内每个成员必须能够得到需求的当前版本，必须清楚地将变更写成文档，并及时通知到项目开发所涉及的人员。为了尽量减少困惑、冲突、误传，应仅允许指定的人来更新需求。这些策略适用于所有关键项目文档。

每一个公布的需求文档的版本应该包括一个修正版本的历史情况，即已做变更的内容、变更日期、变更人的姓名以及变更的原因。你应该使用标准修改符，例如，中划线代表取消，下划线代表添加，在页边空白的竖划线指示每个变动的位置。因为这些符号会弄乱文档，支持修改符的字处理软件使你能够预览和打印编辑后的文档或最终的结果。可以考虑给每个需求标记上版本号，当修改需求后就增加版本号。

版本控制的最简单方法是根据标准约定手工标记软件需求规格说明的每一次修改。根据修改日期或印刷日期区别文档的不同版本容易产生错误，所以不被推荐。使用一种手工方法，任何新的文档的第一版当标记为“1.0版（草案1）”，下一稿标记为“1.0版（草案2）”，在文档被采纳为基线前，草案数可以随着改进逐次增加。而当文档被采纳后被标记为“1.0正式版”。

若只有较小的修改，可认为是“1.1版（草案1）”。若有较大的修改时，可认为是“2.0版（草案1）”（“较大”和“较小”只是一个主观的判断）。这个方式清楚地区分草稿和定稿的文档版本，但要求用手工来操作。

一个具有更高级别的版本控制包括用版本控制工具来存储需求文档，例如用登录（check-in）和检出（check-out）程序来管理源代码。这方面有很多商业配置管理工具。我知道一个项目用诸如Microsoft Word这样的工具管理了几个使用实例文档。这个工具能让每个组员得到每个使用实例文档的先前版本，同时提供一个记录每一个文档变更的日志。对存储在工具中的文档，项目需求分析人员具有对它可以读和写的权利，而其他成员只能读。这个工具在该项目的版本控制计划中工作得很好。

版本控制的最有力方法是用一个商业需求管理工具的数据库存储需求（详情见19章）。这些工具能跟踪和报告每个需求的变动历史，当你需要恢复早期的需求时这很有价值。在添加、变动、删除、拒绝一个需求后，附加一些评语描述变更的原因在将来需要讨论时将会很有用。

16.4 需求属性

除了文本，每个功能需求应该有一些相关的信息或称之为属性与之相联系。这些属性在它的预期功能性之外为每个需求建立了一个上下文和背景资料。属性值可以写在一张纸上，存储在一个数据库或需求管理工具中。商业工具除由系统产生一些属性外，还可以由你自己定义各种数据类型的其它属性。这些工具允许过滤、排序、查询数据库来察看按选择的需求属性的需求集。

对大型的复杂项目来说，丰富的属性类别显得尤为重要。在你的每个需求文档中考虑明确如下的属性：

- 创建需求的时间
- 需求的版本号
- 创建需求的作者
- 负责认可该需求的人员
- 需求状态
- 需求的原因或根据（或信息的出处）
- 需求涉及的子系统
- 需求涉及的产品版本号
- 使用的验证方法或接受的测试标准
- 产品的优先级或重要程度（例如高、中、低或把你能定义的属性来表示成本书第13章所描述的优先级的四个方面：收益、损失、成本、风险）
- 需求的稳定性（在将来需求可能变更的指示器，不稳定的需求意味你应给予较多的关注，因为你将面临不定的、混沌的、或不能重复的业务过程。）

定义和更新这些属性值是需求管理成本的一部分。精心挑选属性最小子集能帮助你有效管理项目。例如，你不必把负责认可需求的人员和需求涉及的子系统都记录下来。如果这样的属性在别的地方已经设置了，在总的开发跟踪系统中就不必在需求数据库中重复设置。

在整个开发过程中，跟踪每个需求的状态是需求管理的一个重要方面（Caputo 1998）。在每一可能的状态类别中，如果你周期性地报告各状态类别在整个需求中所占的百分比将会改

进项目的监控工作。假如你有清晰的要求，指定了允许修改状态信息的人员和每个状态变更应满足的条件，跟踪需求状态才能正常工作。工具能帮你跟踪每次状态改变的日期。

表16-1建议了几种需求状态。一些专业人员添了一些状态如“已设计”（表明功能需求部分已设计和评审出来了）或“已交付”（意味着修改后的软件已交付用户进行二版测试），保存一份已提出但没有批准的（即被拒绝状态）需求的记录是非常有用的，因为在开发中这些需求随时会被重新提出。

表16-1 建议的需求状态表

状态值	定义
已建议	该需求已被有权提出需求的人建议
已批准	该需求已被分析，估计了其对项目余下部分的影响（包括成本和对项目其余部分的干扰），已用一个确定的产品版本号或创建编号分配到相关的基线中，软件开发团队已同意实现该项需求
已实现	已实现需求代码的设计、编写和单元测试
已验证	使用所选择的方法已验证了实现的需求，例如测试和检测，审查该需求跟踪与测试用例相符。该需求现在被认为完成
已删除	计划的需求已从基线中删除，但包括一个原因说明和做出删除决定的人员

图16-2以图示的方法跟踪了一个假想为期10个月的项目持续过程中的需求状态，展示了在每个月底各个状态的系统需求的百分比。这个图并不能表示基线中的需求数量是否正在随时间而改变。但它说明了你是如何达到完全验证所有已获批准需求这个目标的。

对这些需求进行分门别类检测要比对每个需求的完成情况都检测要现实一些。软件开发者报告完成任务的百分比时，往往会过分乐观，常常对未完工的需求拥有较大的信心。趋向于“集中”（round up），他们的进展将导致软件项目或主要任务在很长的时间内处在百分之九十完成的状态这种情况。只有当指定的转换条件满足时才能改变需求的状态。某个状态的改变会导致更新需求跟踪能力矩阵，该矩阵指示与该需求相关的设计、代码、测试元素。这些将会在18章论述。

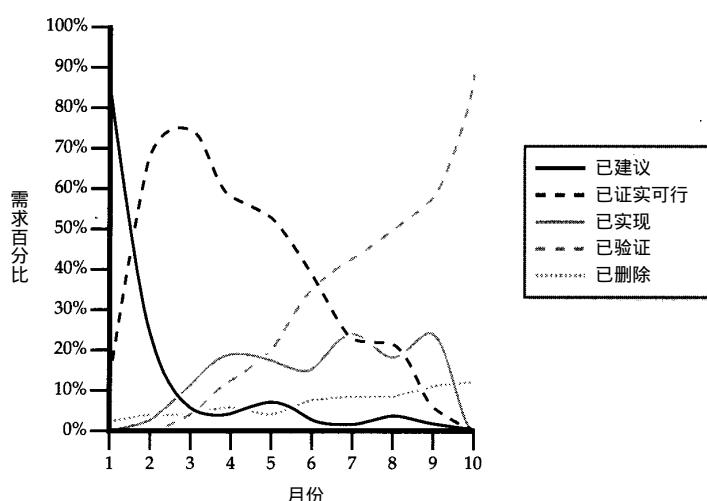


图16-2 在整个项目开发周期中跟踪需求状态的分布

16.5 度量需求管理的效果

在每个项目的工作分类细目结构中需求管理活动应该表现为分配有资源的任务。测算当前项目中的需求管理成本，是计划未来需求管理工作或经费的最佳途径。

一个从未度量过工程任何一个方面的组织通常发现很难开始保持一个耗时记录。测算实际开发和项目管理的工作量要求一个文化上的改变和养成记录日常工作的习惯。然而，测算并不像人们所担心的那样花费时间，了解成员花费在各个项目任务上的确切工作量会使你获得有价值的资料（Wiegers 1996a）。应该注意到工作量计算与翻过的日历时间不成正比，任务进度可能被打断或因同客户协商造成拖延。每个单元的工作时数总和表明一个任务的工作量，这个数据没必要随外界因素变化，但总体上却要比原计划长一些。

跟踪实际的需求管理效果能使你了解组员是否采取了措施进行需求管理。执行需求管理措施不得力，会由于不受约束的变更、范围延伸和遗漏需求等原因而增加项目的风险。考虑一下需求管理的下列活动的效果：

- 提出需求变更和已建议的新需求。
- 评估已建议的变更，包括影响分析。
- 变更控制委员会活动。
- 更新需求文档或数据库。
- 在涉及人员或团队中交流需求的变更。
- 跟踪和报告需求状态。
- 定义和更新需求跟踪能力信息。

尽管忽视和效率不高会随时发生，管理项目需求能确保你投资到需求的收集、归档和分析的努力没有白费。有效的需求管理策略能在整个开发过程中使项目参与者获悉需求的当前状态信息，从而减少大家在需求认识上的差距。

下一步：

- 恰当选择用来描述功能需求生存期的状态。在软件需求规格说明中定义每个需求的当前状态并且在开发过程中不断更新它。
- 定义一个确认需求文档的版本控制计划，把这个计划编写成文档并作为需求管理过程的一部分。
- 把组织用来管理每个项目需求的步骤描述下来。让几个分析员来起草、评审、实验并最终确定过程活动以保证可交付性。确保选择的过程步骤是实际可行和现实的，并且对这个项目有所帮助。

第17章 管理变更请求

一旦执行了软件过程评估，我询问项目组成员是如何接受产品的变更需求的，其中一位回答说“无论何时市场代表对 Bruce或Sandy提出变更要求，他们总是说同意，我们就只好努力去做出来”。他并未正面回答我的问题。不被控制的变更是项目陷入混乱、不能按进度执行或软件质量低劣的共同原因。为了使开发组织能够严格控制软件项目应确保以下事项：

- 应仔细评估已建议的变更。
- 挑选合适的人选对变更做出决定。
- 变更应及时通知所有涉及的人员。
- 项目要按一定的程序来采纳需求变更。

只有项目风险承担者在开发过程中能控制变更，他们才知道将交付什么，哪一项将会导致与目标的差距。对项目越深入了解后，你就越能发现采纳变更需求条件的苛刻。在需求文档中一定要反映项目的变更，需求文档应精确描述要交付的产品，这就是你的原则。要是软件需求文档同产品不一致，那它就毫无用处，甚至就象没有一个软件需求文档来指导开发组开发一样。

当你不得不做出变更时，应该按从高级到低级的顺序对被影响的需求文档进行处理。举个例子，一个已建议的变更可能影响一个使用实例和功能需求但不影响任何业务需求。改动高层系统需求能够影响多个软件需求。如果在最低层需求上做出变更，(典型的情况是一个功能性需求)，可能会导致需求同上层文档不一致。

17.1 控制项目范围的扩展

Capers Jones (1994) 在报告中声称扩展需求对百分之八十的管理信息系统项目和百分之七十的军事软件项目造成风险。扩展需求是指在软件需求基线已经确定后又要增添新的功能或进行较大改动。问题不仅仅是需求变更本身，而是迟到的需求变更会对已进行的工作有较大的影响。要是每个建议的需求都被采纳，对于项目出资者 (sponsor) 参与者与客户来说项目将永远也不会完成——事实上，这是不可能的。

对许多项目来说，一些需求的改进是合理的且不可避免。业务过程、市场机会、竞争性的产品和软件技术在开发系统期间是可以变更的，管理部门也会决定对项目做出一些调整。在你的项目进度表中应该对必要的需求改动留有余地。若不控制范围的扩展将使我们持续不断地采纳新的功能，而且要不断地调整资源、进度、或质量目标，这样做极其有害。这儿一点小的改动，那儿一点添加，很快项目就不可能按客户预期的进度和预期质量交付使用了。

管理范围扩展的第一步就是把新系统的视图、范围、限制文档化并作为业务需求的一部分，正如第6章所描述的。评估每一项建议的需求和特性，将它与项目的视图和范围相比较决定是否应该采纳它。强调客户参与的有效的需求获取方法能够减少遗漏需求的数量，只在做出提交承诺和分配资源后才采纳该需求 (Jones 1996a)。控制需求扩展的另一个有效的技术是原型法，这个方法能够给用户提供预览所有可能的实现，以帮助用户与开发者沟通从而准确

把握用户的真实需求 (Jones 1994)。

事实上，控制范围的扩展的方法是要敢于说“不” (Weinberg 1995)。很多人不喜欢说“不”，开发者只好在各种压力下接受每一项建议的需求。“客户总是对的”，“我们将使客户完全满意”这些话哲理上是正确的，一旦按此办事就要付出代价。忽视代价并不能改变“变更不免费”的事实。我知道一个成功的商业开发公司，它的首席执行官习惯于建议新的特色但开发管理者总是说“现在不行”。“现在不行”比简单地拒绝灵活很多，因为他暗含在后续版本中采纳其特色的希望。把客户提出的所有特色都采纳将会导致错过提交日期，质量的下滑 (slipshod)，开发人员的疲劳不堪。尽管客户并不总对，但他们是上帝，所以你应该尽可能在下一版本中满足他们的需求。

在理想的情况下，在开始构造前应该收集到所有新系统的需求，而且在开发中基本上不变更。这就是“瀑布”型软件开发生存期模型的前提，但在实践中，它却不太有效。当然，某种程度上，对特定的版本应该冻结需求，不再变更。然而，很早确定需求却忽视了有时候客户并不知道需要什么的现实，开发人员应该对用户这些需求变更作出响应。为了对付这些实际情况，你需要有根据地采纳变更过程。

17.2 变更控制过程

一个好的变更控制过程给项目风险承担者提供了正式的建议需求变更机制。通过这些处理过程，项目负责人 (leader) 可以在信息充分的条件下做出决策，这些决策通过控制产品生存期成本来增加客户和业务价值。你通过变更控制过程来跟踪已建议变更的状态，确保不会丢失或疏忽已建议的变更。一旦你确定了一个需求集合的基线，你应该对所有已建议的变更都遵循变更控制过程。

变更控制过程并不是给变更设置障碍。相反地，它是一个渠道和过滤器，通过它可以确保采纳最合适的变更，使变更产生的负面影响减少到最小。变更过程应该做成本档，尽可能简单，当然首要的是有效性。如果变更过程没有效率且冗长，又很复杂，大家宁愿用旧方法来做出变更决定 (或许他们应该那样做)。

控制需求变更同项目其它配置管理决策紧密相连。管理需求变更类似于跟踪错误和做出相应决定过程，相同的工具能支持这两个活动。然而记住它是工具而不是过程。使用商业问题跟踪工具来管理已建议的需求变更并不能代替写下变更需求的内容和处理的过程。

17.2.1 变更控制策略

项目管理应该达成一个策略，它描述了如何处理需求变更。策略具有现实可行性，要被加强才有意义。下述需求变更的策略是有用的：

- 所有需求变更必须遵循的过程，按照此过程，如果一个变更需求未被采纳，则其后过程不再予以考虑。
- 对于未获批准的变更，除可行性论证之外，不应再做其它设计和实现工作。
- 简单请求一个变更不能保证能实现变更，要由项目变更控制委员会 (CCB) 决定实现哪些变更 (本章将讨论 CCB)。
- 项目风险承担者应该能够了解变更数据库的内容。
- 绝不能从数据库中删除或修改变更请求的原始文档。

- 每一个集成的需求变更必须能跟踪到一个经核准的变更请求。

当然，大的变更会对项目造成显著的影响，而小的变更就可能不会有影响。原则上，应该通过变更控制过程来处理所有的变更。但实践中，可以将一些具体的需求决定权交给开发人员来决定。但只要变更涉及两个人或两个人以上都应该通过控制过程来处理。

有一个项目它由两大部分组成，一个是用户集成界面应用，另一个是内部知识库，但缺乏变更过程。当知识库开发人员改变了外部界面但没有将此变更通知应用开发人员，这个项目就碰到了麻烦。还有一个项目，开发人员在测试时才发现有人应用了新的已被修改的功能却没有通知小组中其余人员，导致重做了测试程序和用户文档。采用统一的变更控制方法可以避免这样的问题所带来的错误、开发的返工和耗费时间。

17.2.2 变更控制步骤

图17-1说明了一个描述一项变更控制步骤的模板，它能够应用于需求变更和其它项目变更。下面主要讨论关于过程是如何处理需求变更的。步骤中的4个组件和若干个过程描述将会是很有用的：

- 开始条件 (entry criteria) —— 在执行过程或步骤前应该满足的条件。
- 过程和步骤中所包含的不同任务 (task) 及项目中负责完成它们的角色。
- 验证 (verify) 任务正确完成的步骤。
- 结束条件 (exit criteria) —— 指出过程或步骤完成的条件。

- a. 绪论
 - a.1 目的
 - a.2 范围
 - a.3 定义
- b. 角色和责任
- c. 变更请求状态
- d. 开始条件
- e. 任务
 - e.1 产生变更请求
 - e.2 评估变更请求
 - e.3 作出决策
 - e.4 通知变更人员
- f. 验证
- g. 结束条件
- h. 变更控制状态报告
- 附录：存储的数据项

图17-1 简单变更控制步骤模板

a. 绪论

绪论主要说明此步骤 (procedure) 的目的，并且确定了步骤能够应用的范围。如果步骤仅仅适合特定产品中的变更，在绪论中应该明确表示。绪论还指明是否忽略特定种类的变更。例如对于项目开发过程中产生的过渡或临时产品，我们可能忽略掉变更，同时为了理解文档的其余部分定义了必要的条款。

b. 角色和责任

列出 (按角色分类，而非姓名顺序) 参与变更控制活动的项目组成员并且描述他们的责

任。表17-1提供了一些有关的角色。按你所处的环境和需要调整这些角色和相应的责任，在保持有效性的前提下尽可能使过程简单。一个人不必只担任一个角色。例如，项目管理者也可接收提交的变更需求。对于一些小项目，几个——也可能所有——角色均由一个人担任。

表17-1 变更管理活动中可能的项目角色

角 色	描述及责任
变更控制委员会主席	变更控制委员会的主席，在CCB意见不一致情况下可以独自做出决定
CCB	决定采纳或拒绝针对某项目所建议的变更请求的团体
评估者	应项目管理者要求分析所建议的变更带来影响的人员
修改者	负责实现已经被认可的请求变更，按时更新变更状态的人员
建议者	提交新变更请求的人
项目管理者	负责指定评估者和修改者的人员
请求接受者	接受提交变更请求的人
验证者	负责决定变更是否正确执行的人

c. 变更请求状态

一个变更请要求有一个生存期，相应地有不同的状态。可以使用状态转换图来表示这些状态的变化，如图17-2所示，只有当特定条件满足时才能更新状态。

d. 开始条件

变更控制步骤的基本开始条件是：

- 通过合适的渠道接受一个合法的变更请求。

所有潜在的建议者应该知道如何提交一个变更请求，是通过书面、通过基于Web的表单、或者发一个电子邮件，还是使用变更控制工具。将所有变更控制传递到一个联系点，且为每一个变更请求赋予统一的标识标签。

e. 任务

接收到一个新的变更要求后下一步是评估建议的技术可行性、代价、业务需求和资源限制。变更控制委员会主席要求评估者执行一个系统影响分析（详情见第18章）、风险分析、危害分析及其他评估。这些分析确保能很好理解接受变更所带来的潜在影响。评估者和变更控制委员会同样应考虑拒绝变更所带来的对业务和技术的影响。

制定决策的人应进入变更控制委员会，决定是采纳或还是拒绝请要的变更。CCB给每个采纳的变更需求设定一个优先级或变更实现日期，或将它分配给指定的产品。变更控制委员会通过更新请求状态和通知所有涉及到的小组成员来传达变更决定。相关人员可能不得不改变工作产品，如软件需求规格说明文档、需求数据库、设计模型、用户界面部件、代码、测试文档、用户文档。修改者在必要时应更新涉及的工作产品。

f. 验证

验证需求变更的典型方法是通过检查确保更新后的软件需求规格说明文档、使用实例文档、分析模型均正确反映变更的各个方面。使用跟踪能力信息找出受变更影响的系统的各个部分，然后验证他们实现了变更（详见第18章）。属于多个团组的成员可能会通过对下游工作产品测试或检查工作来参与验证变更工作。验证后，修改者安装更新后的部分工作产品并通过调试使之能与其它部分正常工作。

g. 退出条件

为了完成变更控制执行过程，下列退出条件应该得到满足：

下载

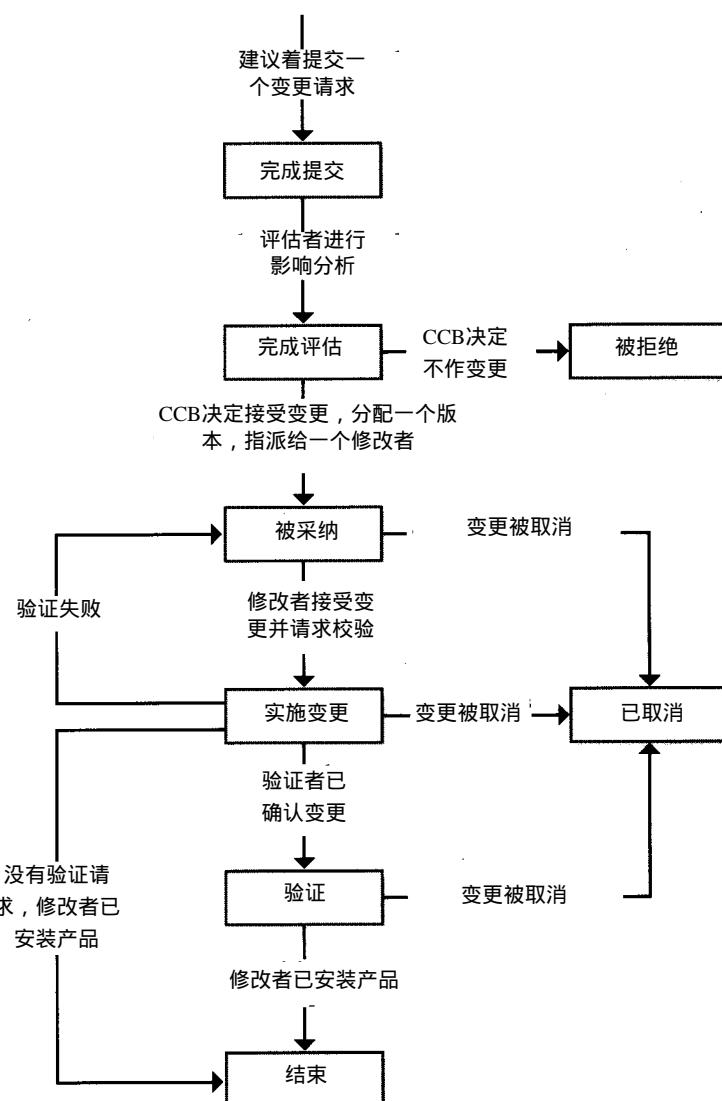


图17-2 变更需求状态转换图

- 请求的状态为“拒绝”，“结束”或“取消”。
- 所有修改后的工作产品安装至合适的位置。
- 建议者，变更控制主席，项目管理者和其他相关的项目参与者已经注意到了变更的细节和当前的状态。
- 已经更新需求跟踪能力矩阵（详见第18章）。

h. 变更控制状态报告

用报告、图表来总结变更控制数据库的内容和按状态分类的变更请求数量。描述产生报告的步骤。项目管理者通常使用这些报告来跟踪项目状态。

附录：存储的数据项

表17-2列出了每一个变更请求存储的一些数据项。当定义自己的列表时，指出什么是必选项，什么是可选项。同时指出每一项的值是由变更控制工具自动修正还是由指定的人员手

工修正。有经验后可能喜欢自己修正数据项，所以当你使用电子表格或一张纸实验处理过程后，再让自动工具着手这事。

表17-2 常见变更请求数据项

数据项名称	定 义
变更由来	请求变更的功能区域，可能包括的团体，有市场、管理、客户、软件工程、硬件工程和测试
变更要求 ID 号	分配给每个请求的标签或顺序号
变更类型	变更请求的类型，例如需求变更，建议性增改，错误报告
提交日期	提交变更请求的日期
更新日期	最近更新变更请求的日期
描述	以自由格式文本描述已请求的变更
实现优先级	由变更控制委员会赋予的每个变更的相对重要性，例如低、中、高
修改者	实现变更的主要负责人姓名
建议者	提交变更请求的人名，也可以存储与此人相关的信息
建议者设置的优先级	建议者赋予每个变更的相对重要性，例如：低，中，高
实现版本	计划中实现此变更的产品版本号
项目	要求变更的项目名称
反映文档	与每个变更相对应的反映文档，可以做出各种反映文档，所有反映文档均应保留
状态	变更请求的当前状态。可以从图 17-2 中选择相应状态
标题	对变更的简短总结（最好仅一行）
验证者	负责决定是否正确实现变更的人名

17.2.3 变更控制工具

自动工具能够帮助你有效地执行变更控制过程（Wiegers 1996a）。有许多人使用商业问题跟踪工具来收集、存储、管理需求变更。可以使用这些工具对一系列最近提交的变更建议产生一个列表给变更控制委员会开会时做议程用。问题跟踪工具也可以随时按变更状态分类报告变更请求的数目。

我曾参与了一个 Web 开发组，我们使用高级配置的问题管理工具来存储软件需求变更请求、问题报告、建议的产品增强、更新 Web 站点内容及新开发项目请求。由于工具的功能、经销商总是频繁变更，所以我给出任何具体的建议。但有一个关于软件工具信息的好站点，地址是 <http://www.methods-tools.com>。挑选工具时可以考虑以下几个方面：

- 可以定义变更请求的数据项。
- 可以定义变更请求生存期的状态转换图。
- 可以加强状态转换图使未经授权的用户仅能做出所允许的状态变更。
- 记录每一种状态变更的数据，确认做出变更的人员。
- 可以定义在提交新请求或请求状态被更新后应该自动通知的设计人员。
- 可以根据需要生成标准的或定制的报告和图表。

一些商业需求管理工具（详见第 19 章）内置有简单的变更建议系统。这些工具可以通过联系链从建议变更找到特定的需求，使得任何时候，无论谁作出变更请求，均可以通过 e-mail 及时通知涉及到的人员。

17.3 变更控制委员会

软件开发活动中公认变更控制委员会或 CCB (有时也称为结构控制委员会) 为最好的策略之一 (McConnell 1996)。变更控制委员会可以由一个小组担任，也可由多个不同的组担任，负责做出决定究竟将哪一些已建议需求变更或新产品特性付诸应用。典型的变更控制委员会同样决定在哪一些版本中纠正哪一些错误。许多项目已经有负责变更决策的人员，而正式组建变更控制委员、制定操作步骤会使他们更有效地工作。

广义上，变更控制委员会对项目中任何基线工作产品的变更都可做出决定，需求变更文档仅是其中之一。重大项目可以有几级控制委员会，有些负责业务决策（例如需求变更），另一些负责技术决策（Sorensen 1999）。有些变更控制委员会可以独立做出决策，而有些只是负责决策的建议工作。高级变更控制委员会做出的决策对计划的影响应比低级的大。小项目中，只需一两个人就可做出所有决策。

看到“变更控制委员会”这个词组，会使某些人想到一群高高在上而且浪费时间的官僚分子。然而，应该想到有变更控制委员会的企业结构可以帮助你很好地管理项目，哪怕是一个小项目。这个结构并不浪费时间或是累赘，相反会很有效率。一个有效率的变更控制委员会定期地考虑每个变更请求，并且基于对由此带来的影响和获益做出及时的决策。变更控制委员会只要能让合适的人做正确的事就足够了，不必追求大而全。

17.3.1 变更控制委员会的组成

变更控制委员会的成员应能代表变更涉及的团体。变更控制委员会可能包括如下方面的代表：

- 产品或计划管理部门。
- 项目管理部门。
- 开发部门。
- 测试或质量保证部门。
- 市场部或客户代表。
- 制作用户文档的部门。
- 技术支持部门。
- 帮助桌面或用户支持热线部门。
- 配置管理部门。

对于小项目只需几个人充当其中的一些角色就可以，并不一定要面面俱到。组建包含软、硬件两方面的项目的变更控制委员会时，也要包括来自硬件工程、系统工程、制造部门或者硬件质量保证和配置管理的代表。建立变更控制委员会在保证权威性的前提下应尽可能精简人员。大团队可能很难碰头和做出决策。确保变更控制委员成员明确担负的责任。有时为了获得足够的技术和业务信息，也可以邀请其他人员参加会议。

17.3.2 变更控制委员会总则

设立变更控制委员会的第一步是写一个总则，描述变更控制委员会的目的、授权范围、成员构成、做出决策的过程及操作步骤 (Sorensen 1999)。总则也应该说明举行会议的频度和

事由。管理范围是指该委员会能做什么样的决策，及哪种决策应上报到高一级的委员会。

1. 制定决策

制定决策过程（程式）的描述应确认：

- 变更控制委员会必须到会的人数或作出有效决定必须出席的人员数。
- 决策的方法，例如：投票、一致通过或其它机制。
- 变更控制委员会主席是否可以否决 CCB 的集体决定。

变更控制委员会应该对每个变更权衡利弊后做出决定。“利”包括节省的资金或额外的收入、增强的客户满意度、竞争优势、减少上市时间。“弊”是指接受变更后产生的负面影响，包括增加的开发费用、推迟的交付日期、产品质量的下降、减少的功能、用户不满意。如果估计的费用超过了本级变更控制委员会的管理范围，上报到高一级的委员会，否则用制订的决策程式来对变更做出决定。

2. 交流情况

一旦变更控制委员会做出决策时，指派的人员应及时更新数据库中请求的状态。有的工具可以自动通过电子邮件来通知相关人员。若没有这样的工具，就应该人工通知，以保证他们能充分处理变更。

3. 重新协商约定

变更总是有代价的。即使拒绝的变更也因为决策行为（提交、评估、决策）而耗费了资源。变更对新的产品特性会有很大的影响。向一个工程项目中增加很多功能，又要求在原先确定的进度计划、人员安排、资金预算和质量要求限制内完成整个项目是不现实的。当工程项目接受了重要的需求变更时，为了适应变更情况要与管理部门和客户重新协商约定（Humphrey 1997）。协商的内容可能包括：推迟“交货”时间、要求增加人手、推迟实现尚未实现的较低优先级的需求，或者质量上进行折衷。要是不能获得一些约定的调整，应该把面临的威胁写进风险管理计划中，这样当项目没有达到期望的结果时就不会有人惊奇了。

17.4 测量变更活动

软件测量是深入项目、产品、处理过程的调查研究，比起主观印象或对过去发生事情的模糊回忆要精确得多。测量方法的选择应该由所面临的问题和要达到的目标为依据。测量变更活动是评估需求的稳定性和确定某种过程改进时机（opportunity）的一种方法，这种时机可以减少未来的变更请求。需求变更活动的下列方面值得考虑（CMU/SEI 1995）：

- 接收、未作决定、结束处理的变更请求的数量。
- 已实现需求变更（包括增、删、改）的合计数量（也可以用在基线上占需求总数的百分比来表示）。
- 每个方面发出的变更请求的数量。
- 每一个已应用的需求（是指已划过基线）建议变更和实现变更的数量。
- 投入处理变更的人力、物力。

可以先用简单的测量法在组织中建立氛围，同时收集有效管理项目所需的关键数据。获得经验后就可以建立复杂的测量方法来管理项目。

图17-3说明了在开发过程中跟踪需求变更数目的一种方法。这个图跟踪新的需求变更建议出现的速率，类似地，还可跟踪采纳的变更需求数量。因为需求在不断变化，所以不必在

定基线前知道实现的变更需求数量。然而，一旦划好了需求基线，应遵循变更控制过程来处理建议的变更，并开始跟踪变更的频率（需求的稳定性）。这种图表的最终趋势应为零。持续高频率的变更隐含了项目超期的风险。同样也表明原来需求的基线确定不完善，应该改进需求获取的策略。

一个项目管理者应该知道频繁的需求变更会使产品不能按时交付。可以通过跟踪产生需求变更的来源深入剖析这个问题。图 17-4 说明了按来源分类的变更请求数量。项目管理者通过图 17-4 应该了解到销售部门造成的需求变更最多。这样，项目管理者就可以与市场代表和项目组一起讨论采取何种措施来减少销售部门提出的变更请求。以数据作为这种讨论的出发点比盲目地开一些面对面的会议更有建设性。

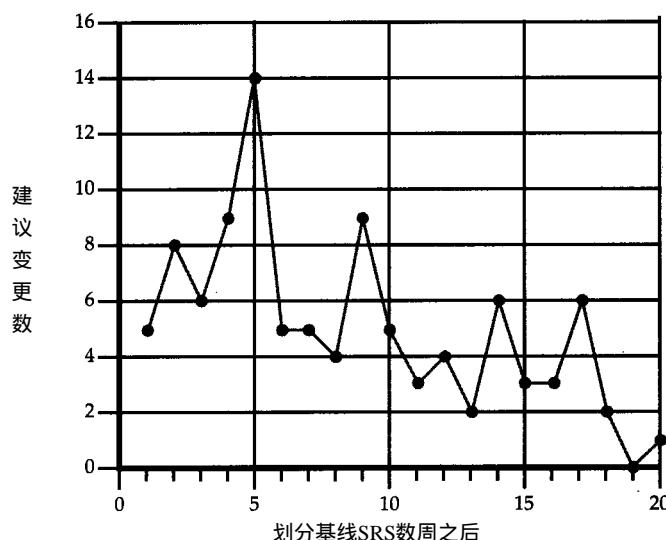


图 17-3 需求变化活动的样本图

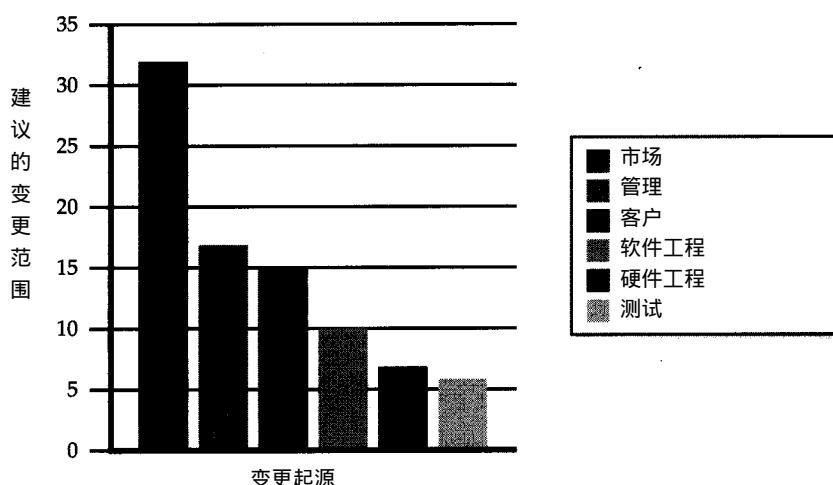


图 17-4 需求变更起源的样本图

现实中的软件项目均有需求变更。严格控制变更管理策略可以减少变更造成的混乱，改

进的需求开发技术可以减少面临的需求变更的数量。效率高的需求获取和管理策略将增强按时交付的能力。

下一步：

- 确定决策制订者并且建立一个变更控制委员会。制订变更委员会的总则，使每个人都理解变更控制委员会的目的、组成及决策制订过程。
- 参见图17-2，为项目建议的需求变更生存期制订一个状态转换图。制定一个描述项目组如何处理建议变更的过程。手工地使用这些过程，直到你确信它是实用的、有效的，并且简单到你能成功使用为止。
- 选择合适的商业问题跟踪工具，它们能与你所处的环境密切配合并且通过裁剪工具使可以支持以前开发出的变更控制过程。

第18章 需求链中的联系链

“开发工作进展如何，Glenn？”在一次项目状态检查会上“化学制品跟踪系统”项目经理Dave问道。

“我没有按计划执行”Glenn说，“我正在应Marcie的要求添加一个销售分类查询功能，比我原先预计的工作量超期多了。”

Dave似乎有点迷惑，“好像在最近的变更控制委员会的会议中我们没有讨论过这个功能。Marcie是通过变更过程来提交要求的吗？”

“没有，她直接给了我这个建议，”Glenn说，“本该请她通过正式渠道的，但这个功能看上去较简单，所以当时我就答应她了。这个功能其实并不简单，每次当我认为该完工了，但总能意识到在另一个文件中漏了一个变更，所以不得不修改它，再测试一遍。原以为花4个小时就可以了，实际上花了4天时间，造成我没能按计划完成任务。我知道延误了工期，那我应该加上这个功能还是放弃它呢？”

一个表面上简单的软件变更往往很复杂，花费时间很难预料。经常很难发现哪怕一个很小的修改会影响到的范围。开发过程中在同意接受建议的变更之前，要确信明白自己在做什么。

本章讲述开发过程中的需求跟踪和需求变更影响分析的相关内容。需求跟踪包括编制每个需求同系统元素之间的联系文档。这些元素包括别的需求、体系结构、其他设计部件、源代码模块、测试、帮助文件、文档等。跟踪能力信息使变更影响分析十分便利，有利于确认和评估实现某个建议的需求变更所必须的工作。

18.1 需求跟踪

跟踪能力（联系）链（traceability link）使你能跟踪一个需求使用期限的全过程，即从需求源到实现的前后生存期（Gotel and Finkelstein 1994）。第1章指出跟踪能力是优秀需求规格说明书的一个特征。为了实现可跟踪能力，必须统一地标识出每一个需求，以便能明确地进行查阅（Davis 1993）。

图18-1说明了四类需求跟踪能力链（Jarke 1998）。客户需求可向前追溯到需求，这样就能区分出开发过程中或开发结束后由于需求变更受到影响的需求。这也确保了需求规格说明书包括所有客户需求。同样，可以从需求回溯相应的客户需求，确认每个软件需求的源头。如果用使用实例的形式来描述客户需求，图18-1上半部分就是使用实例和功能性需求之间的跟踪情况。

图18-1的下半部分指出：由于开发过程中系统需求转变为软件需求、设计、编写等，所以通过定义单个需求和特定的产品元素之间的（联系）链可从需求向前追溯。这种联系链使你知道每

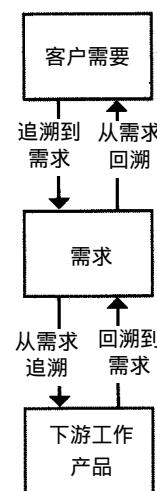


图18-1 四类需求可跟踪能力

一个需求对应的产品部件，从而确保产品部件满足每个需求。第四类联系链是从产品部件回溯到需求，使你知道每个部件存在的原因。绝大多数项目不包括与用户需求直接相关的代码，但对于开发者却要知道为什么写这一行代码。如果不能把设计元素、代码段或测试回溯到一个需求，你可能有一个“画蛇添足的程序”。然而，若这些孤立的元素表明了一个正当的功能，则说明需求规格说明书漏掉了一项需求。

跟踪能力联系链记录了单个需求之间的父层、互连、依赖的关系。当某个需求变更（被删除或修改）后，这种信息能够确保正确的变更传播，并将相应的任务作出正确的调整。图18-2说明了许多能在项目中定义的直接跟踪能力联系链。一个项目不必拥有所有种类的跟踪能力联系链，要根据具体的情况调整。

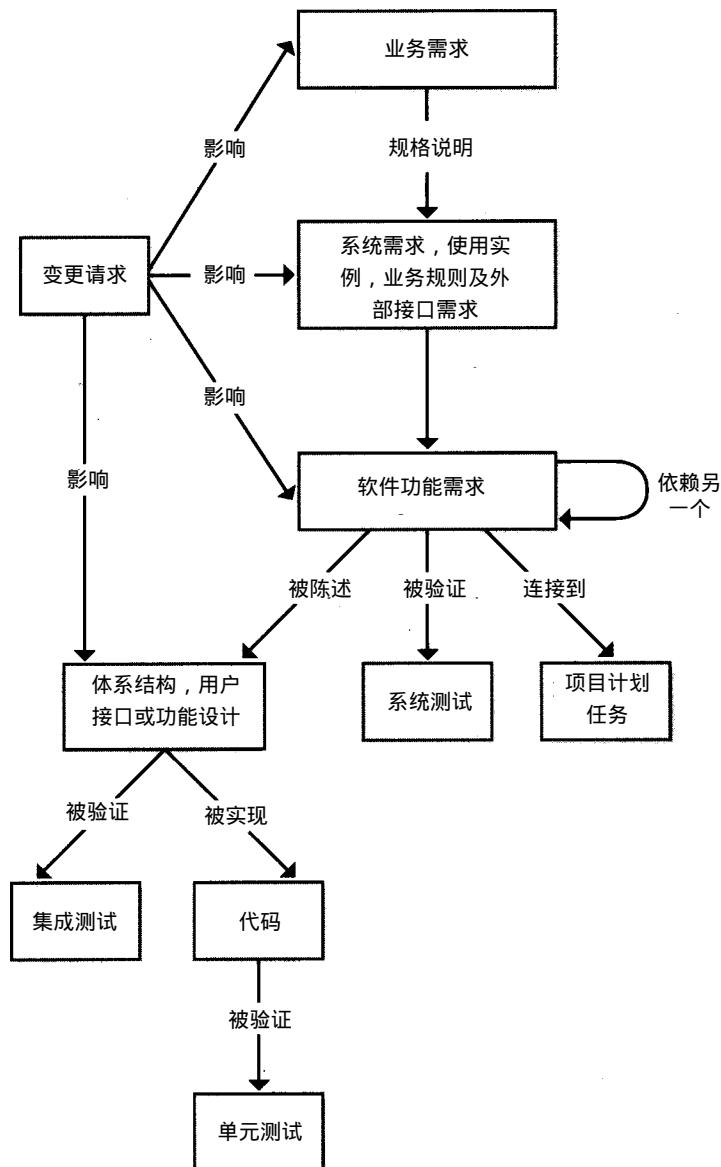


图18-2 一些可能的需求跟踪能力联系链

18.1.1 需求跟踪动机

我有一个尴尬的开发经历，其实是忽略了一个需求，所以在我自认为完成编程后，不得不返工编写额外的代码。就是因为忘了，后来不得不返工。如果忽略某几个需求造成用户不满意或发布一个不符合要求的产品，那就不仅仅是尴尬了。在某种程度上，需求跟踪提供了一个表明与合同或说明一致的方法。更进一步，需求跟踪可以改善产品质量，降低维护成本，而且很容易实现重用（Ramesh 1998）。

需求跟踪是个要求手工操作且劳动强度很大的任务，要求组织提供支持。随着系统开发的进行和维护的执行，要保持关联链信息与实际一致。跟踪能力信息一旦过时，可能再也不会重建它了。由于这些原因，应该正确使用需求跟踪能力（Ramesh et al. 1995）。下面是在项目中使用需求跟踪能力的一些好处：

- 审核（certification） 跟踪能力信息可以帮助审核确保所有需求被应用。
- 变更影响分析 跟踪能力信息在增、删、改需求时可以确保不忽略每个受到影响的系统元素。
- 维护 可靠的跟踪能力信息使得维护时能正确、完整地实施变更，从而提高生产率。要是一下子不能为整个系统建立跟踪能力信息，一次可以只建立一部分，再逐渐增加。从系统的一部分着手建立，先列表需求，然后记录跟踪能力链，再逐渐拓展。
- 项目跟踪 在开发中，认真记录跟踪能力数据，就可以获得计划功能当前实现状态的记录。还未出现的联系链意味着没有相应的产品部件。
- 再设计（重新建造） 你可以列出传统系统中将要替换的功能，记录它们在新系统的需求和软件组件中的位置。通过定义跟踪能力信息链提供一种方法收集从一个现成系统的反向工程中所学到的方法。
- 重复利用 跟踪信息可以帮助你在新系统中对相同的功能利用旧系统相关资源。例如：功能设计、相关需求、代码、测试等。
- 减小风险 使部件互连关系文档化可减少由于一名关键成员离开项目带来的风险（Ambler 1999）。
- 测试 测试模块、需求、代码段之间的联系链可以在测试出错时指出最可能有问题的代码段。

以上所述许多是长期利益，减少了整个产品生存期费用，但同时要注意到由于积累和管理跟踪能力信息增加了开发成本。这个问题应该这样来看，把增加的费用当作一项投资，这笔投资可以使你发布令人满意同时更容易维护的产品。尽管很难计算，但这笔投资在每一次修改、扩展或代替产品时都会有所体现。如果在开发工程中收集信息，定义跟踪能力联系链一点也不难，但要在整个系统完成后再实施代价确实很大。

CMM (capability maturity model) 的第三层次要求具备需求跟踪能力（CMU/SEI 1995），软件产品工程活动的十个关键处理领域有关于它的陈述，“在软件工作产品之间，维护一致性。工作产品包括软件计划，过程描述，分配需求，软件需求，软件设计，代码，测试计划，以及测试过程。”需求跟踪过程中还定义了一些关于一个组织如何处理需求跟踪能力的期望。

18.1.2 需求跟踪能力矩阵

表示需求和别的系统元素之间的联系链的最普遍方式是使用需求跟踪能力矩阵。表 18-1

展示了这种矩阵，这是一个“化学制品跟踪系统”实例的跟踪能力矩阵的一部分。这个表说明了每个功能性需求向后连接一个特定的使用实例，向前连接一个或多个设计、代码和测试元素。设计元素可以是模型中的对象，例如数据流图、关系数据模型中的表单、或对象类。代码参考可以是类中的方法，源代码文件名、过程或函数。加上更多的列项就可以拓展到与其它工作产品的关联，例如在线帮助文档。包括越多的细节就越花时间，但同时很容易得到相关联的软件元素，在做变更影响分析和维护时就可以节省时间。

表18-1 一种需求跟踪能力矩阵

使 用 实 例	功 能 需 求 量	设 计 元 素	代 码	测 试 实 例
UC-28	catalog.query.sort	class	catalog.sort()	search.7
		catalog		search.8
UC-29	catalog.query..import	class	catalog..import()	search.8
		catalog	catalog.validate()	search.13 search.14

跟踪能力联系链可以定义各种系统元素类型间的一对一，一对多，多对多关系。表 18-1 允许在一个表单元中填入几个元素来实现这些特征。这里是一些可能的分类：

- 一 对 一 一个代码模块应用一个设计元素。
- 一 对 多 多个测试实例验证一个功能需求。
- 多 对 多 每个使用实例导致多个功能性需求，而一些功能性需求常拥有几个使用实例。

手工创建需求跟踪能力矩阵是一个应该养成的习惯，即使对小项目也很有效。一旦确立使用实例基准，就准备在矩阵中添加每个使用实例演化成的功能性需求。随着软件设计、构造、测试开发的进展不断更新矩阵。例如，在实现某一功能需求后，你可以更新它在矩阵中的设计和代码单元，将需求状态设置为“已完成”。

表示跟踪能力信息的另一个方法是通过矩阵的集合，矩阵定义了系统元素对间的联系链。例如：

- 一 类 需 求 与 另 一 类 需 求 之 间。
- 同 类 中 不 同 的 需 求 之 间。
- 一 类 需 求 与 测 试 实 例 之 间。

可以使用这些矩阵定义需求间可能的不同联系，例如：指定 /被指定、依赖于、衍生为以及限制/被限制（Sommerville and Sawyer 1997）。

表18-2说明了两维的跟踪能力矩阵。矩阵中绝大多数的单元是空的。每个单元指示相对应行与列之间的联系，可以使用不同的符号明确表示“追溯到”和“从……回溯”或其他联系。表18-2使用一个箭头表示一个功能性需求是从一个使用实例追溯来的。这些矩阵相对于表18-1中的单跟踪能力表更容易被机器自动支持。

表18-2 反映使用实例与功能需求之间联系的需求跟踪能力矩阵

功 能 需 求	使 用 实 例			
	UC-1	UC-2	UC-3	UC-4
FR-1				
FR-2				
FR-3				

(续)

功能需求	使用实例			
	UC-1	UC-2	UC-3	UC-4
FR-4				
FR-5				
FR-6				

跟踪能力联系链无论谁有合适的信息都可以定义。表 18-3 定义了一些典型的知识源，即关于不同种类源和目标对象间的联系链。定义了可以为工程项目提供每种跟踪能力信息的角色和个人。

表18-3 跟踪能力联系链可能的信息源

链的源对象种类	链的目的对象种类	信 息 源
系统需求	软件需求	系统工程师
使用实例	功能性需求	需求分析员
功能性需求	功能性需求	需求分析员
功能性需求	软件体系结构元素	软件体系结构(设计)者
功能性需求	其他设计元素	开发者
设计元素	代码	开发者
功能性需求	测试实例	测试工程师

18.1.3 需求跟踪能力工具

由于联系链源于开发组成员的头脑中，所以需求跟踪能力不能完全自动化。然而，一旦已确定联系链，特定工具就能帮你管理巨大的跟踪能力信息。可以使用电子数据表来维护几百个需求的矩阵，但更大的系统需要更“鲁棒”的解决办法。

第19章描述了具有强大需求跟踪能力的商业需求管理工具。这些工具均使用如表 18-2 的跟踪能力矩阵。可以在工具的数据库中存储需求和其他信息，定义不同对象间的联系链，甚至包括同类需求的对等联系链。有一些工具需要区分“追溯到(跟踪进)”与“从……回溯(跟踪出)”关系，自动定义相对的联系链。这就是说，如果你指出需求 R 追溯到测试实例 T，工具会自动定义相对的联系“T 从 R 回溯”。还有一些工具可以在联系链某端变更后将另一端标为“可疑”。可以让你检查确保知道变更的后续效果。

这些工具允许定义“跨项目”或“跨子系统”的联系链。我了解到一个有 20 个子系统的大项目，某些高层产品需求建立在多个子系统之上。有些情况下，分配给一个子系统的需求，实际上是由另一个子系统提供的服务完成的。这样的项目采用商业需求管理工具可以成功地跟踪这些复杂的跟踪能力关系。

18.1.4 需求跟踪能力过程

当你应用需求跟踪能力来管理工程时，可以考虑下列步骤：

- 1) 决定定义哪几种联系链，可以参见图 18-2 来进行。
- 2) 选择使用的跟踪能力矩阵的种类，是表 18-1 还是表 18-2。
- 3) 确定对产品哪部分维护跟踪能力信息。由关键的核心功能、高风险部分或将来维护量

大的部分开始做起。

- 4) 通过修订过程和核对表来提醒开发者在需求完成或变更时更新联系链。
- 5) 制定标记性的规范，用以统一标识所有的系统元素，达到可以相互联系的目的（Song et al. 1998）。若必要，作文字记录，这样就可以分析系统文件，便于重建或更新跟踪能力矩阵。
- 6) 确定提供每类联系链信息的个人。
- 7) 培训项目组成员，使其接受需求跟踪能力的概念和了解重要性、这次活动的目的、跟踪能力数据存储位置、定义联系链的技术——例如，使用需求管理工具的特点。确保与会人员明白担负的责任。
- 8) 一旦有人完成某项任务就要马上更新跟踪能力数据，即要立刻通知相关人员更新需求链上的联系链。
- 9) 在开发过程中周期性地更新数据，以使跟踪信息与实际相符。要是发现跟踪能力数据没完成或不正确那就说明没有达到效果。

18.1.5 需求跟踪能力可行吗，必要吗？

你可能会认为建造一个需求跟踪能力矩阵与它的价值相比麻烦多过益处，或者虽然简单但却不实用。看一下这个例子：某个会议的参加者在飞机制造厂工作，有一次他告诉我他所在公司最新型的喷气客机用户需求说明书足有一人高，而且有完整的需求跟踪能力矩阵。我非常高兴听到开发人员如此严格地管理他们的软件需求。对有很多子系统的巨大产品进行跟踪能力管理是一项巨大的工程，但他们知道这很必要。

并不是所有的公司都会因为软件问题而造成严重的结果，然而应该严肃地对待需求跟踪，尤其对涉及你业务核心的信息系统。有一次当我在一个专题会上描述完跟踪能力后，在场的一个大公司的CEO问道：“为什么不对战略商业系统作一个跟踪能力矩阵？”这是一个很好的问题。考虑了应用技术的成本和不使用的风险后，才能决定是否使用任何改进的需求工程实践。随着软件的发展，要把时间投向回报丰厚的地方。

18.2 变更需求代价：影响分析

许多开发人员有过类似本章正文前的那种对话情况。一个表面上很简单的变更转变成很复杂的局面。只要允许需求变更或添加新特性，这种情况就免不了。开发人员往往对建议的软件变更成本或其它衍生结果不——或不能——提供出准确的评估。“变更是免费的”这种误解是造成项目范围延伸的一个原因。人们往往只有在知道变更的成本后才能做出理智的选择。

影响分析是需求管理的一个重要组成部分（Arnold and Bohner 1998）。影响分析可以提供对建议的变更的准确理解，帮助做出信息量充分的变更批准决策。通过对变更内容的检验，确定对现有的系统做出是修改或抛弃的决定，或者创建新系统以及评估每个任务的工作量。进行影响分析的能力依赖于跟踪能力数据的质量和完整性。

没有人愿意做一个费时费力还要担心意想不到情况的需求变更。在职业生涯中，绝大多数开发人员会遇到要求添加“没有代价且不影响进度的变更”的要求。对这样令人奇怪的要求的正确回答是“不行，”变更只能在项目时间、预算、资源的限制内进行协商。

18.2.1 影响分析过程

项目变更控制委员会通常会请资深开发人员对提出的需求变更申请进行影响分析。为了帮助影响分析员理解接受一个建议变更的影响，可设计一系列问题核对表，如图 18-3所示。第二个核对表如图 18-4，用来帮助确定涉及的软件元素。熟练以后，可以按照具体情况修改。

- 基准（线）中是否已有需求与建议的变更相冲突？
- 是否有待解决的需求变更与已建议的变更相冲突？
- 不采纳变更会有什么业务或技术上的后果？
- 进行建议的变更会有什么样的负面效应或风险？
- 建议的变更是否会不利于需求实现或其它质量属性？
- 从技术条件和员工技能的角度看该变更是否可行？
- 若执行变更是否会在开发、测试和许多其它环境方面提出不合理要求？
- 实现或测试变更是否有额外的工具要求？
- 在项目计划中，建议的变更如何影响任务的执行顺序、依赖性、工作量或进度？
- 评审变更是否要求原型法或别的用户提供意见？
- 采纳变更要求后，浪费了多少以前曾做的工作？
- 建议的变更是否导致产品单元成本增加？例如增加了第三方产品使用许可证的费用。
- 变更是否影响任何市场营销、制造、培训或用户支持计划？

图18-3 建议的变更涉及的问题核对表

- 确认任何用户接口要求的变更、添加或删除。
- 确认报告、数据库或文件中任何要求的变更，添加或删除。
- 确认必须创建、修改或删除的设计部件。
- 确认源代码文件中任何要求的变更。
- 确认文件或过程中任何要求的变更。
- 确认必须修改或删除的已有的单元、集成或系统测试用例。
- 评估要求的新单元、综合和系统测试实例个数。
- 确认任何必须创建或修改的帮助文件、培训素材或用户文档。
- 确认变更影响的应用、库或硬件部件。
- 确认须购买的第三方软件。
- 确认在软件项目管理计划、质量保证计划和配置管理计划等中变更将产生的影响。
- 确认在修改后必须再次检查的工作产品。

图18-4 变更影响的软件元素核对表

下面是一个评估需求变更影响的简单例子。许多评估问题的出现是因为评估者没有完全按此行事。所以，这个影响分析方法强调广泛的任务确认。对于重大的变更，小组——不只是一个开发者——应该做影响分析和工作量估算来确保不忽略重要的任务。

- 1) 按图18-3进行一遍。
- 2) 按图18-4进行一遍，要使用有效的跟踪能力信息。有一些需求管理工具包含影响分析报告并且能发现受变更影响的系统元素。
- 3) 使用如图18-5所示的一张工单（worksheet）来评估预期任务要求的工作量，绝大多数的变更仅要求工单所列任务的一部分。
- 4) 求评估工作值的总和。
- 5) 确认任务执行的顺序，这些任务如何同当前的计划任务配合？

6) 决定变更是否处于项目的临界路径。如果一个处于关键路径的任务延期，项目的完成之日将遥遥无期。每个变更都会消耗资源，如果能避免变更影响关键任务，则变更不会造成整个项目延期。

- 7) 估计变更如何影响进度和费用。
- 8) 通过与其它任意需求的收益、代价、成本和技术风险的比较来评估变更的优先级（有关详情请见第13章）。
- 9) 向变更控制委员会报告影响分析结果，他们可以在采纳或拒绝变更的决策过程中使用这些评估信息。

在绝大多数实例中，完成这些过程不会超过一到二小时。对于一个繁忙的开发人员来说似乎浪费了很多时间，其实为了确保开发人员对有限的资源的精打细算，这是一笔很小的投资。如果不用系统的评估技能从容地评估出变更影响，就去做吧；但要小心别让自己进入“流沙区”。

工作量 (劳动时数)	任 务
——	更新软件需求规格说明书或需求数据库
——	开发并评估原型
——	创建新的设计部件
——	修改已有的设计部件
——	开发新的用户界面部件
——	修改已有的用户界面部件
——	开发新的用户文档和帮助文件
——	修改已有的用户文档和帮助文件
——	开发新的源代码
——	修改已有的源代码
——	购买和集成第三方软件
——	修改构造文件
——	开发新单元测试和综合测试
——	进行单元测试和综合测试
——	写新的系统测试实例
——	修改已有的系统测试实例
——	修改自动测试驱动程序
——	进行回归测试
——	开发新报告
——	修改已有的报告
——	开发新的数据库元素
——	修改已有的数据库元素
——	开发新的数据文件
——	修改已有的数据文件
——	修改各种项目计划
——	更新别的文档
——	更新需求跟踪能力矩阵
——	检查工作产品
——	根据测试和检查情况返工
——	总计劳动时数

图18-5 评估需求变更的劳动时数

18.2.2 影响分析报告模板

图18-6是一个模板，可以用来报告进行需求变更的影响分析。使用模板可以帮助变更控制委员会找到有用的信息来作出正式的决策。实现此项变更的开发人员可能需要详细的分析情况和工作量计划工单，但变更控制委员会仅需要影响分析的总结。可以视情况调整模板。

变更请求ID_____
标题_____
描述_____
分析者_____
日期_____
优先权评估：
相关收益_____ (1-9)
相关代价_____ (1-9)
相关成本_____ (1-9)
相关风险_____ (1-9)
最终优先级_____
预计总耗时_____ 劳动时数
预计损时_____ 劳动时数
预计对进度的影响_____ 天数
额外的成本影响_____ 金额
质量影响_____
被影响的其他需求_____
被影响的其他任务_____
要更新的计划_____
综合的事项_____
生存期成本事项_____
可能的变更所需检查的其他部件_____

图18-6 影响分析报告模板

下一步：

- 为你当前开发的系统最核心部件建立一个 15~20需求的跟踪能力矩阵。尝试一下表 18-1和18-2的方法。随着项目的开发逐步扩展矩阵。评估一下什么方法最有效以及什么样的收集和存储跟踪能力信息的方法最适合你。
- 下一回评估一个需求变更请求，首先用旧方法来预算耗时，再用本章的影响分析方法计算一下。当最终实现了这个变更，再比较一下两个方法哪个估算更准确。随着经验的积累可以修改核对表和工单。
- 当你有机会对一个文档不全的项目做维护时，应该用反向工程的分析方法来处理要修改的部分，并且把经念教训记录下来。对你所处理的难题建立一部分跟踪能力矩阵，以便以后在其基础之上工作的人有据可查。当你的工作小组继续维护该产品时，进一步扩充跟踪能力矩阵。

第19章 需求管理工具

本章以前的章节讨论了建立自然语言的软件需求规格说明，该说明书除了包括业务需求和使用实例的书面文档以外，还包含功能性和非功能性的需求。基于文档存储需求的方法有若干限制。例如：

- 很难保持文档与现实的一致。
- 通知受变更影响的设计人员是手工过程。
- 不太容易做到为每一个需求保存增补的信息。
- 很难在功能需求与相应的使用实例、设计、代码、测试和项目任务之间建立联系链。
- 很难跟踪每个需求的状态。

需求管理工具使用多用户数据库保存与需求相关的信息，让你不必担心以上的问题。小一点的项目可以使用电子表格或简单的数据库管理需求，既保存需求文本，又保存它的几个属性（Sommerville and Sawyer 1997）。大项目可以从使用商业需求管理工具中获益，其中包括让用户从源文档中产生需求，定义属性值，操作和显示数据库内容，让需求以各式各样的形式表现出来，定义跟踪能力联系链，让需求同其他软件开发工具相连等功能。在考虑自行开发工具前先调查一下是否有可用的成熟工具。

我把这些工具称为需求管理而不是需求开发工具。这些工具不会帮助你确认未来的客户或者从项目中获得正确的需求。然而，你可以获得许多灵活性，可用来在整个开发期间管理需求的变动，使用需求作为设计、测试、项目管理的基础。这些工具不会代替已定义用来描述如何获取和管理需求的处理过程。尽管其他方法同样可以完成工作，但为了高效率就应该使用工具。不要试图把使用工具作为缺乏方法、训练或理解的补充。

从本章节中可以获知使用需求管理工具的好处和需求管理工具一般所具有的功能。表 19-1列出了一些这样的商业需求管理工具。本章节不涉及产品之间横向比较，因为这些工具更新速度较快。甚至价格、支持平台、卖主均变动频繁。可以使用表 19-1中的Web地址获得有关工具的最近信息，注意，这些 Web地址本身也可能变化。有关这些需求管理工具的特性比较及其它几个工具的介绍可以查阅系统工程国际委员会的网址（Web地址 <http://www.incose.org/toc.html>），该网址上同时还提供了如何挑选需求管理工具的指导（Jones et al. 1995）。

这些工具最大的区别是以数据库还是以文档为核心。以数据库为核心的产品（例如 Caliber-RM 和 DOORS）把所有的需求、属性和跟踪能力信息存储在数据库中。依赖于这样的产品，数据库可以或是商业（通用）的或是专有的，关系型或面向对象的。可以从不同的源文档中产生需求，但结果都存在数据库中。在大多数情况下需求的文本描述被简单地处理为必须的属性。有一些产品可以把每个需求与外部文件相联系（微软的 Word 文件，Excel 文件，图形文件，等等）。通过这些文件提供额外补充性的需求说明。

以文档为核心的方法使用 Word 或 Adobe 公司的 FrameMaker 等字处理程序制作和存储文档。RequisitePro 通过允许选择文档作为离散需求存储在数据库中以加强以文档为核心的处理方法的能力。只要需求存储在数据库中，你可以定义属性和跟踪能力联系链，如同以数据库为核

心的工具。该工具同时提供一些机制同步数据库和文档的内容。QSSrequireit不使用分离的数据库；而是在Word需求文档中的文本后面插入一个属性表。RTM Workshop两方面都包括在内，尽管是以数据库为核心，但允许从Word中维护需求。

表19-1 一些商业的需求管理工具

工具	卖主	以数据库或文档为核心
Caliber-RM	Technology Builders, Inc., http://www.tbi.com	以数据库为核心
DOORS	Quality Systems and Software, Inc., http://www.qssinc.com	以数据库为核心
QSSrequireit	Quality Systems and Software, Inc., http://www.qssrequireit.com	以文档为核心
RequisitePro	Rational Software Corporation http://www.rational.com	以文档为核心
RTM Workshop	Integrated Chipware, Inc http://www.chipware.com	以数据库为核心
Vital Link	Compliance Automation, Inc., http://www.complianceautomation.com	以文档为核心

除了入门级的QSSrequireit，其他工具都不便宜，但相对于需求管理的高成本而言，投资购买工具还是值得的。注意，工具的代价不仅仅只是许可证费。还包括每年的维护费用，还应有安装软件，执行管理，获得开发商的支持和咨询，训练用户，升级的费用。购买前一定要权衡利弊。

19.1 使用需求管理工具的益处

即使你善于收集项目的需求，在开发过程中自动化的工具仍能可以帮助你处理这些需求。随着开发的进行，开发组成员慢慢记不清需求细节，这时商业需求管理工具就变得十分有用。以下是一些工具可以帮你执行的任务：

1) 管理版本和变更 项目应定义需求基线，基线是每个版本所包括的需求的集合。一些需求管理工具提供灵活的设定基线功能。这些工具可以自动维护每个需求的变动历史，这比手工操作要优越得多。可以记录变更决定的基本原则并可根据需要返回到以前的需求版本。通常这些工具包括一个内建的变动建议系统，它可以与变更请求所涉及的需求直接联系。

2) 存储需求属性 对每一个需求应该保存一些属性，正如16章描述的，有关人员应能看到这些属性，选择合适的人员更新这些属性值。需求管理工具产生几个系统定义的属性（例如，需求创建日期和版本号），同时允许定义不同数据类型的其它属性。可以通过排序，过滤，查询数据库来显示满足属性要求的需求子集。

3) 帮助影响分析 通过定义不同种类的需求，子系统的需求，单个子系统和相关系统部件——例如：例子、设计、代码和测试——等各个部分之间的联系链，工具可以确保需求跟踪。联系链可以帮助用来对特定需求所做的变动进行影响分析，即通过确定影响涉及的系统部件来做到这一点。最好的是这些工具可以查到功能需求的来源。

4) 跟踪需求状态 利用数据库保存需求可以很容易知道某个产品包含的所有需求。在开发中跟踪每个需求的状态将可以支持项目的全程跟踪。当项目管理者知道某个项目的下一版

本中的百分之五十五的需求已经验证过了，百分之二十八已经实现但还没有验证，百分之十七还没有实现时，他就对项目状况有了很好的了解。

5) 访问控制 可以对个人、用户小组确定访问权限。绝大多数工具允许共享需求信息，对于地域上分散的组可以通过 Web 网页使用数据库。数据库在需求这一级别通过锁机制进行多用户管理。

6) 与风险承担者进行沟通 典型的需求管理工具允许小组成员通过多线索电子对话讨论需求。当讨论达成一个新的结果时或某个需求修改后，自动电子邮件系统就会通知涉及的人员。

7) 重用需求 由于在数据库中保存了需求，在其他项目或子项目中重用需求变为可能。还可以避免信息冗余。

19.2 商业需求管理工具

商业需求管理工具允许定义不同种类的数据库元素，例如业务需求、使用实例、功能性需求、硬件需求、非功能性需求和测试。这样就可以区分软件需求规格说明中的需求对象及其它有用信息。所有的工具提供了强大的功能用来定义每类需求的属性，这一点是它们相对于基于文本的软件需求规格说明方法的优势。

绝大多数需求管理工具某种程度上同 Word 集成，典型的方式是在 Word 上添加工具条。但 Vital Link 是基于 FrameMaker，而不是 Word。高级的工具提供丰富的输入、输出文件格式。有些工具允许从文档中挑选特定的文本，把它们看作离散需求，就如同在数据库中添加新需求。当你挑选好作为需求的文本时，工具通常高亮显示需求然后插入到 Word 书签和隐藏的文本中。还可以把文档编成不同的风格来扩展每个需求。文字处理后的文档可能不太完美，但可以通过使用文档风格和关键字来纠正。

工具对每个需求不仅有统一的内部标识符，还支持层次编码的数字标签。这些标识符通常是一个短文本字首，例如 UR 代表用户需求（User Requirement），之后再跟一个唯一的整数。高级的工具提供类似于 Windows 资源管理器的层次显示方法用来操作需求层次树。DOORS 工具可以使你看到层次结构的软件需求说明书。

工具的输出能力包括以用户定义格式或表单报告格式生成需求文档的能力。Caliber-RM 强大的文档加工功能（称为“Document Factory”）使你能在 Word 中用简单的命令定义一个软件规格说明模板，以指示页面布局、样板文本、从数据库中选取的属性及使用文字的方式。Document Factory 以用户定义的查询条件从数据库中筛选信息，并用所定义的模板产生一个定制的文档。因此，软件需求规格说明本质上是一个产生自数据库筛选内容的报告。

所有的工具都有在需求同其他系统元素间定义联系链的健壮跟踪能力。RTM Workshop 允许为每个项目中的存储对象类别建立一个 ER 图，从而为项目定义一个由 ER 图组成的类别图表。通过定义两类别中（或同类别的）对象的联系链和基于图表中定义的类别联系可以实现跟踪能力。当完成以上工作后，一旦某个变更被采纳，工具自动根据跟踪信息把涉及的需求表示为“可疑的”。从而帮助你分析需求变更的影响。

其他特点还包括：建立用户小组，定义用户或用户小组对项目、需求、属性和属性值的读、写、创建和删除权限。甚至还有些工具允许把非文本的 Excel 工单或图像对象作为需求的一方面。还包括一些学习帮助功能，例如示教和例子项目，帮助尽快上手。

这些产品展示了在应用开发中同其他工具（例如：测试，模型设计，问题跟踪，项目管

理工具)相集成的趋势。当选择一个需求管理产品时,考虑一下是否能与现有工具配合使用(交换数据)。下面是已介绍产品的一些工具连接例子:

- 在RequisitePro中不仅可以建立需求与Rational Rose的使用实例间的联系,还可以建立与Rational TeamTest的测试实例间的联系。
- DOORS允许建立需求与Rational Rose的设计元素间的联系。
- RequisitePro和DOORS能够建立需求与Microsoft Project中的项目任务间的连接。
- Caliber-RM通过一个中央通信框架允许需求不仅能建立Select Software Tools' Select Enterprise的使用实例、类或处理设计元素间的联系,还可以建立存储在Mercury Interacitve's TestDirector的测试实体间的联系。在Caliber-RM的数据库中就可以直接使用这些联系。

19.3 实现需求管理自动化

所有这些工具都可以把需求管理提高到一个新的层次。然而,用户的勤奋刻苦是成功的关键因素。对于有奉献、守纪精神,知识丰富的用户即使不好的工具也能获得成功,而缺乏热诚和训练的用户即使有最好的工具也不能顺利使用。在购买需求管理工具前一定要花费时间先学习它。因为有一个学习曲线问题,你不要寄希望在工具上的投资会马上产生回报;当然也不要在一个新工具第一次使用就应用到一个关键项目上,并寄希望于它来获得项目成功。正确做法是:在应用到关键项目前,一定要先在实验性项目上使用以积累经验。

在对平台、价格、使用方式和需求范例(是以数据库还是以文档为核心)进行考虑之后选择一个适合你开发环境的工具。下列过程可帮助选择一个好的工具:

- 1) 为需求管理工具定义项目需求。确定下列事项:最重要的功能是什么,是否要与其它使用的工具连接以及通过Web远程数据处理是否重要。决定是使用数据库存储全部数据还是只存储一部分。
- 2) 列出影响决策的10~15个因素。既要有主观的也要有客观的因素(如裁剪能力、有效性和GUI的效率)。
- 3) 对步骤2中列出的因素打分(总计100分)。对更重要的因素可以打更高的分。
- 4) 获得有关可用的需求管理工具的最新信息,根据影响决策的因素对候选工具排序。对客观因素的评分只有在使用每个工具后才能进行。开发商的展示可能会增加一些感性认识。但展示往往不全面,所以最好还是亲自使用一下(几个小时)。
- 5) 根据给每个因素的加权值来计算每个候选工具的得分,从而确定最合适的产品。
- 6) 从候选工具的其他用户那里获得一些体会,可以通过在线论坛获得经验,对自己的判断和开发商的投标进行补充。
- 7) 从候选工具中前三名的开发商处得到评估拷贝。确定候选工具前先定义一个评估处理过程,确保获得足够的信息做出好的决策。
- 8) 最好用一个实际的项目来评估工具,不要仅用工具所带的示教项目进行评估。完成评估后,如有必要调整排名分数。找出得分最多的工具。
- 9) 经过对排名、许可权费、开发商后续支持费、当前用户的输入、工作小组主观印象等的考虑之后做出决定。

考虑到将会花很大气力将项目的需求存入数据库、定义属性、设置跟踪联系链、更新数

据库、定义特权和训练用户。应该发动全体成员尽量挖掘产品的潜力。一定避免临时开发自己的需求管理工具或者用一些通用的办公自动化产品临时拼凑。似乎这个方法是一个容易的解决方案，但很快就不能适应要求的强度。

如果知道一个工具不能克服处理缺陷，你可能愿意用商业需求管理工具加强软件需求管理。一旦需求管理工具能够帮助你，你很快就会爱不释手。

下一步：

- 分析现有需求管理过程的缺点，从而确定是否有必要购买商业需求管理工具。确信理解现有缺点的原因，而不是想当然认为工具会纠正它们。
- 在采购工具前，先估计一下组员是否认为有必要采用工具。根据已有的经验来确定如何做才能成功。

附录 当前需求实践的自我评估

本附录包括 20 个问题，您可以用它们来校准您当前的需求工程策略，以确定在哪些领域需要加强。从每个问题的四个可选项中，选出一个最能描述您当前处理软件需求问题方法的答案。如果您想确定自我评估的质量，为每个“a”答案给 0 分，“b”给 1 分，“c”给 3 分，“d”给 5 分。最高的可能得分是 100 分。但是，不要太在意分数的高低，用自我评估发现能应用新策略的机会，您的组织将会从中受益。每个问题都指出了与该问题主题相关的章节。

某些问题可能并不适合您组织所开发的软件。环境不尽相同，您的项目也许不需要最全面、最复杂的方法。例如，对于市场上没有先例的高度创新的产品，由普通产品概念得到的需求可能就不适合。但是，应该承认非正式的需求方法增加了大量返工的可能性。从总体上遵照以下的策略，大多数组织都将从中受益。

1. 项目范围是如何确定、交流和使用的？【第 6 章】

- a. 设计产品的人通过心灵感应与开发组织进行交流。
- b. 有书面的项目任务陈述。
- c. 使用标准的任务和范围文档模板，所有项目成员都能访问这个任务和范围文档。
- d. 评估所有建议的特性和需求变更，确定它们是否与文档中的任务和范围相符。

2. 客户如何与确定的和表征的产品进行交流？【第 7 章】

- a. 不能确定谁是客户。
- b. 销售也许知道谁是客户。
- c. 通过管理，从销售调查和从现有客户基础中确定目标客户。
- d. 销售、管理和关键客户代表不同的用户类别，软件需求规格说明概括了他们的特征。

3. 您如何得到用户需求的输入？【第 7 章】

- a. 开发人员已经知道需要建设什么。
- b. 销售能提供用户的观点。
- c. 调查或访问典型用户的中心小组成员。
- d. 明确代表不同用户类别的个人参与项目，并约定其责任与权利。

4. 如何培训您的需求分析员，他们是否富有经验？【第 2 章】

- a. 他们丝毫没有经验，未接受过特殊的开发需求培训，他们宁愿编写代码。
- b. 分析员是受过一两天软件需求训练的开发者，以前具有和用户交互的经验。
- c. 他们经过短期培训，在采纳技术和主持小组会议方面具有丰富的经验。
- d. 我们具有专业或系统分析员，他们在与用户合作方面具有广泛的经验。他们同时理解应用领域和软件开发过程。

5. 如何将系统需求分配到产品的软件部分？【第 7 章】

- a. 软件是用来弥补硬件的不足。
- b. 软件与硬件工程师讨论哪个子系统应该实现什么功能。
- c. 系统工程师分析系统需求并将其中一些分配给软件。

- d. 系统需求的一部分分配给软件子系统并跟踪明确的软件需求。清楚地定义子系统界面并文档化。
6. 用什么方法来分析客户的问题？【第 8 章】
- a. 我们的开发者很聪明，他们理解问题很深入。
 - b. 我们询问用户想要什么，记录下来，将其实现。
 - c. 我们与用户讨论他们的业务需求与当前的系统，然后写一个软件需求规格说明。
 - d. 我们观察用户如何完成他们当前的任务，将他们当前的工作流程模型化，了解他们希望新系统完成什么功能。这向我们表明他们的部分过程如何自动化，告诉我们什么软件特性是最有价值的。
7. 用什么方法来确定所有明确的软件需求？【第 8 章】
- a. 我们从一个概要的理解开始编写代码，并修改代码直到完成。
 - b. 管理或销售提供了一个产品概念，开发者制定需求。由销售来确定开发是否有遗漏，有时销售也负责向开发者转告产品概念的变更。
 - c. 销售或客户代表告诉我们产品应该具有什么特性和功能。
 - d. 我们为产品与不同用户类别的代表进行有组织的会见或专题讨论。我们通过使用实例来理解用户任务，然后从使用实例中得到功能需求。
8. 如何将软件需求写成文档？【第 9、10 章】
- a. 我们搜集口头历史、e-mail 信息、语音邮件信息、会见记录和会议记录。
 - b. 我们编写非结构化的叙述性的文本文档，或者画出结构化的或面向对象的分析模型。
 - c. 我们结合一些用标准概念表示的图形分析模型，在与标准软件需求规格说明模板一致的基础上，用结构化的自然语言书写需求。
 - d. 我们将需求存储在一个数据库或商业需求管理工具中，将分析模型存储在 CASE 工具中。每个需求将和一些属性共同存储。
9. 如何获取非功能性需求，如软件质量属性，并将它们编写成文档？【第 11 章】
- a. 什么是“软件质量属性”？
 - b. 我们通过用户界面的 beta 测试来得到用户喜欢什么的反馈。
 - c. 将某些属性，如操作与安全需求，编写成文档。
 - d. 我们通过与用户交谈来确定每个产品的重要质量属性，然后将它们在软件规格说明中用准确的、可验证的方式记录下来。
10. 如何标记单个需求？【第 9 章】
- a. 我们采用叙述性文本段落，明确的需求并未单独确认。
 - b. 我们采用加重号和数字的列表。
 - c. 我们采用层次数字方案，如“3.1.2.4”。
 - d. 每个单独的需求都有一个独立的、有意义的标记，它不会随其它需求的增加、移动或删除而遭到破坏。
11. 如何建立单个特征或需求的优先级？【第 13 章】
- a. 它们都很重要，否则我们不会首先将它们记录下来。
 - b. 客户告诉我们什么需求对他们最重要。
 - c. 根据客户的意见所有的需求都标记为高、中或低优先级。

d. 我们使用分析过程，来评估与每一个使用实例、特征或功能需求相联系的价值、代价和风险。

12. 采用什么技术来准备局部的解决方法，并验证对问题的相互理解是否一致？【第 12 章】

- a. 没有任何技术，我们只管开发系统。
- b. 我们开发一些简单的原型并从询问用户获得反馈。有时，我们不得不分发原型的代码。
- c. 我们开发原型不仅用来作为用户界面的模型，也在适当的时候作为概念的技术证据。
- d. 我们计划开发抛弃型的报告和电子原型来帮助我们改进需求。有时我们也开发评估模型。我们在原型的基础上通过结构化的评估脚本来获得客户反馈。

13. 如何评估需求文档的质量？【第 14 章】

- a. 我们认为我们的需求相当好。
- b. 我们巡回传递需求文档来获得反馈。
- c. 分析员和一些开发者进行非正式的评审。
- d. 在客户、开发者和测试者的参与下，我们对软件需求规格说明和分析模型进行正式的审查。我们针对需求编写测试用例，用它们来确认软件需求规格说明和模型。

14. 如何分辨不同版本的需求文档？【第 16 章】

- a. 自动生成文档的打印日期。
- b. 我们为每一个文档版本采用一个序列号，如 1.0, 1.1 等等。
- c. 我们采用一种区分方案能将草稿版本和基线版本、主要修改和次要修改区别开来。
- d. 需求文档通过版本控制存储在配置管理系统中，或者需求存储在保存每个需求修改历史的商业需求管理工具中。

15. 如何跟踪软件需求到它们的来源？【第 18 章】

- a. 不跟踪。
- b. 我们知道许多需求的来源。
- c. 所有需求都有一个确定的来源。
- d. 我们在软件需求和一些客户需求陈述、系统需求、使用实例、标准、规则、结构要求或其它来源之间建立全面的双向跟踪。

16. 如何利用需求作为开发项目计划的基础？【第 15 章】

- a. 交付日期在我们搜集需求之前就已经确定了。我们不能修改进度或需求。
- b. 在交付日期之前我们进行一个快速的缩小范围的过程来去掉一些特性。
- c. 项目计划的第一次反复确定收集需求所需的计划，项目计划的余下部分在我们得到需求后完成。但是，在此之后不能修改计划。
- d. 我们通过需求估算产品的大小，并在对实现要求功能所需工作的估算的基础上制定进度和计划。如果需求变更或者进度延期，通过协商来更新计划和协定。

17. 如何利用需求作为设计的基础？【第 15 章】

- a. 我们并不进行明确的设计。
- b. 如果有所编制的需求文档，我们在编程时也许会参考它们。

- c. 需求文档包含用户界面设计和我们计划实现方案的其它方面。
 - d. 设计者审查软件需求规格说明以确保它能作为设计的基础。我们在单个功能需求和设计元素之间具有全面的双向跟踪。
18. 如何利用需求作为测试的基础？【第 15 章】
- a. 需求和测试之间没有直接的联系。
 - b. 测试者根据开发者的陈述来进行测试。
 - c. 我们根据使用实例和功能需求来设计系统测试用例。
 - d. 测试者检查软件需求规格说明书以确保需求是可验证的，并开始计划测试过程。
我们将系统测试回溯到明确的功能需求。测试的进展部分地由需求覆盖来度量。
19. 如何确定和管理每个项目的软件需求基线？【第 16 章】
- a. 什么是“基线”？
 - b. 虽然客户和经理不再提出要求，但我们仍然收到大量的变更和客户意见。
 - c. 虽然我们定义了需求基线，但它不能总是与过去作出的变更保持一致。
 - d. 当初始基线定义时，需求存储在一个数据库中。当批准需求变更时更新数据库和软件需求规格说明。一旦确定了基线则保存对每个需求的变更历史。
20. 如何管理需求的变更？【第 17 章】
- a. 常常有未经控制的变更蔓延到项目中。
 - b. 当需求阶段完成后，通过冻结需求来拒绝变更。
 - c. 我们采用标准表格将变更请求提交到一个建议中心。由项目经理决定采纳哪些变更。
 - d. 变更通过一个确定的变更控制过程来进行，该过程使用一个工具来搜集、存储和交流变更请求。在变更控制委员会决定是否批准每个变更并评估其影响。

参 考 文 献

- Ambler, Scott. 1995. "Reduce Development Costs with Use-Case Scenario Testing." *Software Development* 3(7):53–61.
- . 1999. "Trace Your Design." *Software Development* 7(4):48–55.
- Andriole, Stephen J. 1996. *Managing Systems Requirements: Methods, Tools, and Cases*. New York: McGraw-Hill.
- Arlow, Jim. 1998. "Use Cases, UML Visual Modeling and the Trivialisation of Business Requirements." *Requirements Engineering* 3(2):150–152.
- Arnold, Robert S., and Shawn A. Bohner. 1998. *Software Change Impact Analysis*. Los Alamitos, CA: IEEE Computer Society Press.
- Bass, Len, Paul Clements, and Rick Kazman. 1998. *Software Architecture in Practice*. Reading, MA: Addison-Wesley.
- Beizer, Boris. 1990. *Software Testing Techniques*, 2d ed. New York: Van Nostrand Reinhold.
- . 1999. "Best and Worst Testing Practices: A Baker's Dozen." *Cutter IT Journal* 12(2):32–38.
- Beyer, Hugh, and Karen Holtzblatt. 1998. *Contextual Design: Defining Customer-Centered Systems*. San Francisco: Morgan Kaufmann Publishers, Inc.
- Boehm, Barry W. 1981. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall.
- . 1988. "A Spiral Model of Software Development and Enhancement," *IEEE Computer* 21(5):61–72.
- Boehm, Barry, et al. 1976. "Quantitative Evaluation of Software Quality," *Second IEEE International Conference on Software Engineering*. Los Alamitos, CA: IEEE Computer Society Press.
- Booch, Grady, James Rumbaugh, and Ivar Jacobson. 1999. *The Unified Modeling Language User Guide*. Reading, MA: Addison-Wesley.
- Brooks, Frederick P., Jr. 1987. "No Silver Bullet: Essence and Accidents of Software Engineering," *Computer* 20(4):10–19.
- Brown, Norm. 1996. "Industrial-Strength Management Strategies," *IEEE Software* 13(4):94–103.
- Caputo, Kim. 1998. *CMM Implementation Guide: Choreographing Software Process Improvement*. Reading, MA: Addison-Wesley.
- Carnegie Mellon University / Software Engineering Institute. 1995. *The Capability Maturity Model: Guidelines for Improving the Software Process*. Reading, MA: Addison-Wesley.
- Carr, Marvin J., Suresh L. Konda, Ira Monarch, F. Carol Ulrich, and Clay F. Walker. 1993. *Taxonomy-Based Risk Identification* (CMU/SEI-93-TR-6). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.
- Charette, Robert N. 1990. *Applications Strategies for Risk Analysis*. New York: McGraw-Hill.
- Christel, Michael G., and Kyo C. Kang. 1992. *Issues in Requirements Elicitation* (CMU/SEI-92-TR-12). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.
- Cockburn, Alistair. 1997a. "Goals and Use Cases." *J. Object-Oriented Programming* 10(5):35–40.
- . 1997b "Using Goal-Based Use Cases." *J. Object-Oriented Programming* 10(6):56–62.
- Cohen, Lou. 1995. *Quality Function Deployment: How to Make QFD Work for You*. Reading, MA: Addison-Wesley.
- Collard, Ross. 1999. "Test Design." *Software Testing and Quality Engineering* 1(4):30–37.
- Constantine, Larry. 1998. "Prototyping from the User's Viewpoint." *Software Development* 6(11):51–57.
- Davis, Alan M. 1993. *Software Requirements: Objects, Functions, and States*. Englewood Cliffs, NJ: PTR Prentice Hall.
- . 1995. *201 Principles of Software Development*. New York: McGraw-Hill.
- DeGrace, Peter, and Leslie Hulet Stahl. 1993. *The Olduvai Imperative: CASE and the State of Software Engineering Practice*. Englewood Cliffs, NJ: Yourdon Press/Prentice-Hall.

- DeMarco, Tom. 1979. *Structured Analysis and System Specification*. Englewood Cliffs, NJ: Prentice-Hall.
- Dorfman, Merlin, and Richard H. Thayer. 1990. *Standards, Guidelines, and Examples on System and Software Requirements Engineering*. Los Alamitos, CA: IEEE Computer Society Press.
- Ebenau, Robert G., and Susan H. Strauss. 1994. *Software Inspection Process*. New York: McGraw-Hill.
- Fagan, Michael E. 1976. "Design and Code Inspections to Reduce Errors in Program Development." *IBM Systems Journal* 15(3):182-211.
- Freedman, Daniel P., and Gerald M. Weinberg. 1990. *Handbook of Walkthroughs, Inspections, and Technical Reviews: Evaluating Programs, Projects, and Products*, 3d ed. New York: Dorset House Publishing.
- Gause, Donald C., and Brian Lawrence. 1999. "User-Driven Design." *Software Testing & Quality Engineering* 1(1):22-28.
- Gause, Donald C., and Gerald M. Weinberg. 1989. *Exploring Requirements: Quality Before Design*. New York: Dorset House Publishing.
- Gilb, Tom. 1988. *Principles of Software Engineering Management*. Harlow, England: Addison Wesley Longman Ltd.
- Gilb, Tom, and Dorothy Graham. 1993. *Software Inspection*. Wokingham, England: Addison-Wesley.
- Glass, Robert L. 1992. *Building Quality Software*. Englewood Cliffs, NJ: Prentice-Hall.
- . 1999. "Inspections—Some Surprising Findings." *Communications of the ACM* 42(4):17-19.
- Gotel, O., and A. Finkelstein. 1994. "An Analysis of the Requirements Traceability Problem." In *Proceedings of the First International Conference on Requirements Engineering*, 91-104. Los Alamitos, CA: IEEE Computer Society Press.
- Grady, Robert B., and Tom Van Slack. 1994. "Key Lessons in Achieving Widespread Inspection Use." *IEEE Software* 11(4):46-57.
- Ham, Gary A. 1998. "Four Roads to Use Case Discovery: There Is a Use and a Case for Each One." *CrossTalk* 11(12): 17-19.
- Hohmann, Luke. 1997. "Managing Highly Usable Graphical User Interface Development Efforts." <http://members.aol.com/lhohmann/papers.htm>
- Hsia, Pei, David Kung, and Chris Sell. 1997. "Software Requirements and Acceptance Testing," in *Annals of Software Engineering*, Nancy R. Mead, ed. 3:291-317.
- Humphrey, Watts S. 1997. *Managing Technical People: Innovation, Teamwork, and the Software Process*. Reading, MA: Addison-Wesley.
- IEEE. 1992. IEEE Std 1061-1992: "IEEE Standard for a Software Quality Metrics Methodology," Los Alamitos, CA: IEEE Computer Society Press.
- . 1997. *IEEE Software Engineering Standards Collection*. Los Alamitos, CA: IEEE Computer Society Press.
- . 1998. IEEE Std 830-1998: "IEEE Recommended Practice for Software Requirements Specifications." Los Alamitos, CA: IEEE Computer Society Press.
- International Function Point Users Group. 1999. *Function Point Counting Practices Manual, Version 4.1*. Westerville, OH: International Function Point Users Group.
- Jackson, Michael. 1995. *Software Requirements & Specifications: A Lexicon of Practice, Principles, and Prejudices*. Harlow, England: Addison-Wesley.
- Jacobson, Ivar, Magnus Christerson, Patrik Jonsson, and Gunnar Övergaard. 1992. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Harlow, England: Addison Wesley Longman Ltd.
- Jarke, Matthias. 1998. "Requirements Tracing." *Communications of the ACM* 41(12):32-36.
- Jones, Capers. 1994. *Assessment and Control of Software Risks*. Englewood Cliffs, NJ: PTR Prentice Hall.
- . 1996a. "Strategies for Managing Requirements Creep." *IEEE Computer* 29(7):92-94.
- . 1996b. *Applied Software Measurement*, 2d ed. New York: McGraw-Hill.
- Jones, David A., Donald M. York, John F. Nallon, and Joseph Simpson. 1995. "Factors Influencing Requirement Management Toolset Selection." *Proceedings of the Fifth Annual*

Symposium of the National Council on Systems Engineering, Volume II. Seattle, WA: International Council on Systems Engineering.

Jung, Ho-Won. 1998. "Optimizing Value and Cost in Requirements Analysis." *IEEE Software* 15(4):74–78.

Karlsson, Joachim, and Kevin Ryan. 1997. "A Cost-Value Approach for Prioritizing Requirements." *IEEE Software* 14(5):67–74.

Keil, Mark, and Erran Carmel. 1995. "Customer-Developer Links in Software Development." *Communications of the ACM* 38(5):33–44.

Kelly, John C., Joseph S. Sherif, and Jonathon Hops. 1992. "An Analysis of Defect Densities Found During Software Inspections." *Journal of Systems and Software* 17(2):111–117.

Kosman, Robert J. 1997. "A Two-Step Methodology to Reduce Requirement Defects." In *Annals of Software Engineering*, Nancy R. Mead, ed. 3:477–494.

Kovitz, Benjamin L. 1999. *Practical Software Requirements: A Manual of Content and Style*. Greenwich, CT: Manning Publications Co.

Kruchten, Philippe. 1996. "A Rational Development Process" *CrossTalk* 9(7):11–16.

Larman, Craig. 1998. "The Use Case Model: What Are the Processes?" *Java Report* 3(8):62–72.

Lawrence, Brian. 1996. "Unresolved Ambiguity." *American Programmer* 9(5):17–22.

———. "Designers Must Do the Modeling." *IEEE Software* 15(2):31, 33.

Leffingwell, Dean. 1997. "Calculating the Return on Investment from More Effective Requirements Management." *American Programmer* 10(4):13–16.

Martin, Johnny, and W. T. Tsai. 1990. "N-fold Inspection: A Requirements Analysis Technique." *Communications of the ACM* 33(2):225–232.

McCabe, Thomas J. 1982. *Structured Testing: A Software Testing Methodology Using the Cyclomatic Complexity Metric*. National Bureau of Standards Special Publication 500-99.

McConnell, Steve. 1996. *Rapid Development: Taming Wild Software Schedules*. Redmond, WA: Microsoft Press.

———. 1998. *Software Project Survival Guide*. Redmond, WA: Microsoft Press.

McGraw, Karen L., and Karan Harbison. 1997. *User-Centered Requirements: The Scenario-Based Engineering Process*. Mahwah, NJ: Lawrence Erlbaum Associates.

Musa, John, Anthony Iannino, and Kazuhira Okumoto. 1987. *Software Reliability: Measurement, Prediction, Application*. New York: McGraw-Hill.

Nelsen, E. Dale. 1990. "System Engineering and Requirement Allocation." In Thayer, Richard H., and Merlin Dorfman, *System and Software Requirements Engineering*. Los Alamitos, CA: IEEE Computer Society Press.

Pardee, William J. 1996. *To Satisfy & Delight Your Customer: How to Manage for Customer Value*. New York: Dorset House Publishing.

Porter, Adam A., Lawrence G. Votta, Jr., and Victor R. Basili. 1995. "Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment." *IEEE Transactions on Software Engineering* 21(6):563–575.

Poston, Robert M. 1996. *Automating Specification-Based Software Testing*. Los Alamitos, CA: IEEE Computer Society Press.

Putnam, Lawrence H., and Ware Myers. 1997. *Industrial Strength Software: Effective Management Using Measurement*. Los Alamitos, CA: IEEE Computer Society Press.

Ramesh, Bala, Curtis Stubbs, Timothy Powers, and Michael Edwards. 1995. "Lessons Learned from Implementing Requirements Traceability." *CrossTalk* 8(4):11–15, 20.

Ramesh, Balasubramanian. 1998. "Factors Influencing Requirements Traceability Practice." *Communications of the ACM* 41(12):37–44.

Regnell, Björn, Kristofer Kimbler, and Anders Wesslén. 1995. "Improving the Use Case Driven Approach to Requirements Engineering." *Proceedings of the Second IEEE International Symposium on Requirements Engineering*, pp. 40–47. Los Alamitos, CA: IEEE Computer Society Press.

Rettig, Marc. 1994. "Prototyping for Tiny Fingers." *Communications of the ACM* 37(4):21–27.

Robertson, James and Suzanne Robertson. 1994. *Complete Systems Analysis: The Workbook, The Textbook, The Answers*. New York: Dorset House Publishing.

———. 1997. "Requirements: Made to Measure," *American Programmer* 10(8):27–32.

- Robertson, Suzanne, and James Robertson. 1999. *Mastering the Requirements Process*. Harlow, England: Addison-Wesley.
- Rubin, Howard. 1999. "The 1999 Worldwide Benchmark Report: Software Engineering and IT Findings for 1998 and 1999, Part II." *IT Metrics Strategies* 5(3): 1-13.
- Rumbaugh, James. 1994. "Getting Started: Using Use Cases to Capture Requirements." *J. Object-Oriented Programming* 7(5):1-12, 23.
- Schneider, G. Michael, Johnny Martin, and W. T. Tsai. 1992. "An Experimental Study of Fault Detection in User Requirements." *ACM Transactions on Software Engineering and Methodology* 1(2):188-204.
- Smith, Craig. 1998. "Using a Quality Model Framework to Strengthen the Requirements Bridge." *Proceedings of the Third International Conference on Requirements Engineering*, pp. 118-125. Los Alamitos, CA: IEEE Computer Society Press.
- Sommerville, Ian, and Pete Sawyer. 1997. *Requirements Engineering: A Good Practice Guide*. Chichester, England: John Wiley & Sons.
- Song, Xiping, Bill Hasling, Gaurav Mangla, and Bill Sherman. 1998. "Lessons Learned from Building a Web-Based Requirements Tracing System." *Proceedings of the Third International Conference on Requirements Engineering*, pp. 41-50. Los Alamitos, CA: IEEE Computer Society Press.
- Sorensen, Reed. 1999. "CCB—An Acronym for 'Chocolate Chip Brownies'? A Tutorial on Control Boards," *CrossTalk* 12(3):3-6.
- Thayer, Richard H., and Merlin Dorfman, eds. 1997. *Software Requirements Engineering*, 2d ed. Los Alamitos, CA: IEEE Computer Society Press.
- The Standish Group. 1995. *The CHAOS Report*. Dennis, MA: The Standish Group International, Inc.
- Thompson, Bruce, and Karl Wiegers. 1995. "Creative Client/Server for Evolving Enterprises." *Software Development* 3(2):34-44.
- Voas, Jeffrey. 1999. "Protecting Against What? The Achilles Heel of Information Assurance." *IEEE Software* 16(1):28-29.
- Wallace, Dolores R., and Laura M. Ippolito. 1997. "Verifying and Validating Software Requirements Specifications." In Thayer, Richard H., and Merlin Dorfman, eds., *Software Requirements Engineering*, 2d ed., pp. 389-404. Los Alamitos, CA: IEEE Computer Society Press.
- Weinberg, Gerald M. 1995. "Just Say No! Improving the Requirements Process." *American Programmer* 8(10):19-23.
- Whitmire, Scott A. 1995. "An Introduction to 3D Function Points." *Software Development* 3(4):43-53.
- . 1997. *Object-Oriented Design Measurement*. New York: John Wiley & Sons.
- Wiegers, Karl E. 1996a. *Creating a Software Engineering Culture*. New York: Dorset House Publishing.
- . 1996b. "Reducing Maintenance with Design Abstraction." *Software Development* 4(4):47-50.
- . 1998. "The Seven Deadly Sins of Software Reviews." *Software Development* 6(3):44-47.
- Wieringa, R. J. 1996. *Requirements Engineering: Frameworks for Understanding*. Chichester, England: John Wiley & Sons.
- Williams, Ray C., Julie A. Walker, and Audrey J. Dorofee. 1997. "Putting Risk Management into Practice." *IEEE Software* 14(3):75-82.
- Wilson, Peter B. 1995. "Testable Requirements—An Alternative Sizing Measure." *The Journal of the Quality Assurance Institute* 9(4):3-11.
- Wood, David P., and Kyo C. Kang. 1992. *A Classification and Bibliography of Software Prototyping* (CMU/SEI-92-TR-13). Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.
- Wood, Jane, and Denise Silver. 1995. *Joint Application Development*, 2d ed. New York: John Wiley & Sons.
- Zultner, Richard E. 1993. "TQM for Technical Teams." *Communications of the ACM* 36(10):79-91.

后记

对一个成功的软件工程项目来说，最重要的是理解需要解决的问题和如何解决它。工程项目的需求提供了成功的基础。如果开发小组和客户在产品的功能和特性上没有达成一致的协议，最可能的后果是发生人人都避之不及的不愉快的软件意外事件。如果您当前的需求策略不能达到预期的效果，请仔细地从本书所提供的众多方法中选择一些方法进行尝试。有效软件需求工程的关键要素包括：

- 尽早包括广泛的客户代表
- 反复地和渐增地开发需求
- 用多种方式表达需求以确保每个人都能理解它们
- 与所有相关组织协商以确保需求的完整性和正确性
- 控制需求变更的路径

改变一个软件开发组织的工作方式是十分困难的。不能否认目前的开发方法不如想象中的那么好，并且很难确定接下来该尝试什么。很难找到时间学习新技术、开发改进过程、试验和调整它们，并且将它们推广到整个项目小组或组织中。很难说服小组成员和风险承担者认识改变是必须的。但是，如果不改变工作方式，你就没有理由相信当前的项目会比上一个项目做得更好。

软件过程改进的成功依赖于：

- 明确组织中的难点所在
- 一次集中于几个改进领域
- 为改进活动规定明确的目标并制定行动计划
- 明确人和与组织变更相关的因素
- 说服高级经理关注改进过程作为对业务成功的一种战略投资

当你为改进需求工程策略制定政策和着手实施时，请将这些工程改进准则铭记于心。将适合您的组织和小组的实用方法落到实处。如果你积极应用公认的好策略并依赖于常识，你会因为其带来的好处在处理项目的需求方面得到显著的提高。最后，请记住没有优秀需求的软件就像一盒巧克力：你无法知道你将得到的是什么。