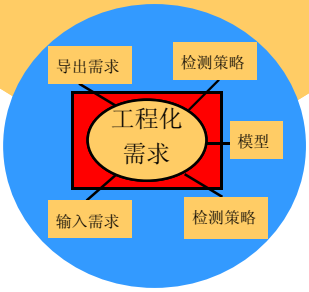
系统，工程，流程



Telelogic

需求工程

介绍
DOORS —
一种需求管理工具

- 总结了需求工程实践的最新技术
- 讲述了适合多种系统开发背景的灵活过程
- 介绍了丰富可跟踪性的重要新概念

作者：M. Elizabeth C. Hull，Ken Jackson，A. Jeremy J. Dick

翻译：韩柯

编审：Telelogic 中国公司

前言

虽然需求工程是常识，但是感觉起来很难而且不容易被很好地理解。正是由于这些原因，使得它实施得不是非常好。机构日益增长的压力，常常变成不引入更严格需求工程方法的一个主要理由。但是，正是这些压力才使得需求工程师在帮助机构正确开发上具有更重要的作用。

系统工程对于当今业界是至关重要的，而需求工程又是整个过程的一个重要方面。好的过程是需求工程的关键——它能够决定如何高效、快速地生产产品。在把“产品上市时间”和满足用户需求作为关键成功因素的全球市场竞争中，好的过程尤其重要。

需求工程也有管理的内容，因此与需求和管理有关的问题往往被结合起来，以说明怎样利用需求来管理系统开发。

本书着重介绍需求的工程化方法以及怎样帮助系统工程师创建更好的需求。本书展示了一种通用过程，来帮助读者更好地理解需求工程的本质。接下来，针对开发的问题领域和解决方案领域，对这种通用过程进行了实例化。本书也讨论了系统建模概念，并介绍了被广泛使用的各种技术和方法。本书的一个重要特色是探讨了可跟踪性的建立以及捕获可跟踪性的方法和能够根据可跟踪性导出的指标。最后本书概要地介绍了 DOORS 这种需求管理工具，并通过案例研究来说明本书所介绍的过程以及 DOORS 工具的功能。

本书适合做为业界的系统和需求工程师阅读，作为实践者，这些读者迫切需要使用需求工程来开发系统的知识。本书适合用作为计算机科学、软件工程和系统工程等高年级本科生的需求工程教材，也适合计算机科学和其它工程类研究生阅读。

本书介绍的方法以需求工程的最新研究成果为基础，不仅吸收了学术界的观点，也在很大程度上建立在业界实际工作经验基础之上，这会帮助系统工程师能够更成功地管理需求（和项目）。本书是当今需求工程这种高速发展领域中最佳实践的一个缩影。

M. Elizabeth C. Hull，Ken Jackson，A. Jeremy J. Dick

（2002 年 5 月）

致谢

感谢那些个人和组织以各种方式提供的帮助

感谢 Richard Stevens，他以多年的需求管理工作经验为我们提供了大量的帮助，并奠定了此书的基础。他是 QSS 公司的创始人之一，这个公司开发了 DOORS 工具。

感谢 Astrium 的 Les Oliver 帮助我们开发了“达成共识”、“鉴定”与“满足”等需求工程流程的状态图。

感谢 Praxis Critical System 公司提出了设计依据的最初概念，此概念最后成为本书丰富可跟踪性的内容。

感谢 Keith Collyer, Jill Burnett 与 Telelogic 公司其他同事为本书提供的思想及评审、注释、建议与鼓励。

目录

第1章 引论 1

对不知道航行目的地的人来说，没有顺风。
（哲学家 Lucius Annaeus Seneca，公元3－65年）

1.1 需求引论

如果系统开发项目曾经需要过“顺风”的话，那么他们现在一定更需要。高速变化着的技术和日益激烈的竞争为开发过程带来与日俱增的压力。有效的需求工程是机构的核心力量能够指导机构把握航行的方向并顺应大潮起伏的节奏。

软件是目前新产品变化的决定性力量，这是由三个关键因素决定的：

1. 任意的复杂性。 最复杂的系统一般都是含有软件的系统，并且软件常常深深地嵌入在系统的组件中。这种产品的复杂性只会受到产品设计者想象力的制约。
2. 即时发布的特点。今天，公司可以构思一种新产品，用软件来实现并迅速发布到全世界。例如，汽车制造商可以改进其诊断系统中的软件，然后用一天的时间，就可以通过电子媒介传到世界各地数以万计的所有汽车销售商中。
3. “商品化”的组件。现在的系统是通过购入技术和现成组件构建的，产品开发周期被大大压缩。

总的来说，这些发展趋势使竞争的骤然加剧，也使得在不需要大型工厂的情况下，新技术的优势可以快速转化为垄断利润。这种趋势的结果使得缩短开发周期和调动技术的时间压力大大增加。但是，仅仅缩短“产品上市市场时间”是不够的，真正的目标是缩短“将恰当的产品投放到市场的时间”。建立需求使我们能够就“恰当的产品”达成一致，并直观地看到这种“恰当的产品”。系统工程过程的一个关键部分，就是通过需求工程首先定义问题范围，然后将所有后续开发信息与问题范围关联起来。只有通过这种方式，我们才能控制和把握项目活动，开发出既合适又经济的解决方案。

需求是所有项目的基础，要根据需求定义潜在的新系统中所有 stakeholers（利益相关人）——用户、客户、供应商、开发商、商家——所需要的内容，定义为了满足这些需要系统必须完成的工作。为了使每一个人都了解需求，需求一般要以自然语言描述，因此存在着这样的挑战：在不借助行话或习语的情况下，完整、无歧义地捕获需要或问题。一旦相互沟通并达成一致，需求

就会推动项目活动。但是，stakeholers 的需要可能是多种多样的，有些需要还可能相互矛盾。这些需要在开始时可能并没有被清楚地定义，可能会受他们不能控制的因素制约，也可能会受随时间变化的其它目标的影响。没有相对稳定的需求基础，开发的项目就只能陷入困境。这就像既不知道目的地，也没有航海图的出海航行一样。需求既提供“航海图”，又提供朝向目的地航行的把舵方法。

需求共识可为系统开发计划及完成时进行验收提供基础。当必须综合考虑多方面的实际问题时，这种需求共识是必不可少的，当在开发过程中出现不可避免的变更要求时，它也是至关重要的。如果没有以前系统足够详细的模型，怎样评估变更的影响呢？如果要求恢复变更以前的需求又要如何应对呢？

即使定义了要解决的问题和潜在的解决方案，我们也必须评估如果不能提供令人满意的解决方案的风险。在缺乏令人信服的风险管理策略时，没有多少投资商或 Stakeholders 会支持这样的产品或系统开发。需求使人们可以在开发的早期就能管理风险。远在投入大量开发资源之前，就可以追踪与需求冲突的风险、评估其影响、理解减缓风险和善后计划的作用。

因此，需求是构成以下活动的基础：

- 项目规划；
- 风险管理；
- 验收测试；
- 综合平衡；
- 变更控制。

表1.1 项目失败的原因

● 不完整的需求	13.1%
● 缺乏用户的参与	12.4%
缺乏资源	10.6%
● 不实际的预期	9.9%
没有执行层的支持	9.3%
● 需求/规格说明发生了变化	8.7%
缺少规划	8.1%
● 不再需要	7.5%
资料来源：Standish 集团 1995 和 1996 年； 《科学美国人》1994 年 9 月号	

表1.2 项目成功的因素

● 用户的参与	15.9%
管理层支持	13.9%
● 清晰的需求描述	13.0%
合适的规划	9.6%
● 现实的预期	8.2%
较小的里程碑	7.7%
有才能的员工	7.2%
● 产权	5.3%
资料来源：Standish 集团 1995 和 1996 年； 《科学美国人》1994 年 9 月号	

项目失败的最常见原因并不是技术原因，表 1.1 列出了项目失败的主要原因。表中的数据来自 Standish 集团 1995 和 1996 年的调查，它给出了项目组陈述的项目失败的各种原因所占的比重。标有符号●的原因与需求直接相关。

这些问题主要分为三大类：

1. 需求——或者是组织得很差、表述得很差、与用户的关联弱、变更太快或不必要的变更，或者是不切实际的期望。
2. 资源的管理问题——没有足够的资金、缺乏支持、没有合适的纪律原则和规划，其中很多问题都是由于缺乏需求控制所引起的。
3. 政策——这是产生头两个问题的原因。

所有这些问题都可以以相当低的成本来解决。

从表 1.2 可以看出，项目成功因素与失败因素并不是一一对应的。管理层支持和合适的规划显然很重要——项目越大、开发时间越长，失败的可能性就越大(《科学美国人》1994 年 9 月号)。

本书讨论一般的需求工程方法和有针对性的需求管理。它解释了 stakeholder 需求和系统需求之间的差别，并讨论需求怎样被用于管理系统开发。本书还说明如何利用从 stakeholder 需求、通过系统需求到设计的可跟踪性来度量进展、管理变更、评估风险。本书通篇都考虑需求以及用来满足需求的设计组件的可测试性问题，以及如何形成可检验或可验证的需求。它强调了生成能够容易地集成和测试的设计的必要性。

需求管理与项目管理有重要的接口，本书第 8 章“需求工程的管理问题”将讨论这方面的问题。

1.2 系统工程引论

本书不仅仅讨论软件需求问题，需求工程的原则和实践也适用于软件只占很小部分的整个系统。

例如，考虑从伦敦到格拉斯哥的西海岸干线铁路系统。对于系统的高层需求来说，可以是从伦敦的Euston火车站到苏格兰的格拉斯哥全程开行时间应该控制在250分钟以内。提出这条需求需要综合考虑系统的所有主要组成部分：

- 列车及其速度；
- 铁轨及其支持高速列车行驶的能力；
- 火车站和火车站员工，及列车在车站的等待时间；
- 火车司机及其控制列车的能力；
- 信号子系统；
- 列车控制和检测子系统；
- 电力传输子系统。

虽然信号和控制子系统中的软件对满足这些需求中具有至关重要的作用，但是不能单独交付这些软件。完整的解决方案应包括整个系统。事实上，大多数需求都是由作为系统整体行为所表现出的属性来满足的。

那么，这里的“系统”含义是什么呢？系统是一个：

一组组件——包括机器、软件和人员——他们以组织协同的方式达到某种所需结果——所要满足的需求。

因此系统包括人员。在西海岸干线中，司机和车站员工及他们所接受的培训和所使用的过程，与软件和机器组件一样重要。

由于组件必须彼此协同，因此组件之间的接口是系统（以及需求）工程中至关重要的考虑因素，即人员和机器组件之间的接口、机器组件之间的接口、软件组件之间的接口。铁路系统中机器到机器接口的一个例子是列车轮子与铁轨的接触方式。除了物理上的考虑外（通过设计使列车能够沿铁轨行驶而不会滑出轨道），经过铁轨的电流可能被用来作为列车控制子系统的一部分来检测列车的位置。

“系统”概念的核心是“新出现的属性”的思想。这意味着系统的有用性不是取决于系统的任何具体部分，而是系统组件交互的行为所显示的新出现的属性。新出现的属性可能有利用价

值，可以被设计到系统中，以使系统有用；或者新出现的属性有不期望的副作用，例如对环境的危害。系统工程的任务就是利用有价值的新出现的属性，而避免不需要的新属性。

另一个重要的概念是“系统的系统”。每个系统都可以被用于构造更大的外围系统。例如，西海岸干线是更大铁路系统的一部分，与其它干线和支线线路构成新的铁路网。整个铁路系统又是更大运输系统的一部分，与所有种类的公路和空运网络交互。运输系统本身是提供货物和人员交通的必要基础设施，也是构成国家经济系统的一部分。而国家又是世界的一部分，以此类推。

正确地理解系统的需求，就是理解系统的外围系统。系统的正确功能往往取决于外围系统所提供的条件。例如，直升飞机的飞行能力取决于地球所提供的环境，地球的重力场和空气。

再举一个非常简单的例子：杯子。杯子显然有一些组件：一个杯把和一个碗型的容器。这些组件有什么用途呢？杯碗是用来盛液体的，杯把的用途是让人拿住碗，同时又不会被烫伤。我们可以由此得出杯子的用途，也就是需求，是使人能够将热的液体送到嘴里而液体不会溅出，人也不会被烫伤。但是还有其它一些因素需要考虑。

- 杯子本身是没有用的，它要依靠人胳膊的机械运动来达到其目的。
- 杯子的杯碗部分要依靠重力的存在才能发挥作用。杯子还必须被正确使用：杯子拿反了就会使液体倒出，可能导致烫伤。

这种简单杯子要具备其用途的能力，最终取决于：

- 从组件交互所表现出的属性；
- 与外部组件的合适接口；
- 正确嵌入外围系统——由人手拿住，并用胳膊举起来；
- 适当环境的存在——在失重条件下将需要另一种解决方案。

总之，需求工程必须把系统的本质考虑进去。最重要的考虑是新出现的属性、外部环境的制约与保障、以及与周围系统的接口。

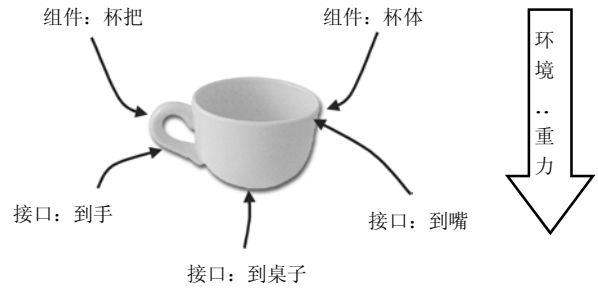


图1.1 作为非常简单系统的一个杯子

1.3 需求与质量

没有需求的所造成的后果有很多，并且是多种多样的。我们周围有大量证据说明系统失败是因为没有合适地组织需求。不过，虽然系统初看起来可能很正常，但是如果这不是用户想要或需要的系统，那么这个系统也是没有用的。

考虑需求和质量之间的关系很有意思。术语“质量”可以用不同的方式来理解。当说到高质量的汽车时，人们可能会提到劳斯·罗伊斯、奔驰或美洲虎。这种“质量”和“豪华”两个概念的内在混淆，在为一年一度的 RAC 拉力赛选最佳赛车时就会暴露出来。不管是劳斯·罗伊斯，还是奔驰或美洲虎都不会选中，因为这些车都没有合适的重量/功率比、离地距离和稳定性等性质。最近的历史说明，在同类中具有最佳质量的汽车是一种 Skoda 车，并不是豪华汽车，但是 Skoda 具有完成其功能的真正“质量”。

质量是“对用途的适合性”，或是需求的符合性，即提供能够满足客户需要的某种功能，并保证考虑了所有 stakeholders 的需要。

第 8 章将会介绍，需求工程是其它管理因素（例如成本和进度）的辅助，因为需求工程强调交付产品的质量。所有管理决策都是成本、进度和质量的折衷，这三个因素构成相互关联的三个轴。

由于需求工程是应用于开发生命周期初期的学科，通过适当的需求管理来对质量进行提升会产生很大的比例放大效果。开发早期阶段投入的相对很少的工作，在后期阶段能够得到几倍的回报。格言“质量是免费的”（Phil Crosby 的一本书的书名）仍然是正确的，因为开发工作开始时投入的工作，会节省本来必须要在后期投入的大量工作。改进需求意味着改进产品的质量。

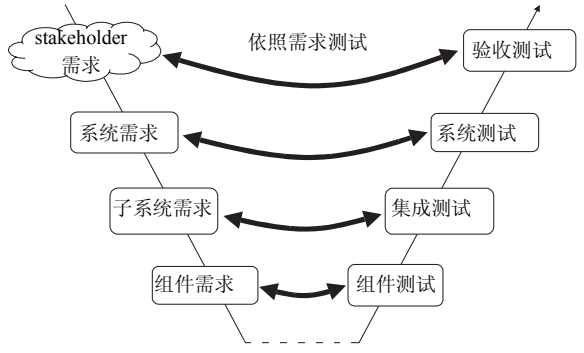


图1.2 V字模型中的需求

1.4 需求与生命周期

人们经常错误地认为，需求工程只是在产品开发开始时需要执行和完成的一个阶段。本节的目的就是说明需求工程在开发的各个阶段都有着至关重要的作用。

让我们先考虑开发过程的最后工作：验收测试。根据什么验收系统呢？答案是 stakeholder 需求。因此我们立即可以看出，最初开发的需求在开发的最后阶段仍然要使用。

表明各个开发阶段的经典 V 字模型就是以测试和需求之间的关系为基础的。图 1.2 给出了开发各个阶段的这种关系。

V 字模型还把开发分成层次，每个层次都研究适于相应开发阶段所关心的问题。尽管每个层次可以使用略有不同的过程，但是需求使用的基本模式是相同的，第 2 章在介绍一般过程时还会讨论这一点。图 1.3 给出了各个层次需求工程的主要考虑。

需求在机构中可以充当的另一种角色是充当项目之间一种沟通的手段。这是一个好想法，因为很多机构都希望：

- 尽可能扩大项目之间的产品重用；
- 管理相似产品系列；
- 通过程式化管理来协调活动；
- 通过学习其它项目的经验来优化过程。

一组好的 stakeholder 需求可以为高级管理层提供所要开发内容的简捷的非技术描述。类似地，系统需求也可以对开发项目作很好的技术归纳。这些描述可以用作与其它活动进行比较的基础，如图 1.4 所示。

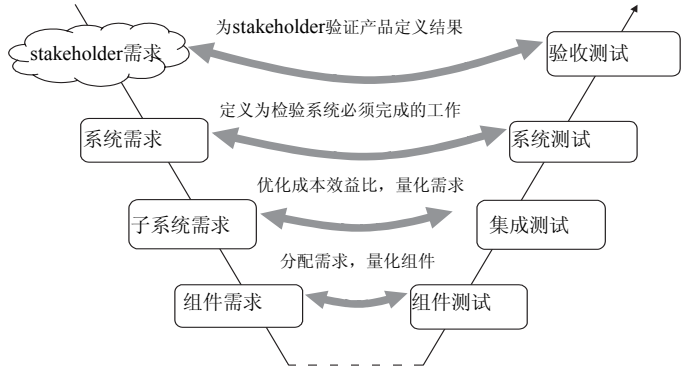


图1.3 不同层次中的需求工程

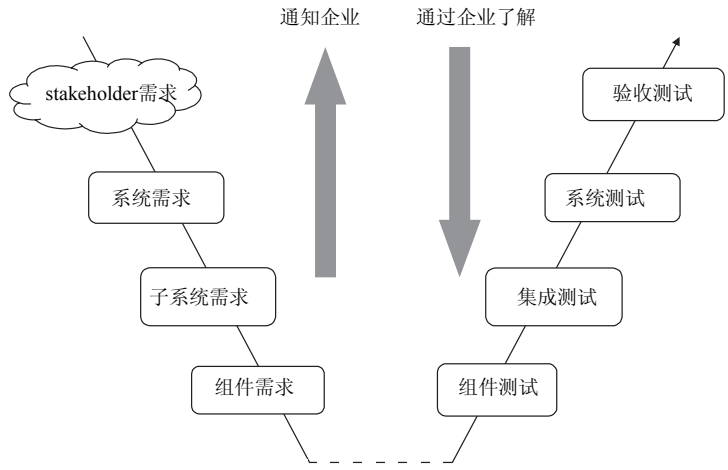


图1.4 企业需求工程

如果需求在系统开发中发挥这样的核心作用，则需求就需要被维护。在修改产品设计时没有同时更新需求以反映设计上的变化会为开发的后续阶段积累大量问题。因此，需求工程与变更管理密切关联。

不管变更是由项目内部提出的——例如由于设计细节引起的技术问题，还是从项目之外提出的——例如涉及 stakeholder 的需要，都需要对变更对质量、成本和进度的影响进行评估。这种评估是构成以下活动的基础：

- 接受或拒绝变更（这是一种选项）；
- 与客户/供应商协商变更成本；
- 组织返工。

要进行这种影响分析的关键是需求的可跟踪性，第 1.5 节和第 2 章、第 7 章还要进一步讨论这个问题。可以说，变更管理是需求工程过程的一个组成部分。这种角色如图 1.5 所示。

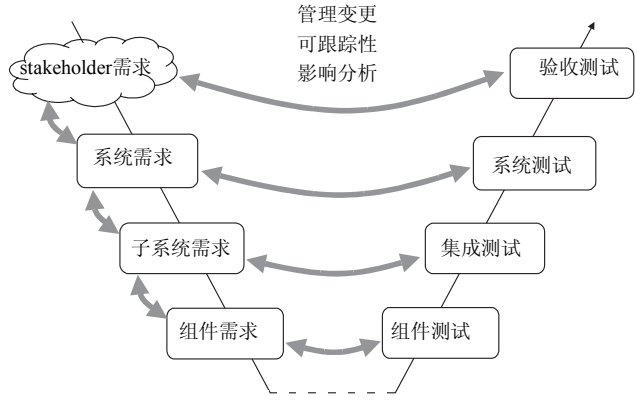


图1.5 变更管理

除了变更管理之外，管理人员控制项目的能力会由于好的需求工程而得到很大提高。如果没有需求，就没有办法判断项目的进展情况，甚至不能判断项目的方向是否正确。当需要变更时，就没有什么依据可以对变更进行评估。不仅如此，当管理人员真正需要介入时，这种介入也只能是技术层面上的，这与他们的角色很不相称，并且会干扰工程师正常充当的技术角色。在适当层次上很好描述的需求会为管理人员提供关于项目的恰当视图以使其能够发挥自己的作用。

总之，需求是所有系统健康开发的基础，它们会从头到尾、从上到下地影响整个开发工作。如果没有有效的需求工程，开发项目就会像在风浪中飘浮着的没有舵的船！最重要的是，采用好的需求管理，听取用户和客户的声音可终止传言游戏，并成为整个开发过程中的一条清晰的沟通线索。

1.5 需求的可跟踪性

在需求工程背景下，可跟踪性就是理解怎样把高层需求，即目标、渴望、期望、需要，如何转换为下层需求，因此主要关注的是信息层次之间的关系。

在业务背景下，人们可能关心：

- 业务远景如何解释为业务目标；
- 业务目标如何实现为业务组织和过程。

在工程背景下，人们可能关心：

- stakeholder 需求如何被系统需求满足；

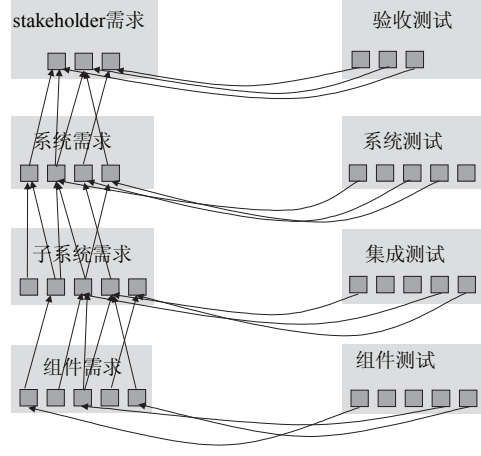


图1.6 需求的可跟踪性

- 系统需求如何划分为子系统；
- 子系统如何实现为组件。

利用可跟踪性可以在以下方面获得收益：

- 更有把握达到目标。建立和构造可跟踪性，可更好地反映目标是如何被满足的。
- 能够评估变更的影响。只有在存在可跟踪性信息的情况下，各种形式的影响分析才有可能。
- 改进下级机构的可审计性。可以更清晰地反映供应商对整体的贡献。
- 能够跟踪进度。如果所做的只是创建和修改文档，则很难度量进展。围绕可跟踪性的过程能够精确度量早期阶段的进展。
- 能够平衡成本与收益。将组件与需求关联起来，可以针对成本评估收益。

可跟踪性实际上是多对多关系——即一个下层需求可能与多个上层需求关联，一个上层需求也可能与多个下层需求关联。实现某种形式的可跟踪性最简单的方法，是将一层中的需求语句与另一层中的需求语句联系起来。需求管理工具通常通过在文档段之间的拖拽来支持这种关联。这种链相当于类似 Web 页面的超链，但是最好能够双向传导。图 1.6 给出的是通过各层需求的向下连接并与测试信息相连的可跟踪性。

箭头方向遵循一种特殊约定：信息回溯到它所响应的信息上。这样约定有若干原因：

- 这种约定一般可反映信息创建的时间顺序：总是反向连接到较老的信息上；
- 这种约定一般对应由于拥有关系产生的访问权限：一种信息拥有从文档外出的链，另一种信息拥有进入链。

可以使用各种形式的可跟踪性分析来支持需求工程过程（如表 1.3 所示）。

如果所选择的工作产品出现变化，影响分析可用于确定开发中的哪些工作产品会受到影响。这一点可通过图 1.7 说明。这种影响是潜在的，如果存在这种影响的话，工程师必须进行创造性的分析，以确定影响的确切性质。

表1.3 可跟踪性分析的类型

分析类型	描述	所支持的过程
影响分析	跟踪进入链，回答：“如果这种情况发生变化会出现什么情况？”	变更管理
来源分析	跟踪外出链，回答：“为什么会出现这种情况？”	成本/收益分析
覆盖率分析	统计拥有链的语句，回答：“我是否覆盖了所有内容？”常常用于度量进度。	一般工程管理报告

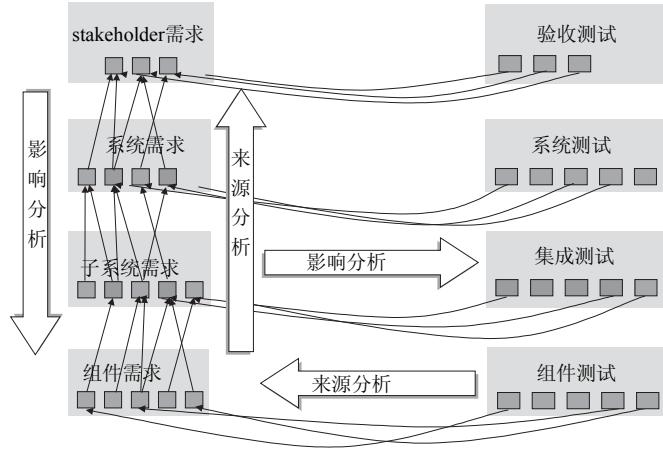


图1.7 影响与来源分析

来源分析与影响分析的方向相反。当选择了下层的工作产品，例如需求、设计单元或测试，使用可跟踪性链可以确定引起这些下层工作产品的高层需求是什么。不能进行这样反向连接的设计单元有可能只会增加成本而不产生任何效益。

最后，覆盖率分析可以用来确定所有需求都可以跟踪到下层并与测试相连。如果没有这种跟踪，则可以相当肯定地说，这些需求不会被满足，也不会被测试。当然，存在这种链也并不能保证需求就会满足——这也同样需要创造性的工程判断。

覆盖率分析还可以用来度量进展：系统工程在满足 stakeholder 的需求方面已经走了多远？假设编写系统需求以满足 stakeholder 要求的任务被交给工程师。当他们编写系统需求时，要反向关联到他们所要满足的 stakeholder 需求。（通过在其开展工作时进行这种关联，建立可跟踪性不会增加多少额外负担，而在 stakeholder 需求和系统需求编写完成之后建立这种可跟踪性要困难得

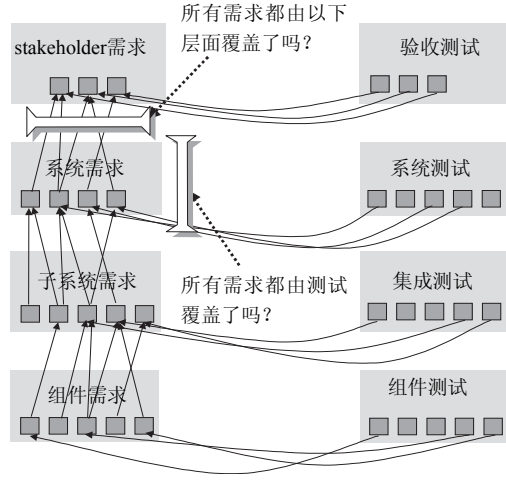


图1.8 覆盖率分析

多！）

现在，在任务的任何阶段，工程师的工作进展都可以通过到目前为止所覆盖的 stakeholder 需求的百分比来度量。这是在开发早期阶段非常有用的管理工具。

同样的原理也可以用于度量测试计划的进展。到目前为止，占多少百分比的需求已经定义了测试？图 1.8 归纳了覆盖率在这两方面的作用。

由于可以进行这些类型的分析，因此可跟踪性是处于需求工程过程核心位置的一种简单概念。第 7 章将详细讨论更先进形式的可跟踪性。

1.6 需求与建模

理解需求管理和系统建模之间的差别是非常重要的，这些是不应该等同起来的相互支持的活动。

具体的模型决不会表达系统的所有内容——如果能够表达所有的内容，它也就不再是模型了。因此，人们经常使用多种不同的有可能存在内部关联的模型来反映系统的不同侧面。需求的完整描述（通常以文本形式）要从建模活动中导出并将之整合，并且必须覆盖那些没有被建模的层面。

模型是系统的抽象，关注的是系统的某个方面，以排斥系统的其它方面。在这个意义上说，抽象就是避开干扰，忽略那些尽管很重要，但是与特定模型无关的细节。模型在这方面的优点是它能够被用于收集、处理、组织和分析较少的相关信息，应用各种特殊的技术进行与其相关的研究。

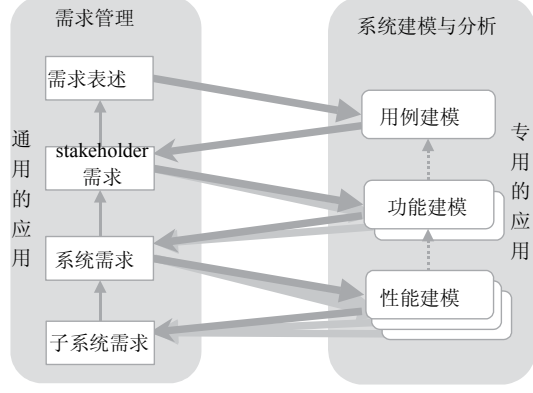


图1.9 需求与建模

如果必须管理大量复杂信息，建模可以提供一种聚焦手段，将有特定用途的数据子集收集在一起，然后再返回到整体问题上来。通过一次只集中关注少量信息，有助于对系统的整体把握。

图 1.9 显示出需求和系统建模所具有的相互关联的作用。模型可帮助需求工程师分析特定层次上的需求，以便：

- 与客户沟通，改进对被开发系统的相互理解；
- 分析系统，确定提供所急需的属性（并排除不需要的属性）；
- 通过导出下层新需求，确定如何满足本层需求。

所用模型的性质随层次的不同而不同。在顶层可使用用例模型，例如“stakeholder 场景”被用来导出 stakeholder 需求的最初描述。接下来，可以使用各种功能模型从 stakeholder 需求中导出系统需求。对于软件，这类模型可以包括 UML 类图、消息顺序图和状态图。（有关这些建模手段的进一步介绍，请参阅第 3 章。）

从系统需求转移到体系结构，关注点就转向了各种性能问题。可以使用多种模型确保所选择的体系结构可以满足非功能和功能需求。这类模型可以包括用于评估性能的队列原理、用于评估空气动力性能的风洞、用于评估旅行时间的时间表来建模。

从这些例子可以看出，模型的性质也会随应用系统的不同而不同。时间表建模可能适合于铁路系统设计，但是不适合于飞机设计，而空气动力学模型更适合飞机设计。（当然，空气动力学对于高速火车也很重要。）消息顺序图可以用于通信系统，但是有大量数据的应用系统更适合采用以数据为核心的模型，例如实体-关系图。

虽然模型不同，但是需求管理的原理对所有应用系统都是通用的。由于本书讨论的是需求工程，因此也要涉及与建模和方法密切相关的问题。

1.7 需求与测试

正如上面已经讨论过的那样，测试与各个层次上的需求密切相关。从广义上说，测试是使系统中的缺陷能够被发现或避免的所有活动，而缺陷是对需求的偏离。因此，测试活动包括所执行的评审、检测、模型的分析以及组件、子系统和系统的典型测试。

由于测试活动的多样性，本书使用“鉴定”一词来表示所有这类活动。

鉴定应该尽早开展，因为等到系统差不多已经完成时才进行测试会造成成本巨大的设计变更和返工。系统设计期间开展的最早类型的鉴定活动，包括需求评审、设计审查和各种形式的分析，

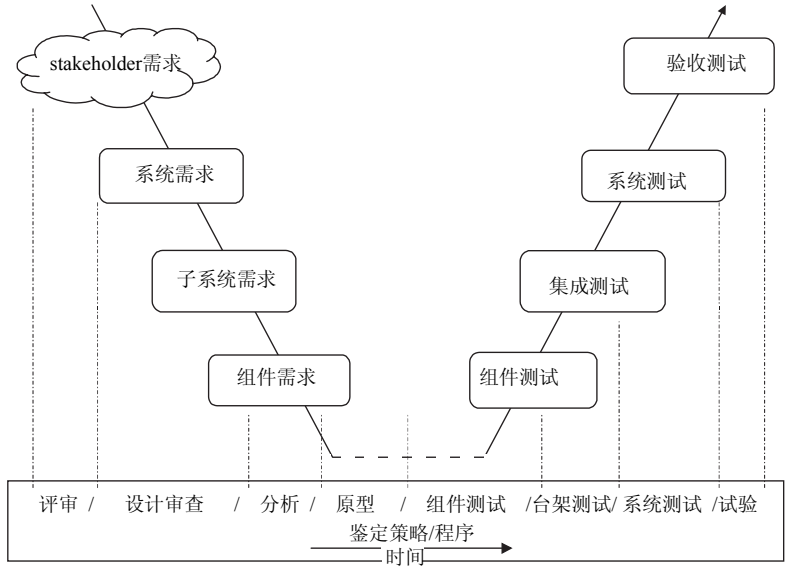


图1.10 鉴定策略与V字模型

应该根据系统模型进行。

图 1.10 显示出 V 字模型随时间展开的鉴定策略。早期鉴定活动与 V 字模型的左侧关联，后期鉴定活动与 V 字模型右侧的测试阶段关联。

单一的 stakeholder 需求一般会导致在开发不同阶段的多种鉴定活动。需求通过有用的新出现的属性来满足，而仅有对组件的鉴定是不充分的，要在能够表现出新属性的层次上进行测试。

1.8 问题与解决方案领域中的需求

系统工程是针对问题的有效解决方案的开发和管理。正如前面已经讨论过的，系统工程是对企业至关重要的分阶段过程，使企业能够在可接受的时间和成本范围内生产出合适的产品。

在过程初期，待建产品的需求定义具有头等的重要性。从管理和工程观点看，应该在“问题领域”和“解决方案领域”之间作出明确区分。这些开发阶段与系统的最高层描述（需要陈述、用例建模和 stakeholder 需求）应该坚实地扎根于问题领域中，而从系统需求开始的后续层次，要在解决方案领域中展开。

表 1.4 给出了问题领域和解决方案领域之间的理想边界，以及最高需求层所充当的角色。

抽象在这里有很重要的作用。最初能力的描述，不应该超出定义问题所必需的内容，并要避免涉及任何具体的解决方案。这使系统工程师能够自由发挥其作用，即不带任何偏见地设计最佳的解决方案。

表1.4 问题与解决方案的空间

需求层	领域	视图	角色
stakeholder需求	问题领域	stakeholder视图	说明stakeholder希望通过使用系统获得什么。避免涉及任何具体解决方案。
系统需求	解决方案领域	分析师视图	抽象地描述系统如何满足stakeholder需求。避免涉及任何具体设计。
体系结构设计	解决方案领域	设计人员视图	描述具体设计如何满足系统需求。

建模有助于导出下一层需求，并有助于考虑各种可能的解决方案，即使是高层解决方案。为了避免不恰当的解决方案偏见，早期建模关注的不应该是待建系统，而是系统的直接周围环境。例如，如果要开发无线电系统为船只导航，则早期建模应该关注船只，而不是无线电。这样才能在外层解决方案背景下描述问题。

同样的原理也适用于系统工程师：他们应该使设计人员能够自由发挥其作用，即根据抽象解决方案进行设计。通过功能建模引入的解决方案要素仍然是高层要素，而将细节留给后续阶段定义。

例如，在交通控制系统中：

- stakeholder 可以从最大交通流量及交通路口风险最小的情况，以及维护成本最低的角度来描述问题；
- 系统工程师可以考虑各种解决方案，例如桥梁、交通灯或环岛，并且确定从开发和维护成本来看，桥梁是解决这个问题的最佳方案；
- 然后设计人员根据实际环境条件，设计具体桥梁。

stakeholder 常常在描述问题时涉及预先想好的解决方案，这时需求工程师要确定使用具体解决方案的理由，或确定这是否是必要的约束条件。例如，客户开始提出要采用交通灯，供应商通过询问问题理解了客户的真正目的是使交通流量最大并驾驶员和行人的风险最小——这样才能建立与解决方案无关的问题描述；这样就能够更好地理解选择解决方案的理由，并可能通过适当建模进行确认，最终产生抽象解决方案的精确和描述清晰的规格说明。

到了构建系统阶段，需要确定是根据问题领域（stakeholder 需求）还是根据抽象解决方案领域（系统需求）来构建系统。解决方案的性质往往事先就知道了，因此根据由解决方案导出的系统需求是合理的。但是，就是根据具体的解决方案构建，在解决方案之前得到纯问题描述的原则，仍然会提供重要的优势。

如果不清晰地区分问题和解决方案，则可能产生以下结果：

- 对真实问题缺乏理解；
- 不能界定系统范围，不能理解系统应包含那些功能；
- 使开发人员和供应商就系统发生争执，因为对系统的惟一描述是采用解决方案的术语来描述的；
- 由于不能自由地设计，因此不能找出优化解决方案。

由于以上原因，本书是通过需求是怎样被获取、建模和表示的方式来区分 stakeholder 需求和系统需求的。

1.9 如何使用本书

本书关注的是需求工程，以及需求工程过程如何帮助系统工程师和软件工程师开发更好的需求。第 1 章已经讨论了需求的重要性，研究了需求工程在开发生命周期各个部分中的作用。

由于本书各章之间相互依赖，因此我们仔细地确定了各章内容，以尽可能减少前面的内容涉及后面内容的情况。虽然最好按顺序阅读各章，但是读者也可以根据以下指南，结合自己的具体目标有效地使用本书。

第 2 章“通用的需求工程过程”，讨论适用于开发各个层次的一般形式的需求工程。虽然这种方法有助于读者更好地理解需求工程的基本概念，但还是有必要的抽象性。不过第 5 章和第 6 章将更具体地讨论通用过程，通过大量例子说明修改通用过程以适应 stakeholder 和系统层需求的开发。

第 3 章“需求工程的系统建模与方法”，讨论系统建模问题，涉及广泛使用的各种手段和方法。这些讨论也为第 5 章和第 6 章做好准备，在这两章中将在 stakeholder 和系统需求背景下讨论具体的建模手段。

第 4 章“编写和评审需求”，讨论构建需求文档结构和需求表达描述问题，也将讨论不同类型的需求描述语言。

第 5 章“问题领域中的需求工程”，将通用过程具体化，以描述用户需求所主要关注的问题领域。

第 6 章“解决方案领域中的需求工程”，针对解决方案领域具体化通用过程，将系统需求细化到子系统和组件中。

第 7 章“高级可跟踪性”，将进一步讨论可跟踪性，着眼于以符合逻辑的依据来获取可跟踪性并讨论可以通过可跟踪性得出的指标。

第 8 章“需求工程的管理问题”，讨论需求管理背景下的项目管理问题，包括三种组织类型。

最后的第 9 章“DOORS：用于需求管理的工具”，将作为一个推进需求管理过程的软件工具的例子来概要介绍 DOORS，通过一个案例研究说明本书所介绍的过程以及工具的功能。

图 1.11 给出了各章之间的依赖关系。

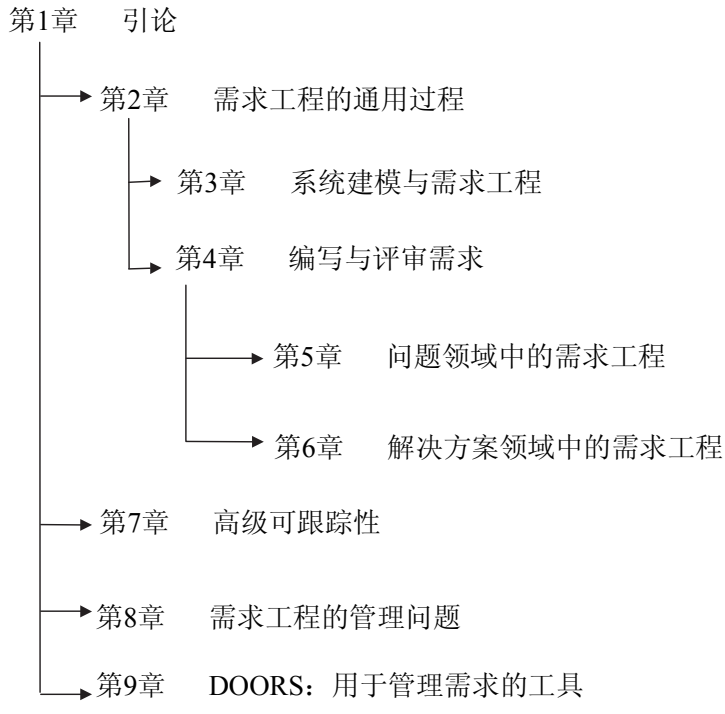


图 1.11 各章之间的依赖关系

第2章 需求工程的通用过程 2

如果你不能把你所做的描述为过程，就不知道自己正在干什么。
(William Edwards Deming，管理顾问，1900－1993)

2.1 引言

本章介绍系统开发过程的概念。首先研究系统开发的方式，来找出可以在很多不同背景下使用的开发模式。这种开发模式被表示为通用过程并在一定程度上进行详细介绍。后续各章将讨论如何将通用过程实例化，以用于具体目的。本章还要研究过程模型和信息模型之间的关系并开发一种通用过程的信息模型。

2.2 开发系统

在系统可以被开发之前，最基本的是确定系统需要。如果不知道系统的用途，就不清楚要开发什么样的系统，甚至在系统开发完成时，也不能确定系统是否满足其用户的需要。Forrest Gump (阿甘) 在归纳这种情况时，很深刻地指出：

“如果你不知道要去哪里，就不太可能最终到达那里。”

需要（the need) 被描述的严格程度，取决于负责描述这种需要的个人性格以及他在机构中的角色。需要在最初可能被描述得很模糊，例如“我希望系统能够提高我们部门的工作效率”。显然，这种“规格说明”不适合用作着手购置系统的基础，但是可以作为研究他实际需要什么的基础。这种研究必须确定该部门当前在哪些方面效率不高，并假设所建议系统将提供的能力会如何用来提高效率。这些活动将模糊需要描述转换为一组需求来作为购置系统的基础，并构成开发 stakeholder 需求的过程。 Stakeholders 不仅包括将直接与系统交互的人，而且还包括对系统的存在有相关利益的其他人和机构。第 5 章将详细讨论创建 stakeholder 需求问题。

图 2.1 显示的是开发过程。在用于过程模型的框图约定中，椭圆表示过程，矩形表示读取或产生的数据或信息。箭头表示数据是读取还是写入。这样，图 2.1 说明“开发 stakeholder 需求”过程是根据“需要描述”来产生“stakeholder 需求”的。此外，它还创建和读取“用例”模型。

一旦有了一组坚实的 stakeholder 需求，定义 stakeholders 将能够使用未来的系统做什么，就可

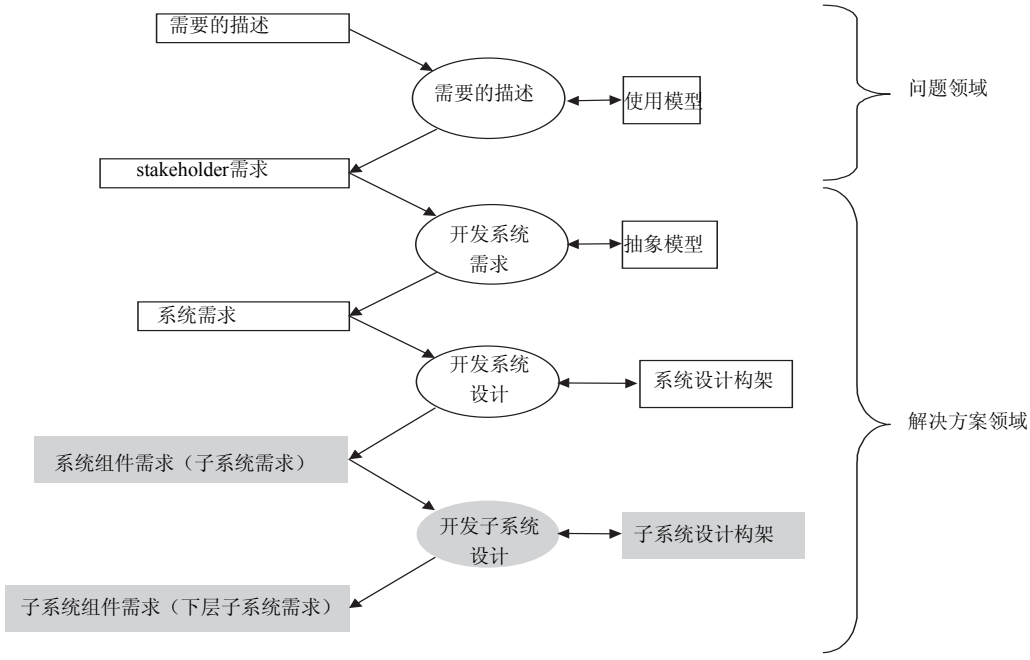


图 2.1 系统开发过程

以开始考虑潜在的解决方案。好的实践不是直接跳到设计，而是首先确定系统必须具备的特征，却不考虑最终的详细设计。这种过程叫做“建立系统需求”。它表达了一种将要创建的系统的抽象模型。这种模型为开发团队的内部讨论提供了基础，进而提供了共同来理解未来解决方案的方法，尽管这是在抽象层次上。这种模型还可以用来向那些希望确保开发人员沿正确方向发展的stakeholders 来解释解决方案的概念。最后，模型提供一种以文档形式表示系统需求的结构。模型中的每个要素都可以构成文档中的一节，将每个需求都放在相关的上下文中，从一致性和完整性角度看，这对于评审完备的需求集合是必不可少的。

通过系统需求，可以考虑不同的设计体系结构。设计体系结构被描述为一组交互组件，它们合在一起提供所需的属性。这些属性被叫做系统新出现的属性，它应该与在系统需求中所描述的系统所需的特性相匹配。设计体系结构定义每个系统组件必须做什么，系统组件如何相互交互，以便产生系统需求所描述的整体作用。换句话说，设计体系结构通过系统组件的功能和交互职责，定义每个系统组件的需求（如图 2.1 所示）。设计体系结构以及系统组件需求，还必须规定其它所需的性质，例如物理尺寸、性能、可靠性和可维护性等。

除了最小的系统，对于所有系统来说，设计体系结构中的组件都太复杂，不能直接实现。这个层次上的组件常常叫做“子系统”，因为相当复杂，本身就可以被看作是系统，但仍然还只是要设计的高层系统的一部分。

建立每个子系统的设计体系结构，然后再使用设计体系结构导出组件需求的过程，与总体系统描述过程很相似。最终要为每个子系统产生子系统设计体系结构和子系统组件需求，如图 2.1 所示。

开发过程的这种描述说明了系统开发发生在多个层次上，在每个层次上都有不同的活动。图 2.1 还说明，每种活动都由模型支持（例如用例模型、抽象模型、设计体系结构），尽管模型的性质有相当大的不同。以下是一个具有共性的例子：每层开发都使用一种模型。本章后续部分将进一步讨论这些相似性，以便定义通用过程的属性。

重要的是要认识到每个层次都有需求：

- 需要（needs）的描述；
- stakeholder 需求；
- 系统需求；
- 系统组件需求；
- 子系统组件需求。

因此，需求工程不是干一次就可以被遗忘的工作。需求工程出现在各个层次上，并且不同层次上的工作常常是并行的。在从系统组件向下的所有层次上都有多个并行的，针对需求的工作。（图 2.1 中相关符号的灰色背景表示这一点。）

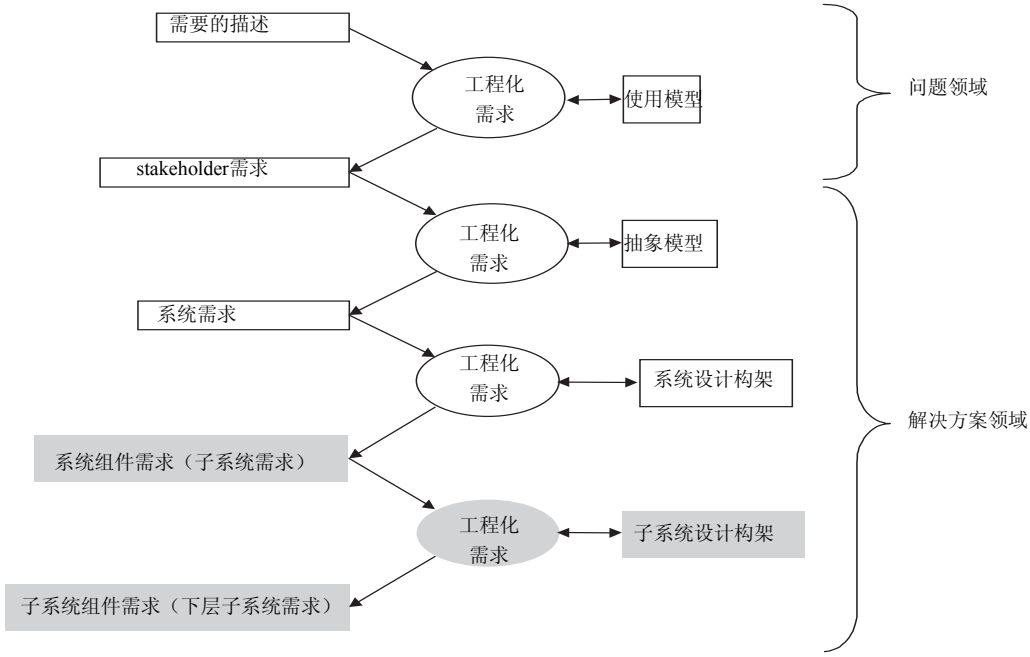


图2.2 需求工程的不同层次

2.3 通用过程背景

考虑开发过程的另一种方式如图 2.2 所示。这个框图说明被同样称作“工程化需求”的开发过程在每个层次都使用，尽管上面给出的解释说明所涉及的工作在每个层次上都不相同。这种看起来很奇怪的描述过程方式，被用来说明在各个层次上所完成的工作实际上具有相当程度的共同性这个事实。本章的目的，就是研究这些共性，提出不仅能够描述共性问题，而且还能够适应不同问题的通用过程。

需要强调的是在多层开发中，开发的每个层次都需要相应的专门技术。在较高层次上，问题领域中的领域知识是至关重要的。在系统层，重要的是采用全系统的观点，以避免对 stakeholder 需求作过窄的解释。在这个层次上，不可避免地要引入解决方案偏爱。有开发过类似系统经验的人员或组织是必要的。类似地，子系统开发人员要有子系统特定领域的经验。

因此，同一个人不太可能承担所有层次的开发工作。即使同一个机构在多个层次上开展工作，所涉及的也很可能是不同的人员，而且常常来自不同部门。因此，引入每个层次的开发是为了响应上一层“客户”，并涉及下一层“供应商”的这种观点很有用。

2.3.1 输入需求与导出需求

图 2.3 显示出图 2.2 的另一种视图，在这种视图中将单个过程分开，强调由一个过程导出的需求成为另一个过程的输入需求，并自然地形成通用的工程化需求过程引入输入需求，生成导出需求的思想（也在图 2.3 中给出）。

2.3.2 验收准则与鉴定策略

在解释工程化需求过程内部细节之前，需要考虑既是输入到过程又是由过程导出的另一类信息。这就是有关需求鉴定策略的信息。

为了充分理解需求的重要性和更好地理解达成令人满意的需求共识是构成良好的开发基础，需要考虑当系统（或组件）被实现时，如何演示需求。对于每个需求，通过用于建立声称要实现需求的系统是否符合客户可以接受的准则，可部分地做到这一点。

这也需要确定检查准则的环境。第 1 章介绍了每个层次的测试计划概念。测试只是鉴定策略的一种类型，其它策略还包括试用、认证和检测。要使用哪种鉴定策略类型取决于系统的性质。例如，有安全方面准则的系统，与比方说管理信息系统相比，要做更仔细得多的检查。

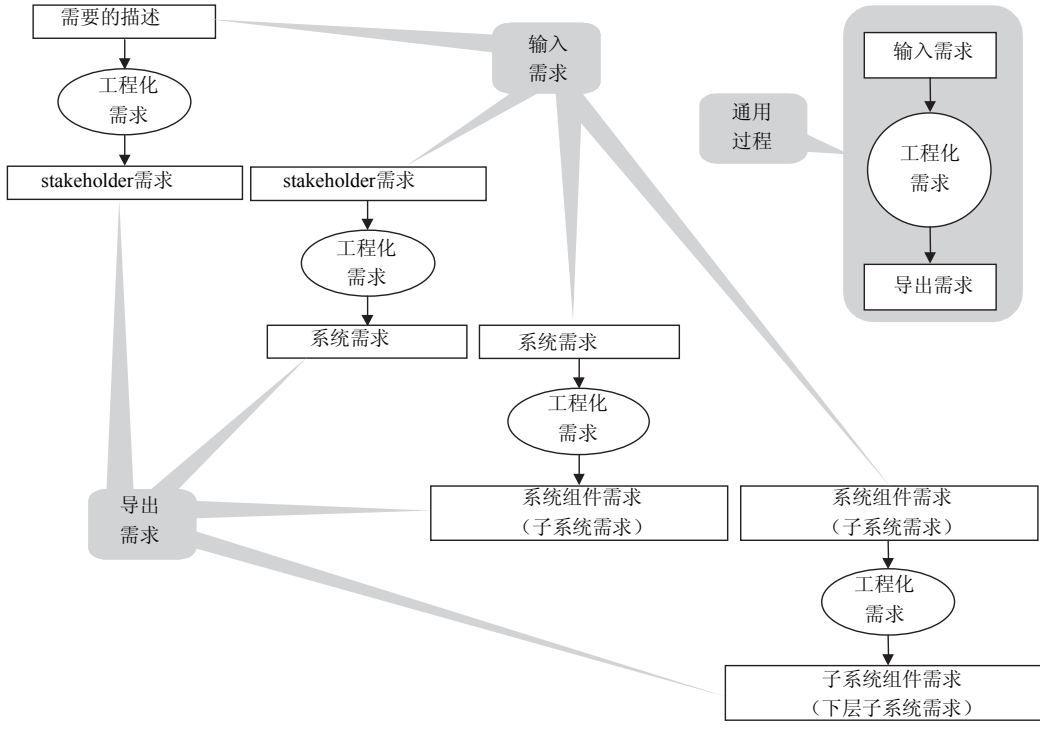


图2.3 标识通用过程的输入和导出需求

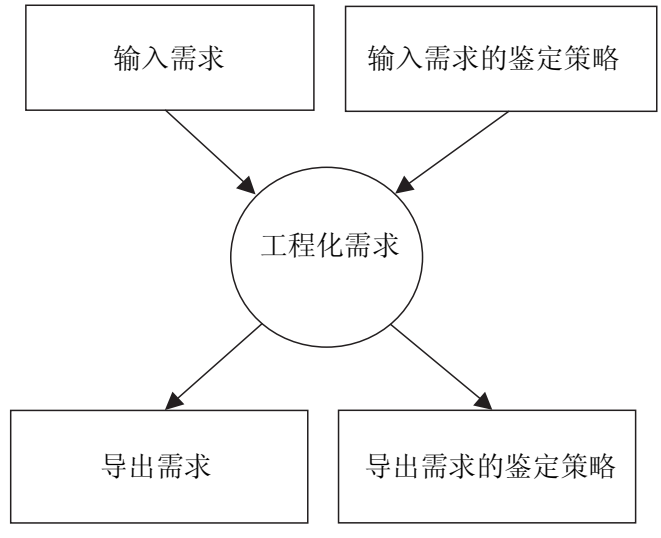


图2.4 鉴定策略是基础

因此，工程化需求的通用过程的完整上下文如图 2.4 所示。

鉴定策略常常对测试设备及现有工具的使用（例如风洞、隔音室等）和特殊诊断功能或监视点等产生新的需求。在有些情况下，会需要一个全新的项目来开发测试设备和其它所需的工具。例如，在航空设备开发中，在设备被安装到飞机上之前，由于成本和安全的原因，需要执行尽可能多的测试。即使设备被安装之后，也需要在试飞之前进行模拟运行。显然，必须使试飞飞行员确信，在第一次飞行之前，设备已达到已知的标准。

在需要制造部件的较低层次中，鉴定策略可能会考虑诸如是供应商还是客户负责每个部件的测试等问题。可能的策略包括交付前对所有部件的完整测试，由供应商执行批量测试，由客户执行随机检查。

2.4 通用过程介绍

介绍了通用过程的背景之后，下面可以深入讨论工程化需求的过程。首先在一切都没有变化的理想环境下来介绍这一过程，然后对过程进行修改以适应变化。

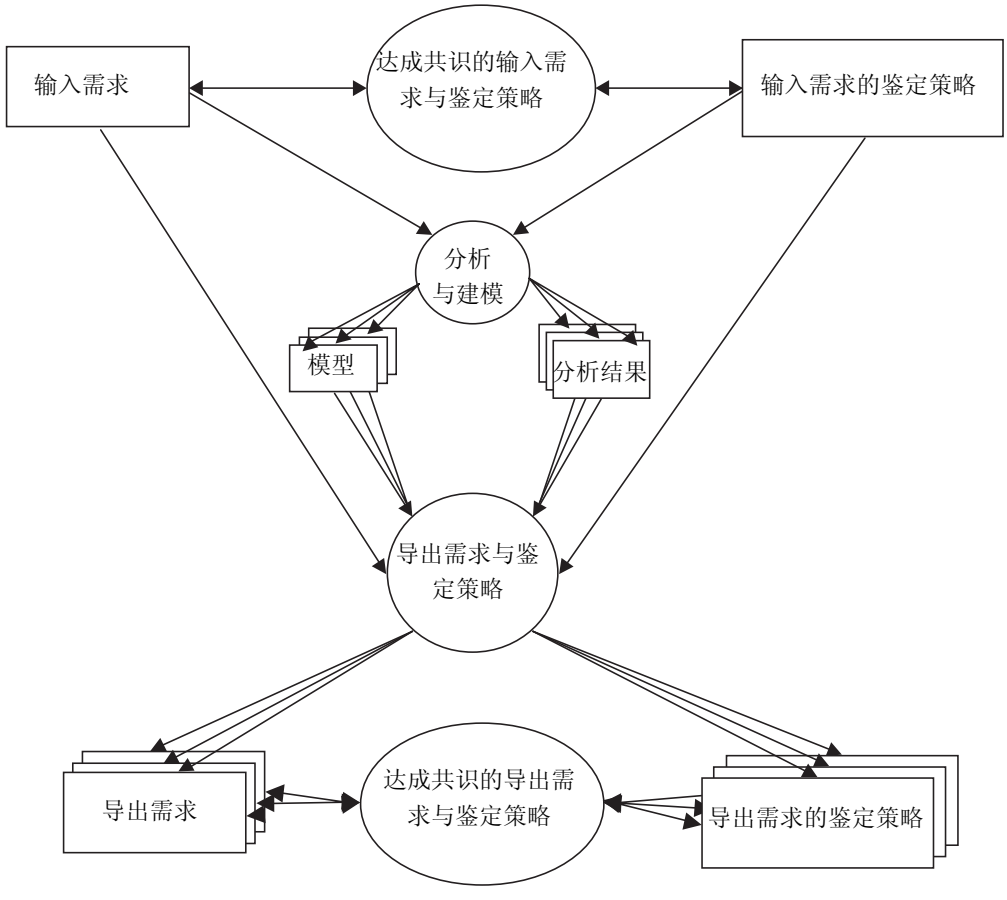


图2.5 理想环境下的工程化需求过程

2.4.1 理想的开发

理想环境下的工程化需求过程如图 2.5 所示。过程一开始就需要与上层客户就项目的输入信息达成共识。这种过程的第二个活动是分析输入信息，并考虑如何开发所需的输出。这种活动常常与就需求达成共识的活动并行开展，几乎总是要需要创建一种或多种模型，并编写分析报告，提供需求来源的基础，以及下层提供者的鉴定策略。当这些需求足够成熟之后，必须与供应商达成共识，以形成下层开发合同的基础。

图 2.5 还说明，可能生成多个导出需求集合。每个集合都必须与相关的供应商达成共识，有些供应商有可能负责多组需求。

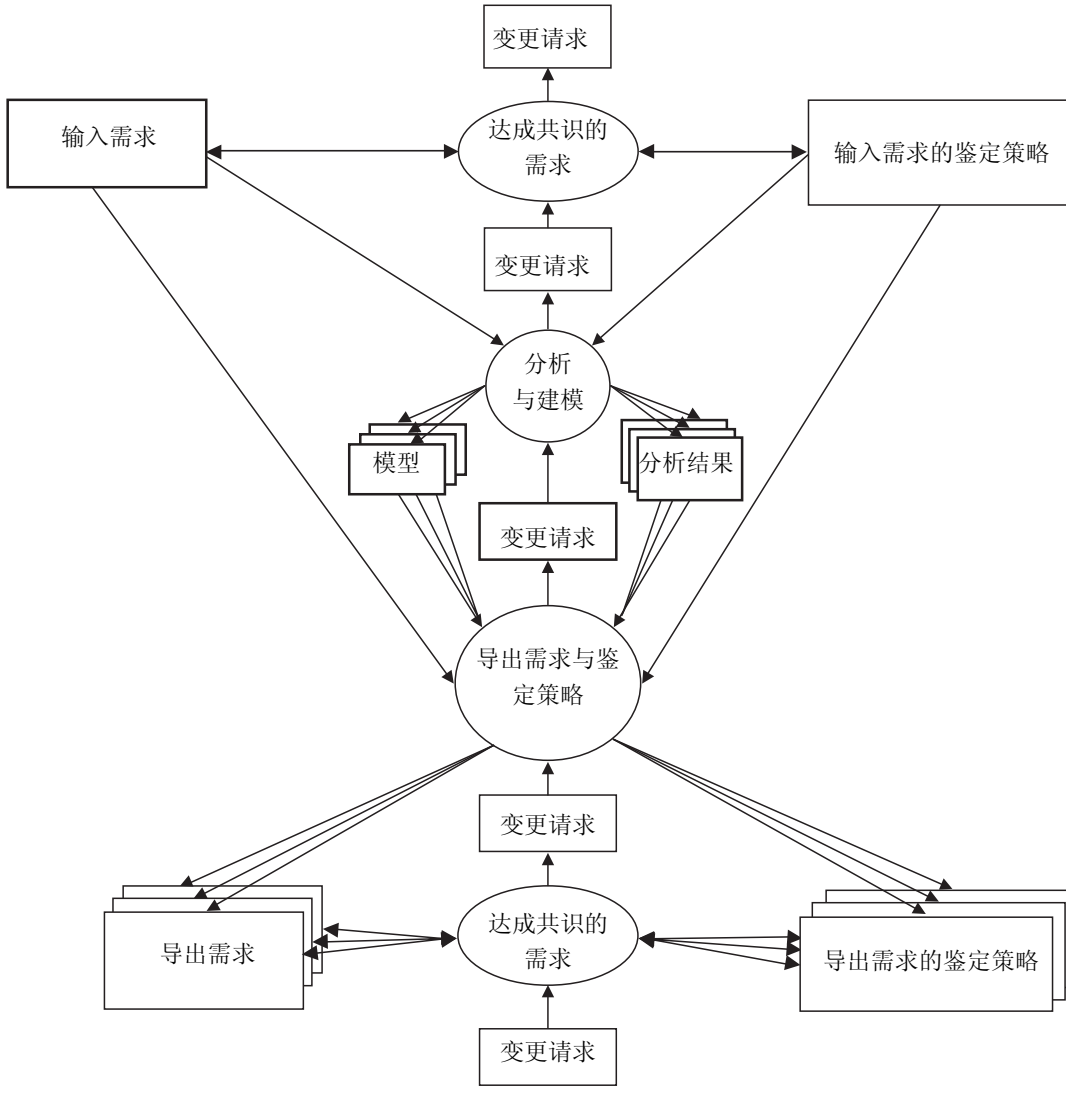


图2.6 变更背景中的加工需求过程

2.4.2 在变更背景下的开发

不幸的是，世界不会静止不变。在系统开发方面尤其是这样。看起来所有人都不断改变自己的想法，或发现以前已经达成共识的内容不再可能。因此，必须修改通用过程，如图 2.6 所示，以响应这种必然的变化。

变更被管理的正式程度，取决于项目的性质和状态。在早期阶段，可以并且必须能够很容易进行修改，以便项目可以展开。不过，到后来的某个时刻开始，必须做出承诺，并遵守正式约定。从这个时刻开始，通常要有一个更正式的约定，项目中的任何人都不能随意去修改。替代的过程是采用首先进行变更申请或提交，然后来根据它们会对项目造成的影响来决定。决策过程通常都要涉及一个人，例如项目经理，必要时在变更控制委员会成员的支持下，负责作出变更决策。同样，这些人员发挥作用的正式程度，也取决于项目的性质。第 8 章将在项目管理背景下，更深入地讨论变更管理问题。

从图 2.6 中可以看出，几乎所有活动都可以导致产生变更，并且这些变更通常会向上层流动。这并不意味着客户永远不改变其想法，也不意味着所发现的问题只是下层细节问题，根据自顶向下的策略来流动。实际情况是，向下的路径已经可以进行正常流动，但是返回路径必须明确地提供。例如，可能会产生变更请求的一种典型情况是在生成导出需求或导出需求的鉴定策略时，发现了很多模型局限或分析结果中的异常。变更请求会建议对模型进行修改，或进行进一步分析研究所发现的问题。类似地，在分析与建模过程中也可能发现输入需求中的问题产生对需求共识过程的变更请求。

2.5 通用过程信息模型

在考虑通用的工程化需求过程的子过程之前，引入支持这些子过程的通用信息模型是很有用的。

用于表示通用过程的框图既包含过程符号，也包含数据或信息符号。框图通过箭头说明要生成哪些信息，以及每个过程所使用的信息。

信息模型的用途是指示有什么类型的信息以及在信息项之间是否可能或应该存在关系。引入状态转移图来指示每种类型信息的状态如何随时间改变也是很有用的。这些状态转移图会直观地表示出过程之间在什么时候以及如何通过信息进行交互。

2.5.1 信息类

在通用过程背景下已经遇到的信息类型包括：

- 输入需求；
- 导出需求；
- 输入需求的鉴定策略；
- 导出需求的鉴定策略；
- 变更请求。

图 2.7 采用统一建模语言（UML）类图表示出这五种类型的信息。类的名称永远在类符号的最上部分（或惟一部分）给出。中间部分（如果有的话）指示类可以拥有的属性名称。下面部分（如果有的话）包含可以对类做的操作（常常叫做“方法”）。

连接类符号的连线表示类之间的关系，在 UML 中这些连线叫做“关联”。因此，一个输入需求可以通过“由满足”关系与一个导出需求联系。类似地，导出需求可以通过反向“满足”关系与一个输入需求联系。（这些符号在 UML 中叫做“角色”。）星号表示关联中可涉及类的零或更多个实例。这样，在图 2.7 所示的模型中，零或更多个输入需求可以被一个导出需求满足，一个输入需求可以被零或更多个导出需求满足。有些读者可能会对 0 下限存有疑问，因为这说明可能会没有任何关联。但是，如果下限被定为 1，则将意味着只有当存在至少有一个导出需求的关联

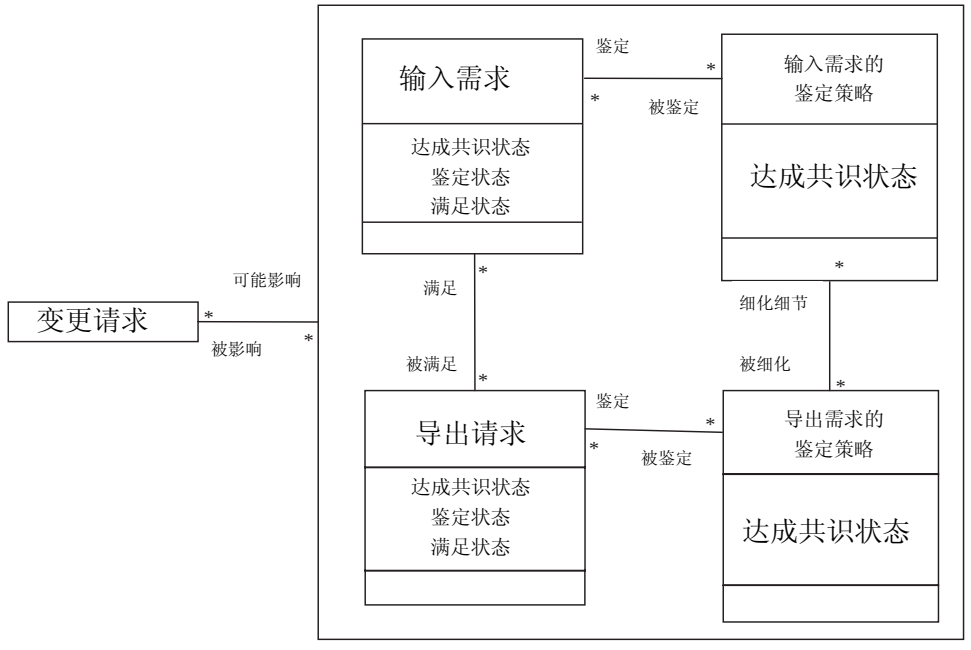


图2.7 通用过程信息模型

存在，输入需求才会存在。显然，这是不可能的。从本质上说，输入需求可以在被生成的导出需求之前存在。因此，这是一种合理的模型，因为项目开发过程中可以存在没有输入需求和导出需求关联的时刻，例如，在建立关联之前的在早期开发阶段。但是，项目经理期望尽快建立连接。这会说明取得了进展，而且所有导出需求都被检验是满足输入需求的，反过来所有输入需求也都得到了满足。

每个鉴定策略类都可以鉴定相应类型的需求，而且导出需求鉴定策略可以为输入需求鉴定提供更详细的信息。

“变更请求”可以用于任何其它四个类。通过将这个四个类包含在一个外矩形中并让关系连线接触这个外矩形来表示。

类符号的中间部分用于定义类将拥有的属性。每个需求类都有三个属性：

- 共识状态；
- 鉴定状态；
- 满足状态。

以下通过 UML 状态图定义这些属性。假设鉴定类的共识状态有“达成共识”和“未达成共识”两种取值。

2.5.2 共识状态

共识状态的状态图如图 2.8 所示。在这种框图中，每个（圆角）矩形表示某个历史点上的一个需求的状态。标出“被评估”的矩形叫做“超状态”，因为这个状态内部包含其它状态。一个状态与另一个状态的连接线表示引起状态变化的转移。

需求状态从所建议的状态开始。当客户满意地认为要发送给供应商的需求已经被充分描述，就会发送出需求。这时共识状态进入到被评估的超状态。在这个状态期间，客户和供应商要进行协商，直到出现达成共识的需求。

一旦处于达成共识状态，需求就会停留在这个状态，直到客户或供应商产生变更请求。如果出现这种情况，则这个需求的状态会重新进入被评估状态，直到出现新的达成共识状态。

在被评估状态中，客户和供应商会交替建议不同形式的需求，直到达成共识。因此，共识状态会处于所示两种状态之一，取决于是哪一方在进行当前的评估。

2.5.3 鉴定状态

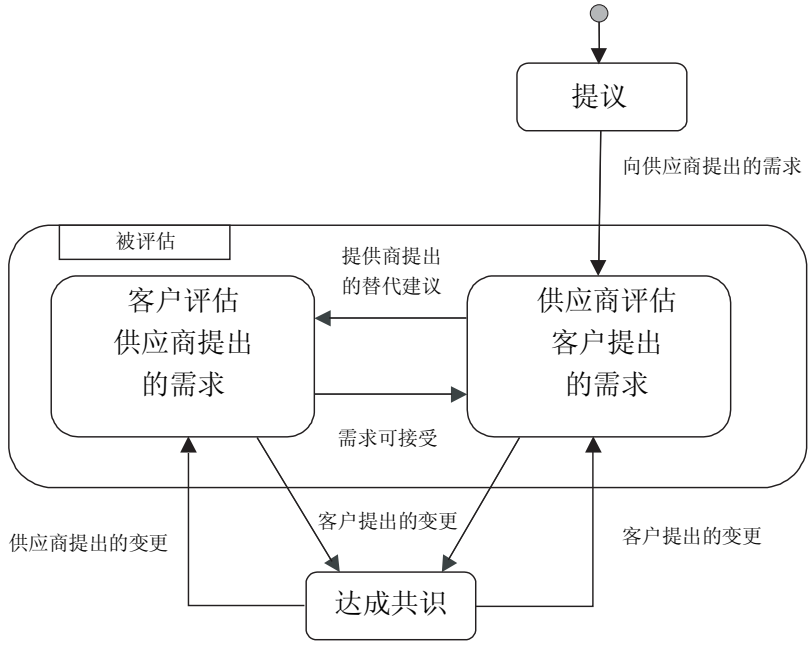


图2.8 共识状态的状态图

需求的鉴定状态在图 2.9 的状态图中给出。初始状态是没有已经确定的鉴定策略。当达成鉴定策略共识时，状态进入“鉴定策略已确定”状态。这种状态一直持续到收到变更请求为止。所请求的变更可能涉及需求本身，也可能与鉴定策略关联。当出现变更请求时，状态会变成“鉴定策略有疑问”，直到评估了变更的影响为止。这种评估要确定现有的鉴定策略是否成立，然后状态可以返回到鉴定策略已确定，否则必须确定是否采用其它鉴定策略，在这种情况下，状态变成尚未确定鉴定策略。

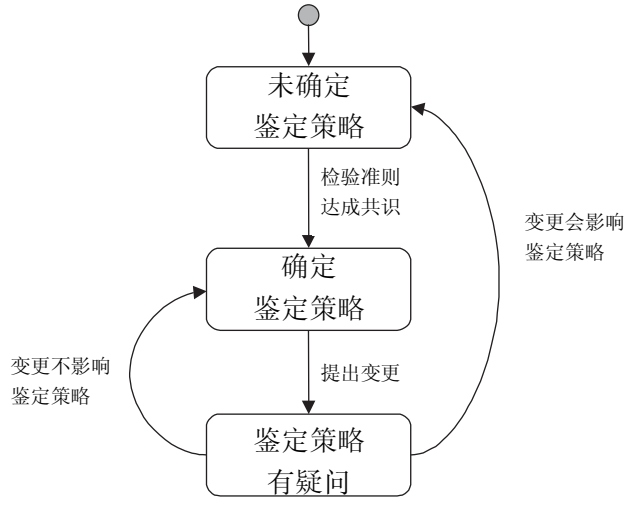


图2.9 鉴定状态

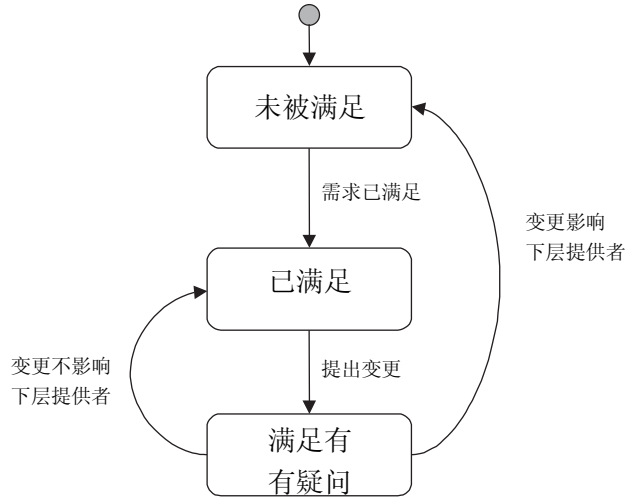


图2.10 满足状态

2.5.4 满足状态

满足状态的状态图如图 2.10 所示。这些状态的逻辑与鉴定状态的逻辑非常相似。开始点是“未被满足”状态，表示没有导出需求与这个需求关联起来。当输入需求被一个或多个导出需求所满足，并且与下层供应商对该需求达成共识，则状态进入“已满足”状态。请注意，在单个输入需求能够达到已满足状态之前，可能会需要对很多导出需求达成共识。

如果有人建议进行变更，则满足状态立即成为“满足有疑问”，不管所建议的变更涉及上层需求还是下层需求。这种有疑问状态一直持续下去，直到所建议变更的影响被评估完毕，这时满足状态可以变成“未被满足”或“已满足”。

2.5.5 信息模型的约束

变更请求将共识、鉴定和满足状态捆绑在了一起。提出变更请求立即会改变所有这三种状态，并要求开展额外的工作，首先确定变更是否会带来影响，其次要说明影响会产生的后果。请注意，满足状态可以对作为满足关系主体的需求产生影响。这种影响会对后续变更的产生潜在的影响，也就是变更的“影响”。

导出需求的共识状态必须与输入需求的满足状态一致，因为输入需求在下层供应商对满足输入需求的所有导出需求达成共识之前，不能达到其满足状态。

2.6 通用过程细节

2.6.1 共识过程

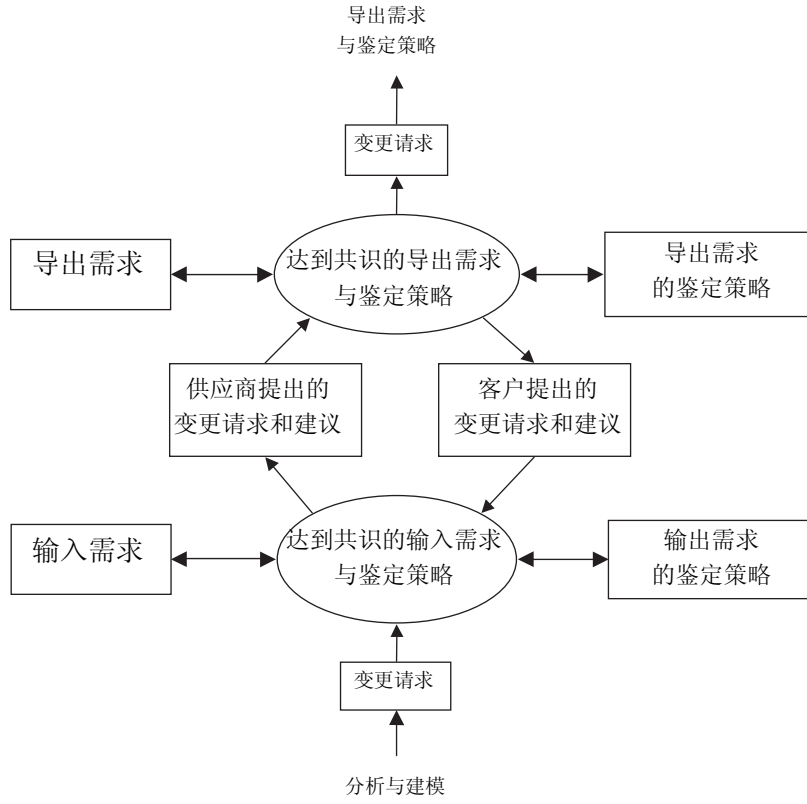


图2.11 共识过程

共识过程永远是在一个层次的供应商和该层以上的客户之间的一种协调活动，如图 2.11 所示。

在着手进行任何导出工作之前，需要评估输入需求，以确定输入需求是否构成后续开发的足够坚实的基础。

这种评估必须回答以下问题：

- 需求是完备的吗？
- 需求是清晰的吗？
- 需求是可实现的吗？
- 鉴定计划是清晰和可接受的吗？

对这些问题的各种回答，很自然地会导出拒绝需求的以下原因：

- 遗漏信息——例如使用“TBA”（要达成共识）、“TBC”（需要完成）或“TBD”（要确定）来做标记；
- 不够清晰——歧义、矛盾、难以理解等；
- 不能实现——没有已知的解决方案；

- 不可接受的鉴定计划。

根据评审，如果需求及其鉴定计划是可接受的，则状态被设置为“已达成共识”。

如果需求是不可接受的，则向客户提供修改表格，将下一步的工作转给客户，共识状态（如图 2.8 所示）变成“客户根据供应商的修改评估需求”。如果客户对修改感到满意，则客户可以将共识状态修改为“已达成共识”。如果不满意，客户会提出进一步修改意见，并转给供应商。共识状态变成“供应商根据客户的修改评估需求”，下一步工作转到供应商。

这种提议和反提议的过程一直持续下去，直到达成共识。当然，也可能永远也不能达成共识，并出现争论。

当任何一方提出变更时，就会进入“在评估中”的超状态，下一步工作转到接受变更的一方。前面介绍过的协商过程持续进行，直到能够达成新的共识。

在达成共识过程中，客户一方可能会提出变更请求，要求修改导出需求。这样会转到导出需求和鉴定策略过程中，以便评估变更的作用，也需要在一条或多条导出需求的一些地方进行调整。当然，所提出的变更请求有可能难以在这个层次上完全处理，需要上交给建模和分析过程处理。这需要将问题逐级上交，由这些层次的人员进行决策。换句话说，需要同时协调多个层次的决策。这种要求完全破坏了“瀑布”生命周期原则。根据这种原则，活动要严格以自顶向下的顺序展开。开发工作并不是串行活动，而是一些协商合作决策的并行工作。

在很多项目中，验收准则和鉴定计划在相当晚的时候才能确定，有时要在对需求本身达成共识之后很久，有时会在在测试开始之前才能确定验收准则和鉴定计划。这是一种很差的实践，通常会由于很晚才变更需求，使得需求很难测试，最终导致延期。

2.6.2 分析与建模

这个过程的分析部分主要是理解输入需求的性质和范围，以便评估满足这些输入需求有可能带来的风险。分析工作的范围可以从可行性研究到探索构建某些关键或高风险组件原型潜在实现的选择。常常需要构建性能模型，以便研究潜在的处理量和应答数据。

这个过程中模型的其它用途包括理解并提供导出需求的结构。理解和构建 stakeholder 需求的最常见的模型是用例或用户场景。这些有助于理解人们怎样使用待开发的系统。

对于构建解决方案域中的解决方案来说，最常见的模型是设计体系结构。设计体系结构标识解决方案的要素，并说明这些要素如何交互。

表 2.1 建模手段举例

飞机工业
空气动力学模型
三维空间模型
重力分布模型
飞行模拟器
铁路工业
时刻表模拟
安全性、可靠性和可维护性模型
汽车工业
款式模型
仪表板模型
空气动力学模型

在大量情况下，模型被用来建立所建议解决方案的设计体系结构。对于存在事实上的体系结构并且发展很成熟的开发领域（例如汽车、电信、飞机等），这些模型常常相当明确。但是对于没有已建立体系结构的全新开发领域，模型可能更抽象一些，以容纳潜在的不同变化。

一般来说，所使用的模型完全依赖于所承担开发工作的性质。前面已经讨论过，所使用模型的类型与具体领域有很大关系。在软件系统中，越来越多地使用了对象模型。表 2.1 给出了在三种工业领域中所使用的不同类型的模型。

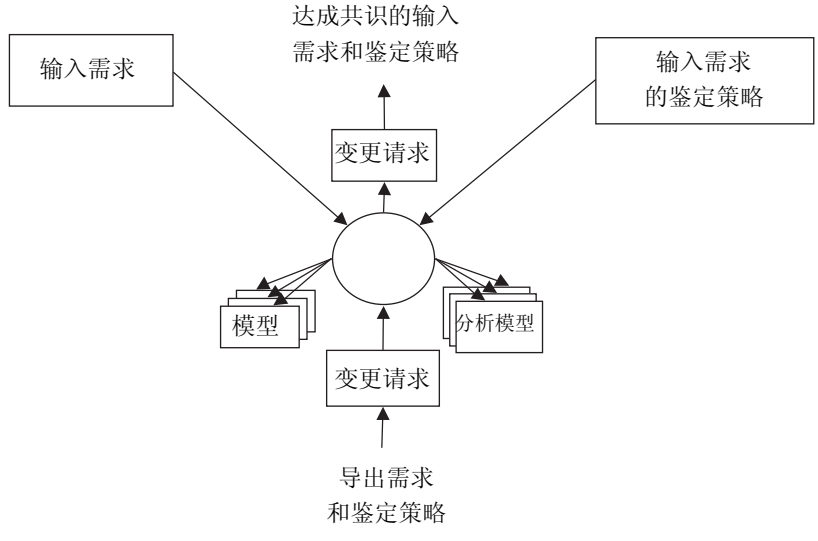


图2.12 分析与建模过程

开发模型的核心是理解输入需求与所建议的鉴定策略和在确定怎样生成导出需求之前试验不同解决方案。这种工作还要考虑导出需求的鉴定策略，而这种考虑又会对测试设备或软件产生需求。这种工作还要标识导出需求的鉴定需求。

分析和建模过程（如图 2.12 所示）可以与共识过程并行展开，因为分析和建模过程很可能产生对需求性质的更深理解。

第 3 章将总结一些广泛使用的建模技术，特别是软件业所使用的建模技术。第 5 章将介绍如何使用用户场景模型来辅助理解 stakeholder 需求，第 6 章将讨论面向功能模型，这种模型有助于制订系统需求框架。

在分析和建模过程中，很有可能产生有关输入需求含义和表达方式方面的更多问题，从而引出变更请求，使得重新进入对需求达成共识的过程。

2.6.3 导出需求与鉴定策略

图 2.13 说明了导出需求和鉴定策略过程。

导出需求

针对导出需求的模型的使用方式有各种各样，但是最初要考虑的是基于设计体系结构导出组件需求的最简单的方式。通过这种方式，可以确定由每个组件必须满足的具体需求。这些需求中

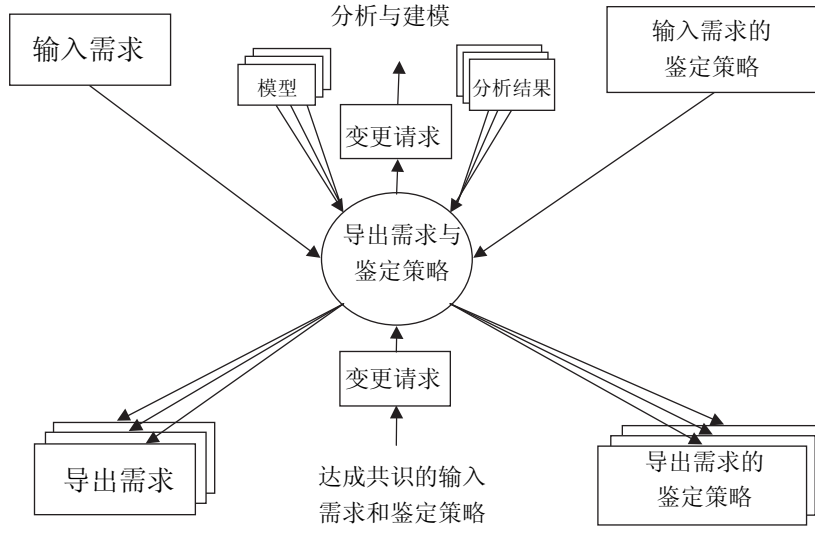


图2.13 导出需求和鉴定策略过程

的一些可能与一个或多个输入需求相同，有些可能要从输入需求中导出，以便将其在组件之间划分。进一步的需求集合包括由组件体系结构或输入需求引入的约束。这些约束包括接口约束，以及诸如重量、体积、功耗和散热这样的可能的物理约束。

在实践中，组件需求的定位或导出工作，可能要在输入需求及其鉴定策略达成最终共识之前进行，但是不会在达成最终共识之前完成这种活动。

除了制订组件需求之外，还需要确定输入需求和导出需求之间的满足关系。这种关系说明哪个输入需求是由哪个导出需求满足的，并可以用来确定：

- 所有输入需求都被满足；
- 所有导出需求都是必要的（即导出需求直接或间接地满足一个或多个输入需求）。

只是断定存在像交叉引用矩阵那样的满足链还不够，还需要对每条满足链作出合理解释。这类合理解释构成满足论据。

在通过模型生成需求过程中，可能会发现一个或多个模型中的缺陷或疏忽。这会导致反过来对建模小组提出变更请求，建模小组会直接修改模型，或进一步澄清或修改输入需求。这样的变更引入过程将持续下去。

导出鉴定策略

前面已经讨论过，满足关系是有关根据输入需求生成导出需求的关系，即如何设计系统。而鉴定策略则计划每个需求将怎样在每个层次上演示。

鉴定策略包括一组鉴定活动，每个鉴定活动都是一种具体的试用、试验或检测。针对每个需求可能有多个鉴定活动。

每个鉴定活动都应该考虑以下问题：

- 适合该需求的活动种类；
- 采取每个活动的阶段，越早越好；
- 活动可能需要的特殊设备；
- 活动获得成功结果需要什么条件。

鉴定计划的结构可以按照阶段组织，也可以按照活动类型组织。

鉴定活动的定义应该适合需求的层次。换句话说，stakeholder 需求会引出验收试验，而系统需求会引出系统测试，即向客户交付之前的测试。不一定要根据 stakeholder 需求定义系统测试，

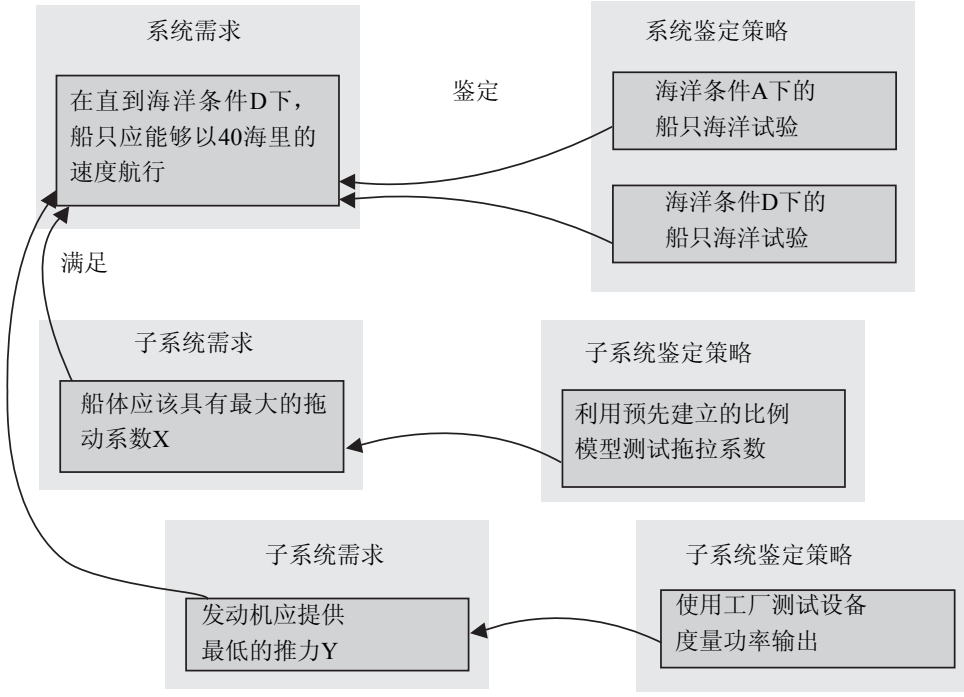


图2.14 鉴定信息

因为从 stakeholder 需求导出的系统需求会有自己的系统测试。

例如，如图 2.14 所示的轮船系统需求被分解为不同子系统，即船体和推进系统的两个需求。要根据系统级需求制订两个鉴定测试，并根据子系统需求制订另外两个鉴定测试。

因此，为了充分理解需求是如何被测试的，既需要满足关系也需要鉴定策略。为了理解高层需求的鉴定状态，需要通过满足关系及鉴定关系来考虑针对下面各个层次的需求鉴定活动结果。

2.7 小结

本章介绍了可以同时应用于系统开发各个层次的一种通用过程。这种通用过程的优点，是可以标识适用于各个层次的公共活动：

- 就输入需求与客户达成共识；
- 分析输入需求，以确定满足这些输入需求的风险和潜在问题；
- 创建一个或多个模型，以研究针对导出需求的可能策略；
- 通过分析和建模信息，生成从输入需求导出的需求；
- 就导出需求与负责实现这些需求的小组达成共识；

- 建立输入需求和导出需求之间的满足关系；
- 建立导出需求与相关的鉴定策略之间的鉴定关系。

这些活动最终根据所提供的信息模型确定信息。信息的当前状态可以用于度量进展，评估所建议变更的影响，以及定义评价项目工作情况的指标。例如，需求状态可以通过三种属性获取：

- 达成共识；
- 满足；
- 鉴定。

所有系统开发中的对于需求的理想状态都应该是：

- 在客户和供应商之间达成共识；
- 针对这种共识，有达成一致的鉴定策略；
- 被下层需求（或设计）满足。

项目实际需求与这种理想状态的偏离程度，反映出从需求管理角度看的项目风险程度，也反映出使需求处于理想状态所必须完成的工作。

艺术与科学在方法上存在共同点。
(Edward Bulwer-Lytton，诗人，1803－73)

3.1 引言

结构化技术和方法的使用是需求工程的一个重要特征。这种技术的基础是系统开发所涉及的符号、表示和模型。系统工程师的创造性和艺术性在很大程度上要通过使用这种建模技术来表达。本章将讨论一些这类表示法，以及使用这些表示法的一些的需求工程方法。

3.2 针对需求工程的图形表示

3.2.1 数据流图

数据流图（DFD）是大多数传统建模方法的基础。数据流图的要素包括：

- 数据流（带标签的箭头）；
- 数据转换（圆圈或“泡泡”）；
- 数据存储（水平平行线）；
- 外部实体（矩形）。

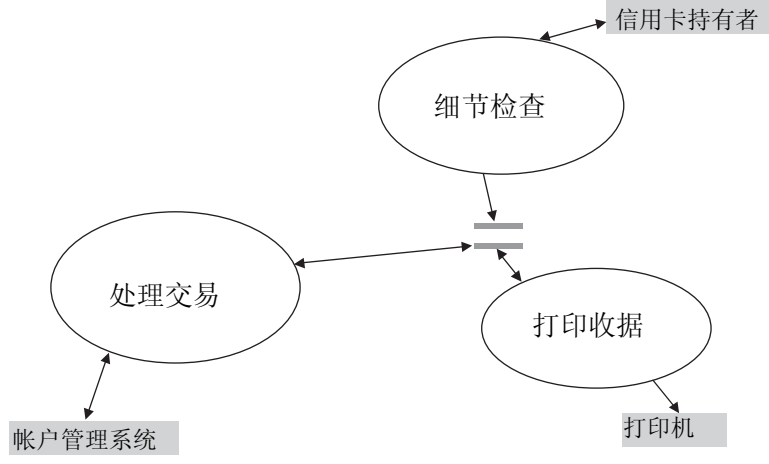


图3.1 数据流图

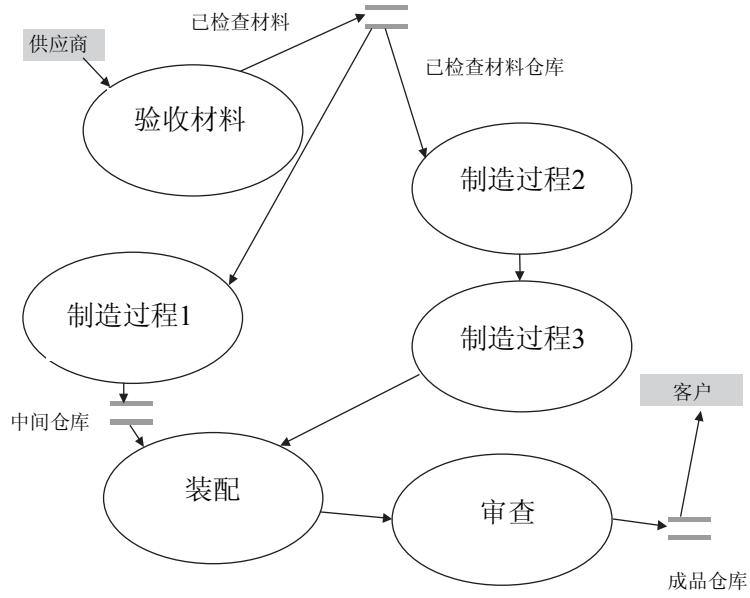


图3.2 某个工厂的数据流图

数据流图是系统结构和接口的最低程度的图形表示。尽管数据流图最初是用来针对数据操作和数据流的，但是它可以用于表示任何类型的流程。

图 3.1 给出的简单例子，显示的是针对传统信息系统使用数据流图的情况。

图 3.2 显示的是针对完全非基于信息的系统的数据流图。

流程表示两个操作之间的信息或物质交换。在现实世界系统中，流程可以是连续的、按需设置的、异步的，等等。数据流图显示物质可以选择的所有通路，但不是控制流。数据流图必须由每个过程、数据存储和流程的文字说明来支持。

数据字典被用来定义所有数据流程和数据存储。每个叶节点泡泡都要定义由系统组件提供的基本功能。这一点用 P-说明 (*P-spec*) 或最低-说明 (*mini-spec)* 表示。这是一种常常以伪代码形式编写的文本描述。

背景图是数据流程图的顶层框图，用来显示与待开发系统交互的外部系统，如图 3.3 所示。

泡泡可以向下层分解。每个泡泡都可以扩展为一个框图，在这个框图本身又可以包含泡泡和数据存储（如图 3.4 所示）。

下面考虑一个“救护车指挥与控制”系统的背景图例子（如图 3.5 所示）。这是系统数据流分析的切入点。

主要的外部实体是打来急救电话的呼叫者，以及由该系统控制的救护车。请注意，记录是系

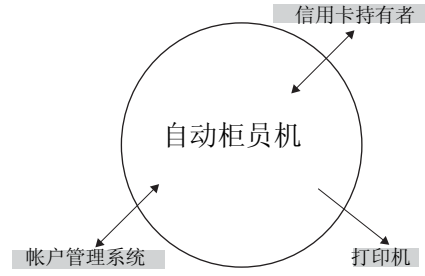


图3.3 背景图

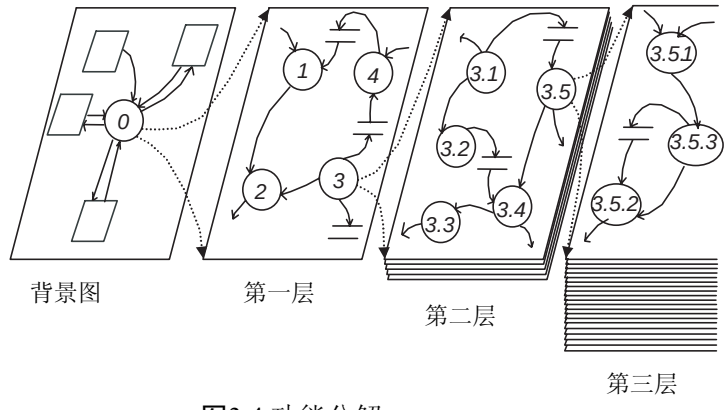


图3.4 功能分解

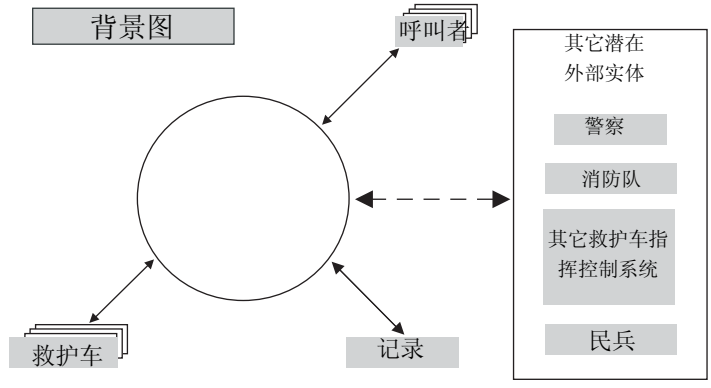


图3.5 “救护车指挥与控制（C&C）”系统的背景图

统的一种重要输出（实际上是一种强制需求），是度量系统“性能”的一种很重要的手段。

实际系统所需的其它潜在外部实体本来都应该在框图上标出，但是为了简洁起见，在这里省略了。

下一步是标识系统的内部功能。通常首先画出作为最低限度分解的每个外部实体的功能，然后画出必须在这些顶层功能之间流动的基本数据，如图 3.6 所示。

接下来对顶层功能进行分解，并包含更多细节，如图 3.7 所示。

由一组数据流图表示的这种功能层次结构，可以被用作导出并结构化系统需求的框架。图 3.8 给出了从图 3.7 中导出的“救护车指挥与控制”系统例子的功能结构，图 3.8 还给出了从这种结构导出需求的一些示例。

这种逐层分解和接口，很好地描述了组件模型，但是不能很好地描述跨系统的“事务”，即从图 3.9 中可以看到的从输入到输出（或完成某种系统动作）的过程。

因此，需要通过事务跨系统所经过的路径、所耗费的时间、以及所吸收的资源，来观察这些事务。仿真系统需求并观察功能操作会说明主要事务，不过表示系统事务的另一种方式，是像 3.10 所示的那样，在数据流图中采用粗箭头作标记。

数据流图是表示结构的一种好方法，但是不是太精确。数据流图在开发完整的系统定义上，不如文字精确，因为接口线可以表示任何内容，而单句可以归纳任何内容。数据流图不能恰当地处理约束。

数据流图可以清晰地显示功能和接口，可以用来标识完整的事务过程，但是不能直接显示这些事务。在理想情况下，我们期望通过“就地展开”方法观察框图内容，以便能够看清将要分解的各个层次中的内容。几乎没有 CASE 工具提供这种功能。

图 3.7 的画法实际上违反了数据流图的规则，因为它给出了将整个系统到多个过程的分解，同时又显示出系统必须与之交互的外部实体。我们通常采用数据流图的务实用法，而不是严格遵循纯理想概念。为了精确遵循数据流图的画法规则，外部实体代理只能出现在背景图中，因此在下层是不应该出现的。但是，如果不显示外部实体，并且使到外部实体的流程断在那里（这是外部实体已经定义了的规则），则数据流图的意义会差很多。

下面小结一下。数据流图：

- 可以显示功能总体结构和流程；
- 可以标识功能、流程和数据存储；

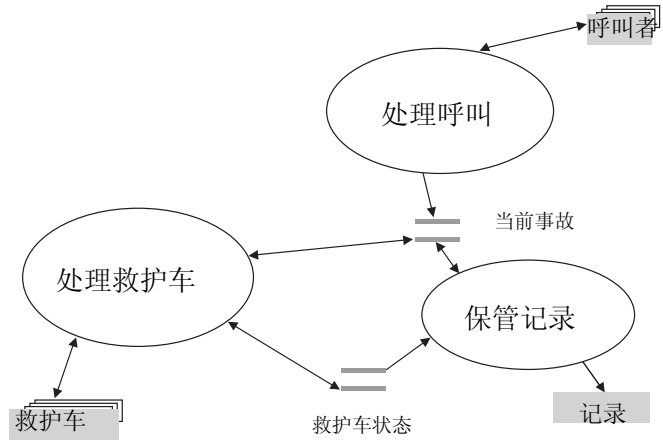


图3.6 “救护车指挥与控制”系统模型

- 可以标识功能之间的接口；
- 可以为导出系统需求提供框架；
- 已经有可用的工具来支持；
- 已经被广泛用于软件开发；
- 适用于一般系统。

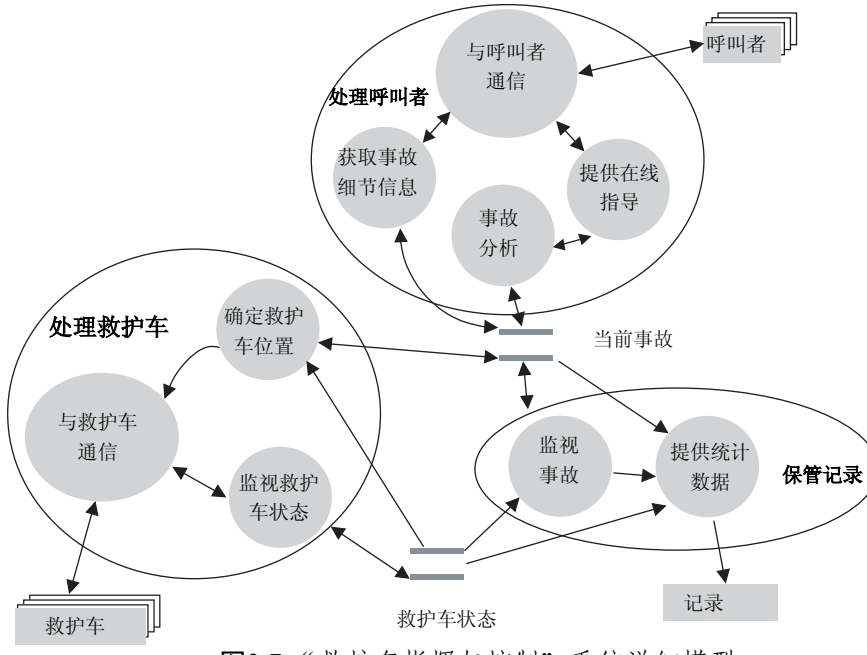


图3.7 “救护车指挥与控制”系统详细模型

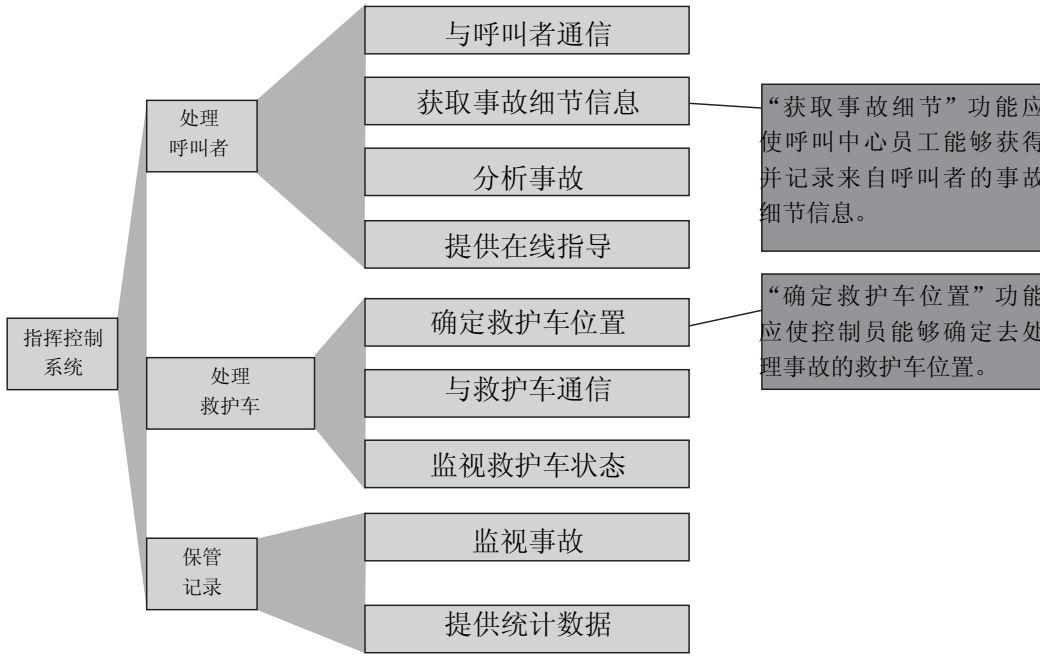


图3.8 “救护车指挥与控制”系统的功能结构

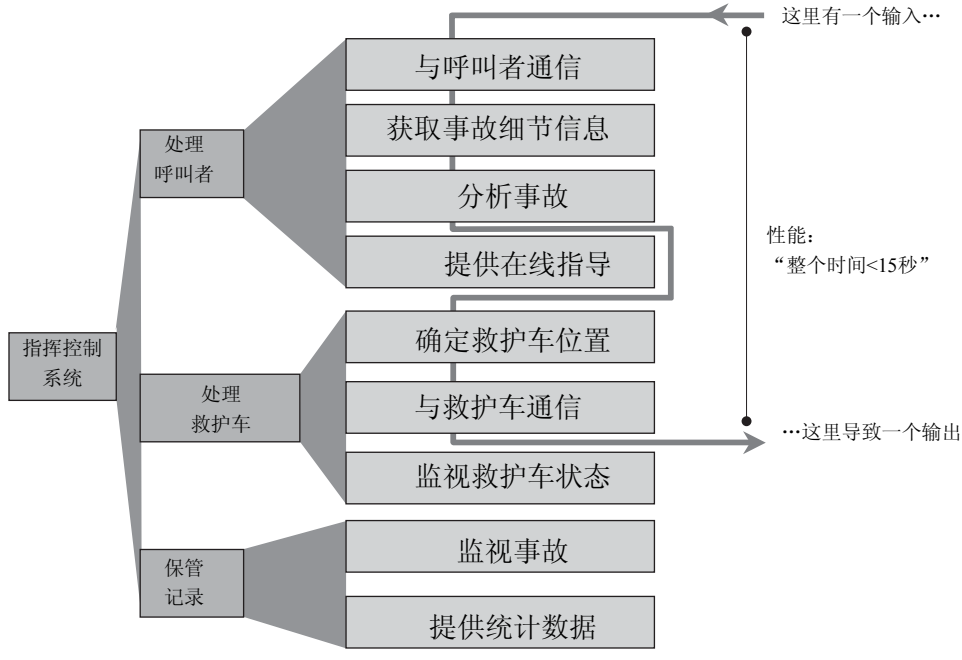


图3.9 系统事务

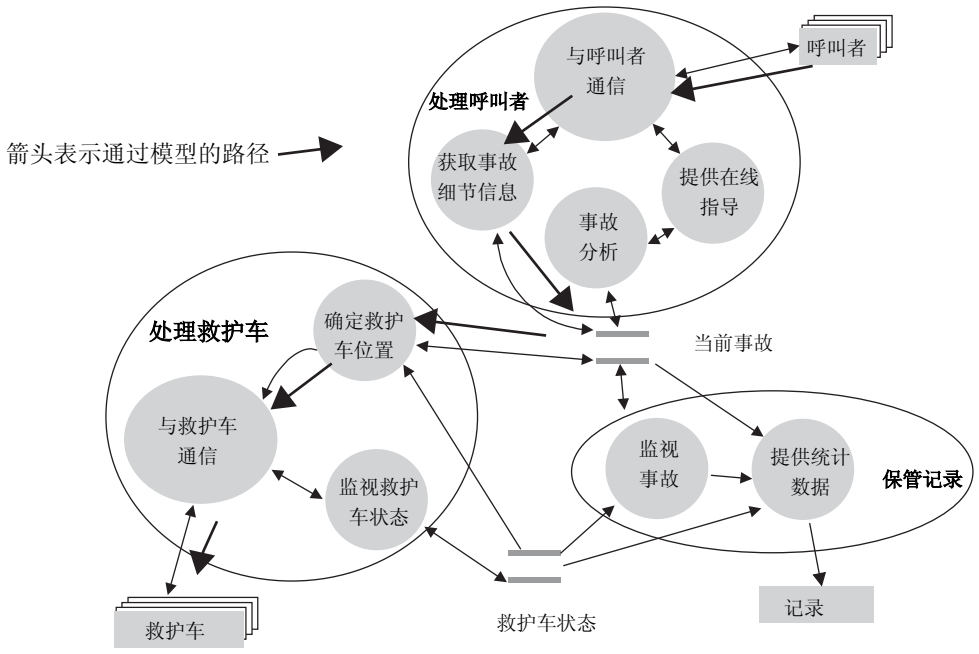


图3.10 “救护车指挥与控制”系统的系统事务

3.2.2 实体关系图

对系统中的信息建模，例如飞行计划、系统知识和数据库记录，常常是很重要的。实体关系图（ERD）是提供对感兴趣的实体，以及这些实体之间存在关系的一种建模手段。实体关系图是Chen（1976）最初提出的，现在已经有很多不同的实体关系图标记。

实体是一种能够被惟一地标识的对象，例如客户、供应商、部件或产品。性质（或属性）是描述实体的信息。关系具有基数，可以表示实体之间关联的性质（一对一、一对多、多对多）。子类型是另一个实体的子集，即如果类型 X 的每个成员都属于 Y，则 X 是 Y 的一个子类型。

实体关系图通过标识系统内部的实体和实体之间的关系，定义系统的一种局部模型。这种模型独立于生成或使用信息所需的过程，因此是供在系统需求阶段所需的重新建模工作使用的一种理想工具。图 3.11 中包括了“救护车指挥与控制”系统例子中的实体。

3.2.3 状态转移图

功能和数据流对于需求定义是不够的，还需要能够表示系统的行为，并且在有些情况下，将系统看作拥有有限个数的可能“状态”，将外部事件看作导致状态转换的触发器。

为了表示这些内容，需要研究系统可以进入的状态，以及系统处在这些状态下如何响应事件。进行这种研究的一种最常见的方式，是使用状态转移图（STD）。状态转移图可以在抽象模型中用于范围很宽的各种不同目的。有关进一步信息，请参阅文献 Ward 和 Mellor（1985）。

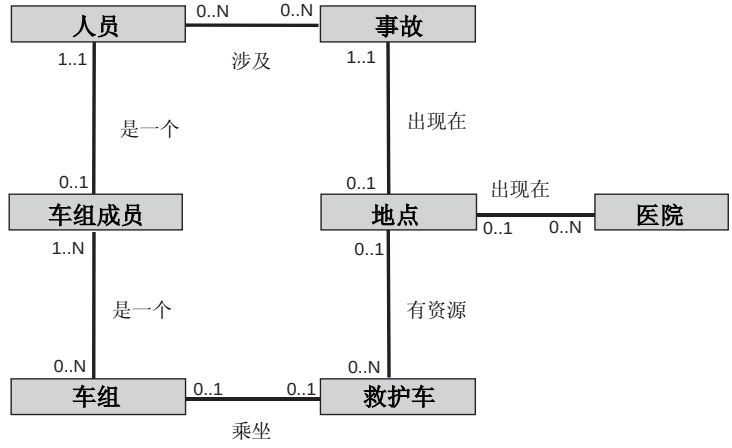


图 3.12 给出了飞机从起飞到返回航站期间内的不同操作模式。

状态转移图也可以定义任何实体响应事件的不同状态。例如，救护车在其生命期内具有的不同状态，包括运行状态、维修状态、待命状态、响应急救求状态等。状态转移图的更细层次可以采用启用、禁止或触发过程响应事件的方式来定义控制过程的控制结构。

与状态转移图有关的两个弱点是：

1　状态移换图不能表示并发活动，也就是说，状态转移图只限于描述有限状态的自动机。

2　状态转移图通常不能表示状态的层次结构。

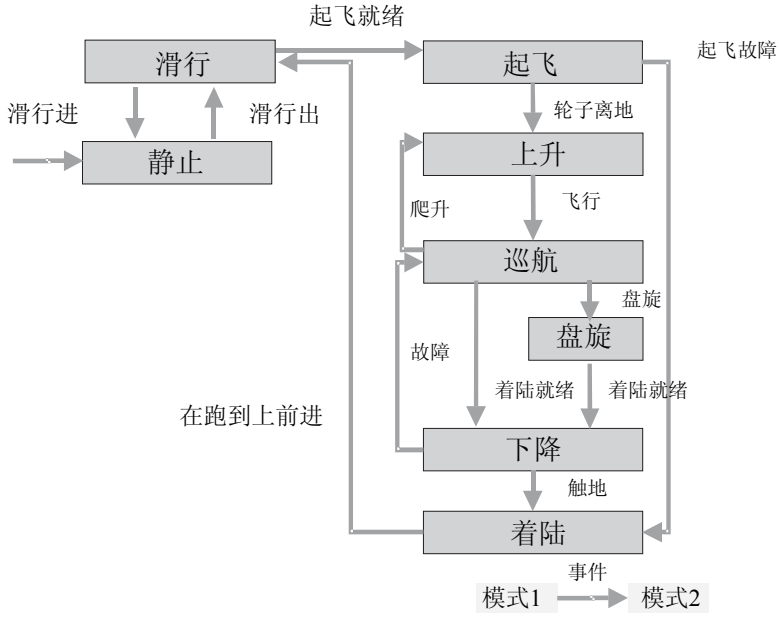


图3.12 飞机飞行状态转换图

3.2.4 状态图

Harel 的状态图通过单一框图来获取内部的层次结构（Harel 1987）。状态图还能够表示并发活动，因此没有有限状态自动机的限制。所以在普遍存在并发活动的实际环境中，状态图要比状态转移图有效得多。状态用带标签的圆角方框表示，层次结构用封闭线表示，状态之间的转移用带有事件描述标签的有向弧线表示。

图 3.13 是对图 3.12 的重画。两个顶层状态是“飞行”和“地面”，在两个状态之间定义了转移。在“飞行”状态中有三组独立状态，在“地面”状态中有“能够滑行”状态和“位于跑道”状态。在“地面”状态中，还有“滑行”和“静止”两个更进一步的状态。

当飞机的轮子离开地面时进入“飞行”状态，当轮子接触到地面时进入“地面”状态。这些状态现在可以进一步分层细分。如果状态细化需要涉及飞机制动和转向，则“地面”状态会变得相当复杂，对这一点大家都不会感到意外。

状态图还进一步引入一个与历史有关的很有用的概念。如果重新进入带有（H）注释的状态，则也要重新进入所离开的子状态。

与状态转移图一样，状态图也用于提供系统的行为描述。但是，状态转移图和状态图之间在每种表示法所获取的内容方面也有一些重要差别。两者的状态、事件和转移描述是相同的。状态转移图对所发生的动作描述得更详细，而状态图在抽象、默认、历史和范围描述方面，有更细化的机制。前面已经介绍过，缺乏层次结构会限制状态转移图，而状态图可以描述整个系统。在这

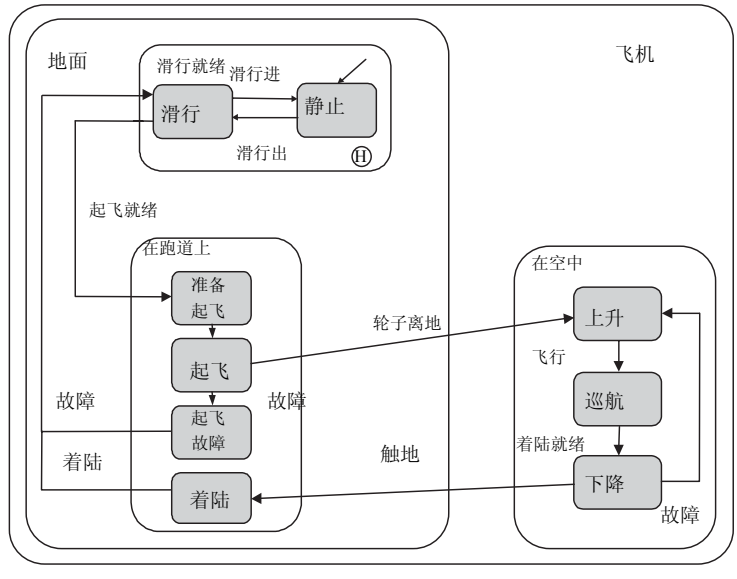


图3.13 飞机飞行状态图

个意义上，状态转移图和状态图具有互补性，状态转移图更适合对问题建模，状态图更适合对解决方案建模。此外，状态图具有正交描述的优点。

3.2.5 面向对象的方法

面向对象提供了与结构化分析方法相当不同的方法。假设条件是：

- 对象可以封装行为（状态和事件）、信息（数据）和一组动作；
- 对象是稳定的，因此是持久和可重用的；
- 对象将尽可能在生命周期的初期定义；
- 同样的对象集合可以用于系统需求和初步设计；
- 可以通过具体化现有对象创建新对象，而不只是创建新类型。

对象描述稳定（并且有希望）可重用的组件。面向对象试图通过要求开发人员选择持久对象来最大化可重用性，即可以用于系统需求和设计的内容。

因此，面向对象的目标是：

- 在相同对象内部封装行为（状态和事件）、信息（数据）和动作；
- 试图定义持久对象，可以在需求和设计阶段使用；
- 通过更详细地定义对象来增加信息；
- 通过具体化现有对象创建新对象，而不是直接创建新对象。

面向对象关注对象的行为以及对象之间的关系。有时设想对象具有扁平组织，但这不是必要的，甚至是不希望的。分析人员寻找的是生存期长的实体，并对围绕这些实体的系统行为建模。这种方法给出一种系统的一致行为定义。系统元素应该是可重用的，因为元素（如果不是其行为）可以渐进增强。

有些方法论学者坚持认为设计（甚至实现）是分析模型的细化。这对于非平凡系统来说可能是一种苛求。但是，从分析、设计到实现的进展，面向对象常常比其它方法清晰得多。分析元素最终应用到实现中的要比结构化分析与设计中的多，这对可跟踪性和可维护性会有极大帮助。

类图

类图是面向对象分析和设计的基本框图标记。面向对象起源于基于计算机的模拟。其基本原理是软件系统的内容应该对现实世界建模。处理这个问题的一种很自然的方式就是在软件中提供表示现实世界实体信息和动作的对象。

例如，在银行系统中，不是提供一个帐户文件和单独的帐户程序，而是提供帐户对象，包含诸如余额和透支限额以及与其它对象的关系，例如帐户拥有者之间的关系等信息。这些对象拥有操作（又叫做方法），以处理按帐户执行的动作，像检查余额、存款和取款等。

这种方法背后的最初思想是使软件开发大大接近于建模，因此更加自然。与很多好思想一样，面向对象也受到实用性的影响，并且很少有面向对象软件系统可以被看作是现实世界的纯粹表示。不过在这种方法中仍然有相当大的优点。

类（或对象）图是大多数面向对象方法的基本工具，如图 3.14 所示。

类图表示关于对象的类及其关系的信息。在很多方面，类图类似实体关系图。与实体关系图一样，类图也显示特定类的对象如何与相同或不同类的其它对象关联。所增加的主要信息是：

- 操作（方法）；
- 泛化的概念；
- 对象内部的属性。

3.3 图形表示与信息

方法使用从自然语言、图形形式，到形式数学的各种表示。这些使用图形表示的方法，通常叫做“结构化方法”，使用数学工具的方法叫做“形式方法”。

方法中所用的图形表示的目的是捕获信息。通过定义图形所代表的一组概念来捕获信息，图形的画法则由语法规则来规定。

上一节已经讨论了图形表示问题，这里将框图例子和框图所表示的概念归纳如下：

数据流图	功能与数据流
控制流图	跨功能的控制流
实体关系图	保留的信息及其关系

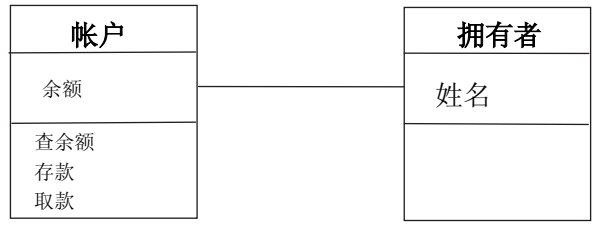


图3.14 类图

状态转移图	状态动作和事件
实体-生命-历史图	实体和事件
对象图	对象及其关系

不同的建模概念并不是独立的，在不同概念之间存在关系。例如：

- 数据流图上的流程，可以携带触发状态改变的事件。
- 状态转移图上的动作，可以由数据流图上的功能执行。
- 状态转换移的状态，可以描述功能、实体或对象的状态。
- 数据流图上的数据存储，可以包含实体关系图的一个或多个实体。
- 关系是事件的一种记忆。

在理想情况下，商业图形工具会尽可能自动地组织这种关系，使我们能够表示其它关系，并找出不一致的地方和遗漏元素。事实上，商业图形工具常常是独立的。

3.4 方法

3.4.1 方法中有什么？

方法比模型更具有描述性——模型告诉我们要做什么，以及以什么顺序做；方法告诉我们什么时候、在什么地方使用图形表示。

本章前几节已经介绍过，系统建模使用了各种不同的图形表示。大多数方法，例如 Yourdon（1990）、DeMarco（1978）、Shlaer 和 Mellor（1988）、Rumbaugh（1991a），这里只列举几个，都是这些概念的再组织，在选择和完成的顺序上有所不同，常常还有一点增强。这些方法的相似性，要比差异显著得多。表 3.1 分类归纳了这些方法。

表3.1 方法中有什么？

概念的已定义类型	功能，流程，实体，事件
这些概念的结构	层次结构，封装
图形表示	数据流图 状态转换图 信息关系图
定义技术	如何创建数据流图 如何创建实体关系图
创建图形表示的序列	功能流程框图，数据流图，状态图，信息关系，控制逻辑

3.4.2 结构化方法

受控的需求描述（CORE）

CORE 最初是在由英国国防部完成的需求分析工作的基础上发展起来的。他们工作的主要发现是在开始评估可能的解决方案之前，方法常常从定义问题解决方案的背景入手，而不是试图定义问题本身。CORE 就是专门针对这种方法的。图 3.15 说明了 CORE 所使用的概念和图形表示。

CORE 的核心概念是视点和叫做视点与其层次结构的相关表示法。视点可以是对待开发系统有一种视图的一个人、角色或机构。（这种概念被 Darke 和 Shanks[1997] 用作用户视点分析的基础）。当用于系统需求时，视点还可以表示待开发系统、系统的子系统、以及可能影响系统必须做什么的系统环境中的系统。视点被组织为层次结构，以提供一种范围并引导分析过程。

如果要举一个例子，那么图 3.16 给出了飞机制动和控制系统（ABCS）通过集思广益得到的可能的初始视点清单。

引入了潜在视点清单之后，要通过对相关的候选视点分组构成层次结构并围绕相关集合画出边界。这个过程重复进行，直到所有候选视点都被圈起并产生层次结构为止。

图 3.17 给出了飞机制动控制系统的部分层次结构。

在 CORE 中，要确定每个视点必须执行的动作。每个动作可以使用或生成信息，或其它与待开发系统有关的内容（例如物品）。通过分析产生的信息通过表格收集表（TCF）记录下来，如表 3.2 所示。

在相邻列之间画出连线，表示流的发生。

通过这种方法分析了每个视点之后，要作为一个组来检查视点层次结构中各个层次的扁平收集表，保证每个视点所期望的输入会由源视点产生，并且每个动作所生成的输出是表示为目的地的视点所期望的。

表3.2 表格收集表

源	输入	动作	输出	目的地
输入来源的视点	输入项的名称	对一个或多个输入执行的动作，以生成所需的输出	由动作产生的所有输出的名称	输出所发送的视点

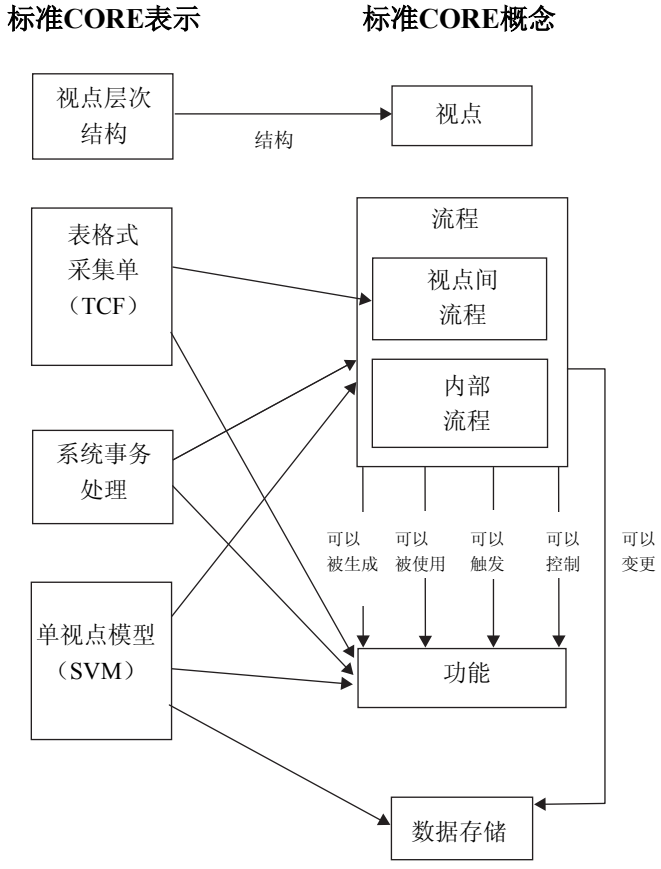


图 3.15 CORE 中的表示与概念

图 3.16 ABCS 初始视点

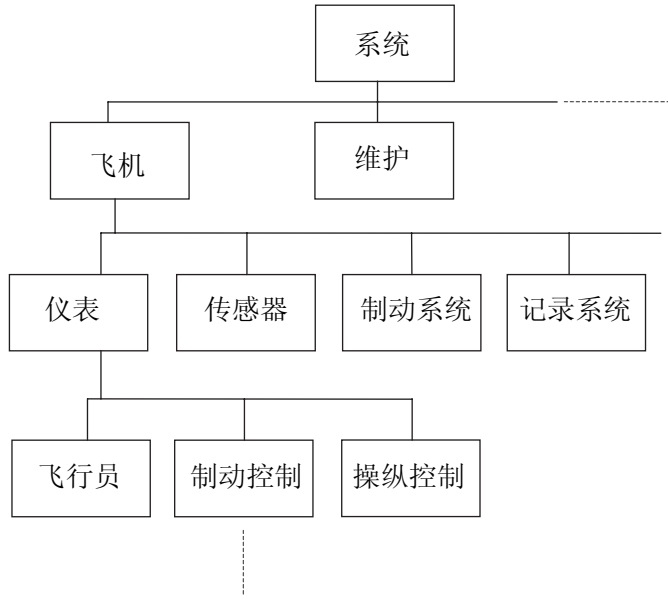
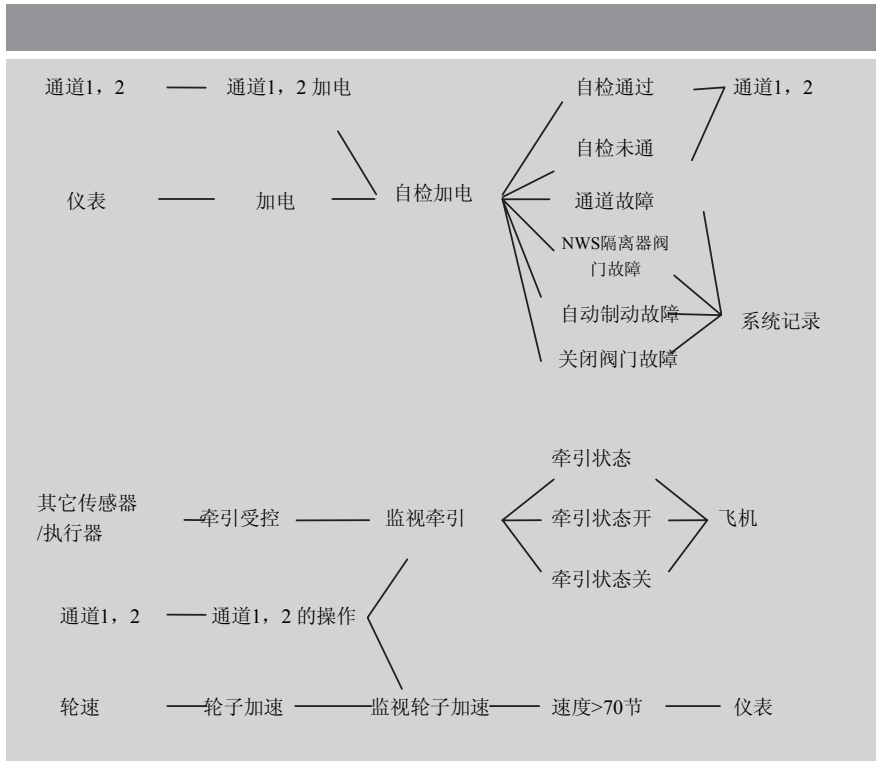


图3.17 层次结构举例

回到飞机制动控制系统例子上来，这个系统的部分 TCF 如表 3.3 所示。

表3.3 TCF举例



进一步的分析包括依次为每个视点开发更详细的数据流模型。这些单一视点模型（SVM）的切入点是在 TCF 中记录的信息，SVM 增加完全位于视点和数据存储内部的流程。SVM 还定义如何通过来自其它动作的流程控制和触发动作。

因此，通过分析视点层次结构中每个层次来自顶向下地推动分析。采用自顶向下的分析，可能很难知道什么时候停止，很难预测分析的最终结果。首先标识视点，然后使用视点控制后续分析的步骤，提供了一种自顶向下进行分析的受控方式。这样可以克服与基于数据流的分析有关的一个主要问题。这种控制元素在 CORE 的全称——“受控的需求描述”中体现出来。

CORE 的另一个主要概念是系统事务。从一个或多个输入、数据流或事件，到一个或多个具体输出流或事件，存在一条贯穿系统的路径。系统事务说明系统预期将如何操作。（这一点与 Jacobson 的用例很相似。）系统事务提供与自顶向下分析正交的一种视图，为讨论非功能需求提供了坚实的基础。

结构化分析与设计技术（SADT）

SADT 是一种以 Ross 在 70 年代承担的结构化分析（SA）工作（Ross 1977）为基础的结构化分析方法，面向图形表示并采纳了一种纯层次结构方法来使用一系列模块化的蓝图，逐步求精直到找出解决方案为止。SADT 的基本元素是方框，表示一种活动（在活动图中）或数据（在数据图中）。方框通过表示所需数据，或由方框所表示的活动（在活动图中）提供的数据的箭头，或提供或使用数据（在数据图中）的过程连接起来。

一共有四种基本与方框关联的箭头，如图 3.18 所示。箭头类型由到方框的连接位置隐含说明：

- 输入箭头从左侧进入方框，表示数据可供由该方框表示的活动使用。
- 输出箭头从右侧离开方框，表示由该方框表示的活动所产生的数据，即输入数据已经被由该方框表示的活动转换，生成这个输出。
- 控制箭头从顶部进入方框，并控制转换发生的方式。
- 机制箭头从底部进入方框并控制活动能够使用外部机制的方式，例如具体算法或资源。

SADT 图由一些带有一组箭头的方框组成。通过分解每个方框并生成层次结构框图可细化问题，如图 3.19 所示。

图 3.20 给出了 ABCS 的一个示例活动图。这种分解持续进行，直到详细程度足以开始进行设计为止。

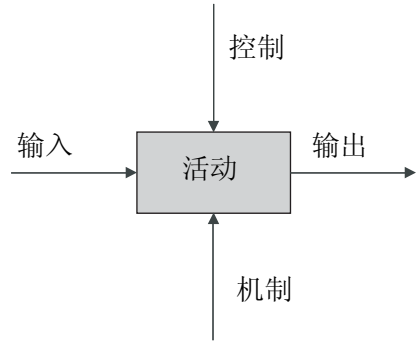


图3.18 SADT方框和箭头

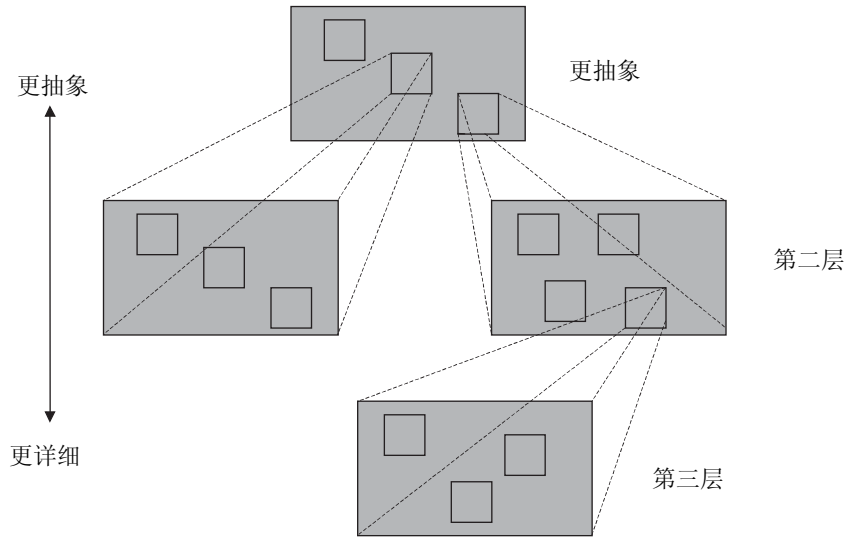


图3.19 使用SADT框图进行分解

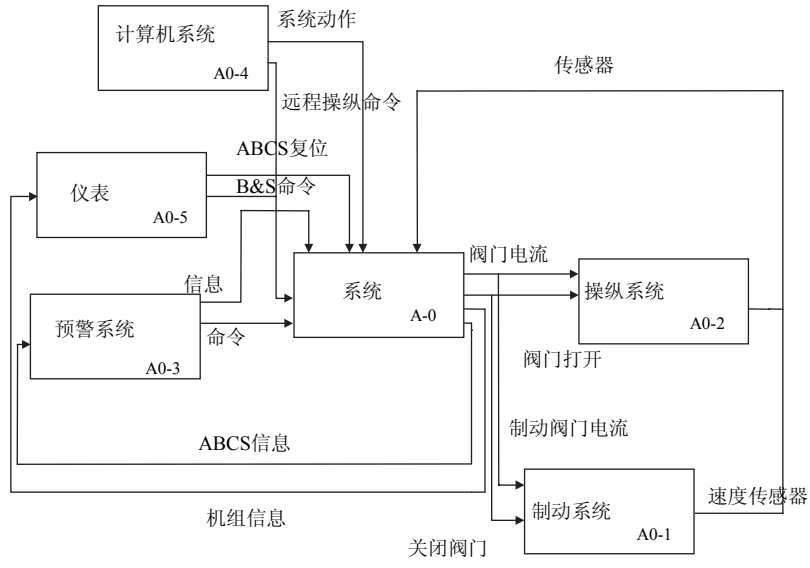


图3.20 SADT举例

3.4.3 面向对象的方法

80年代末90年代初，出现了大量面向对象（O-O）方法，提出了面向对象分析和设计的不同方式。最早使用面向对象方法的是那些对产品上市时间和抗变化能力要求很高的公司，包括电信、财务机构和以后的航空航天、卫生、银行保险、运输等行业。

主要的面向对象方法是面向对象分析（OOA）、Booch、OMT（对象建模技术）和Objectory。Shlaer和Mellor（1998）也提出了面向对象方法，但是他们的方法没有被看作是真正的面向对象方法。不过，Shlaer和Mellor提出的方法在辅助标识对象方面确实起了很大作用。

面向对象分析

面向对象分析（OOA）是由Coad和Yourdon（1991a）开发的。OOA分布在他们所谓的三个层次上。第一层是主题层，关注的是对象标识。在这一层，用户能够通过标识相关的问题领域对象，直接表达对问题领域的理解。第二层叫做属性层，关注的是标识与问题领域对象关联的属性（数据元素）。第三也是最后一层是服务层，用来描述由每个对象执行的服务（或操作）。

在实践中，OOA有助于开发人员标识系统需求，而不是如何确定软件的结构和实现。因此，OOA描述的是现有系统、系统的操作以及软件系统应该如何与其交互。

OMT

OMT方法是由Rumbaugh开发的，其目标是构建一系列对象模型来细化系统设计，直到最终模型适合实现为止，因此被认为是一种完整方法。这种方法通过三个阶段实现。分析阶段产生问题领域的模型。要产生三类的模型，即对象模型、动态模型和功能模型。对象模型是要构建的第一种模型。它使用与OOA类似的标记，以描述对象、对象的类以及对象之间关系的ER（实体关系）建模概念为基础。动态模型表示系统的行为，并使用Harel状态图的一种扩展形式。功能模型描述系统功能如何通过数据流图执行。

这些模型通过迭代方法得到。设计阶段确定模型的结构，实现阶段考虑适当的目标语言构造。这样，OMT不仅覆盖需求捕获阶段，而且还有助于沟通体系结构设计过程。

Booch

Booch方法是最早提出的面向对象方法之一。尽管这种方法确实考虑了分析，但是它的优势还是面向对象系统设计。Booch方法既是渐进的又是迭代的，并且鼓励设计人员通过逻辑和物理

两种角度观察系统来开发系统。

这种方法包括分析问题领域以标识系统中的类与对象集合及其关系。这些内容都是使用图形标记表示的。当考虑类、对象及其所提供的服务的实现时，要进行扩展。使用状态转移图和时序图也是这种方法的一个重要部分。

Objectory

Objectory 是 Jacobson 提出的方法。这种方法的很多思想都与其它面向对象方法类似，但是这种方法的基本特征是场景，也就是用例。用例定义在系统用户（参与者）和系统本身之间发生的交互。在数据流图类型的背景图中，用例被表示为过程泡泡。参与者不一定是人，尽管参与者采用枝状人型表示。因此，系统功能应该能够根据系统用例集合，即用例模型描述。图 3.21 给出的是银行系统的一个用例。

然后，这种模型被用来生成领域对象模型，通过将领域对象划分为三种类型构成分析模型。这三类对象是: 接口对象、实体对象和控制对象。这种分析模型再转换为设计模型，采用框图表示，再通过设计模型实现系统。

3.4.3.5 UML 标记

UML 是将三种接受程度最高的面向对象方法，即 Booch、OMT 和 Objectory 合到一起的一种尝试。在 90 年代中期，Booch、Rumbaugh 和 Jacobson 加入 Rational 公司，开发出一种单一、统

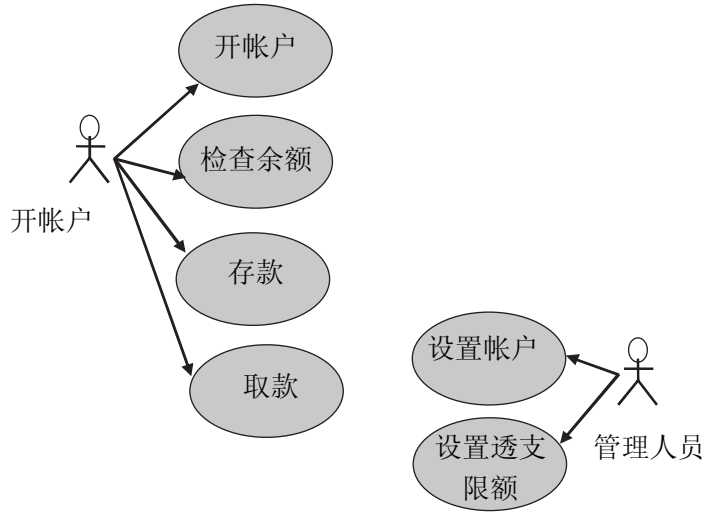


图3.21 银行系统的用例

一和可普遍使用的建模语言。他们主要强调的是产生标记，而不是方法或过程。

UML 由若干模型组成，这些模型合在一起描述待开发系统。每种模型都由以下一种或多种框图组成：

- 用例图（上一节已经介绍过）；
- 类图；
- 行为图；
- 实现图。

这些框图合在一起构成系统的完整描述。

图 3.22 使用了 UML 类图标记，扩展了本章前面介绍过的银行例子，并给出一组扩展类，即“帐户”、“拥有者”、“当前帐户”和“签发支票”，用来对系统建模。从图中可以看出，每个类都有一组关联的属性和操作。

面向对象方法提供了处理基于时间信息的很多方式。其中一种方式是使用前面已经介绍过的状态图，另一种方法是使用 UML 顺序图的行为/交互图，这是一种描述场景之间交互的非常有效的方式（如图 3.23 所示）。有些方法把这种方式叫做事件跟踪图。

这是一种相当简单的框图，但是却显示了这种技术的主要基本原理。在捕获与讨论需求时，与用户一起来画顺序图是非常有用的。提供了另外一种创建和观察场景的方式。如果用于这种目的，则不应该显示系统的内部细节，尽管这条规则在以后的开发过程中可以放宽。

可以看出，一般来说，使用 UML 或其它标记的面向对象建模过程如下：

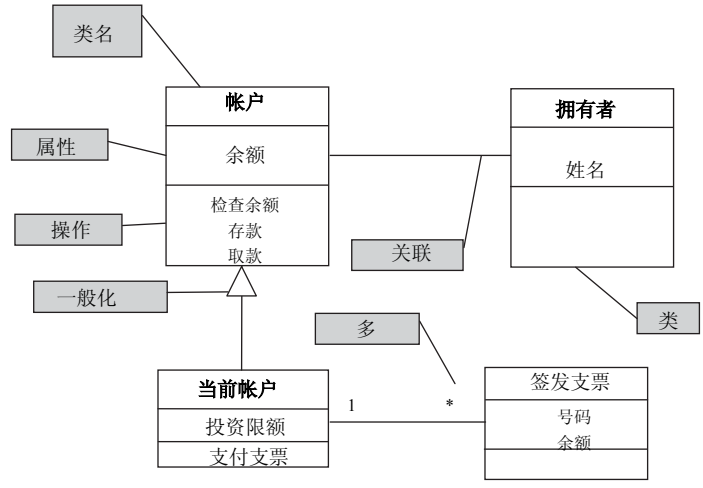


图3.22 经过扩展的UML示例

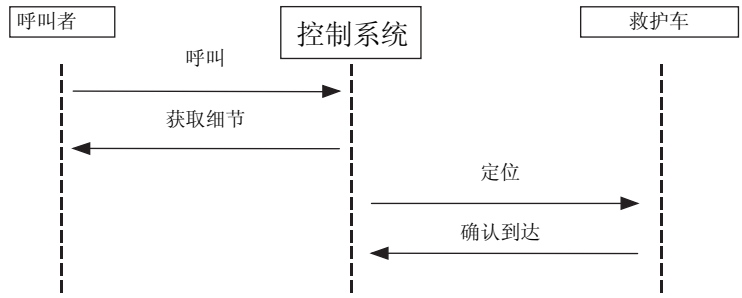


图 3.23 序列图

- 标识系统领域的核心对象；
- 标识类之间的关系；
- 定义每个类的属性和相关操作。

下一个步骤是定义在对象之间传递的消息，最终细化模型。

3.4.4 形式化方法

形式化方法提供基于数学的更严格的表示，可以用于推导规格说明一致性以及实现正确性的数学证明。可以进行严格检查，能够消除某些类型的错误。对于特定类型的系统，例如核电站、武器和飞机控制系统来说，这样做是很有必要的。

Z（Spivey,1989）、VDM(Jones,1986)、LOTOS(Bjorner,1987) 和 B(Abrial,1996) 方法是最常见的用于功能形式化定义的形式化方法。LOTOS（时序规格说明语言，Language of Temporal Ordering Specification）、VDM（维也纳定义语言，Vienna Definition Language）和 Z 都是由 ISO（国际标准化组织）标准化的形式化方法。B 和 LOTOS 模型是可执行的，而且 B 模型可以细化为代码。

形式化方法特别适合关键系统，即可能造成极大潜在物资或人员损失的系统，在这些系统中使用数学的严格方法是值得的。

形式化方法正在变得逐渐重要起来。如果形式化方法的范围再扩大一些以说明范围更广的系统问题，则形式化方法会更有用。

Z：一种基于模型的形式化方法

Z 是一种基于一阶谓词逻辑和集合论的形式化规格说明标记。这种标记把数据表示为集合、映射、元组、关系、序列和笛卡尔积，此外还提供一些操纵这些类型数据的功能和操作符号。

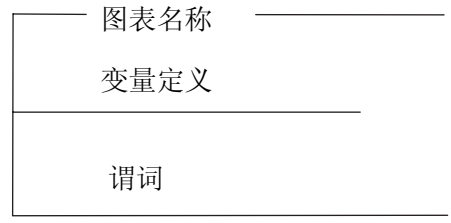


图3.24 Z图表

Z 规格说明采用一种叫做“图表”的小型易读的方框标记表示。图表采用标记图部分和谓词部分形式。标记图部分是一个变量声明表，谓词部分由单个谓词组成。对图表命名可产生名称和图表之间的一种语义等价关系（如图 3.24 所示）

采用 Z 编写的规格说明被表示为一组图表，每个图表都引入一些规格说明实体，并声明实体之间的关系。这些图表提供一种框架，在这种框架中可以渐进地开发和表示规格说明。

图 3.25 给出了图书馆“发放”操作的一个 Z 规格说明，整个图书馆系统的总行为在叫做“图书馆”的图表中描述。标记“Δ图书馆”叫做 delta 图表，表示“发放”操作会造成图书馆中的状态改变。

图 3.25 中的图表将输入与输出区分开，将状态前和状态后区分开。这些操作是：

“？”表示到操作的输入变量；

“！”表示操作的输出变量。

操作后的状态用单引号“’”表示，例如库存’，将其与操作前的状态区分开来。

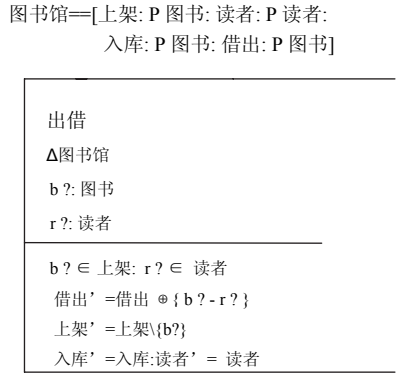


图 3.25 图表示例

3.5 小结

本章讨论了系统建模问题，重点讨论了解决方案领域的系统建模问题。本章给出了各种技术和方法，既有已经经过时间考验的，也有最近才开发出来的。这些技术和方法都在业界得到了广泛应用。本章为将为后续各章讨论的对 stakeholder 和系统需求建模问题提供基础。

第4章 编写与评审需求

4

写得简单易懂与写得好是同样困难的。
（William Somerset Maugham，作家，1874－1965）

4.1 引言

需求工程是一种技术过程。因此，编写需求与其它类型的写作不同。编写需求当然与写小说或写类似本书的著作不同，甚至与诸如编写操作手册和用户指南这样的“技术类”写作也不相同。

本章的目的，是讨论各个开发层次通用的需求编写问题。一旦通用过程被实例化，就要采用特定的原则和技术来确定需求的结构并描述需求。

在编写需求文档时，需要仔细综合考虑以下两个方面：

- 提高需求文档可读性的需要。
- 提高需求集合可处理性的需要。

第一个问题关注文档的结构，关注文档的组织方式和流程怎样帮助评审人员把单个需求描述放在整体背景下考虑。第二个问题关注单个需求描述的质量，使用清晰性和准确性的语言，以及如何将需求分割为单一的可跟踪项。

有需求经验的工程师逐渐认识到，仅靠字处理软件还不足以管理一组需求，因为单个描述需要标识、分类和跟踪。例如，一个传统问题是使用段号标识需求：在中间插入一个新需求，突然后续需求标识符都需要改变。

同样，试图直接用数据库管理需求的工程师很快会发现，有很多单个需求语句的表难以管理。尽管能够标识、分类需求并对需求排序，但是丢失了由文档提供的上下文信息；当单一语句与其所处的总体背景分开后，会失去意义。

因此，需求在文档和个性方面都需要维护。

编写和评审需求（或编写和评审所有其它种类的文档）应该是密不可分的，因为编写好的需求的准则，恰恰就是需求评审应该采用的准则，因此本章将编写和评审需求问题放在一起讨论。

表4.1 需求的 stakeholders

stakeholders	角色
作者	编写需求并进行修改
出版者	发放需求文档，并归档
评审者	评审需求，并提出修改建议
实现者	分析需求，并讨论修改

4.2 对需求的要求

在讨论如何编写需求文档和说明之前，最好还是先归纳一下编写需求的目标和目的，这有助于理解本章提出的一些原则的原因。

首先要标识需求的 stakeholders，如表 4.1 所示。

表 4.2 列出对各类 stakeholders 与编写需求文档和说明有关的能力要求。这些都是需要在编写需求方面做到的最基本要求，包括标识、分类、详细描述、确定状态、跟踪、背景关联以及检索。需求的表达和组织方式，对需求的“可使用”方式有很大影响。

表4.2 编写需求所需的能力

能力
● 唯一地标识需求各个语句的能力。 ● 以多种方式对需求各个语句的分类能力，例如： 按重要性分类； 按类型分类（例如功能、性能、约束、安全性）； 按紧急程度分类（如果必须提供）。 ● 跟踪每个需求语句状态的能力，以支持多个过程，例如： 评审状态； 满足状态； 鉴定状态。 ● 以多种方式详细描述需求的能力，例如通过提供： 性能信息； 鉴定； 测试准则； 根本原因； 注释。 ● 在文档总体背景下研究需求描述的能力，即考虑与其相关的描述。 ● 根据具体分类或背景，浏览整个需求文档并找出需求的能力。 ● 跟踪任何单个需求语句的能力。

4.3 确定需求文档的结构

需求文档可能很长。例如，航空母舰完整子系统需求的书面需求能够装满很多文件柜。不知道供应商对要成火车车皮地交付的大型系统会有什么反应。在这种情况下，一种能够被很好理解、清晰的整个需求文档的编写结构，对于复杂文档的有效管理是至关重要的。

为需求确定恰当的结构可有助于：

- 最小化需求的总量；
- 理解大量信息；
- 找出与具体问题有关的需求集合；
- 发现遗漏和重复；
- 消除需求之间的矛盾；
- 管理迭代（例如延迟提出的需求）；
- 拒绝差的需求；
- 评估需求；
- 在多个项目中重用需求。

文档一般是分层的，对于多个层次采用节和小节来组织。文档层次是分类的有用结构，确定需求文档结构的一种方式，是使用通过标题结构能够对需求语句编目的节。采用这种方式，需求语句在文档中的位置代表其一级分类。（二级分类可以通过指向其它节的链或通过属性给出。）

第 3 章描述了系统模型如何在系统分析中经常使用层次结构，例如；

- stakeholder 场景中的目标或能力分解；
- 数据流图中的功能分解；
- 状态图中的状态分解。

当需求从这类模型中导出时，所产生的一种层次结构可以用作需求文档标题结构的一部分。

除了需求说明本身，需求文档可以包含各种技术和非技术文本，以支持对文档的理解。这些文本可能包括：

- 提供需求背景的背景信息；
- 描述外围系统的外部背景，常常叫做“领域知识”；
- 需求的范围定义（包含什么，不包含什么）；

- 在需求描述中所使用的术语定义；
- 连接文档各个节的描述性文本；
- stakeholder 描述；
- 在导出需求中使用的模型归纳；
- 其它引用文档。

4.4 关键需求

很多机构都使用“关键需求”的概念，特别是在 stakeholder 层次上。关键需求常常叫做 KUR（关键用户需求）或 KPI（关键性能指标），是从系统总体中抽取的反映系统本质的一个很小的子集。

选择关键需求时的指导原则类似于 Jerome K. Jerome 提出的“一船三人”原则，即在规划行程时，发现：

泰晤士河的上游不能通行大到能够承载他们所携带的必需品的船只……

乔治说，“我们不应该考虑我们会用到的东西，而只能考虑缺了它们，我们就不能做事的东西。”

每个关键需求都会产生这样的负面问题：

“如果解决方案不能提供这种能力，为什么还要买呢？”

也就是说，如果在系统层：

“如果系统不能做这个，为什么还要这个系统呢？”

从这个角度说，关键需求就是绝对必需的需求。（当然什么事情都是可以协商的，但是涉及关键需求的协商，必须非常小心地进行。）

如果可能，所有关键需求都要通过性能属性量化。这样做可以使关键需求成为 KPI，用于评估针对这个需求的其它建议，或用于归纳项目进展的关键指标。

4.5 使用属性

在前面几章讨论的过程，以及表 4.2 给出的能力已经说明，简单文本说明不足以充分定义需求，每个需求还要携带其它分类和状态信息。

[SH234] 救护车控制系统应能够处理最多100个并发急救呼叫。
来源 : 托马斯
优先级 : 强制
发布 : 1
评审状态 : 已接受
可检验 : 是
检验 : 先通过仿真,然后通过系统测试。

图4.1 需求的属性

补充信息不能影响需求的文本说明，而是应该放在需求附件的“属性”中。属性使与单一需求关联的信息能够被结构化，以便于处理、过滤、排序等。属性可以用来支持表 4.2 中的很多能力，使需求能够为进一步行动排序或检索，并使需求开发过程本身是可控的。图 4.1 给出了一个带有一些属性的需求例子。

所使用的具体属性取决于需要支持的确切过程。有些属性完全是自动确定的，例如日期、数字; 有些是用户提供的，例如优先级; 有些属性是标志，在分析工作完成后设置，例如可检查性。表 4.3 给出的属性分类建议，部分来源于英国 INCOSE 协会 (国际系统工程协会）需求工作组所完成的研究工作。

4.6 保证需求之间的一致性

管理大量需求经常要考虑的一个问题是能够标识矛盾的需求。要找出相隔很多页的两个矛盾描述很困难。可以采用什么手段辅助找出这些潜在不一致性呢？

一种方法是以多种方式对需求分类，并通过过滤和排序手段将说明相同问题的少量描述归在一起。很多需求都涉及系统的很多方面。例如，一条主要涉及发动机性能的需求，可能还涉及特定的安全性要素。因此，这种描述既可以在发动机性能背景中看到，也可以在安全性背景中看到。为了做到这一点，可以为需求指定第一级和第二级分类，就像第 1 章 1.3 节讨论过的那样。在典型情况下，每个需求都有一个第一级分类（可能利用需求在文档中的位置）和多个第二级分类，可能使用链或属性。

现在彻底的评审过程可以包括采用在第一级和第二级分类使用的关键词系统地过滤描述。例如，过滤与安全性有关的所有需求会集中找出与第一级分类相当不同的描述。然后对这些相近的描述进行评审，以发现潜在的矛盾。

表4.3 属性分类

分类	取值举例
标识	
标识符	唯一参照
名称	归纳需求主题的唯一名称
本质特征	
基本类型	功能、性能、质量因素、环境、接口、约束、非需求
质量因素子类型	可用性、灵活性、完整性、可维护性、可移植性、可靠性、安全性、可支持性、可负担性、可使用性、技巧
产品/过程类型	产品、过程、数据、服务
定量/定性类型	量值、性质
生命周期阶段	概念前、概念、开发、制造、集成/测试、部署/交付/安装、运行、支持、拆除
优先级与重要性	
优先级（一致级别）	键值、必备、可选、要求，或 必须、应该、可以、希望（MoSCow）
重要性	1到10
来源与拥有关系	
导出类型	分配、分解
来源（出处）	文档或stakeholder名称
拥有者	stakeholder名称
批准负责人	名称或人员
背景	
需求集合/文档	（最好根据需求在结构化文档中的位置来处理）
主题	
范围	
检验与确认	
检验与确认方法	分析、检测、系统测试、组件测试
检验与确认阶段	（参见生命周期阶段）
检验与确认状态	质疑、通过、失败、未决定
满足论据	选择分解的根本原因
确认论据	选择检验与确认方法的根本原因
过程支持	
协议状态	提出、正在评估、达成共识
鉴定状态	未鉴定、已鉴定、质疑
满足状态	未满足、已满足、质疑
评审状态	将要评审、被接受、被拒绝
详细描述	
根本原因	有关提出需求原因的文本说明
注释	用于澄清的文本注释
问题	提出需要澄清的问题
响应	对澄清作出的响应
其他	
成熟度（稳定性）	变更数量/次数
风险等级	高、中、低
估计成本	
实际成本	
产品版本	满足需求的产品版本

4.7 需求的值

有些需求是不可协商的。如果不能满足这类需求，产品就没有用处。

其它需求是可协商的。例如，如果要求系统至少支持 100 个并发用户，但是所交付的解决方案只支持 99 个并发用户，则这个系统很有可能对于客户仍然具有价值。

获取需求的值问题会是一种挑战。需要找出一种表达思想的方式，虽然目标可能是 100 个并发用户，75 个是可接受的，但是低于 50 个是不能接受的；而且 200 个可能更好。

获取需求值的一种方法是提供多个性能取值。以下是一种三价值方法示例：

- M：强制低限（或高限）；
- D：所要求的值；
- B：最佳值。

这三个值可以放在三个单独的属性中，或在带标签的文本中表示，例如“系统应该支持 [M：50，D：100，B：200] 个并发用户。”

另一种方法是通过提供将性能映射到某种值表示的函数，通常是 1 到 100 的数字，表示需求的取值。图 4.2 给出了不同形状值函数的四个例子。函数（a）显示出上面给出的例子，并发用户的数量应该尽可能地多，但是多于最低数是强制的需求值。函数（b）是二选例子：是否超过 100 个并发用户。200 个并发用户并不增加额外价值。函数（c）给出要最小化取值（例如重量），而函数（d）显示的是要优化的值（例如发动机的每分钟转数）。

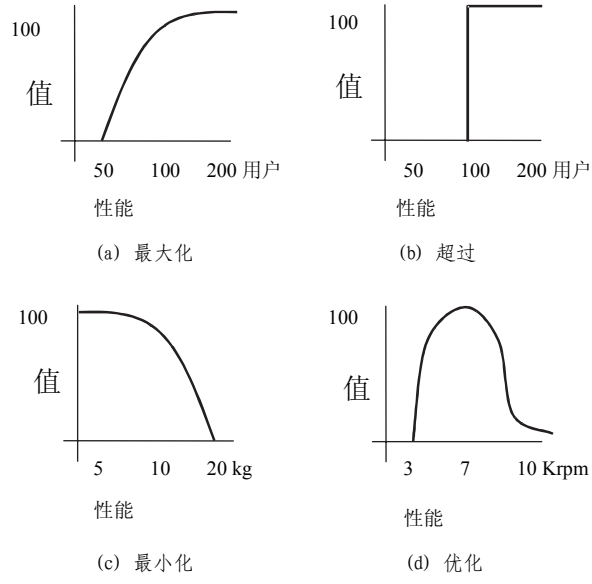


图4.2 典型的价值函数

这是一种表示值的很直观的方法，看一眼取值曲线的形状就知道需求的性质：最小化、最大化、优化等。这还使工程师能够理解自己所拥有的自由度，以在设计解决方案时，通过全面折衷考虑不同需求的要求，交付总体价值最佳的产品。这就是为什么这种方法常常用于投标评估过程的一部分，以判断不同方案相对价值的原因。

可以使用一个属性将值函数表示为一组性能/值的匹配。

4.8 需求的语言

使用一致语言有助于标识不同类型的需求。一个简单的例子是使用“将”作为指示文本中存在需求的关键词。有些方法走得更远，使用“将”、“应该”和“可能”指示需求的不同优先级。

所使用的语言随要表达需求层次不同而不同。主要区别是问题领域中的 stakeholder 需求和解决方案领域中的系统需求之间的区别（请参阅第 1 章 1.7 节）。

正如将在第 5 章所强调的那样，stakeholder 需求主要考虑的是能力和关于能力的约束。能力描述应该表达由一个或多个已标识 stakeholder 类型（或用户组）（单一）的能力。stakeholder 类型应该在需求文本中描述。

典型的能力需求具有以下形式：

The <stakeholder type> shall be able to <capability>.

<stakeholder 类型>应该能够<能力>。

如果有一些只与需求关联的性能或约束问题，可以也在文本中描述，例如给出以下形式：

The <stakeholder type> shall be able to <capability>
within <performance> of <event>
while <operational condition>.

<stakeholder 类型>应该能够在<运行条件>下，在<事件>的<性能>中具有<能力>。

例如，以下能力需求具有附加的性能和约束要求：

武器操作人员应该能够在严峻的海面条件下，在雷达捕获目标 3 秒内发射导弹。

另一种不太常见的情况是单一性能属性与多个能力关联。例如，可能需要在指定时刻提供多种能力。在实践中，这些能力通常是高层能力的细分，性能属性应该附加到这些细分能力上。

但是，约束常常独立于能力被单独描述，要么因为约束适用于整个系统，要么因为约束适用于不同的能力。一般来说，stakeholder 需求中约束的基础是最低可接受性能，或通过与外部系统（包括法律和社会系统）交互的要求导出。

典型约束需求具有以下形式：

The <stakeholder> shall not placed

in breach of <applicable law>.

<stakeholder>不应该破坏<适用规则>。

例如： **救护车驾驶员不应该违反国家道路交通规定。**

由于约束需求属于解决方案领域，因此系统需求语言有一点不同。系统需求关注的是功能，约束关注的是系统。语言依赖于与需求有关的约束和性能种类。以下是一个具有能力性能的功能例子：

The <system> shall <function>

not less than <quantity> <object>

while <operational condition>.

<系统>将<功能>在<运行条件>下，不低于<量><对象>。

例如：**某通信系统在没有外部电源的情况下，支持不低于 10 个用户的电话连接。**

以下是另一种形式，表示的是周期性约束：

The <system> shall <function> <object>

every <performance> <units>.

<系统>将在每个<性能><单位>实现<功能><对象>。

例如：**某咖啡机应该每 10 秒钟生产一杯热饮。**

下面一节进一步讨论这个问题。

4.9 需求样板文件

第 4.8 节介绍的需求语言是采用样本文件表示的。本节扩展这种概念，将其用于约束需求的采集和表示。

像第 4.8 节中的例子那样地使用样本文件，是标准化需求语言的一种好方法。可以按照表示特定需求的不同方式进行收集与分类样本文件来构成样本选择集。随着机构经验的不断积累，可以扩展样本选择集，并在不同项目中重用。

现在，通过样本文件表示需求已经成为以下活动过程：

- 从样本选择集中选择最适合的样本文件；
- 提供数据，以补充尚未确定的内容。

需求可以引用样本文件单个文档范围内的实例，尚未确定的内容实际上可以作为需求的属

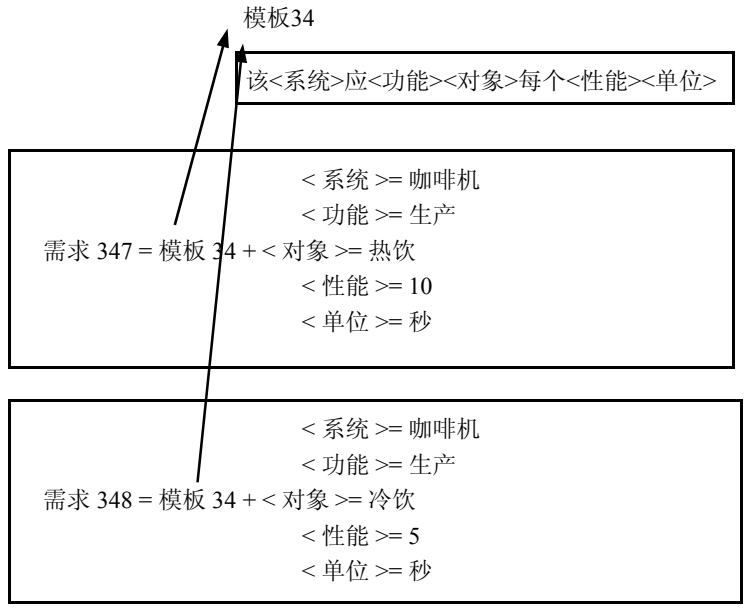


图4.3 全局模板

性单独收集。图 4.3 说明了这一点。

根据这种信息，必要时可以生成需求的文本形式。将模板分开具有以下优点：

- 可以通过全局变更来调整风格：为了修改特定需求的表达方式，需要编辑的只有特定的相关样本文件。
- 可以更容易地处理系统信息：例如，将所有“<运行条件>”未确定内容收集到单独的属性中，便于根据运行条件排序和过滤。
- 可以保护保密信息：如果需求包含机密或保密信息，则可以使用样本文件，仅将这部分需要保护的所有语句分离开。

这最后一点值得更详细地详细描述。在军事或商业敏感项目中，需要限制某些信息的可用性，但并不是所有信息。相当常见的情况是，单个需求语句包含不同层次的机密信息。例如，显然军舰要发射导弹，需要保密的是与这种能力有关的参数：就绪状态、发射频率和射程等。采用样本文件可以不因为语句中有些要素是保密的就隐藏整个语句，而是显示出该语句，只是隐藏一些敏感属性。当然，不同读者能够看到不同的属性集合。

由于约束的差别这样大，因此约束一般是最难表示的，正是在这一点上样本文件可以发挥其主要作用。以下是捕获约束需求的步骤：

1 首先收集所有能力需求。

2 列出一个清单，包含可能需要表示的所有不同类型的约束。如果这种清单依据的是以前

同类系统的开发经验，则应该有各种类型的样本文件，否则必须定义合适的样本文件。

3 对于每种能力，要考虑各种约束，并确定是否需要捕获约束。为此可以使用一张大表，在每个字段中，通过输入该需求的适当下属从句，指示约束所处位置; 如果不需要约束，则在适当的字段中输入“无”。

4 选择能够最好地匹配要描述约束的样本文件，并实例化。

5 当所有“字段”都被考虑之后，整个过程结束。

这个过程可以回答两个常见问题：

- 如何表示约束需求？（使用样本文件。）
- 如何知道什么时候已经收集了所有约束？（使用这种系统化的覆盖手段。）

表 4.4 给出了一些按约束类型分类的示例样本文件。请注意，类似的分类约束可能有多种方式可以表示，约束也可能有复合分类。只有用粗体给出的样本文件部分实际与约束有关。

表4.4 约束需求的示例样本文件

约束类型	样本文件
性能/能力	The <system> shall be able to <function> <object> **not less than <performance> times per <units>.** <系统>**每<单元>**将能够<功能><对象>**不低于<性能>次**。
性能/能力	The <system> shall be able to <function> <object> **of type <qualification> within <performance> <units>.** <系统>**在<性能><单元>内**将能够<功能>**类型<限定>**<对象>。
性能/能力	The <system> shall be able to <function> <object> **not less than <quantity>** <object>. <系统>将能够<功能>不低于<数量><对象>。
性能/时间线	The <system> shall be able to <function> <object> **within <performance> <units> from <event>.** <系统>**在<性能><单元>内**将能够**根据<事件>**<功能><对象>。
性能/周期性	The <system> shall be able to <function> **not less than <quantity>** <object> **within <performance> <units>.** <系统>在<性能><单元>内，将能够<功能>**不低于<数量>**<对象>。
交互操作性/能力	The <system> shall be able to <function> <object> **composed of not less than <performance> <units> with <external entity>.** <系统>将能够<功能>**由不低于<性能><单元>组成的<外部实体>**<对象>。
稳定性/周期性	The <system> shall be able to <function> <object> **for <performance> <units> every <performance> <units>.** <系统>将能够**每<性能><单元>对<性能><单元>**<功能><对象>。
环境/可操作性	The <system> shall be able to <function> <object> **while <operational condition>.** <系统>在<运行条件>下，将能够<功能><对象>。

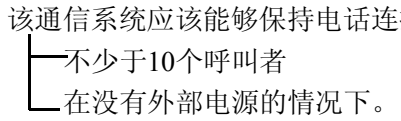


图4.4 作为从句的性能和约束

4.10 需求的粒度

使用需求样本文件，有利于将一些约束和性能描述作为能力或功能需求的子句。在有些情况下，可能需要建立到这些子句的双向跟踪。

这样就产生信息粒度问题。在需求管理中，要把“原子”分得多细呢？

需求语句可以分解为从句，只要工具支持能够保证从句在上下文中永远是可视的。一种图表是分层扩展需求以使从句成为主要需求的下层需求，如图 4.4 所示。而主要的需求自身是可读的（和可跟踪的），虽然从句为了跟踪是可以单独引用的，但是只有在其“父”语句背景下才有意义。

现在的可跟踪性可以引用具体从句，但是从句只能在上级语句的背景下引用。例如，图 4.4 中在用斜体表示的背景下可以引用的可跟踪语句是：

- 通信系统要能够保持电话连接。
- *通信系统要保持*不少于 10 个用户的*电话连接*。
- *通信系统要*在没有外部电源的情况下*保持电话连接*。

可能存在多种方式组织从句层次结构。例如，假设“在没有外部电源条件下”有多种能力要求。图 4.5 给出的是一种组织方式。

现在能够引用的可跟踪语句是：

- *在没有外部电源条件下，*通信系统要支持电话连接。
- *在没有外部电源条件下，通信系统要支持*不少于 10 个用户的*电话连接*。
- *在没有外部电源条件下，通信系统要支持*不少于 15 个救护车驾驶员的*无线连接*。

的确，作为一种一般原则，需求的组织要使父对象集合能够为每个语句提供完整的上下文，包括节标题和小节标题。

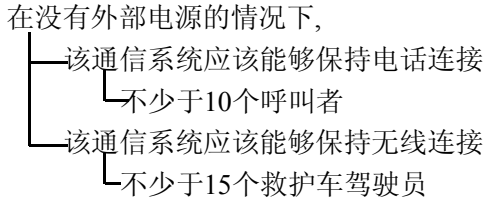


图4.5 从句的另一种组织方式

4.11 编写需求语句的准则

除了语言之外，还有一些需求语句应该满足的特定准则。这些准则归纳如下：

- 原子性：每个语句都只携带单个可跟踪元素；
- 唯一性：每个语句都可以被惟一地标识；
- 可行性：在成本和进度限度内，在技术上是可行的；
- 合法性：在法律上是可行的；
- 清晰性：每个语句都可以被清晰地理解；
- 准确性：每个语句都是准确、精确的；
- 可检验性：每个语句都是可检验的，并知道如何检验；
- 抽象性：不强迫针对下层的特定的设计解决方案。

此外，对于需求集合整体还有一些准则：

- 完备性：提供所有需求；
- 一致性：没有相互矛盾的需求；
- 不冗余性：每个需求只描述一次；
- 模块化：属于同一个问题的需求语句放在一起；
- 结构化：需求文档有清晰的结构；
- 被满足：已经达到一定程度的可跟踪性覆盖率；
- 被鉴定：已经达到一定程度的可跟踪性覆盖率。

以下给出两个“恶梦”般的实际需求的例子。

1 系统在任何时候都要达到最大额定值，能够提供 125%最大额定值的紧急情况除外，除非紧急情况持续时间超过 15 分钟，在这种情况下，额定值将降到 105%，但是如果只能达到 95%，则系统要采用降低额定值例外处理，并至少在 30 分钟内维持所述额定值的 10%。

2 系统要提供通用字处理功能，这种功能要便于未经过培训的员工使用，要在细缆以太局域网上运行，以太网通过每个系统中的集成接口卡，将各个系统接在一起，如果必要可以使用额外内存。

这些例子包含了一些常见问题。读者应该避免出现以下问题：

- 避免凌乱：能做到简捷是最好的，需求不能像小说那样读；
- 避免有漏洞的从句：例如“如果必要”，这样的句子使需求失去意义；
- 避免在一段中包含多个需求：常常以出现词汇“并且”为标志；
- 避免猜测；
- 避免模糊性词汇：通常、一般、常常、正常情况下、典型情况下；
- 避免模糊性术语：用户友好、万能、灵活；
- 避免理想化的思考：100%地可靠、使所有用户满意、安全、在所有平台上运行、永远不失效、处理所有意外失效、可针对未来所有情况升级。

分析第一个例子会发现，其中给出了 12 个需求。更好的办法是清晰地标识飞机的四种不同运行模式：正常、紧急、紧急情况持续 15 分钟以上和降低额定值预期，并单独表示每种模式的需求。

请注意第二个例子中的漏洞从句。从句的作用范围很不清晰。一种解释是“如果必要……系统要提供通用字处理功能。”那么这是需要的，还是不需要的呢？

4.12 小结

需求获取中最困难的事就是开头。拥有一种方法非常重要，但是首要任务还是从第一天开始将需求写下来，并提供给其他人征求意见。以下给出的是一种可以安全展开工作的步骤：

- 一开始就定义提纲结构，最好是层次式的，并随着工作的深入不断改进。
- 尽可能快地写下需求，即使这些需求并不完备。
- 提前确定用于为文本描述分类与详细描述的属性是什么。
- 快速产生一个初始版本，以便立即得到反馈。
- 随着工作的输入不断完善需求，去掉重复部分、难以保证的设计和不一致性。
- 不断集思广益并进行非正式评审，快速改进版本。
- 向用户请教要比由“专家”分析好得多。

编写需求要遵循的规则包括：

- 使用简单、直接的语言；
- 编写可测试的需求；
- 使用经过定义和达成共识的术语；
- 一次编写一个需求。

第5章 问题领域中的需求工程 5

并不是他们看不到解决方案，而是他们看不到问题。

（Gilbert Keith Chesterton，作家，1874－1936）

5.1 什么是问题领域？

问题领域是系统要在其中使用的领域。因此，从操作的观点来看需求是非常重要的。系统或任何其它产品要使一些人或设备能够做一些事情。系统的这种能力是问题领域中的需求工程的核心。因此，面对通过潜在用户引出需求的困难，工程师很想询问用户这个问题：

你们想要系统做什么？

有些用户对于系统要做什么几乎没有什么想法。已经有现有系统的用户通常对如何改进系统会有看法，但是如果没有现有系统，则不能用这种方法启发用户。对于了解系统能够提供什么的用户，得到问题的答案可能比较容易，但是他们很可能会提出一种解决方案，因为问题关注的是将由待开发系统提供的功能。

为了避免这种过早转入解决方案领域的情况发生，需要问的问题应该是：

你想要的系统的目的是什么？

当考虑系统的用途时，人们立即就会考虑自己要通过系统能够做什么，而不是如何做。陈述用户要达到的目的可以不涉及任何实现或解决方案，而把解决方案方面的问题留给系统工程师和结构设计师。

有人可能会提出不同意见：即使在问题中提到“系统”也可能产生误导，问题应该简化为：

你想要能够做什么？

对这个问题的回答应该具有以下形式：

我想要能够做……

这种回答叫做能力需求，是问题领域中需求的一种关键形式。

认识到问题领域需求工程的基本任务就是提炼能力需求之后，下一个问题就是：

应该向谁问问题？

这带来将一些人标识为“stakeholders”的问题。这些人或机构与待开发系统有直接或间接的关系。

最后，必须检查什么类型模型与问题领域相关。显然，所使用的任何模型都必须被stakeholders理解，因为他们将有责任验证这些模型。由于stakeholders是因为有问题领域中的专门知识才被选出的，因此他们一般不想或不能理解即使技术含量很小的任何模型。例如，如果你要进入汽车展示大厅，查看展示的汽车，那么你很可能不会对发动机管理系统的状态转移图感兴趣，更可能关注的是加速性能、燃油效率、舒适程度以及车内娱乐设备等方面的汽车性能。换句话说，要考虑的是在长途旅行中汽车驾驶起来会怎么样。在你的头脑中，考虑的是想象中的驾车旅程，考虑的是汽车在旅行中用得着或能够提供方便的各个方面。这是一个使用场景的例子。

人们发现，使用场景是对人们想做什么或想要能够做什么进行建模的一种很好的方法。这种方法与他们对其工作或问题的思考直接关联起来。场景可以通过stakeholders构建，然后用作讨论所需能力的基础。

在问题领域中的需求工程的最后一个问题是可能会有一些压倒性的约束。在购买汽车的例子中，你的资金预算可能有限，也可能希望汽车在指定时间段内交付，可能希望驾驶成本在一定水平之下。

下面可以考虑如何实例化通用过程来创建stakeholder需求。

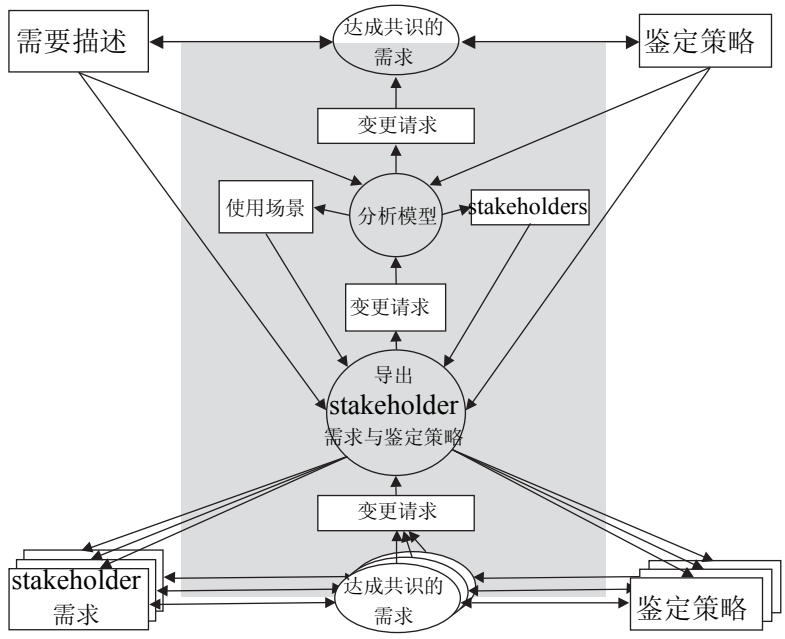


图5.1 责任人需求过程

5.2 实例化通用过程

图 5.1 给出的是提炼 stakeholder 需求的通用过程的实例化。切入点是需要的描述。需要的描述可能是一件小事，例如可能是首席执行官（CEO）给首席技术官（CTO）的一个电子邮件，告诉他需要领先一步推出一种新产品。需要的描述也可能是已经经过研究考虑过可能的选项，并且给出某些使用场景的运行文档概念已经产生。

图 5.1 说明，分析和建模过程要产生一组使用场景加上 stakeholders 清单。所导出的需求就是 stakeholder 需求。

后面几节将详细介绍分析和建模、导出 stakeholder 需求以及鉴定策略过程。

5.3 与客户就需求达成共识

stakeholders 需求过程开始阶段的达成共识过程通常很不正式。需要描述很可能就是简单的文件，这些文件从需求的观点看还没有达到工程水准。换句话说，这些文档可能包含混合一些描述性信息的需要的含糊描述。没有包含能够形成满足关系的原子需求。从这一点看，stakeholder 需求过程与其它需求过程不同，因为要从相当模糊的描述开始。获取 stakeholder 需求的一个关键要素是确定待开发系统的范围。这种工作通常在确定了使用场景之后进行。

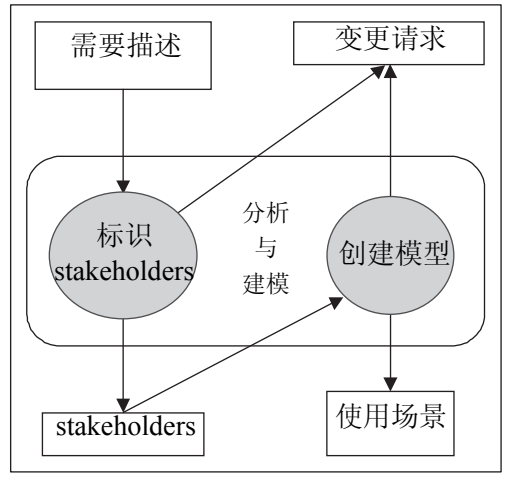


图5.2 stakeholder需求的分析与建模过程

5.4 分析与建模

问题领域的分析和建模过程的实例化如图 5.2 所示。第一个活动是标识 stakeholders，然后在 stakeholders 的帮助下确定使用场景。

5.4.1 标识 stakeholders

前面已经介绍过，stakeholders 可以是对待开发系统有看法、负责或受待开发系统影响的任何人或机构。stakeholders 的类型随系统性质的不同而不同。例如，系统可能是一种消费产品，也可以是空中交通管制或铁路运输等公共服务。

对待开发系统有看法的人包括将直接使用系统的人。请注意，这些人可以包括一般大众，例如飞机或火车乘客，或不一定是旅客但却受事故影响的人。对系统负责的人，可以是负责运营该系统的经理，或安全负责人。

以下列出可能的 stakeholders，可以用于判断是否已经标识了所有 stakeholders 的基础。这个列表并不是完备的，只是为列出 stakeholders 提供指南。

- 经理: 负责待开发系统开发经费或运行经费的人。吸收高级策略制订者是很好的计划，他们可以研究所提议的开发是否与公司或机构的目标和文化一致。
- 投资者：已经作出或正在被邀请对待开发系统投资的人，或负责开发或运营该系统的机构。
- 系统用户: 显然这是非常重要的一组 stakeholders。他们与由新系统或服务所提供的能力有直接的利益关系。请注意，有些用户可能并不直接与系统交互。例如，哈博望远镜的用户是天文学家。他们发出指令在特定方向上拍摄照片，并在到达时接收信息，但是并不直接控制望远镜本身。现有系统的用户也是该系统问题知识的一个有价值的来源。他们可以就希望看到系统被怎样改进发表很有价值的意见。
- 维护和服务员工: 虽然他们的主要责任是在系统交付之后保持其正常运行，但是他们在系统必须帮助他们完成维护工作上会提出重要的需求。
- 产品处置人员: 随着环境保护立法的发展，这是一种日益重要的角色。通过他们得到的需求会对设计产生很大影响，特别是要使用的材料。
- 培训人员: 与维护人员一样，这些人在使系统易于使用，并最终易于培训用户使用方面，有既得的利益关系。培训人员可能还会要求系统能够在同时存在实际数据和培训数据的模式下运行，而不会影响系统的安全运行。

- 系统买家: 对于公共服务和其它大型系统，购买系统的人可能不直接涉及开发或运行，但是在以成本效益比观点界定系统范围上具有重要作用。对于基于产品的开发来说，买家可能是实际用户，例如移动电话用户、汽车驾驶员等。
- 销售和市场开发人员: 这些人在确定新系统的能力方面具有至关重要的作用，特别是基于产品的开发，因为对于大量生产的消费产品，不可能接触所有的潜在用户。
- 可使用性和效率专家: 这些人知道如何优化系统，以提高使用效率。这些因素包括人机工程学、易于学习以及在面对压力可靠地运用系统的能力（例如空中交通管制）。
- 运行环境专家: 通常新系统不是为在“干净的”环境中运行所创建的，而是必须与现有系统交互。此外还有其它环境问题，例如在禁止系统污染环境的地方控制排放，以及另一个方面的问题，即系统必须能够容忍所在的工作环境（例如极端气象条件、浸入水中等）。
- 政府: 规则、规定和法律确定并影响系统可以做什么，不可以做什么。
- 标准: 现有和未来的标准会影响待开发系统的目标。可能是一些国际标准，例如 GSM 移动电话标准，也可能是国家标准或公司内部标准。
- 公众意愿和意愿领导者: 世界不同地区有不同的意愿。对于瞄准国外市场开发的产品，必须考虑这些因素。
- 监管机构: 这些组织可能要求收集一定的证据，作为确认或授权过程的一部分。这方面的例子包括英国的铁路监管局和美国的食品与药品管理局（FDA）。

得到了潜在的 stakeholder 类型之后，需要确定哪些类型的 stakeholder 是有关的，以及如何联系每种类型的 stakeholder。在有些情况下，例如系统用户，可以直接联系。在另一些情况下，例如一般公众，则不能直接联系。对于能够直接联系的 stakeholder 类型，需要确定提名谁为 stakeholder，而对于不能直接联系的 stakeholder 类型，需要确定谁来担当 stakeholder 的“角色”，并以他们的名义发表意见。通过这种过程最终确定 stakeholders（如图 5.2 所示）。

5.4.2 创建使用场景

大多数对话都是建立在一组发言人达成共识的假设基础上。这些假设可以解释为其相互理解的模型。试图在没有达成共识的基础规则上讨论需求是不会有结果的。

讨论能力需求的一种基本构造机制是运行或使用场景。使用场景产生一种按时间分层组织的结构。stakeholder 需求使用场景作为确定框架的一种手段，在这种框架中可以展开有意义的对话。

场景能够激励 stakeholders 考虑他们正在做的工作，以及他们希望怎样完成这些工作。事实上，他们是在预演他们所希望的工作方式。一旦就场景达成共识，就可以生成单个需求，精确地定义在场景的每个点上 stakeholders 希望能够做什么。

场景为与 stakeholders 一起探索需求提供了一种极好的方法，场景本质上就是 stakeholders 希望达到的目标。场景是 stakeholders 通过时间产生的结果序列（或达到的状态）。如图 5.3 所示，使用场景可以表示为一种目标层次结构，并表示由系统向 stakeholders 提供的能力，但不说明如何提供。换句话说，用户场景是一种能力层次结构。

面向时间的特性，使开发人员能够预演系统将要提供什么，stakeholders 可以一步步地深入发现遗漏或重叠的要素。因此这种结构可避免对于解决方案的过分承诺，同时又能很好地定义问题。

在创建使用场景时，可遵循一种经过清晰定义的方法。要询问 stakeholders 的基本问题是“你希望达到什么目标？”或“你希望处于什么状态？”然后这种方法从最后状态开始，再进行扩展，询问达到最终目标需要经过哪些状态或中间步骤。然后将这些状态作为层次结构树进行研究。这样就出现以下过程：

- 从最终目标开始。
- 导出达到那一点所必需的能力。
- 将大的步骤分解为较小的步骤。
- 使需求层次化。

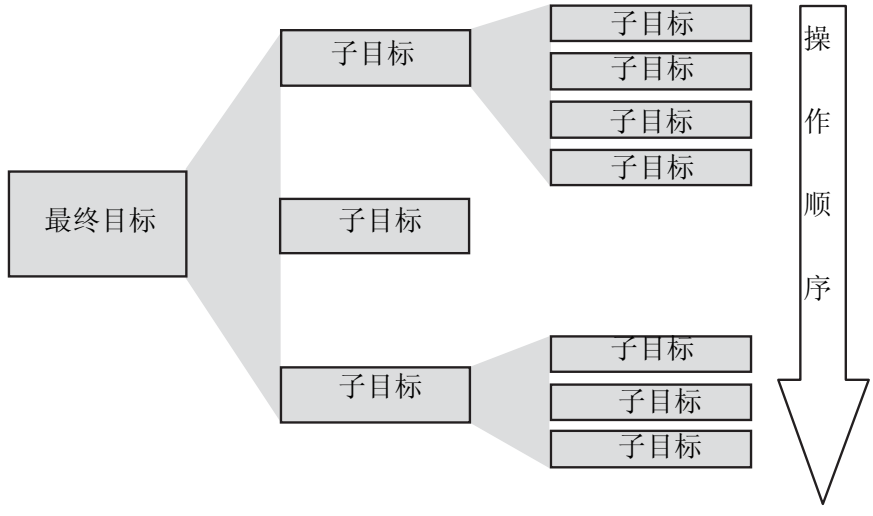


图5.3 作为目标层次结构的用户场景

- 在每个阶段非正式地评审。
- 提防定义解决方案。

如果 stakeholder 发现很难定义中间阶段，则可以要求 stakeholder 描述典型情况，了解 stakeholder 在所描述的某种情况下会做什么非常重要。如果系统是全新的，则可能需要使用自己的想象力。可以猜测在每一步 stakeholder 要做什么，或期望发生什么，或达到什么。这时标识是否有阶段是可选的、是否有重复，也是很重要的。不同的条件会导致不同的顺序吗？

stakeholder 还需要标识能力的顺序，以及这种顺序是固定的还是可变的。如果是可变得，在什么环境下变化。例如，在可以画画之前，必须先有画纸（或画布等）、颜料和画笔，但是先准备好哪一样并不重要。这种信息可提供顺序变更或并行处理的机会。

正如所有形式的需求捕获一样，重要的是要接受 stakeholder 提供的所有信息。这些信息以后总是可以细化的。这往往需要要求 stakeholder 解释信息的具体含义。

场景表示由待开发系统（采用问题领域观点描述）提供的，组织为层次结构的能力，而不说明如何提供这些能力。场景能够发挥作用看起来有以下原因：

- 场景使 stakeholder 能够走通操作使用情况；
- 可以找出遗漏的步骤；
- 不同 stakeholder 可以有不同的场景；
- 可以标识时间构造。

使用场景的特征

图 5.4 给出了带着能够用汽车运输的帆船外出为基础的一个场景例子。这个例子包含从将帆船装在汽车上，准备起航，操作帆船，直到回家的旅程中的各个方面。

这个场景还说明了一些其它问题：

- 一般地讲，它遵循时间顺序；
- 其节点是高层能力；
- 显示替代能力；
- 显示周期性重复行为；
- 显示顺序不重要的地方（并行分支）；
- 显示例外情况。

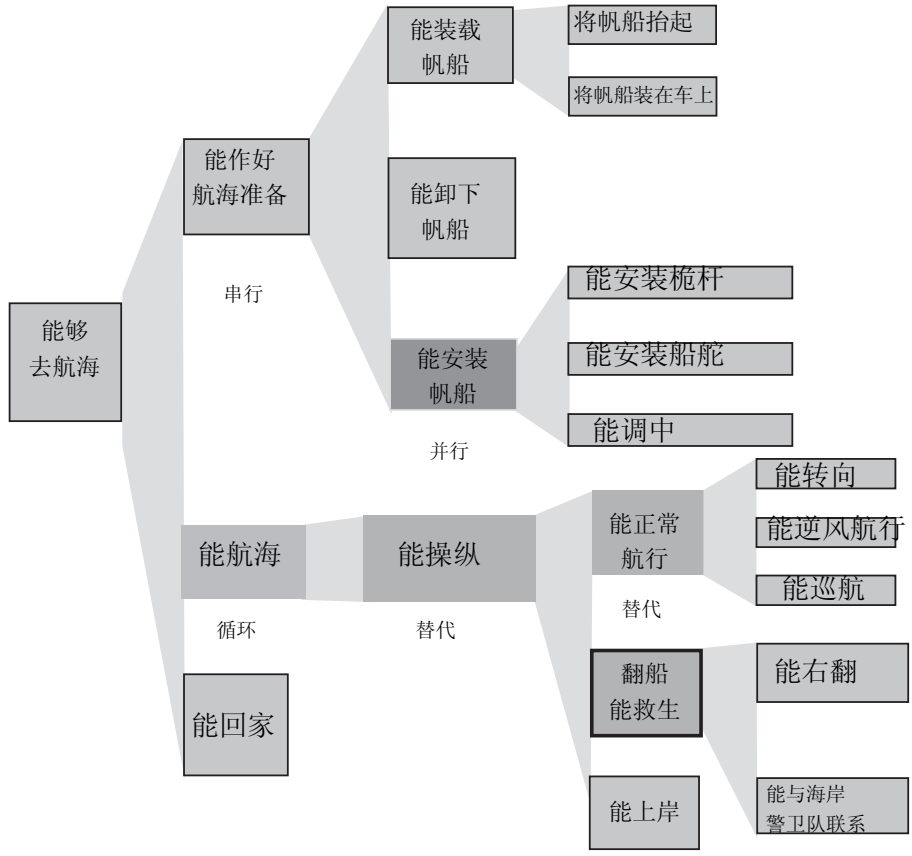


图5.4 使用场景举例

使用时间顺序是重要的，这样做不仅能够提供便于 stakeholder 理解的简单框架，还有助于将 stakeholder 需求放在一定的背景环境中。

重要的是要把所有节点都描述为合适层次上的能力。在这些节点的名称中采用“能够……”有助于避免将能力看作功能（这样会转向实现细节）。

场景提供了研究例外情况的非常强的方法。在很多系统中，处理例外的功能要比提供 stakeholder 所需的主要能力更复杂。在获取例外信息时，可以问 stakeholder 诸如“在这种状态下会出现什么问题？”或“在到达这个状态过程中，会出现什么问题？”这类的问题。通过询问如果出现问题应该怎么办（或发生什么），可以研究恢复行动。

在图 5.4 所示的例子中可以看出，该场景包含当出现翻船事故时需要进行通信。如果没有场景，这种需求可能不能被发现。

这个例子还说明场景如何便于发现遗漏的需求部分。在这个例子中，遗漏了“能够运输被装载的船”（到航海地点）以及“能够下水”的能力。

创建场景的目的，是便于理解和沟通。场景本身并不是需求，而是用于提炼需求的一种结构。场景通过提供操作使用的各个方面，有助于找出完整的需求集。任何一种建模技术都不会表示所有可能的概念，对给定操作的建模并没有单一的正确方法。不同的人会提出不同的模型。

5.4.3 确定系统范围

在准备场景时，最好能够把边界定义得比预期系统边界略宽一些。这样做可以确保所采用的视图不会“看不清周围事物”，便于将系统放在背景环境中。有时最基本的问题是确定系统的边界放在什么地方，并确定系统的范围。

一旦完成完整的场景集合之后，还可以细化系统的范围。在估计了系统开发成本之后，这类决定可能必须更改。这种估计可以由具有待开发系统领域的系统开发经验的人员完成。纯粹以场景为基础的评估还很粗，因此一定会有与场景关联的很大程度的不确定性。但是作出这种估计，可以给出所建议的经费预算是否是在可接受范围内的初步信息。

5.5 导出需求

我们将导出需求过程和鉴定策略过程分开，两部分内容在本节和下一节分别讨论。

针对问题领域实例化的导出需求过程如图 5.5 所示。关键活动是捕获需求和定义容纳这些需求的结构。一旦确定了结构和候选需求之后，可以将候选需求放在结构中。在实践中，这两种活动并行进行，并随着使用结构的经验不断增多，要不断修改结构。因此，不是用一个单独的活动处理候选需求，并把需求放在结构中，如图 5.5 说明的是，两种活动对确定结构化需求都有作用。

当结构被确定之后，可以评审并细化需求和结构。

5.5.1 定义结构

结构对于处理整个生命周期中的所有复杂要素都是至关重要的。通常逐个捕获 stakeholder 需求，然后再整理并放在结构中。有些方法假设：

- stakeholder 需求本质上是无结构的；
- 对设计的可跟踪性已经足够；
- 我们永远也不会看到完备的需求模型，因为一次只需要研究一个需求。

这些方法都与质量无关，只是从开发人员的短期利益考虑。

需求需要组织，当单个需求出现时就要有好的结构来管理。在结构和是否需要需求结构上的

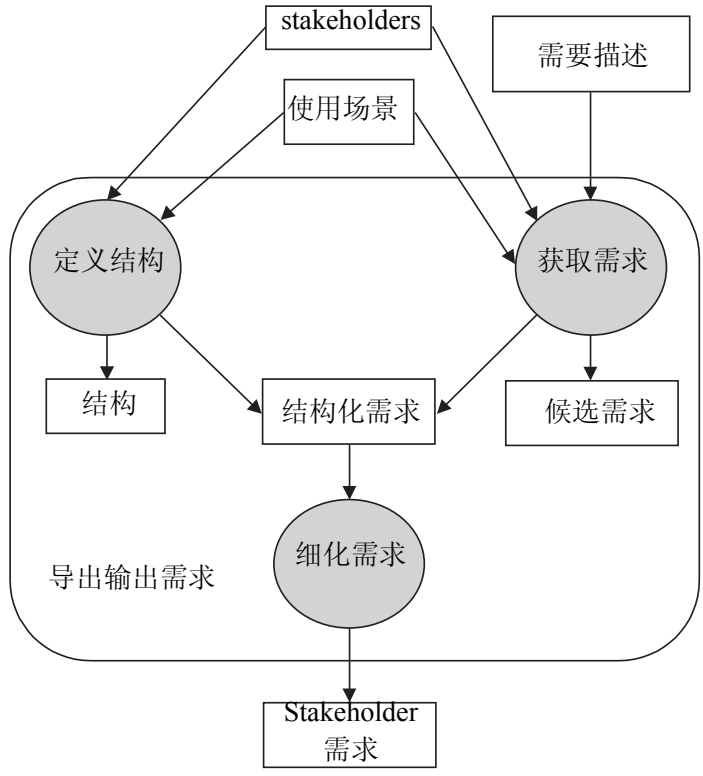


图5.5 问题领域需求的导出输出

争论，与问题领域和解决方案领域中的需求工程争论一样。因此将其并在第 4 章中一起讨论。本章认为提供可理解的结构至关重要，因此问题就归结为如何导出用于 stakeholder 需求的结构。

stakeholder 需求的主要结构化概念是使用场景。但是取决于系统的性质，可能会有很多这类场景。建议花些时间和精力来以尝试合并场景，如果可能的话，构成一个单一的整体场景。显然这并不是永远都可能，但是进行尝试是好想法。除了其它结果之外，这样做确实使人可以意识到系统的整体外延，并常常暴露出很多问题。

为了说明有时可以用来合并场景的方法，下面举一个经营餐厅的例子。有三个场景可以用来描述餐厅：

1 餐厅的生命周期情况－－拥有者场景；

2 餐厅的一天情况－－经理场景；

3 餐厅的一餐情况－－顾客场景。

这些场景如图 5.6 到 5.8 所示。

餐厅生命周期场景中的第一个目标是拥有者获得餐厅，然后是餐厅的经营阶段，最后售出餐厅。

餐厅一日场景考虑餐厅一天中的不同状态。第一个目标是补充食品和饮料库存。这些场景部分会说明有多个供应商，但是交付食品和饮料的顺序并不重要。有人可能会说，在餐厅开业之前没有必要完成补充库存工作，但是为了给出合理的例子，规定在餐厅向顾客开放时，不接受任何送货。这样，一日场景的区间从营业开始，整理好餐厅中的一切，记录好要在第二天中需要的库存补充，到一天工作结束为止。

客户一餐场景是很明显的状态序列。

如果观察这些场景是如何放在一起的，就会发现：

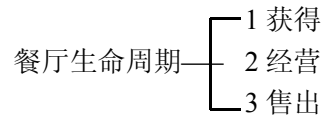


图5.6 餐厅历史场景

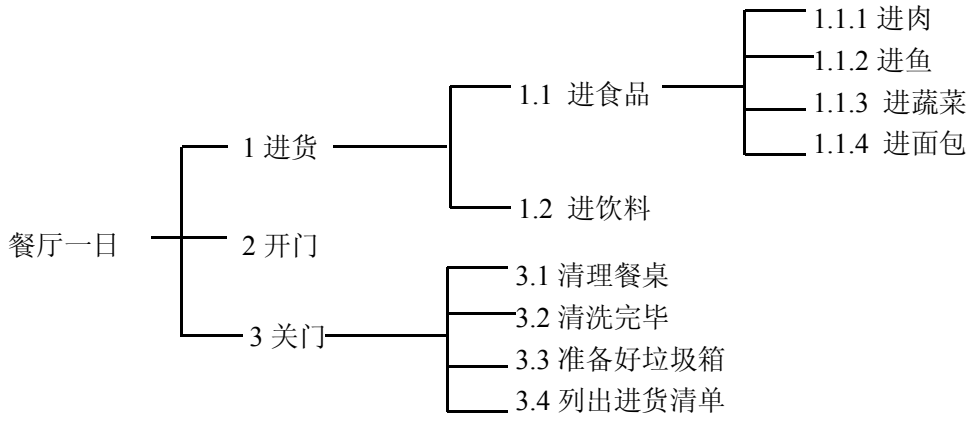


图5.7 餐厅一日场景

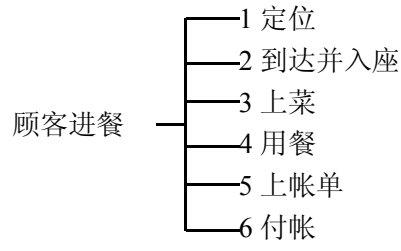


图5.8 餐厅一餐场景

- 餐厅一日场景可以是餐厅生命周期场景中经营状态的重复场景；
- 一餐场景可以是餐厅一日场景营业状态的并行重复场景。

因此，这三种不同 stakeholder 场景的总体结构可以按图 5.9 画出，并作为需求文档中能力标题的结构。

当然，在有些情况下是不能将场景放在一起的。这里没有简单答案。如果所有合并尝试都不成功，则可以一个接一个地单独使用场景。因此，stakeholder 需求文档的结构会是一个场景序列，每个场景都嵌入自己的需求。从本质上看，结构是通过所有 stakeholder 驱动的，即使采用这种方法，也应该尝试将一个场景嵌入到另一个场景中。

但是必须小心地保证没有重复。如果有重复，则所重复的部分必须只出现一次。有两种方法可以使用。第一种要求去除共同项，将其放在各自的单独部分中。然后每次出现都必须在合适的时候引用所分开的部分。另一种方法是将重复部分放在文档的第一个场景中，然后供所有其它出现引用。

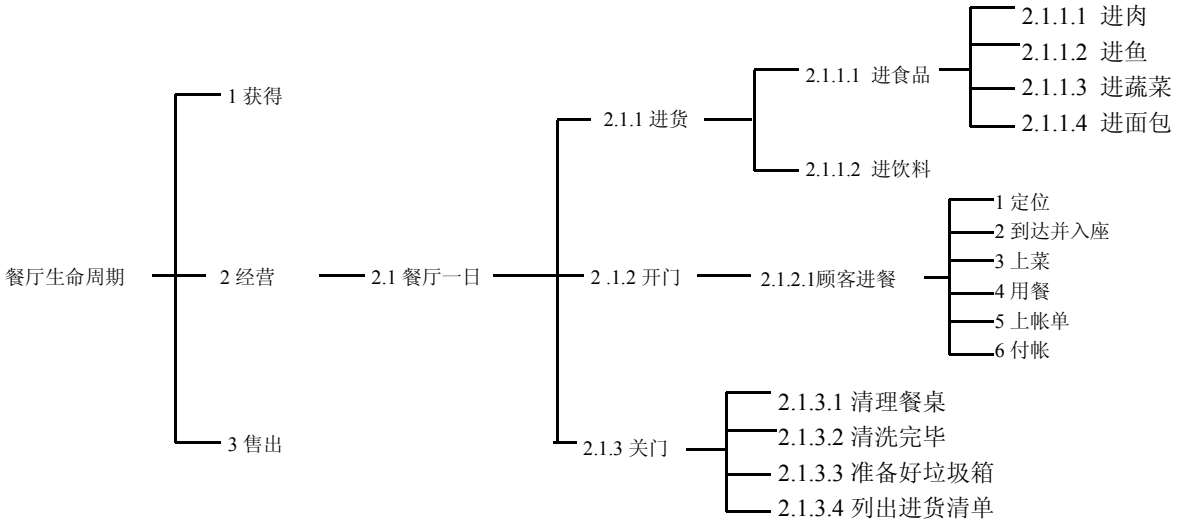


图5.9 餐厅能力的总体场景和结构

5.5.2 捕获需求

stakeholder 需求的来源

stakeholder 需求可以通过不同来源获得，例如：

- 与 stakeholders 进行面谈；
- 场景探索（一般通过 stakeholders 面谈）；
- 描述性强的文档（可能通过研究或市场调查）；
- 要被升级的现有系统；
- 现有系统存在的问题和变更建议；
- 类似系统；
- 产品或需求本身的原型，可以是部分系统、实验模型甚至简单草图；
- 新技术提供的机会（经过 stakeholders 同意）；
- 研究；
- 调查表；
- 人机工程学研究或录像分析。

与 stakeholders 面谈

为了完成这个任务，需求工程师必须是好的沟通者，要通过与 stakeholder 面谈发掘出其真实需求。这是一种有很强心理学要求的任务，与系统开发的技术和运营没有什么共同之处。重要的是要注意，提取 stakeholder 需求是一种人的问题，而不是技术问题，因此提前进行准备，以便理解 stakeholder 的领域这一点很重要。

重要的是要采用 stakeholder 的语言讨论 stakeholder 的领域问题，而不是最终产品或其它技术问题。在面谈时，应该通过问题，使 stakeholders 走通自己的工作过程。要作全面记录，以便以后组织到结构化需求集合中，并返馈给 stakeholder。面谈是一种交互过程，重要的是需求工程师不要作出评判，而应该不断反复地询问“为什么？”。可以采用多种方式问这个问题，例如：“这样做的目的是什么？…”或“你能提供这个问题的更多背景信息吗？…”显然，不能指望需求工程师成为 stakeholder 领域的专家，因此需要在各种场合澄清这一点。不要担心会问出（看起来）很傻的问题，最傻的问题就是没有提出问题！但是，stakeholder 最终要对所提供的需求负责，这一点很重要。

要在面谈中讨论场景问题！

以下是一些与 stakeholder 面谈的小提示：

- 要与所有类型的 stakeholder 面谈；
- 要很认真地面谈；
- 面谈时要作记录，并请 stakeholders 在面谈记录上签字；
- 找出哪些场景与正在面谈的 stakeholders 有关，并与他们深入讨论，请 stakeholders 说明在场景的每个阶段他们都要能够做什么；
- 如果有必要，要随着讨论的深入创建新的场景，并根据这些场景开发需求；
- 尝试发现每个需求对各类 stakeholders 的相对重要性；
- 如果 stakeholders 对任何需求回答得不清晰，则首先要问清楚该需求的目的是什么，然后问清楚如何演示所提出的需求；
- 询问 stakeholders 所知道的所有约束；
- 使 stakeholders 意识到他们提出的需求将帮助塑造系统；
- 启发并鼓励 stakeholders 回答；
- 不要评判 stakeholders 需求；
- 尽快将笔记整理为单个的需求，然后复述出来。

通常所询问的问题会从一般逐步深入到特殊。重要的是要确保覆盖所有方面，定义哪些领域是无关的。要在总结面谈经验的基础上，根据 stakeholders 和环境确定问题的形式。

从非正式文档中提取需求

在诸如信件、研究、活动表和其它类型的描述性材料中，都可能隐含地包括需求。这种用户需求不应该继续隐含，而应该带到明面上。在做这种工作时，重要的是要记录 stakeholder 需求的出处，换句话说，必须记录来源。不仅如此，通过这种方式提取的需求必须要经过 stakeholders 之一“证实”。

通过场景标识能力需求

当编写完场景大纲之后，可以直接根据场景来假定能力需求。有时所需要的只是对状态的简单解释。例如“准备航海”可以解释为“用户使航海船只处于航海就绪状态”的能力。在其它情

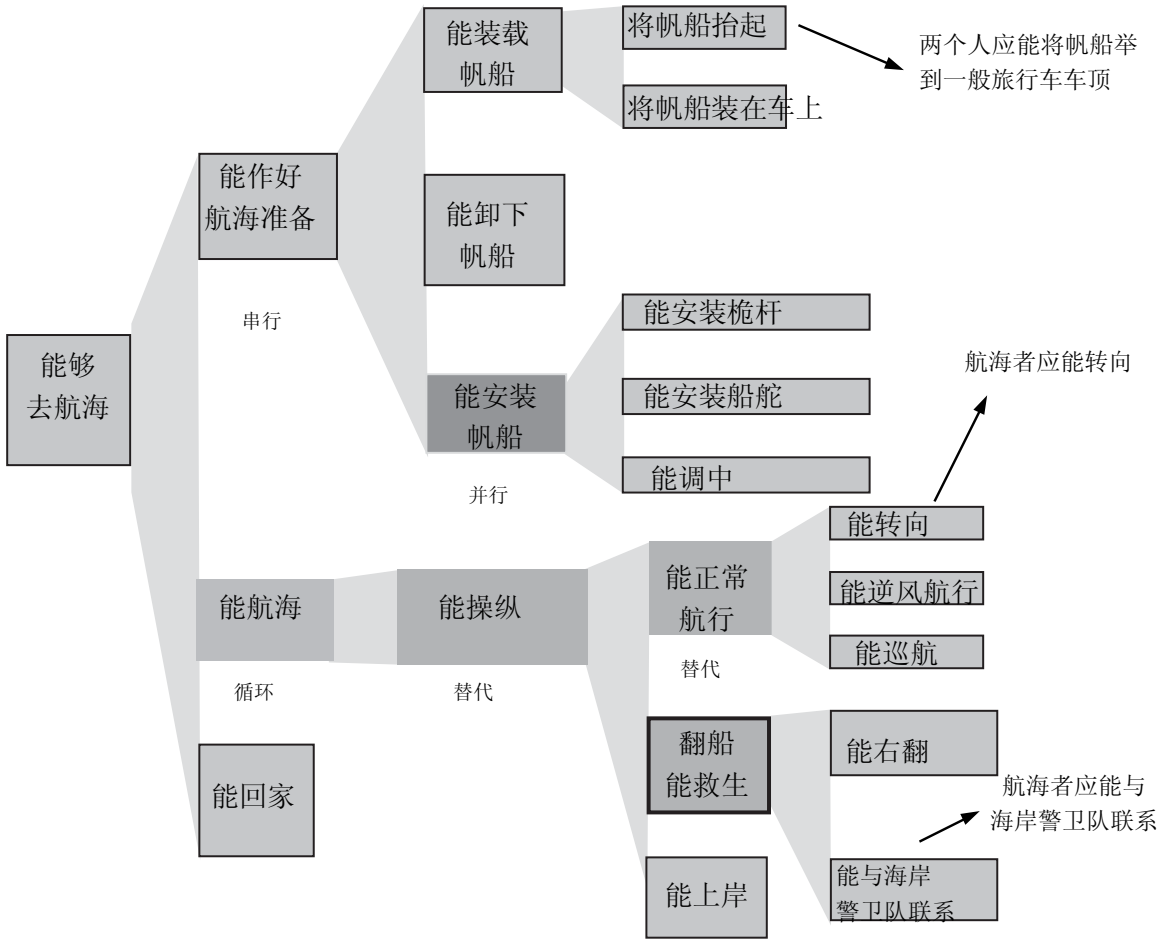


图5.10 通过场景导出能力需求

况下需要更多的工作。图 5.10 给出了一些例子，尽管这些例子描述得也并不是非常好。考虑这个需求：

两个人应该能够举起船，放到一般旅行车的车顶。

这会产生以下问题：

1 这些人有多强壮？

2 什么是“一般旅行车”？

这些问题最终都必须回答。不过在收集需求时，重要的是要将需求写下来。需求开始格式不太好并不重要，因为还要改进。关键问题是不要遗漏！借用一条著名谚语对这种方法作个总结：

值得做的事就是即使做不好也值得去做的事！

有关恰当地形成需求的更多信息，请参阅第 4 章。

需求研讨会

收集 stakeholder 需求的另一种方式是举行需求研讨会。这可以是快速引出并捕获需求的一种很好的方法。重要的是，从一开始就要使 stakeholders 处于一种有益的环境中，使他们意识到捕获需求并不困难，不需要占用很长时间。研讨会应该有议程，但也应该是迭代式的。如图 5.11 所示，应该向 stakeholders 说明期望从他们那里得到什么。例如，stakeholders 需要理解以下概念：

- stakeholder；
- 使用场景；
- 能力需求。

取决于研讨会的起点，在会议开始时可能已经有了需求草稿。也可以将与会者分成小组，编写待开发系统的场景。然后再在全体会议上评审所编写的场景，根据评审意见，对场景进行必要的修改，然后以这些场景为基础提取需求。

一旦可能，要尽快向全体与会者提交需求草案，并鼓励进行批评和讨论。不同类型 stakeholders 之间有可能进行讨论，会为需求增加重要的价值。这常常是这类 stakeholders 第一次坐在一起。通过 stakeholders 之间的交互，编写更深刻地反映每类 stakeholders 所需能力，以及这些能力如何与其他 stakeholders 所需能力匹配的需求，总是会得到令人感兴趣和满意的结果。

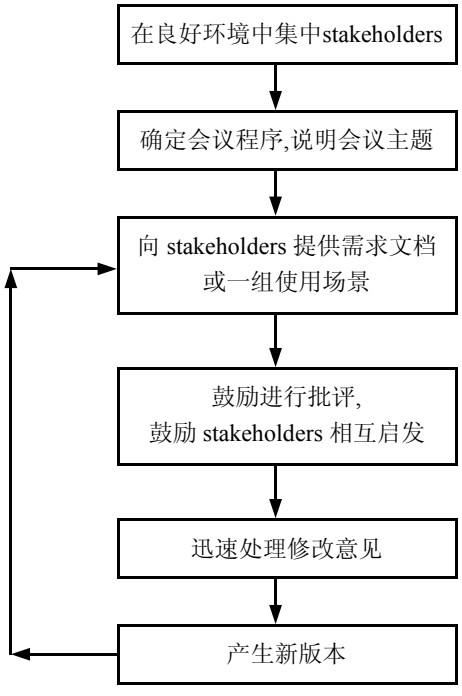


图5.11 需求获取研讨会

在研讨会上使用大屏幕投影机，全体与会者参与需求的联机编辑，但是将与会者分成小组，分别编写需求的一部分，然后再评审全部需求，效率可能更高。采用这种方式，编写典型项目的需求需要 3 到 4 天。

研讨会的关键要素是首先确定动力，然后不断推动。举行研讨会的要求可能很高，但是研讨会的结果对于各个方面都会是很有价值的。

要代表所有 stakeholders，并且 stakeholders 都可以作出决定，这一点至关重要。

通过经验了解需求

系统的真正用户所报告的问题是一种宝藏，但是这种信息常常被抛弃。由于这些信息与问题关联在一起，因此人们多少报有消极态度，但是这类信息具有真实的价值。显然，发现问题越早，变更成本越低，但是允许太轻易地进行变更会毁了项目。在迭代式开发中，常常是将变更推迟到系统的下一次审查之后进行。

通过原型获取需求

在开发前所未有的系统时，原型是无价之宝。原型可以用于向 stakeholders 提供一种系统能做什么的概念。在开发很难想象的基于软件的用户界面系统时，原型也非常重要。使用原型的问题是，开发人员的注意力会分散，并花费太多的时间和工作量。因此，原型开发应该总是被看作是具有其自身需求的小型子项目。应该永远明确开发原型的目的，原型通常要提供更深刻的认识，便于更容易、更准确地编写 stakeholder 需求。

原型法有以下三个方面的问题：

1 开发人员会分散精力，在细节方面钻得太深。

2 原型容易使 stakeholders 陷入实现问题中。

3 原型给 stakeholders 留下的印象太深，以至于使他们想要正式使用原型。

头两个问题通过恰当地提出原型需求可以解决。为了解决第三个问题，重要的是要使 stakeholders 真正充分认识到原型的演示性质，因为原型是一种部分系统、实物模型甚至一组简单的草图。

stakeholder 需求中的约束

约束是不为系统添加任何能力的一种需求，它控制一个或多个能力所要交付的方式。

例如，考虑以下句子：

顾客点菜后，要在 15 分钟内上菜。

这并没有使系统发生本质上的变化，只是量化了所提供的服务。

但是需要注意的是，每个都是合理的大量约束可能使开发不能进行，因此既需要单个地分析约束，也需要系统地分析约束。

当已知设计时，应该对每个约束分析费用与效率比，或对系统的影响。约束可能会引出某种功能，例如警告机制或备份机制。在已知设计之前，只能猜测约束的成本。不幸的是，这种猜测又要取决于设计选择，不过可以进行某种最低假设，因为太多不必要的约束会毁了系统。

在默认情况下，约束适用于顶层能力及其所有从顶层能力继承的子能力。可适用性要尽可能向层次结构的下层推（如图 5.12 所示），以便限制约束的可适用性，降低约束的影响。如果约束只适用于一个能力，则该约束可以写作能力的一部分。

需要注意 stakeholder 约束和系统需求约束之间的差别。stakeholder 约束指 stakeholder 想要的结果，系统约束是影响产品质量的“专业”或工程约束。所有 stakeholder 约束都必须在系统约束中解决。有时需要重新建立，有时可以不经过修改地直接使用。

细化需求

在每个需求的背景环境中进行评审，并保证：

- 需求属于它所在的位置；
- 符合第 4 章所讨论过的良好需求的编写准则。

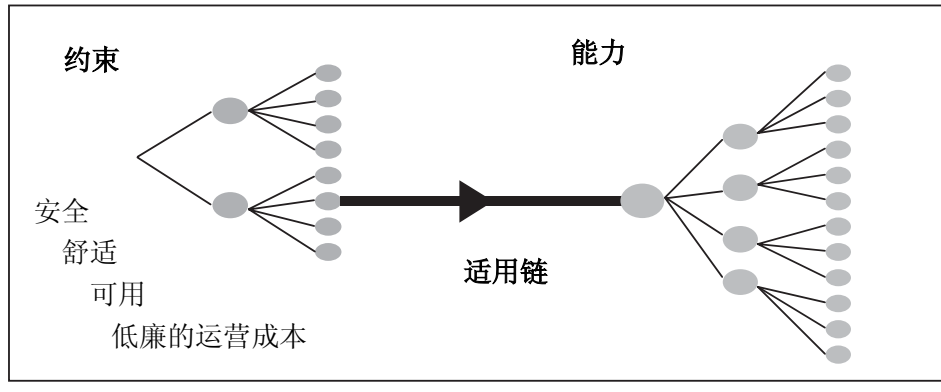


图5.12 能力与约束

5.6 导出鉴定策略

有两种子过程可以用来导出鉴定策略，如图 5.13 所示。以下介绍这两种子过程。

5.6.1 定义验收准则

理解将使 stakeholder 满意并说明需求被满足的验收准则，是需求获取的基本和关键部分。询问问题：

什么能够使你确信这个需求已经被满足？

常常可以更清晰、更有重点地形成需求。因此，这个问题常常在与 stakeholders 面谈时使用。这种问题可以用两种方式来回答：

1 stakeholders 可以定义一种操作背景，通过这种背景可以演示需求；

2 stakeholders 可以定义必需演示的要达到的量化水平。

第一种回答可直接用于创建一组必须作为鉴定策略一部分所提供的测试、试用或演示的过程。第二种回答说明试验测试或演示的“通过标准”，也就是指示需求的验收准则。

每个需求的验收准则定义什么将是所采纳鉴定方法的成功结果。验收准则通常记录在与该需求关联的属性中。换句话说，在需求与需求的验收准则之间，通常存在一对一关系。在餐厅例子中，经营该餐厅的验收准则可以是，餐厅是“成功的”。成功可以通过多种方式度量，例如：

- 效益；
- 投资回报；
- 消费指南、报纸文章等中给出的声誉；
- 用餐厅的订餐情况所表示的未来客流量。

不同的 stakeholder 对于成功会有不同的看法。例如，财务经理对头两个要素更感兴趣，但是

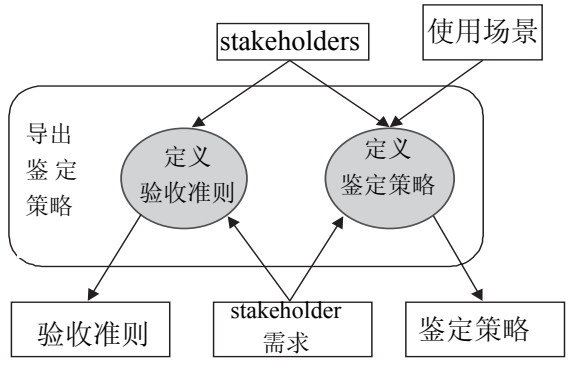


图5.13 导出鉴定策略的过程

总经理一定会对后两个要素更感兴趣。

因此，重要的是通过可能拥有自己观点的所有 stakeholder 来确定验收准则。

5.6.2 定义鉴定策略

证实可接受性的方式，在很大程度上取决于应用程序的性质以及其获取的方式。对于大型的一次性系统，例如空中交通管制系统，需要保证已经恰当地提供所有功能，并且交通控制人员对于繁忙时对可以容易、快捷地使用系统而感到满意。这既需要测试也需要试用。首先，必须在轻载荷情况下演示系统的能力。如果这种能力是不可接受的，则没有必要继续进行需要多得多的投资的测试，例如繁忙时间的实际试用。

还必须考虑鉴定策略的成本。进行全面试用非常昂贵，因此总是要逐步扩展。例如，大多数船只在进行航海试验之前都要先进行港口试用。

还必须考虑总成本，但是必须权衡在运行使用中没有发现系统重大问题的风险。因此，对于有很大安全、环境或资金风险的系统，必须小心设计鉴定策略，以保证渐进但是稳定地建立起对系统的信心。另一方面，如果功能失效的后果相当轻微，则可以采用不太昂贵的方法。底线是一个不能（以某种方式）被演示的需求就不是一个需求。恰当设计的需求是可以很容易地被理解和演示的需求。

5.7 小结

stakeholder 需求集合必须尽可能地小，并易于理解。stakeholder 需求必须是非技术的，同时又是现实的。必须关注角色和责任，重要的是恰当地区分不同的 stakeholders。

导出 stakeholder 需求时的常见问题是：

- 过于强调解决方案；
- 忽视定义要解决的实际问题；
- 没有理解 stakeholders 必须拥有并赞同这些需求。

stakeholder 需求应该尽可能快地编写，要以 stakeholder 喜欢和熟悉的词汇定义 stakeholder 所要求的能力。因此必须关注 stakeholder 的领域，而不是解决方案。stakeholder 需求应该是结构化的，并可以跟踪到信息源。stakeholder 需求是由 stakeholders 拥有的，由预算负责人确定范围，常常由需求过程专家编写。

第6章 解决方案领域中的需求工程 6

永远不要告诉别人该怎样做事，告诉他们要做什么，他们的天才表现会使你吃惊。

（George Smith Patton，将军，1885－1945）

6.1 什么是解决方案领域？

解决方案领域是工程师利用其聪明才智解决问题的领域。区分解决方案领域与问题领域的主要特征是，解决方案领域中的需求工程从给定的一组需求开始，而问题领域的需求工程从模糊的目标或希望列表开始。把输入需求扩展到解决方案范围，完全取决于开发解决方案的客户机构内人员的素质。在理想世界中，所有需求都被清晰地说明，并且是单独可测试的。

正如第 2 章所讨论的那样，解决方案极少能够通过一个步骤获得（如图 6.1 所示）。

在每个层次中，都要进行建模和分析来首先理解输入需求，然后为下一层次导出需求提供坚实的基础。设计的层次数由应用领域的特性以及开发设计的创新程度决定。不管需要多少层，了解各个步骤应该引入多详细的解决方案，即“如何做”，总是至关重要的。

在解决方案领域中的每个层次上，工程师都必须作出决策，向最终解决方案发展。所有这些

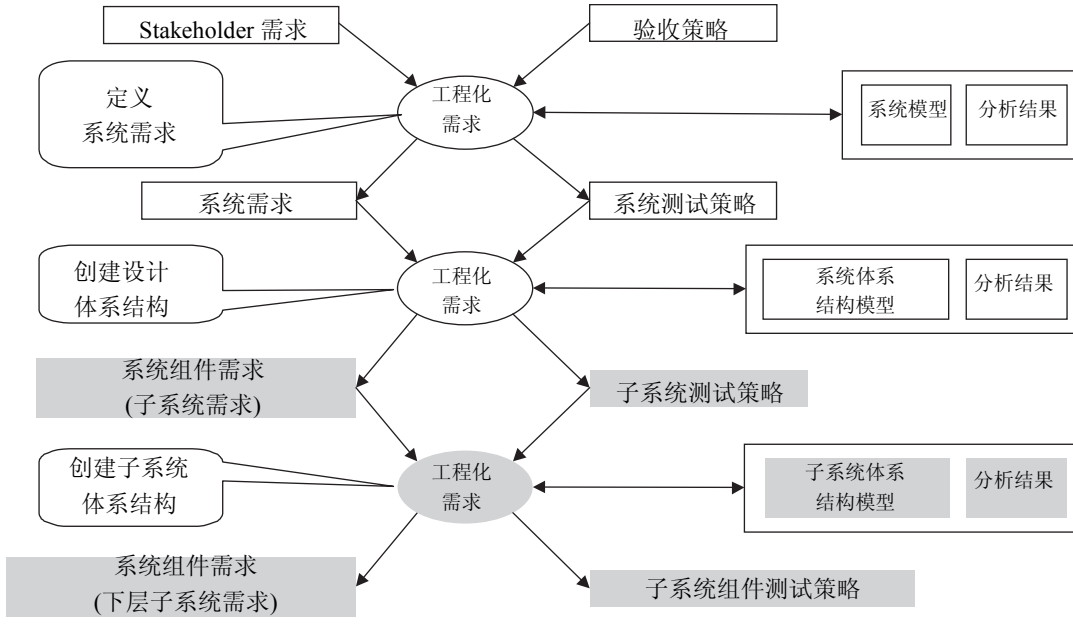


图6.1 通用过程可能的实例化

决策，本质上都会压缩可利用的设计空间；也就是说，排除一定的设计选项，但是没有决策就不可能有进展。工程师总是会过早地受到过于深入的诱惑。必须避免被诱惑，以便使创造性和智慧性相结合来产生创新的解决方案，否则由于不成熟的设计决策所造成的局限性而会使得创新设计变得不可能。

在典型情况下，解决方案领域中的系统开发第一层，是将 stakeholder 需求转换为一组系统需求。这些系统需求必须定义为了解决 stakeholder 需求所产生的问题，系统必须做什么。图 6.1 中的通用过程上层的实例化说明了这个第一层。

不成熟的设计细节问题在第一个步骤中影响特别大。必须使我们能够在不涉及不必要的细节的情况下，在抽象层上定义系统的功能，创建图 6.1 给出的系统模型。

定义系统需求之后的下一步，是创建由图 6.1 所示的通用过程的第二个实例化所说明的体系结构设计。这必须采用交互作用的组件来表达，组件交互作用会生成系统需求所标识的的新属性。通过体系结构设计过程（如图 6.1 所示）产生的导出需求，定义组件供应商必须满足的每个组件的需求。

开发向设计的更深层次推进，逐步到实现细节。

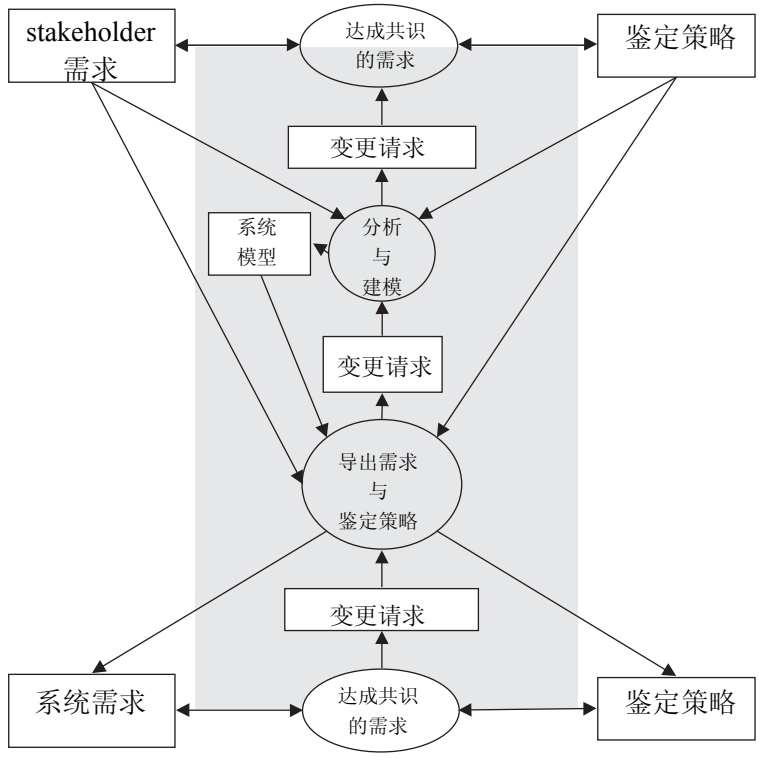


图6.2 创建系统需求的通用过程实例化

本章集中讨论从 stakeholder 需求到系统需求的转换，因为在大多数开发中这一点最成问题，因为在典型情况下，常常太早地添加了太多的细节。

6.2 从 stakeholder 需求到系统需求的需求工程

这种转换的通用模型的完整实例化，如图 6.2 所示。

与所有实例化一样，过程是从对输入需求的共识开始的，对于通用过程的实例化来说，就是 stakeholder 需求。达成共识的过程不能假设已经根据本书前面给出的指南产生了输入需求。需要考虑输入需求以及相关的鉴定策略的优点，并将评审准则严格、全面地应用于 stakeholder 需求。

6.2.1 生成系统模型

为了避免过于深入细节的倾向，工程师应该总是在模型背景下工作（如图 6.1 所示），这种模型对于定义待开发系统应该做什么而不是怎样做，已经够充分详细了。在导出需求中应该提供的细节层次，取决于需求工程的开发层次，但是最高限度永远是“不增加超出必要的细节”。深入细节的诱惑在顶层时总是最大的，这时要将采用问题领域的术语表示的 stakeholder 需求，转换成高层系统需求，说明系统必须做什么以解决 stakeholder 提出的问题。这会产生困难，因为需要是在抽象层次上工作的。创建提供系统需求的抽象系统模型永远都会产生问题。在这之下的各个层次，开发工作要在设计体系结构背景下推进。工程师会习惯于这种细节层次，因为他们可以参与确定系统将来的工作方式。即使在这些层次上，也必须小心推进，保证所引入的细节是合适的。接下来，应该通过一起运行的组件来描述体系结构模型，但是还应该很小心，要保证从要求组件做什么，而不是如何做的角度定义组件。换句话说，组件应该被描述为“黑盒子”，只考虑组件满足需求所定义的总体目标，不考虑其内部细节。

本章下一节集中讨论系统模型的准备，以导出系统需求，然后介绍如何将这种方法用于更详细的层次上。

6.2.2 创建系统模型以导出系统需求

必须在适当的抽象层次上创建系统模型，以便系统模型能够包括：

- 系统必须具备的内部功能，也就是必须关注系统必须做什么，而不是为了避免使设计空洞而去描述系统应该怎样做；

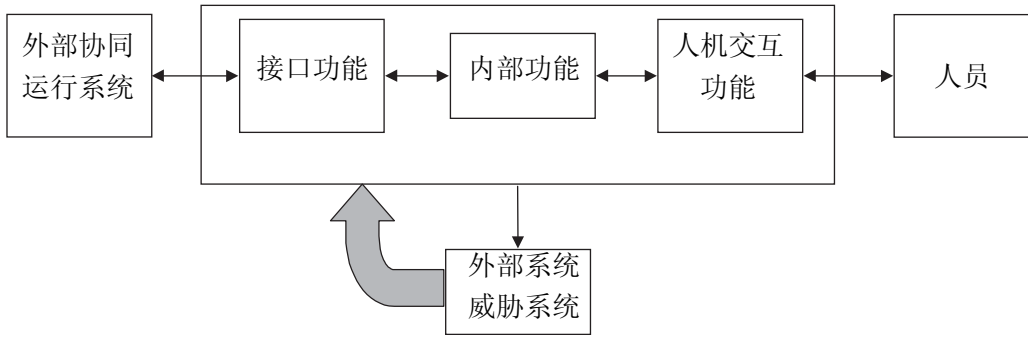


图6.3 系统环境与功能类型

- 使系统能够与环境中的其它系统交互的必要功能；
- 使人员能够成功地与系统交互的必要功能；
- 防止系统由于环境中其它系统（威胁）的存在而失效的功能。（请注意，有些系统可能会受到潜在的威胁，例如来自周围设备的电磁辐射。）

图 6.3 采用框图的形式，说明了这类功能相互交互，以及与系统环境中其它要素交互的方式。很明显，定义系统所处环境下的系统背景应该考虑：

- 要求新系统与之协同工作的现有系统；
- 需要与系统交互的人员类型；
- 系统必须抵御的威胁。

可以通过多种方式表示功能，例如：

- 类图中类的操作或方法；
- 消息顺序图；
- 状态转换图；
- 功能流框图；
- 数据流图中的过程。

在实践中，会需要使用多种模型以覆盖所要求的很多不同问题。每个模型都包含一组已定义的类型信息，并且每种建模手段都包含其自己的语义。有些模型的信息可能相当地独立于其它模型的信息，而有些信息可能出现在多个模型中。对于后一种情况，重要的是当信息变化时，这种变化要反映到所有出现这种信息的其它模型中。在理想情况下，通过集成的建模工具可以自动地做到这一点。如果建模工具不能集成，则必须极为小心，确保所有变更都被同样地用于所有有关

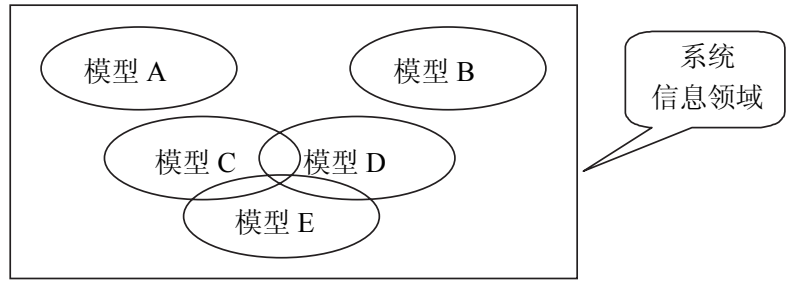


图6.4 系统模型的范围

模型中。图 6.4 给出的维恩图说明，有些模型表示信息孤岛，有些模型具有采用不同形式表示的公共信息。图 6.4 还说明，有些系统信息有可能没有通过任何模型表示。

内部功能

这是创建系统需求的基本要素，因为内部功能是定义系统要做什么的主要焦点。需要创建一种结构或模型能够做为创建系统需求的基础。这种模型必须能够以某种形式表达分解成模块或高层组件的系统，例如子系统。这里使用“模块”或“组件”这样的术语，是为了使开发人员更多地用实现，而不是规格说明的术语来思考。一般来说，这样做被认为是差的实践，特别是基于软件的系统。在一般系统中，向更物质化模型推进的需要，不会被认为有很大的问题，因为应用领域一般来说更具有物质化性质。

为了取代可能会导致不成熟实现的术语，有一种有意思的倾向（有人可能叫做“时尚”）是使用“对象”一词作为分解要素，特别是基于软件的系统，因为对象可以指问题领域中的内容。这种方法有助于避免这种不成熟性被深入到解决方案问题中。这样，功能可以作为对象的方法或操作，以及对象之间的交互作用。

使用这种面向对象的方法，还可以使建立从系统需求到 stakeholder 需求的可跟踪性变成一种比较容易的工作，因为同样的对象一般在问题领域和解决方案领域中都可以看到。

系统模型除了说明系统必须做什么之外，可能还需要说明系统的预期行为。有多种方式可以表示这类信息。这个领域的模型通常表示有若干并发活动的“角色”以某种方式交互这种事实。这种表示的例子包括消息顺序图和行为图。消息顺序图早已在电信界使用，行为图最早在 70 年代的美国弹道导弹预警系统（BMEWS）中使用，后来在工具中实现，例如 Ascent Logic Corporation 公司的 RDD-100 和 Vetch Corporation 公司的 CORE。行为可以通过状态转移图或状态图建模。状态转移图有只能对状态序列建模的限制，并且被建模的内容任何时刻都只能处于这些状态中的

一个。状态转移图不能直接表示层次状态，层次结构中的每一层都必须使用不同的状态转移图表示，在有些情况下，这意味着在一定时刻会有一组活动的状态转移图。这样的状态转移图组会很难理解。为了避免出现这种复杂情况，最好使用状态图，因为状态图可以直接处理状态层次结构，还可以处理并行状态。

在任何系统中，都需要考虑是否有要处理的信息。有些系统，例如保险公司系统，要求跨越很多年使用所收集和维护的信息。而另一些系统，例如空中交通管制雷达数据处理系统，有些信息的寿命很长，例如航班计划；有些信息，由于数据的性质，如飞行着的飞机的当前位置，而很快就过时了。因此，必须研究信息需求以确定：

- 信息的寿命，即信息有多长时间可用，必须维护多长时间？
- 信息的新鲜度，即信息必须有多新（秒、分还是小时）？

还非常需要了解必须包含的信息量。这可能对系统设计有很深的影响。

接口功能

有必要定义要求与所有其它系统交互的性质。交互可能会涉及系统之间的信息或物质的流动。流动可以是单向的，也可以是双向的，传输能力可能有限制，可能需要为被搁置的信息提供临时存储（例如数据缓冲器或仓库）。有时可能还有应答速度要求，一个系统必须对来自另一个系统的激励作出响应。

接口的性质有很大不同。但是，必须总有说明每个成员承担要做的事的基线参照，或作为接口部件提供。这些责任常常形成接口控制文档（ICD）。如果接口受国家或国际标准控制，则这些标准就成为所有有关各方行动的接口控制文档。如果接口定义得不够完备，也必须形成书面的责任（即接口需求）并达成共识。众所周知，这些需求的控制很难，因为常常没有明确权力机构来控制接口。结果，接口各方很可能会有自己版本的文档，更糟糕的是，各方一般都对接口有自己的解释。

接口文档通常由负责两个（或多个）系统之间接口的机构控制。如果要开发的是新系统，则这类机构很难定义。现有系统常常早已开发，而且可能没有恰当地形成接口文档。不仅如此，开发机构可能早已不再对现有系统负责，已经将系统移交给高层客户或其它运营管理机构了。

必须小心地保证所有接口责任都准确地反映在合适层次上的导出需求中，并尽可能清晰地定义接口控制权威部门。

人机交互功能

人员与系统交互的关键问题，是所要求的是什么交互。用户工作的背景也很重要，这会影响用户工作的方式。例如，在标准办公室环境中工作的用户不会寒冷，能够不带手套方便地工作。其他用户可能会在严酷的环境中操作系统，例如在很低的气温环境，或必须穿戴防护服装的危险环境。在这些情况下，显示器或键盘的设计必须注意到这些问题。

安全功能

系统必须运行的环境也会有重大影响，特别是安全和保密。例如在银行系统中，需要保证信息和钱不会提供给未授权人员。在汽车（系统）中，需要保证踩下制动板时汽车会停下来。

有些系统可能要运行在与待开发系统竞争的系统环境中。这种竞争可能是商业竞争，例如在线银行。迅速提供所需的所有系统可能性具有头等重要性。

其它“竞争”系统还包括会干扰系统正确操作的系统，通过例如通过无线电传输误导系统，或使敏感接受设备过载的系统。一个例子是在正在飞行的飞机上使用移动电话，有可能干扰飞机的导航系统。

系统事务处理

重新研究根据 stakeholder 提供的信息为系统开发的使用场景是值得的，如果还没有这样的场景，可创建一组相关场景。这些场景可用于系统模型，以保证这些场景能够在被描述的系统内实现（如图 6.5 所示）。走通这些“系统事务处理”，可再次保证系统功能的有些要素没有通过转变为对象建模或功能分解而丢失。（请注意，这里的“系统事务处理”与第 3 章描述的 CORE 方法中的术语不同。）

在图 6.5 中显示的为用户系统作事务处理的系统事务处理是从用户场景中导出的。图 6.5 还说明，可以有从被开发系统必须与外部系统交互的方式中导出的其它事务处理。

系统事务处理鼓励系统工程师后退一步，从“整体”的角度观察系统。人们太容易关注细节、忘记整体了，即细节部分怎样协同工作来达到整体目标？

操作的模式

在有些情况下可能需要不同功能。基于信息的系统的典型例子是需要能够为员工提供培训，

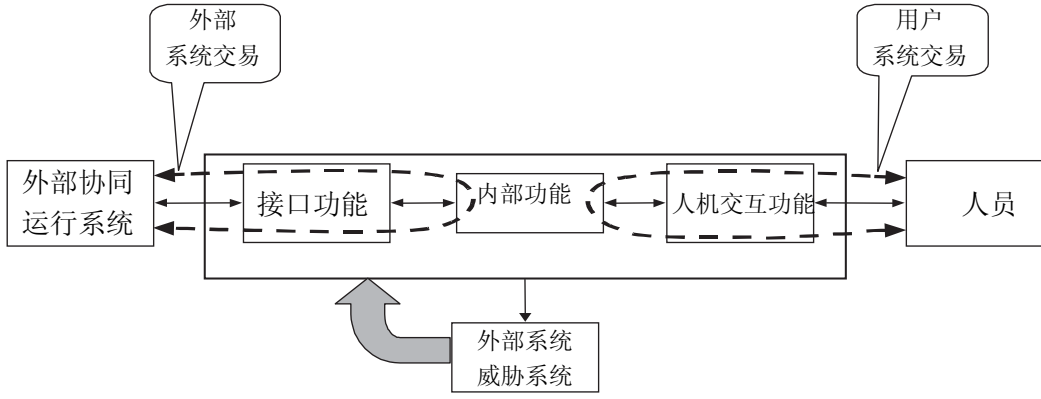


图6.5 系统事务处理

同时又不会破坏系统中数据的完整性。其它例子包括失效之后的操作低效运行模式，或军用系统中不同的模式取决于警戒的当前状态。这些可能会与 stakeholder 需求中的使用场景有关。

6.2.3 银行例子

在这个管理信息系统的例子中，主要关注的是对必须处理的信息建模，但是很清楚，还有很多其它方面的问题应该解决。因此有可能使用多个模型，一个模型关注信息，其它模型关注信息的流程和保密。

图 6.6 给出了银行例子的另一种抽象的模型。这个模型标识了设备可能安放的位置类型，并从这里开始事务处理。

内部功能

主要的内部功能是支持由银行提供的服务，例如当前帐户、存款帐户、贷款与投资证券。为了支持这些服务，系统必须能够采集、更新并维护信息。这里至关重要的是信息的类型（或类）（例如帐户、客户等）、信息类型之间的关系（例如，一个客户允许拥有多少帐户？）以及每种类型的说明、新鲜度和量。

重要的是要确定信息是怎样收集、分发和操作的。

银行系统的一个更重要的问题是信息源和事务的数量和位置，包括支行、自动柜员机和信用卡销售机点。

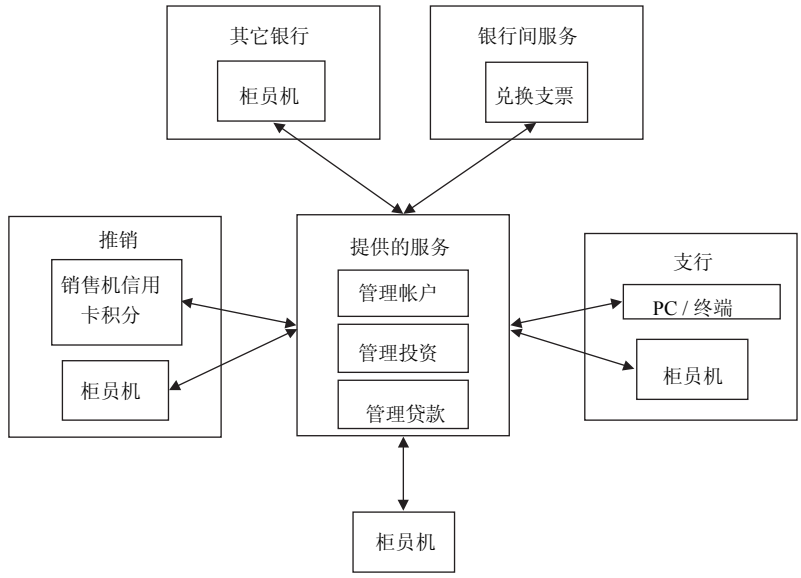


图6.6 银行例子的一种抽象模型

从性能角度看，重要的是理解系统必须能够承受可能出现的负载情况，例如事务数量和混合类型事务。显然，事务每天每时都不相同。可能存在现有基础设施带来的约束，例如运输线或其它通信机制。

接口功能

这类系统的主要接口是到其它银行的接口，以进行转帐和使用其自动柜员机。

银行还有进行清支票等工作的现有系统，这个系统是与多个银行联合创建的。

银行系统还非常可能利用外部供应商提供的电信服务。

人机交互功能

信息系统一般要与范围很广的不同类型用户打交道。对于银行来说，包括：

- 普通公众——必须能够不进行先期培训地使用自动柜员机，甚至在线网络设施——即用户界面必须直观。
- 前台员工——必须能够快速使用系统，以便向正在排队的客户提供快捷和高效的服务。这些前台员工需要培训，这种界面最重要的是，当员工经过培训之后应该能够“平稳”地使用系统。

- 各层经理——有些经理可能不像前台员工那样具有较多的计算机基础知识（当然，有些经理是从精通系统的前台员工提拔的）。针对经理提供的功能可能包括部分前台员工的功能，但是要包含更多从比单个帐户更宽的信息集合中归纳信息的功能。这些功能可能包括汇总报表、支行或地区业务数据汇总等。请注意，这类信息都要求维护一段时间，以便演绎发展变化趋势和其它历史信息，并将其表达出来。
- 政策制订者和市场开发员工——需要相当不同的能力，可能包括引入新业务产品的能力。
- 系统维护人员——必须能够更新系统功能。在理想情况下，在系统全面运行时他们也应该能够做这些工作，但是在实践中，维护人员可能要全部或部分地关闭系统（通常是在半夜很短一段时间内），以便确保数据的完整性。

安全功能

银行系统的安全性极为重要。关键要素是要保护作为业务中心的信息的完整性。

所使用的显而易见的机制包括信用卡或借记卡上的人员标识编号（PIN），以及支行、自动柜员机等之间的加密传输。

必须考虑的其它方面是，在计算机出现失效、电源故障或通信故障的情况下，也需要系统能够工作。这类功能与对风险的感知有关。对抵抗风险的程度在很大程度上取决于对风险的预见。

最后，也是最重要的，需要考虑来自黑客、盗贼或其它具有欺诈企图人员的威胁。软件必须提供足够的防护，保护银行及其客户免受这些威胁的影响。

系统事务

每类用户都可能是这类系统的 stakeholder。因此，可能对每类用户都有一组使用场景。对于银行客户来说，这些使用场景包括正常使用功能，例如取款、存款和转帐，或者人工操作，或者自动完成，例如直接借款、工资支付等。还有一些使用得比较少的其它事务，例如协商个人贷款或抵押贷款。

对每类用户来说，都值得考虑可能会引入的负载，以便能够估计响应时间。当然，响应时间并不是固定不变的，而是要取决于当前负载情况，而当前负载又取决于一天中的什么时间，以及一周内的哪一天。

必须考虑基于 Web 设施的日益增多的使用，预计其引入的额外系统负载。

运行模式

占主导地位的运行模式是正常模式。但是可能还有其它模式，包括培训、备份与恢复操作，以及系统升级。

6.2.4 汽车例子

第二个例子描述一个更物质化的系统，不过有意思的是能够发现同样类型的信息仍然存在，尽管是以完全不同的形式存在。

这个例子的最大问题是系统模型是否是物理模型，并在怎样的程度上进行抽象。新汽车不大可能会与以前车型的体系结构有很大的不同——它仍然会在每个角上有一个车轮，有发动机、变速箱、悬挂、挡风玻璃等。因此，汽车的系统模型可能相当多地直接引用体系结构物理对象，如图 6.7 所示。图中的箭头表示在箭头方向上有“某种影响”。驾驶员踩下制动板，制动板启动制动。车体和部件之间的固定连接，采用双端箭头表示，说明在两个方向上有依赖关系，例如发动机固定在车体上，车体有容纳发动机的安装位置。

但是，新车型在有些方面会有很大的差异——例如汽车电子控制系统——虽然很抽象但在确定最佳解决方案上会显示其优势。大家都能在一定程度上相当好地理解汽车功能，所需要的是对功能的量化。例如，大家都知道汽车能够把人员及其行李，或其它东西从一地送到另一地。在 stakeholder 需求中，要描述的关键问题是：

- 有多少人？
- 有多少行李？
- 汽车有多舒适？
- 汽车的行驶速度有多快？
- 汽车加速度有多大？
- 汽车成本是多少？
- 要向驾驶员提供什么信息？
- 车内提供什么娱乐设施？
- 必备安全功能有哪些？
- 汽车将在哪里使用？

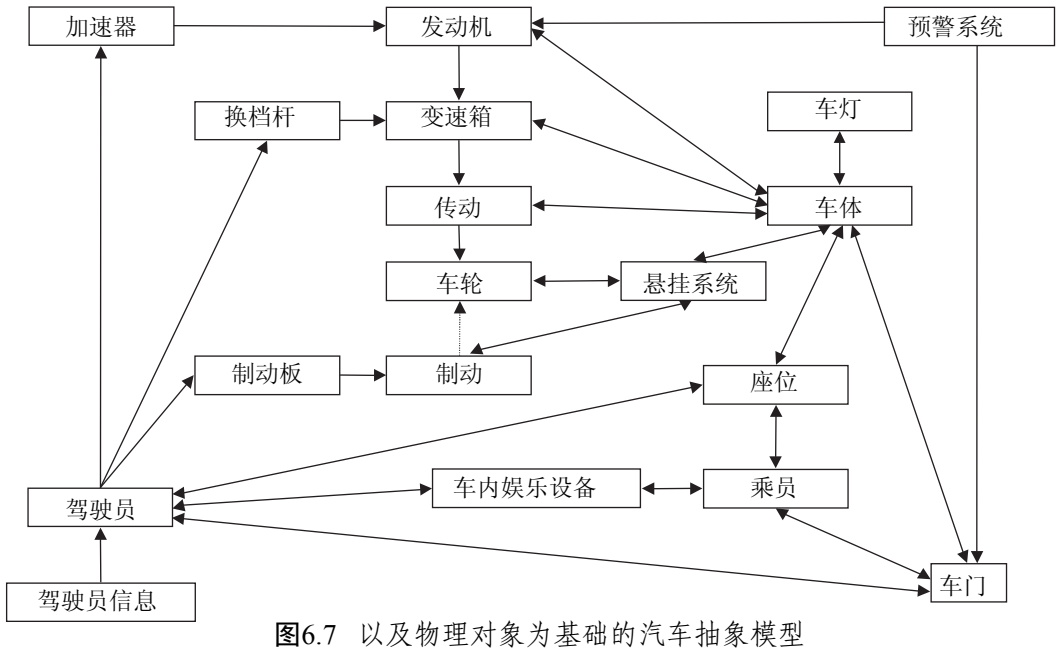


图6.7 以及物理对象为基础的汽车抽象模型

内部功能

在功能级上必须描述的关键需求包括：

- 汽车加速度。这要求在发动机动力、汽车总重量、车体阻风性和车轮牵引力之间综合平衡考虑。
- 汽车开行距离。这要求在发动机的燃料效率、燃料能力、使用手动还是自动变速箱以及汽车驱动方式之间综合平衡考虑。
- 汽车的舒适程度。这会影响汽车的成本和重量，不同体形的乘员也会得到不同的最终结果。

请注意，这些关键问题并不是独立的。这种情况在系统工程中很典型。正是这种相互影响，才推动模型向更抽象的层次发展。例如，以上因素会随所使用发动机和燃料类型的不同而有很大差别。燃料类型包括：汽油、柴油和液化丙烷气（LPG）。燃料效率和发动机重量，燃料和燃料箱，对于这三种情况都会有很大不同。因此需要确定：

- 是在这时作出选择，还是
- 先不确定所有选择，还是
- 向客户提供一种、两种或三种类型的可选选项。

设计的性质会受这些决策的很大影响。可能要评估多种选项，每种选项都比整体模型更详细。有些选项也可能在一开始就被取消，例如 LPG 燃料。

接口功能

有人可能会认为汽车在需要与其它系统接口方面是孤立的。这一点正越来越不成立。一个小例子是，汽车通常都有无线电接收机，需要兼容一定的解调标准，以便能够解码所发送的信号。

随着汽车精细程度的提高，必须遵循越来越多的标准。例如，安装了 GPS 导航系统的汽车，必须理解如何接收并解码导航系统所依赖的卫星信号。能够向驾驶员提供交通流量信息的汽车，必须能够与有关的信息供应商有接口。在未来，导航系统有可能受交通流量信息的影响，因此会引入更多（内部）接口。

对于现代汽车，所接受的服务方式是一项重要考虑。汽车常常要保存在操作使用时的事件信息，以便服务技术员能够用来诊断问题，或引导检查，或更换失效或接近可用寿命的零件。这是一个部分安装在运行系统（即汽车），部分安装在完成维护工作的汽修站的测试系统例子。

人机交互功能

汽车“用户界面”的很多问题，都要由很多年来形成的习惯确定。例如，脚踏板的相对位置（加速器在右侧，制动在左侧，如果有离合器，则在最左侧）在全世界都是一样的。

其它方面，例如左手或右手驾驶、仪表的位置、挡风玻璃雨刷，在不同地区有不同的本地习惯。

另一方面，娱乐系统、导航系统和其它不太常见的系统，还没有达成共识的约定，因此设计人员可以根据自己的选择，随便提供界面。与大多数电子系统一样，需要使界面便于需要使用系统的任何人使用（或可以使用）。这一点相当困难，因为能够提供的唯一解释就是用户指南。不可能让技术员和乘员参加培训课程，作出有关需要使用系统人员的文化程度或经验的任何假设也都是不合适的。

安全功能

汽车的主要安全功能就是确保汽车及其乘客的安全。此外，防盗的功能也变得越来越重要。

安全功能从制动系统开始。重要的是如何时刻都向驾驶员提供有效的制动能力。提供冗余能力的双路液压制动，在任何一个液压装置失效时仍然能够提供制动能力，这就是达到这种目标的一种方式。系统模型可以直接包括实现，也可以只包含对制动能力的需要。对于后一种情况，在单个液压装置失效时必须仍然能够提供制动能力这个事实，必须在模型之外增加。

请注意，以上讨论都假设制动能力是靠液压装置提供的！可以包含详细设计的一些问题，尤其是已经有了公认的详细设计，否则要根据开发机构引入到输入需求中的业务目标作出决策。

安全性其它方面还包括 ABS 制动和减缓碰撞作用的气囊。同样，这些内容可以明确地包含在模型中，也可以让设计人员在给定自由度内变换解决方案。

防盗功能从车门上的锁开始。防盗功能可以通过报警系统和发动机固定器进一步增强。这里的限制因素是额外功能引入的成本，以及客户准备为防盗功能支付多少钱。但是还有其它因素，例如竞争对手的汽车所提供的功能，以及保险公司的态度。这些不仅对必须提供的功能，而且对评价这些功能的方式都有很强的影响。

系统事务

对于汽车有很多可能的事务，所有事务都以旅途为基础，但是具有具体的目标或特征。例如：

- 驾驶员，城里购物旅行——离开停车场，开车，存车，锁车，开车锁，往车上装货，离开停车场，开车，存车，卸货，锁车。
- 驾驶员，高速公路旅行。
- 驾驶员，机场旅行（带行李）。
- 驾驶员，出现事故的旅行。
- 乘客——上车，系安全带，旅行，解安全带，下车。
- 修车店技术人员——重复提供服务，长/短间隔。
- 车主——购买，厌烦，出售/丢弃。
- 销售人员——反复尝试销售，最终售出，保修期

这些事务的每一个都可能增加新的需求，例如装载行李能力或维护设施。

因此，重要的是全面考虑所有这些事务，并理解每种事务所隐含的需求。当然，这并不意味着要满足所有这些需求。可能有些要被拒绝，因为太昂贵或被认为与所考虑的市场无关。事务也可能会导致针对不同市场创建不同模型。

操作模式

可以想象，汽车所处的主要地形会影响汽车的操作。例如在山区，变速箱会自动地选择较低档位，而且发动机管理系统会考虑空气含氧量，并据此改变汽油和空气的比例。此外，这些也可

能是选购或驾驶时可以选择的选项。

还有一种很重要的操作模式是维护模式，在这种模式下，发动机管理系统要下载所采集的信息，供维护系统和技术人员分析。

一种更极端的模式是，一组汽车在高速公路上排成“车队”，所有汽车都以相同的速度开行，间隔达到最小。这时的汽车要作为一个集体来控制，本地汽车的驾驶设施要被停用。

6.2.5 通过系统模型导出需求

创建需求文档结构

前面的讨论说明，系统模型可能由很多独立和潜在重叠的模型组成。开始可以通过任何这些模型导出需求，就像在讨论银行和汽车例子时所间接提到的那样。但是，困难的是找出一种结构，在这种结构中放入所有这些导出需求，使得每个需求在结构中都有一个明显的位置，并且所有空出的部分都是根据设计空出的，而不是偶然空出的。第 4 章把这种结构叫做“文档结构”。

建议选择一个模型作为生成文档结构的主要来源。被选择的模型应该有最宽的范围，因为系统需求必须覆盖完整的系统，而不是一小部分。在实践中，这种决策往往很明显。对于面向数据的系统，例如银行例子，数据模型常常是最佳焦点，因为所有功能都在一定程度上与建立、传播、更新或保护数据有关。对于更物质化的系统，例如汽车例子，常常最好使用根据系统物理结构（假设有这样的物理结构）导出的模型，因为大多数需求都要引用一个或多个这类要素。

导出或分配需求

一旦就结构达成共识，就可以采集已经导出的所有需求，并将其放入这种结构中。可能可以将一些输入需求直接分配给文档结构。如果是这种情况，则往往意味着输入需求过于详细，过于接近实现。

在形成需求时，第 4 章归纳的编写好文档的所有规则都应该发挥作用。请注意，最重要的规则是将每个需求都写成单一的可测试语句。在形成每个需求时，需要反向建立到新导出需求满足全部或部分的一个或多个输入需求的可跟踪性。

在考虑可测试性时，值得思考确定测试成功与否的评判准则。应该为每个需求建立这种验收准则的文档。有时准则可以嵌入到需求文本中的性能条款中。另一种方法是将准则写入伴随该需求的单独属性中。

在考虑可测试性和性能时，考虑如何组织测试，或成功实现的其它证明是至关重要的。这自然会产生鉴定策略和以大纲形式标识必要的试验和检测集合的问题。

在这种背景下，考虑所需的测试用具或特殊测试设备也很重要。这些可能要求单独的开发，在某些情况下，其自身也可能变成单独的项目。这方面的进一步考虑，是研究内置测试和提供监视点。内置测试正在变得日益重要，尤其是在与安全有关的领域。例如在汽车例子中，大多数电子系统都有在汽车发动机启动时运行的内置测试。监视点是能够提供重要信息的地方，如果没有监视点这些信息本来是不可视的。监视点的一个简单例子是汽车上的油压表。银行系统的一个信息例子是显示整个银行网络当前事务处理率的屏幕。

应该考虑的最后一组需求是约束。约束并不增加新的功能，但是要控制提供功能的方式。在系统需求层，有些约束可能直接来自 stakeholder 需求。例如，系统可以占用的空间可能是有限的，stakeholder 可能会坚持认为要把现有系统作为新系统的子系统使用。

一些其它约束来源包括：

- 设计决策——例如采用双液压制动系统的决策。
- 应用系统本身——例如，设备必须能够抵抗汽车在开行时所产生的振动。

6.2.6 与设计团队就系统需求达成共识

创建系统需求的最后一个步骤是与负责开发设计的团队就需求达成共识。这要使用第 2 章介绍的达成共识过程，这里不需要进一步的解释。

6.3 从系统需求到子系统的需求工程

创建了系统需求，下一步在逻辑上就是产生一种设计体系结构，其组件是待开发系统的主要子系统，如图 6.8 所示。通常，这个过程首先要符合客户的输入需求。必须把系统需求的评审准则，结合第 2 章和第 4 章给出的一般准则，用作符合输入需求过程的基础。需求应该独立于实现，除非有会限制设计的具体需要。对于后一种情况，这种需求必须明确地写为约束。往往会给出很多约束，因为“这是客户要求的”。对所有设计约束提出疑问永远是好的实践，特别是当约束是隐含而不是明确的时候。有时因为懒惰，以及工程师倾向于过早地过于深入细节，采用设计术语描述需求。

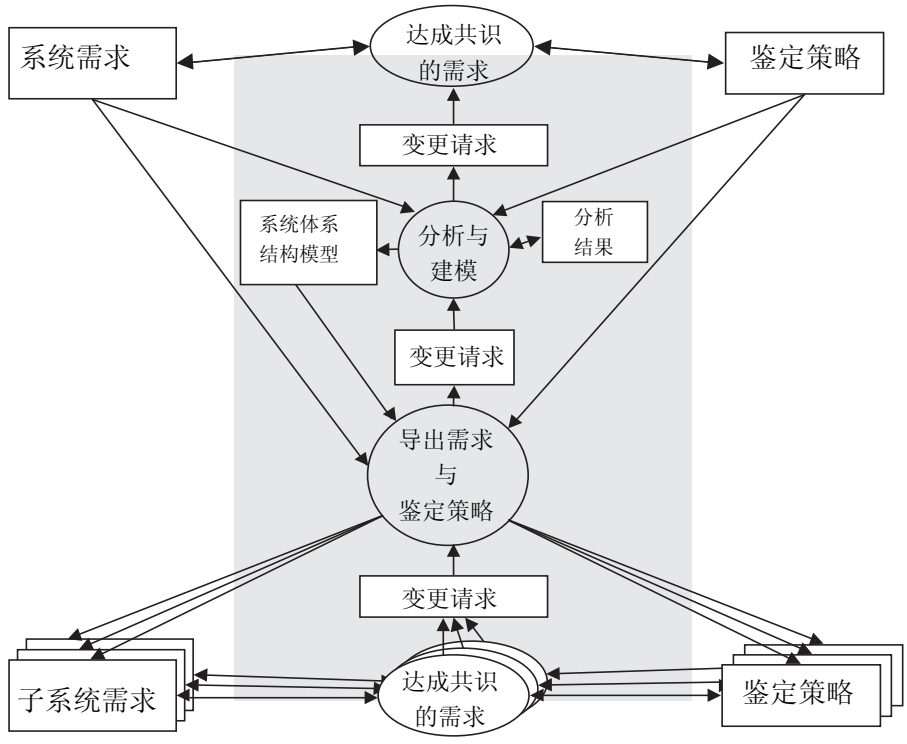


图6.8 通用过程的实例化，以通过系统需求创建子系统需求

支持达成共识过程的必要分析工作，有助于使设计人员理解什么是所期望的，使他们能够开始考虑可能的解决方案。

6.3.1 创建系统体系结构模型

体系结构模型标识系统组件以及系统组件的交互方式。设计人员必须理解组件如何协同工作以开发系统的新属性，即说明新属性如何满足输入需求。设计人员还必须能够预测是否有肯定不需要的新属性，例如灾难性失效，或其它安全或环境危害。但是，可能由给定设计所生成的新属性，虽然不是客户所要求的，但是完全可以接受。必须与客户讨论这种额外能力。客户可能会修改输入需求，以要求这些属性，客户也可能要求禁止这些属性。最终设计人员可能发现，根本不可能或不能以合理的成本满足这些需求。

只有当生成并检验了设计体系结构，这种可能性才会明确。一旦有了设计，就可以比以前更准确地预测系统的成本和开发时间。因此，可以进入与客户的新一轮协商，推敲客户提出的输入需求，就问题或成本作出必要的让步。

在很多情况下，值得考虑两种或更多其它设计，然后比较每种设计的相对优点。同样，这也会导致与客户的进一步协商（折衷），以确定成本收益比最合适的选项。

建立了达成共识的体系结构之后，每个组件都必须通过内部功能及其与其它组件和外部系统的交互义务来描述。

6.3.2 通过体系结构设计模型导出需求

通过组件描述可以导出需求。所导出的需求必须说明组件必须提供的功能，必须使用或提供的接口，以及组件必须符合的约束。这些约束可能直接来自整个系统的约束（例如所有组件都必须使用具体的电子技术），也可能通过整个系统的约束导出（例如系统的整体重量被分解到组件上）。如果将组件本身看作是一个系统，则组件（即子系统）需求本质上是对那个组件的系统需求。

因为每个需求都是导出的，因此应该反向跟踪到部分或全部满足的一个或多个输入需求。

还必须确定测试每个需求的策略。这已经不是第一次考虑可测试性了。可测试性是设计最重要的问题，必须随着设计的创建而考虑。

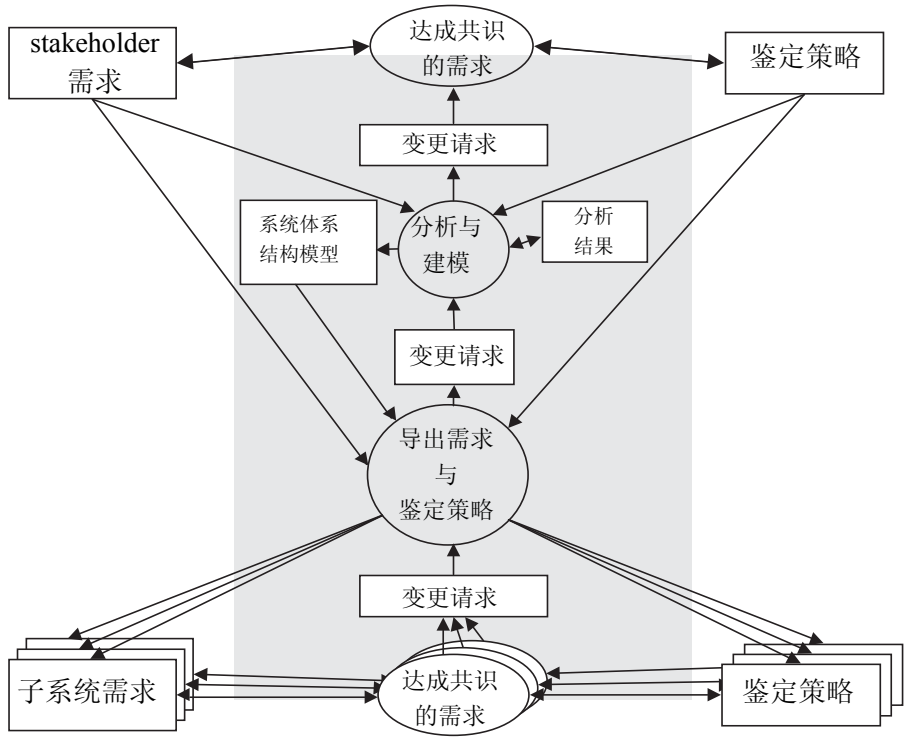


图6.9 将责任人需求直接转换到子系统需求

6.4 使用设计体系结构的其它转换

随着开发从一个层次深入到更低的层次上，每个层次都会引入自己的体系结构设计模型（如图 6.1 所示）。上一节已经介绍了在每个层次上要遵循的过程。因此创建子系统之后再往下的层次，就是创建每个子系统的组件，并层层深入下去。

有一种特殊情况，所使用的体系结构模型是这种规则的一个例外。如图 6.9 所示，设计体系结构和后续的子系统需求，都直接通过 stakeholder 需求产生。只有当事先知道系统体系结构模型时才会出现这种情况。这种情况的例子包括已经讨论过的一些物理系统，例如汽车和飞机。另外一类重要的应用系统是电信。电信系统的总体设计体系结构，必须符合控制该应用领域的电信标准。需要审议的是，这种常常直接通过标准得到的过程的输入需求，是否是真正的 stakeholder 需求，或实际上是系统需求。不管怎样解释这些需求，在转换时，通常都要相当直接地将输入需求分配到子系统需求。同样，这也意味着这类标准要在详细层次上提供需求。

6.5 小结

本章描述了解决方案领域的性质，解释了将需求工程应用于把 stakeholder 需求转换为系统需求，并进而转换为子系统需求和组件需求的方式。

本章使用了两个相当不同的例子，描述必须用来定义解决方案领域需求的功能类型。这些例子说明，除了所需的内部功能之外，必须增加额外功能，以满足外部协同系统、人机交互以及保护系统免受外部系统威胁的要求。

第7章 高级可跟踪性 7

首先，我要说明“谁？”、“什么？”、“何时？”、“何地？”，然后说明“方式？”、“出处？”和“原因？”，并详细说明“为什么？”

（《星空探索指南》，Douglas Noel Adams，作家，1952 - 2001）

7.1 引言

情况往往是，具体设计的实际依据，以及对系统组件如何协同工作得到最终结果的理解，都留在工程师的思想中。经过很多月或很多年，当最初的设计人员早已离开，或其记忆已经模糊时，这种理解的丧失可能严重妨碍系统的进化、维护或重用能力。

本章首先给出通过捕获与问题、解决方案和设计之间的关系有关的根本原因来维持对系统更好理解的手段。这种初步叫做“丰富可跟踪性”的方法，以第 1 章介绍的“基本可跟踪性”基本概念为基础，并在后续各章中使用。

丰富可跟踪性可以基本说明“为什么？”，可跟踪性的“方式？”、“出处？”和“原因？”通过本章介绍了另一个主题解决：即与可跟踪性有关的指标。

7.2 基本可跟踪性

有很多种表示多对多关系的方式。一个顾问刚好在一个国防工程承包商办公室将被可跟踪性审计员检查之前进行了访问。沿长长的楼层一侧，是摊开着的需求文档，楼层的另一侧是程序清单。可跟踪性采用文档之间的一段段绳子表示。占地、费时、不可维护、不可转换。不过也完成了部分工作。

很多工程师会看到作为相关文档的附录中采用矩阵形式表示的可跟踪性。例如，矩阵的一个方向表示用户需求，另一个方向表示系统需求，在每个有关系的格子中作出标记。

这种方法有多个缺点：

- 如果在两个方向上有大量语句，纸张或屏幕幅面会太小，不能显示足够信息；
- 可跟踪性关系一般很稀疏，使得矩阵中的大多数格子都是空的，浪费空间；
- 研究由多个单独矩阵表示多层可跟踪性是非常艰苦的工作；
- 有关可跟踪性的信息与需求本身的细节分离。

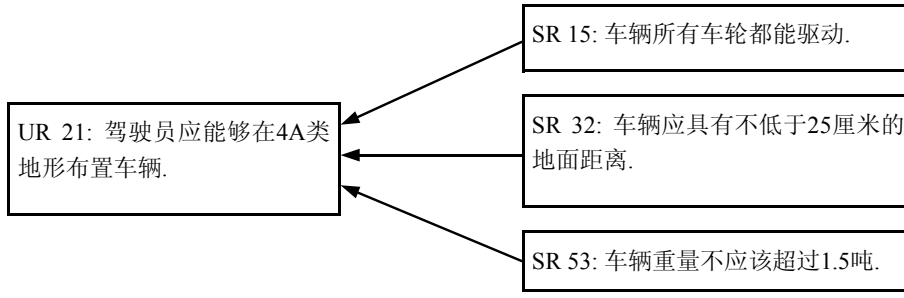


图7.1 基本可跟踪性举例：车辆

另一种方法是使用超链文档，其中的语句可以根据需要高亮显示，链接到其它语句，或贯穿语句，如果很聪明，还可以利用超链双向贯穿语句。现在可跟踪性信息在语句文本中是可视的，但是在以下方面仍然存在问题：

- 为了执行分析，可能必须使另一端文本可视，才能走通链；
- 当超链的另一端被删除时很难发现，使得链悬空，很难维护可跟踪性。

不管使用哪种方法，除非有工具支持，否则可跟踪性很难维护。

可跟踪性最简单的形式，是通过链接语句结合某种数据库支持。如果将链接信息单独放在文档中会很有帮助。重要的是语句能够被单独和唯一地标识。

记住这些分析后，可跟踪性实现的核心功能是：

- 能够创建语句之间的链，形成所允许的关系；
- 能够以受控方式删除语句之间的链；
- 能够同时观察所选择关系两端语句的文本（或其它属性）；
- 能够执行覆盖率分析，以显示被所选关系覆盖和没有覆盖的语句；
- 能够执行单层或多层影响分析，以显示被影响的语句集合；
- 能够执行单层或多层导出分析，以显示原始的语句集合；
- 能够执行向上和向下覆盖率分析，以显示所选关系覆盖和没有覆盖的语句。

图 7.1 给出了一个基本可跟踪性例子。一个用户需求跟踪到三个对应的系统需求。图中，用户需求的文本以及对应的系统需求集合都是可视的。将这两种信息结合起来，使得评审可跟踪性变得很容易。图 7.2 给出了第二个例子。

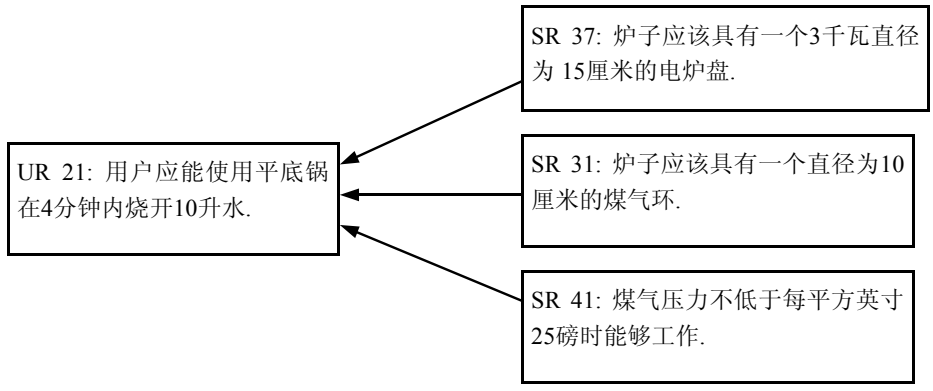


图7.2 基本可跟踪性举例：炉子

7.3 满足论据

第 7.2 节讨论的基本可跟踪性实现，对于很多机构都是一个重要发展步骤。的确，改变机构文化，采用即使是这种简单方法，其本身也可能是一大步。但是，总是还有更多可以做的。

在图 7.1 所示的例子中，三个系统需求在一定程度上充分满足了用户需求。但是，由非专家来检验这种断言是困难的，因为没有给出推理。

更好的方法是为每个用户需求提供一个“满足论据”。对于如图 7.1 所示的基本可跟踪性，所提供的唯一信息是三个系统需求在满足论据中充当某种角色，但是没有确切地指示论据是什么的东西。

丰富可跟踪性是一种捕获满足论据的方法。这种方法是用户需求和对应的系统需求之间的另一种描述，如图 7.3 所示。

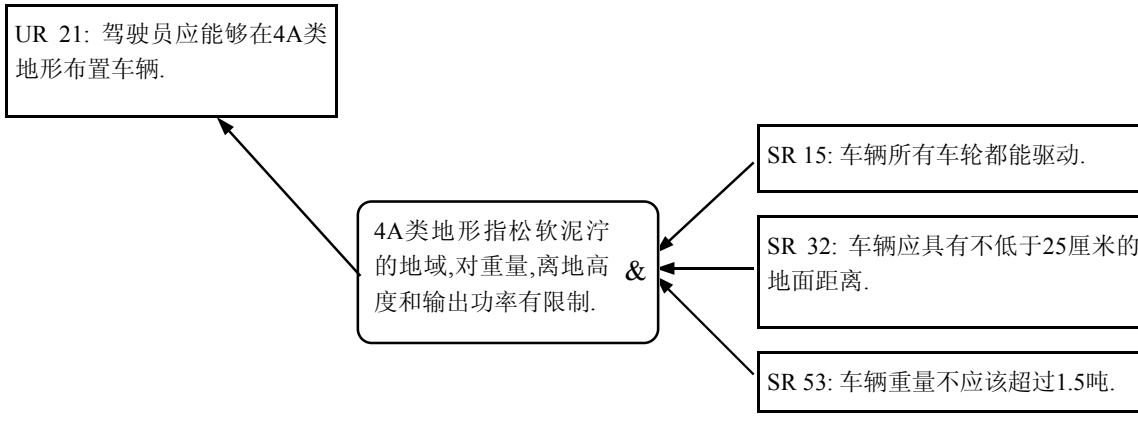


图7.3 丰富可跟踪性举例：汽车

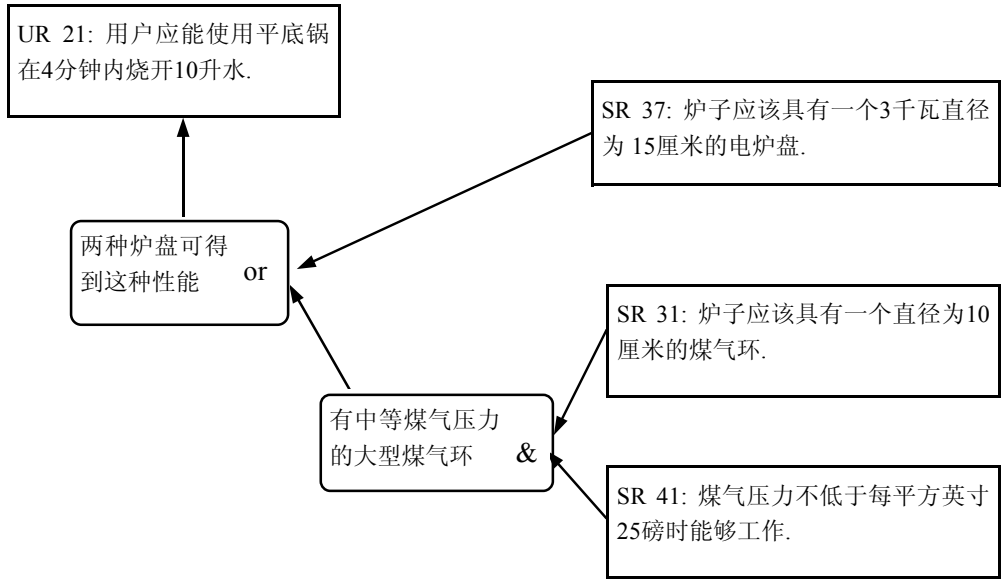


图7.4 丰富可跟踪性举例：炉子

不仅满足论据采用文本方式描述，而且还使用逻辑关系操作符描述系统需求合并的方式：

- 与（&）操作符表示用户需求满足论据成立要求有全部系统需求；
- 或（or）操作符表示用户需求满足论据成立要求有任意一个系统需求。

图 7.4 给出了一个或的例子，提供电热圈，或燃气圈，或两种圈，都可以满足。请注意论据的两层逻辑关系结构。

现在提供了越来越多的有关被满足的用户需求的信息，即使不是领域专家也能够评估论据的重要方面。文字有助于评估论据逻辑的正确与完整性。操作符使论据结构更准确。

请特别注意，对于图 7.2，系统需求集合表示替代解决方案很不清晰，而对于图 7.4，这种事实被表达得非常具体。如果不能提供电热圈，则通过燃气圈仍然可以满足需求。

作者最初接触丰富可跟踪性是通过“西海岸铁路现代化”项目，Praxis Critical System 公司的一个团队提出了一种需求管理过程和使用了“设计评价”的数据模型。相同的概念也可以在各种类似的方法中找到，其中的满足论据有不同叫法，例如“需求的详细细节”、“可跟踪性基本原理”、“策略”等。

除了低层需求之外，满足论据可能还取决于很多内容。图 7.5 给出了一个使用“领域知识”支持论据的例子。领域知识是关于现实世界的一种事实或假设，本身并不约束解决方案。在这个例子中，领域知识描述是满足论据的基本部分，用斜方框表示。

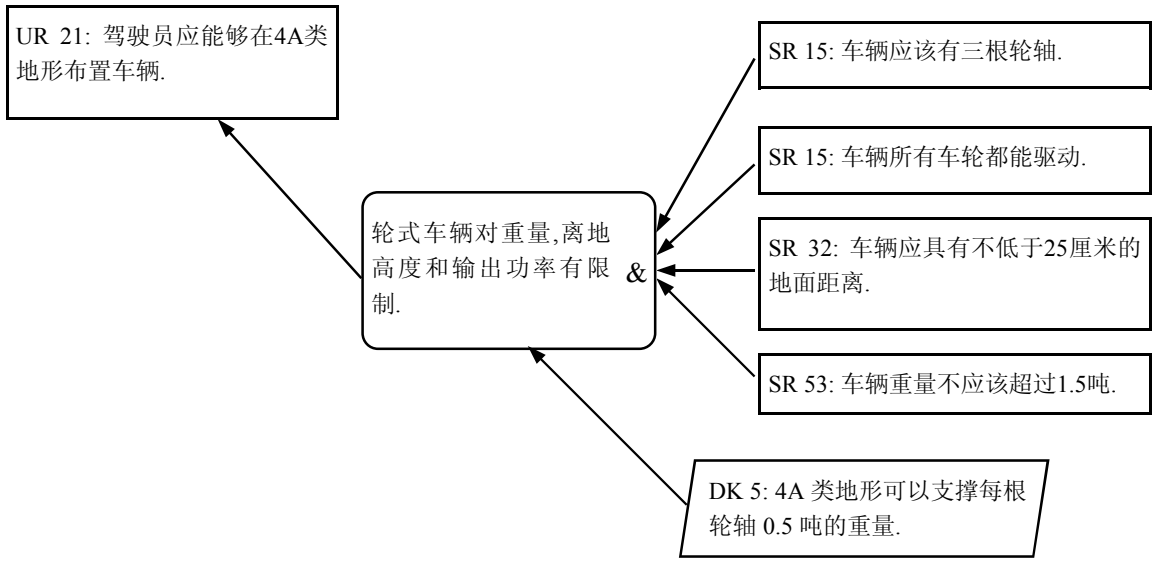


图7.5 领域知识的角色

捕获这种假设很重要，因为现实世界以及关于现实世界所作的假设都在变化。一旦捕获了这种假设，则可以使用导出分析来理解变化着的假设对系统满足其需求能力的影响。

这方面的一个例子是纽约地铁。70 年代发生了一系列事故，原因是错误地假设了列车的刹车距离。虽然最初经过确认，但是经过若干年后列车变重了，刹车距离增加了，原有假设不再有效。虽然信号系统的运行最初是正确的，但是没有进化，变化了的假设意味着从特定时间开始，信号系统不再满足需求。

通过有效的可跟踪性，可以提供记录并跟踪这种假设角色的能力。

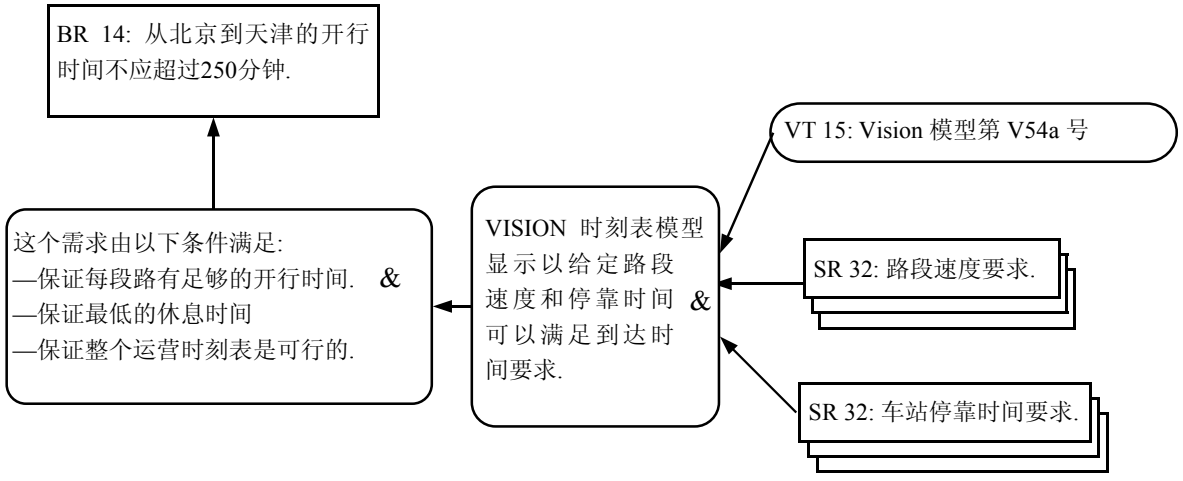


图7.6 建模的角色

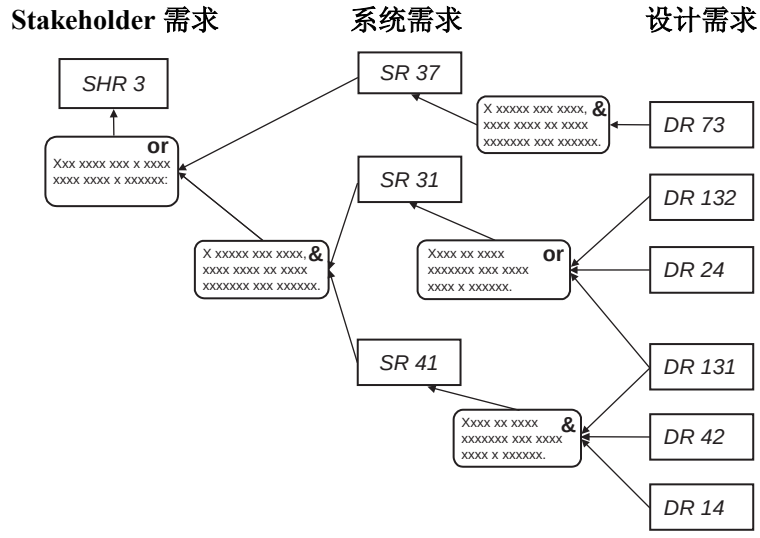


图7.7 丰富可跟踪性的多个层次

非需求信息在满足论据中起作用的另一个例子来自建模活动。满足论据常常通过复杂的建模活动中导出，其完备的细节过于详细，不能在丰富可跟踪性中捕获。

图7.6给出了来自一个铁路项目的例子，其中的满足论据取决于使用一种叫做VISION工具的复杂时间表建模活动。通过这个建模工具导出了一组假设和子系统需求，并按丰富可跟踪性结构形成文档。这个参考模型采用圆角方框表示。

在这个例子中，需要重新考虑的建模活动显然会对分析产生影响。

丰富可跟踪性当然也可以被用于多层需求或对象。图 7.7 给出了三个层次以及层次之间的可跟踪性。

7.4 需求分配

满足论据常常是琐碎的，可能只是将相同需求分配给一个或多个子系统或组件。这有时叫做需求“分配”或“向下流动”。

如果只使用纯粹的向下流动的需求，变更过程可以被简化。改变高层需求会自动流向到低层。

丰富可跟踪性的简单扩展能够捕获这种情况。表示“相同性”的一种新值被加到用于注释论据的“与”和“或”操作符上。图 7.8 给出了一个这种例子。符号“＝”用于表示相同性。

流到2个子系统的需求

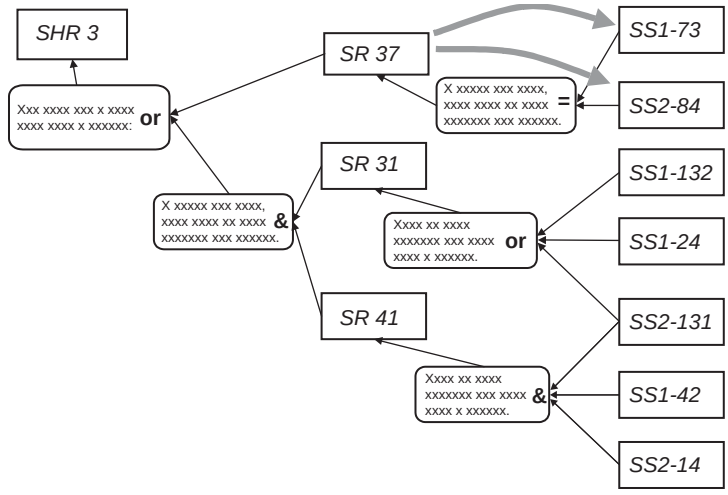


图7.8 使用“相同性”的需求流下

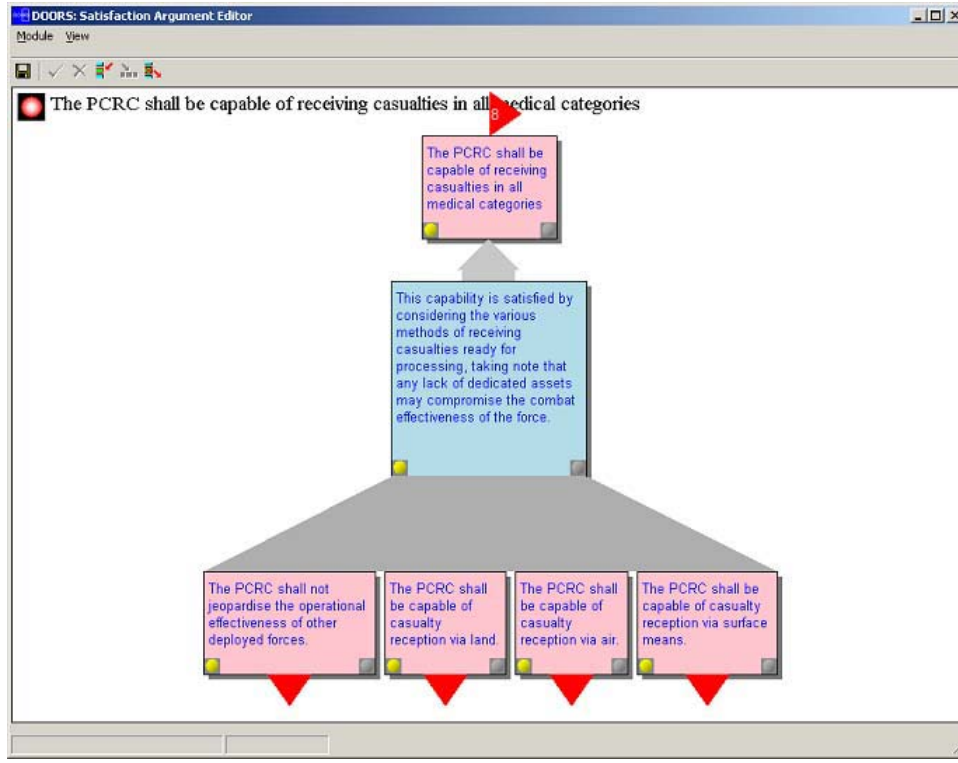


图7.9 针对满足论证的评审工具

7.5 评审可跟踪性

每次评审可跟踪性时，应该结合其满足论据进行。根据丰富可跟踪性，可以建立一次只关注一个需求及其满足论据、下层需求的评审过程。

图 7.9 给出的是在一个国防项目中使用的工具屏幕，用来评审需求和满足论据。屏幕上显示的是用来评估需求，以及该需求被满足方式的合适的信息包。

深色三角表示沿可跟踪性层次的流向，或流向相同层次的下一个需求。

7.6 满足论据的语言

与需求一样，采用某种统一方法对满足论据会有很大帮助。关键的是要用“这个需求将由……满足”这种描述模式，头脑中重点考虑的是要编写的语句种类。

虽然需求应该是严格原子化的（请参阅第 4 章），但是满足论据并不受此限制。但是，如果语句太复杂，应该采用结构化的论据。

满足论据的重复模式可以相同，在这种情况下，使用样板语句选择集会有很好的效果。

7.7 丰富可跟踪性分析

丰富可跟踪性中存在满足论据，并不排除执行第 1 章介绍的基本影响和导出分析的能力。满足论据通过所提供的信息，即对存在的理由，可对影响的性质补充重要的线索。

满足论据的逻辑结构（与、或）可为其它类型的分析提供机会。例如，可以通过分析逻辑结构，说明满足特定目标的自由度。

以图 7.4 为例。UR13 的逻辑结构可以通过表达式“SR37 or （SR31 and SR41）”获得。利用逻辑规则，将这个表达式转换为一种特殊形式，显示满足该需求的方式个数：

SR37 and (not SR31) and (not SR41)

(not SR37) and SR31 and SR41

SR37 and SR31 and SR41

在简单情况下，这种分析看起来没有什么用，但是对于有复杂交互的在多个层次的数百个需求的复杂情况，这种分析很有用。有人可能希望知道是否存在满足该需求的方式，如果没有，是什么造成其不能满足，冲突在哪里。

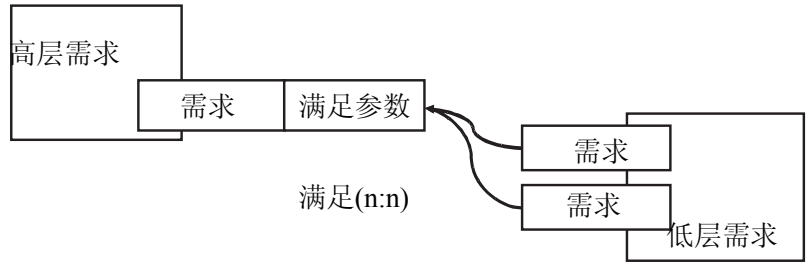


图7.10 单层丰富可跟踪性

7.8 针对鉴定的丰富可跟踪性

丰富可跟踪性可以用在任何可跟踪性关系中。到目前为止的讨论都是以满足关系为基础的，但是也适用于鉴定。在这种情况下，“满足论据”可以叫做“鉴定论据”或“鉴定推理”。使用满足论据所具有的所有优点，都适用于鉴定策略。

7.9 实现丰富可跟踪性

以下介绍实现丰富可跟踪性的两种方法：单层法和多层法。

7.9.1 单层丰富可跟踪性

如图 7.10 所示，在这种方法中，每个高层需求都把一个满足或策略语句作为属性，多个下层需求可以从这个属性流出，构成一种多对多的满足关系。使用另一个属性（在图 7.10 中没有画出）确定论据的类型是联合还是分离。

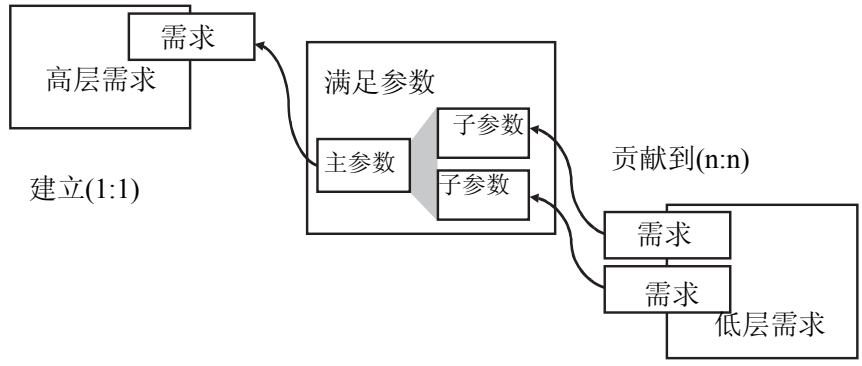


图7.11 多层丰富可跟踪性

7.9.2 多层丰富可跟踪性

这种方法将满足论据构造为多层结构：将主论据挂到（作为一个属性或连到“确定”关系中）要确定的需求上，构成从主论据引出的子论据层次结构。低层需求通过“构成”关系链到子论据上。这种结构如图 7.11 所示。

有些实现把论据层次结构的深度限制为 2，使用主论据，即满足论据，以及解释组成需求所起作用的单层子论据。

7.10 可跟踪性指标

由于可跟踪性概念在需求工程中处于如此中心的地位，因此研究什么样的过程度量可以用于需求的向下推进关系是很有意义的。

从满足关系开始，向下通过需求层次推进，可跟踪性有三个重要的方向：

- 宽度：关系向上、向下覆盖层次的程度怎样？
- 深度：关系可以向下（或向上）延伸到多少层？
- 成长度：关系通过层可以向下扩展多少？

为了便于确定这些方面的什么因素有助于度量需求工程过程，需要区分以下两类指标：

- 层的指标：与开发的单个阶段有关的度量，例如只与系统需求层有关；
- 全局指标：跨多个开发阶段的度量。

以下讨论这三个方向及其平衡问题。

7.10.1 宽度

宽度与覆盖率有关，层指标也与覆盖率有关。第 1 章已经讨论过的，覆盖率可以用来度量在单个层建立可跟踪性的过程进展。宽度度量需求在多大程度上被相邻的上层或下层所覆盖（即研究鉴定时的“两侧”）。

7.10.2 深度

深度研究从给定层次开始，可跟踪性向上或向下扩展的层次数，是一种全局指标。一种应用与确定最下层需求的来源有关。有多少组件需求实际上来自 stakeholder 需求？有多少组件需求来自设计？

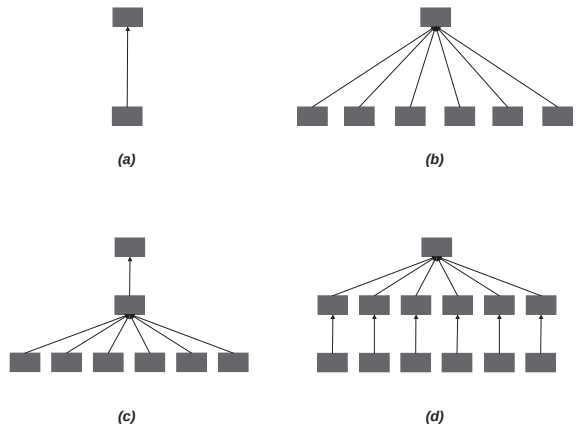


图7.12 可跟踪性成长度

7.10.3 成长度

成长度更有意义，它与潜在的变更影响有关。下层有多少需求与顶层的单个需求有关？

图 7.12 对比了四种情况。

在（a）中，单个需求被下一层的单个需求满足。成长度因数为 1。在（b）中，单一需求被六个需求满足，成长度因数为 6。这说明了两种需求之间差别的什么问题呢？存在以下可能：

- 需求（b）可能描述得很差，需要分解为多个需求；
- 需求（b）本质上比（a）复杂，因此需要特别关注；
- 变更需求（b）产生的影响比变更（a）大，因此需要特别关注。

当然，一个层次上的明显不平衡可以通过下一层解决。这种情况可以通过（c）和（d）说明，其两层以下的成长度因数是相同的。从这两个例子中可以得出什么结论呢？存在以下可能：

- （c）中的顶层需求层太高了；
- （d）中的中间需求层太低了。

只有当特定机构在开发特定类型的系统上积累了相当多的经验之后，才能够开始确定各层次之间需求的预期成长度因数。但是可以直接将检查需求之间成长度的平衡，作为标识潜在伪劣需求或不平衡在过程应用的一种手段。

7.10.4 平衡

指标的一个思想是研究两个给定层次之间个体需求的成长度因数分布，检查这种分布外层四分内之一中的个体需求。目标是标识具有异常高或低的成长度因数的需求，并进行特别关注。

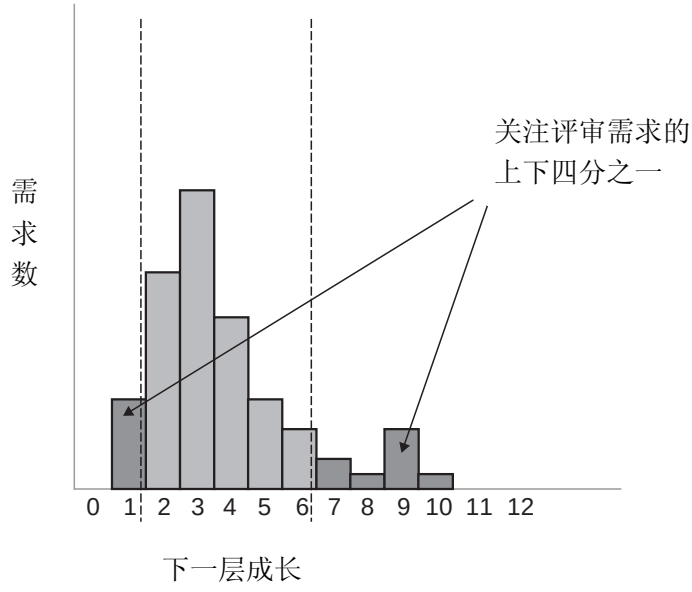


图7.13 需求成长度的频率分布

图 7.13 给出了一种典型的成长度分布情况，采用成长度速率和拥有该成长度速率的需求数表示。大多数需求都在 2 和 6 之间，少量需求只有 1，或大于 6。正是这最后一种需求需要标识并特别关注。

以上讨论的都是向下成长度，即研究从另一个需求中流出的需求数量。在另一个方向上会出现什么情况，即流入另一个需求的需求数量？

请注意，可跟踪性是一种多对多关系，如图 7.14 所示。低层的两个需求有流入其中的多个需求。对于这些需求应该如何描述？这些需求可能比其它需求更关键，因为满足多个需求，因此需要特别关注。

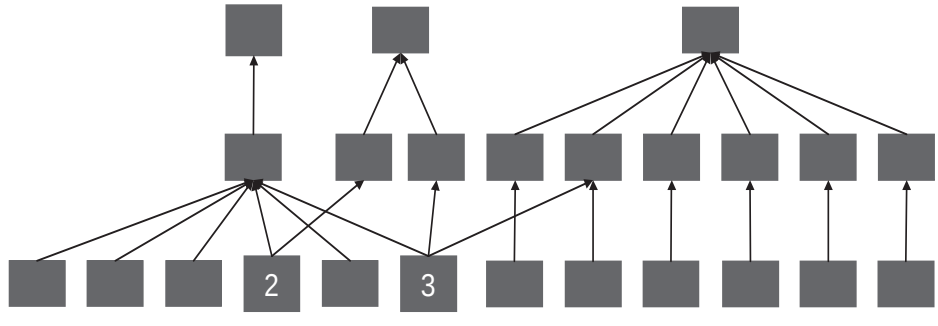


图7.14 需求的关键性

可以使用上游可跟踪性的分布选出这些需求。图 7.15 给出了这种分布的一种典型形状。

7.10.5 潜在变更

变更管理大概是最复杂的需求工程过程。第 2 章详细介绍的过程和信息模型，都利用可跟踪性来确定潜在的变更影响。当提出对一个需求作变更的请求时，所有那些跟踪到该需求的需求都要转到怀疑状态，直到工程师确定实际影响为止。

因此，提出单个变更请求，会突然引入对系统的潜在的一连串的变更。在这种情况下，非常需要进行追踪，并估计会产生的工作。

图 7.16 说明了变更影响的复杂性。提出一个针对一个最高层需求的变更请求，图（a）利用向下可跟踪性显示潜在影响。带有白圈的方框是要进行变更评估的对象。

图（b）使用向上影响表示潜在变更。出现这种情况，是因为有一个从两个高层需求流下的下层需求。需要评估这些变更的对上影响，因为下层需求中的变更，可能会引发对高层需求的重新协商。在这个例子中，所有需求突然都成了潜在的变更对象！

当然，工程师在评估实际影响时，可能会发现这些需求中的一些根本就不是变更对象，潜在变更级联也谢天谢地能够剪裁，有时可以作相当大的剪裁。

变更状态可以直接采用仍然处于怀疑状态的需求个数度量。当提出变更请求时，所有其它能够向下或向上跟踪的需求，都被标出处于怀疑状态。此后，随着对每个需求进行评估，受怀疑需

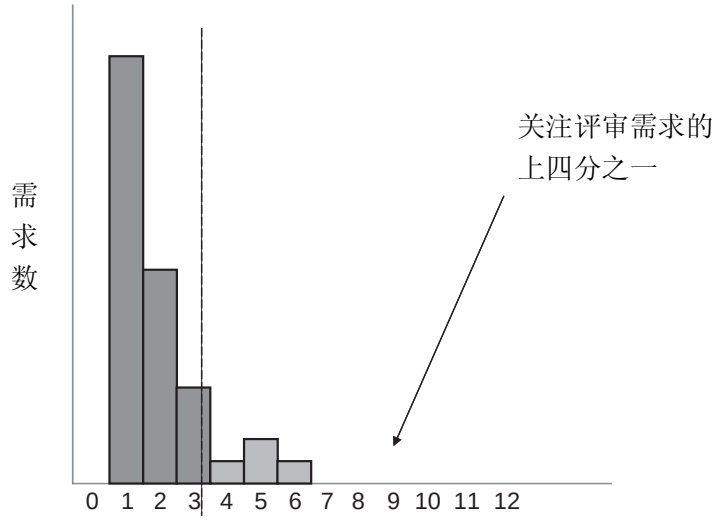


图7.15 需求关键性的频率分布

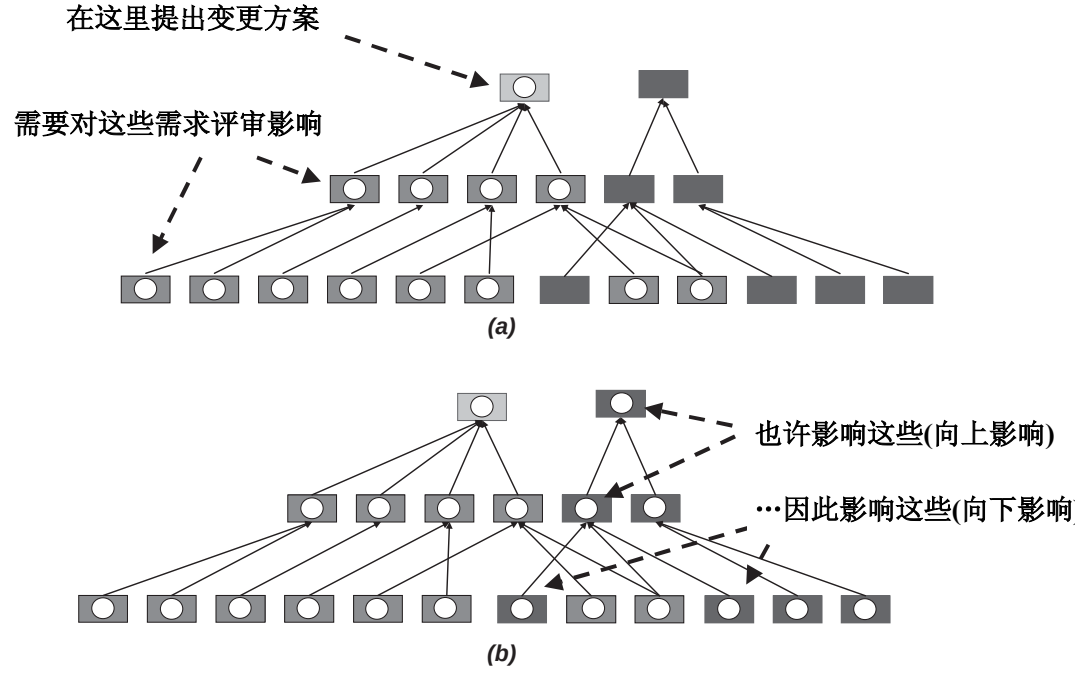


图7.16 由一个变更请求引出的潜在变更

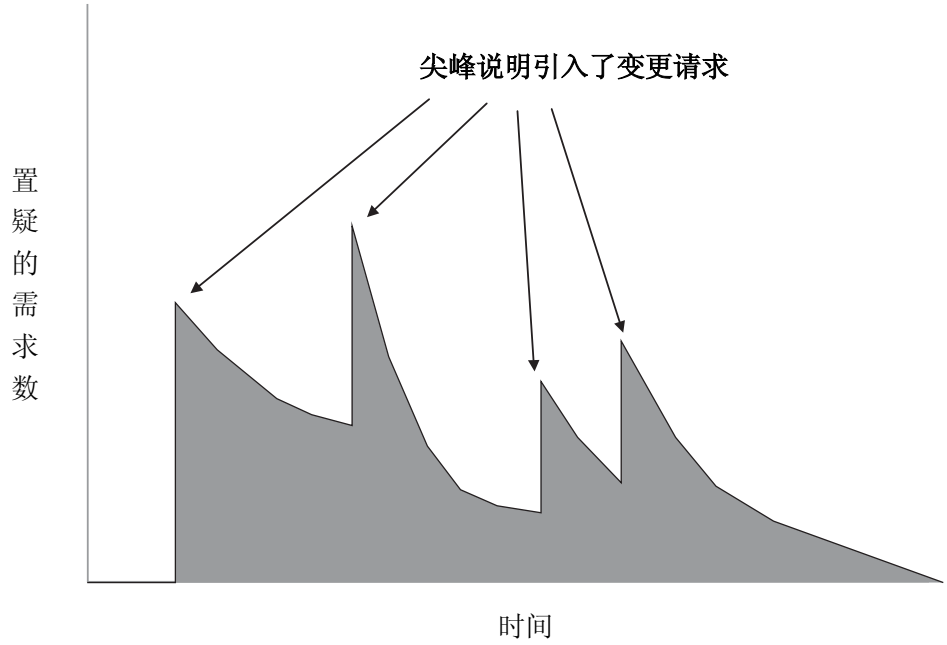


图7.17 变更处理进展

求的数量逐渐减少，其状态被复位，可能导致其它需求也被级联复位。这样，每次引入新变更时，系统中残留的变更数量都会达到峰值，然后逐渐衰减，如图 7.17 所示。

以上对变更过程的讨论，说明变更只能通过现有的链集合从需求向需求传播。但是，需求中的变更需要可跟踪性链的增加或删除。链的变化会将变更传播到通过链连接起来的两端需求。

7.11 小结

使用第一章，第 1.5 节介绍的可跟踪性的所有优势，就是通过丰富可跟踪性清晰描述来提高满足需求的信心。获得与可跟踪性有关的根本原因的原则建立了这种信心。

毫无疑问，创建完整的满足论据需要大量工作，特别是对有数以百计需求的复杂系统。

在前面提到的铁路项目中，大约有 500 个满足论据用于将高层需求分解为子系统需求。2 到 5 名工程师的团队在大约 3 年中专门从事这种信息的维护工作。

但是经验也说明，高层信心的提高会使这种投入得多更多回报。铁路项目投资方提出高层目标的能力，以及详细演示丰富可跟踪性层次如何满足这些目标的能力，是这种概念的一个主要卖点。

大家已经认识到，可跟踪性是过程度量指标的丰富来源。正是通过可跟踪性和相关过程建立的关系，才使这种度量成为可能。

第8章 需求工程的管理问题 8

在理论上，理论和实践之间并没有差别；而在实践上两者是有差别的。

（Yogi Berra，棒球运动员，1925　　）

8.1 管理简介

需求工程过程的管理类似于任何其它过程的管理。在开始之前需要理解要做什么。我们需要理解必须承担的活动种类，需要理解这些活动之间是否存在依赖关系，例如一种活动是否只能在另一种需求完成之后才能开始，需要理解执行这些活动需要什么样的技能。

准备一个计划，关注由每种活动所产生的输出，这是一种好的实践。输出可以被看到，并提供了工作已经完成或正在完成的切实证据。

根据所有这些信息，可以生成一个计划，包括已经标识的将要承担的活动、执行这些活动的人员以及由这些人员完成活动的时间。然后就可以根据这个计划展开工作，经理可以依据计划监督工作。在理想世界中，计划要严格按文字执行，不会出现差错，计划完成之日，就是所有工作完成之时。

现实可能有相当大的差距。首先，估计完成某项任务所需的时间和工作量非常困难，除非管理人员过去在处理类似任务上积累了大量经验。第二，在工作进展中会发现一些事先难以预计的困难。例如，计划出于各种原因，在特定时刻可能要依赖一个关键人员，而这个人却不能到位。这些事件会导致计划偏移，并产生修改计划的需要。一旦编制了新计划，整个过程要重复进行。变更计划经常产生的结果是几乎不可避免地要增加成本，完成时间会滞后于以前所估计的时间。另一种方法是使成本和完成时间保持不变，来减少要完成的工作量。在某些情况下，这可能是一种可行的策略。例如，可能需要公司在给定时间（以进行竞争），用给定预算（因为公司只能承担这么多），向市场推出某种新产品，不管这种产品的能力如何（虽然为了避免功能太弱，通常至少要有一种最低限制）。这种情况一般是靠商业压力推动项目的一种方式。

重要的是要认识到，任何项目都会受到以下三种因素的制约：

- 产品能力；
- 成本；
- 时间表。

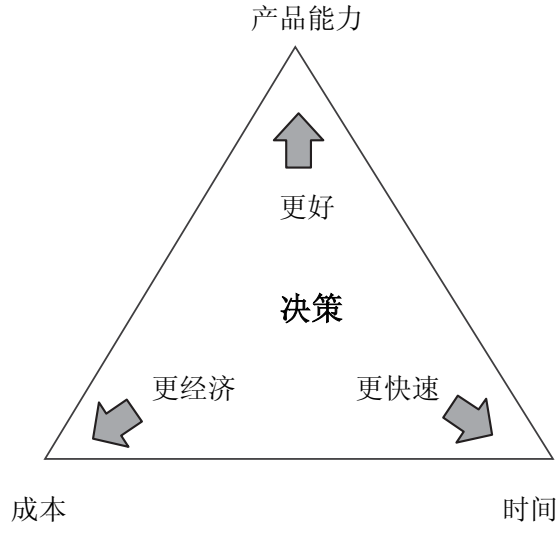


图8.1 能力、成本和时间是相互关联的

这三种因素相互关联，如图 8.1 所示。改变这些因素中的一个，最终会改变至少另外一个因素。图 8.1 还说明项目通过决策进行推动。每个决策都要确定项目在这三种基本因素中的位置。每个项目经理都梦想每个决策都会改进产品能力，同时又会降低成本，缩短开发时间。虽然这是不可能的，但是这种梦很多人还在做。

8.2 需求管理问题

本节将介绍一些使需求管理比其它管理活动更困难的具体问题。第一个问题是，非常少的人具有管理需求的丰富经验。这主要是因为非常少的机构定义了全机构都要遵循的需求管理过程。结果，面对必须说明需求的项目，人们能够利用的经验很少。这使得估计非常困难，因为产生好的估计的一个主要成份是大量的相关经验。因此，起点并不好。这使我想起一个笑话：一个人问另一个人到某个地方怎么走，回答是“我不会从这里走”！

这个问题的推论更加基本。如果没有多少需求管理经验，甚至可能不知道开发需求需要什么活动。本书前面各章已经讨论过这个问题，并可直接构成在多种环境下开发各种类型需求必需活动种类的指南。

第二个问题，是很多人都不能恰当地区分用户或 stakeholder 需求和系统需求，进而常常不能区分系统需求和设计规格说明。换句话说，他们直接进入解决方案，而不是首先定义独立于解决

方案的需求集合。这个问题也同样在前面各章作了讨论。

第三个主要问题是管理需求的方式要取决于完成需求工作的机构类型。在前面各章中，已经讨论了不同类型的需求并说明这些需求是怎样相关的。但是，应用这些过程的方式要取决于应用这些过程的机构类型。共有三种主要机构类型：

- 采购系统并使用这些系统提供运营能力的采办机构。这些机构主要关注产生并管理 stakeholder 需求，然后利用 stakeholder 需求作为验收所交付系统的基础。
- 响应来自采办机构或更高层供应商机构的采办请求的供应商机构。这些机构接收输入需求，并针对这些输入需求开发系统需求（以及后面要生产的设计）来响应这些需求。（供应商也可能会采办下层子系统或组件，但这是相当不同的采办形式，因为这种采办以设计体系结构为基础。）
- 开发并销售产品的产品公司。这些机构收集 stakeholder 需求，不过是通过市场收集，而不是个人或运营机构。市场开发部门通常执行需求采集工作。产品公司开发产品，响应 stakeholder（市场）需求，并销售所开发的产品。在一定程度上，这种机构包含采办和供应，但是完成这种作用的公司部门之间的关系，一般与标准采办和供应商关系不同。

本章稍后还要继续讨论这种机构。

使需求管理比其它管理活动更困难的第四个问题是在生成需求时监视进展相当困难。一个困难问题是要知道需求集合是否完备，以便确定是否停止活动。更糟糕的是，当活动还未完成时，存在确定进展程度的问题。这个问题由于需要评估所生成的需求质量而进一步恶化。有可能生成一个很长的需求清单，但是经理怎样评估每个需求是否都经过很好描述？经理怎样辨别每个需求是唯一的，并且每个需求都是必需的？

最后一个问题是不断变更的问题。需求管理应该主要关注变更管理。所提出的所有变更通常都与一个或更多需求相关。评估所提出变更的影响或冲突的后果常常相当困难，但是如果没有这种知识，就不能估计引入变更对成本和时间的影响。

8.2.1 需求管理问题小结

需求开发所产生的具体管理问题与以下因素有关：

- 规划；
- 监视进展；
- 控制变更。

取决于有关机构，问题会有微妙的不同。因此，本章其余部分将以前面介绍的三种机构为背景讨论每种活动，最后在小结中归纳一些共同方法。

8.3 在采办机构中管理需求

8.3.1. 计划

采办机构项目的切入点是某种形式的概念描述。概念描述最基本的形式只是一种想法，但是通常更具体，组织得也更好。这样做的理由很简单：项目必须得到机构的授权，而授权过程需要形成某种书面说明，以便支持时间和资金（资源）支出。这种说明通常是简要描述用户需要能够做什么（概念），以及说明这种能力能够使运营机构得到收益的支持证明。

概念定义的信息使项目经理能够开始作计划。由于概念定义包含“用户需要能够做什么”，因此立刻就有了系统最初的一组 stakeholder（用户），并勾画出一个或多个场景（能够做某事的能力）。

编写计划的第一步，包括标识更完备的 stakeholder 类型集合，以及更完备的场景集合，覆盖系统预期运营的整个范围，包括有用的不同运营模式。一旦确定了 stakeholder 类型数量之后，就能够制订如何获取需求的详细计划。在计划中要实例化的行动可以包括：

1 计划与每类 stakeholder 中的一个或多个面谈。需求经理负责保证进行面谈得到 stakeholder 的经理认可。认可可能取决于适当的工作规范，以及达成共识的经费预算（这样经理就不会因为其员工缺席面谈而受到批评）。需求经理还应该保证能够与关键操作员工保持联系。stakeholder 的经理不会愿意派出其最得力（有用并掌握信息）的员工参加与其短期利益无关的活动，需求经理要承担向他们说明参加这种活动价值的工作。

2 分配时间来对面谈过程形成面谈报告，并被当事人认可。

3 确定面谈策略，并与 stakeholder 沟通（stakeholder 可能会参与决策过程）。面谈策略确定每次面谈如何进行，例如是 stakeholder 根据提示自己描述场景，还是给出提议场景供其评判。

4 在面谈之前，集中所有面谈人员（但不一定很容易）并解释面谈目的会很有用。如果可以安排这种会议，则会议将提供很好的论坛，以讨论/开发用户场景，并确认所有 stakeholder 类型都已经被标识。

5 就用户场景达成共识并形成文档，这些场景要最好地反映系统在其背景条件下的用途和运行。重要的是要确保场景的面不要太窄。

6 面谈之后，可从面谈报告中提取所建议的 stakeholder 需求，并与面谈人员达成共识。

7 确定一种能够容纳所有 stakeholder 需求的结构。

8 将所有经过标识的 stakeholder 需求放入约定的结构中，并在必要时进行修改。

9 标识并记录可能的约束。有些约束是产品需求，例如几何尺寸。其它是计划约束，例如预算成本和完成时间。产品约束应该写入 stakeholder 需求规格说明中。计划约束（例如预算、进度、资源或质量）属于管理计划，并对计划活动产生影响。

10 确定是否需要补充属性支持需求文本。很多机构都有要求或只是建议使用的标准属性集合。例如：优先级、急需程度、状态、检验方法、验收准则。

11 就每个个体需求的评审和需求整体集合的评判准则达成一致。这些评判准则最好表示为评审人员使用的检查单。在理想情况下，评审准则应该尽早确定，并发放给需求编写人员，使得这些人员在开始编写之前能够了解所要求的内容。

12 定义评审过程，并将评审过程与单个需求的状态联系起来。这种过程可以归纳为状态转移图，如图 8.2 所示。图 8.2 说明，stakeholder 需求的初始状态是“已提出”。当需求管理团队评审了需求之后，则可以转为“已评审”状态。经过评审的需求可以提交给投资方团队作进一步评审，如果通过评审，则进入“已签署状态”。请注意，任何时候“活动”需求都可以被拒绝。必须为每个评审确定评审准则。

13 按照已定义的评审过程要求执行评审。

这个活动清单说明，需要作出多个决策。需求经理的责任是协调其它利益团体，例如参加评审人员、他们的经理以及系统的总投资方。

必须小心评估可能存在的计划约束，以保证计划是可行的、明智的。stakeholder 可能会要求

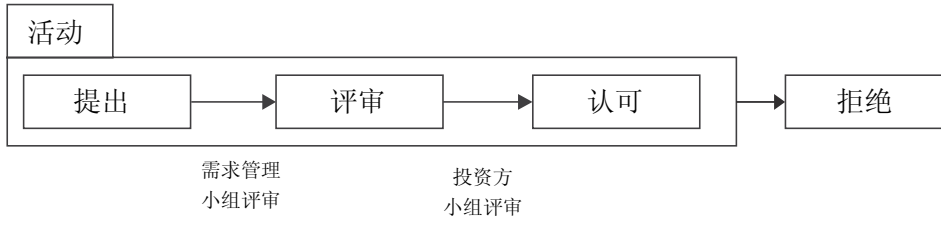


图8.2 stakeholder 需求状态状态转换图举例

系统在很短的时间内，以很低的成本投入运行，但是这也许是不可能的。不现实时间约束的一个主要例子是 90 年代初用于控制伦敦救护车的“伦敦救护车系统”。管理人员要求部署该系统，以供应政府所需的运营统计数据。这种需求被提交给项目，但是绝对不能满足。很多承包商都试图说服救护机构这种需求是不可能满足的，并要求推迟系统的部署时间。管理人员拒绝了这些请求，有很多承包商都退出了竞标。剩下的都是经验不足的承包商，他们试图满足不可能的约束。历史说明他们完全没有满足所要求的底线，并且在开发过程中对很多人造成了严重伤害。

计划的现实性是最基本的职业道德。

8.3.2 监视

一旦计划投入实施，监视就可以开始。明显的监视点是完成计划中每个活动的时候。在早期阶段中，活动主要围绕准备面谈、实施面谈以及整理面谈报告展开。这些活动的评估相当容易。

三个重要里程表有助于定义对其它过程的监视：

- 需求规格说明结构的定义；
- 每个需求所需的属性定义；
- 采用相关检查单评审过程的定义。

一旦有了结构，就可以确定每个方面是否应该有需求但是却没有。这些“漏洞”可以通过具体行动弥补。

一旦确定了属性，就可以监控填充这些属性的进展。

最后，满足评审检查单准则的进展可以通过度量处于特定状态的需求个数进行检查。

8.3.3 变更

在开发 stakeholder 需求过程中，会有一个短暂的快速而密集的变更阶段。在这个阶段，采用某种正式变更控制过程是不明智的，因为变化过于频繁，并且会造成妨碍。但是，在某个时刻会开始稳定，需求经理可以确定从什么时刻开始需求已经足够稳定，所提交的变更要进入更正式的过程中。这种阶段常常只发生在所有需求已经通过评审，并且进入“已签署”状态之后（如图 8.2 所示）。

管理变更是需求开发中的一种至关重要的活动。过程必须采用的正式程度取决于项目的开发状态。重要阶段包括：

- 开发用作竞争竞标过程基础的 stakeholder 需求；
- 签订系统开发合同；
- 设计完成而且制造就要开始；
- 验收试验正在进行；
- 系统投入使用。

这个清单以承担的任务逐渐增加的顺序排列。因此这个清单越往下，要求变更控制过程越正式，变更造成的支出越高。

不管项目处于什么阶段，变更控制过程都要求有以下步骤：

1 记录所建议的变更。

2 判断所建议变更的影响。

3 决定是否接受该变更。

4 决定什么时候实现该变更。

所建议的变更应该说明变更理由，并标识必须变更、补充或删除的 stakeholder 需求。还必须记录提出变更的个人或机构。

在第 2 步，影响取决于提出变更的阶段，并要求说明所修改的需求会怎样影响下游信息，例如系统需求、设计、制造和实际运营。

第 3 步由变更控制委员会执行。这个委员会的构成要取决于机构、系统范围以及开发或系统运营使用阶段。如果接受了变更要求，则需要执行第 4 步。该变更可能必须立即不考虑成本地实现，也可能推迟到下一个系统版本实现。任何中间策略都可能被采用，这明显取决于具体情况。

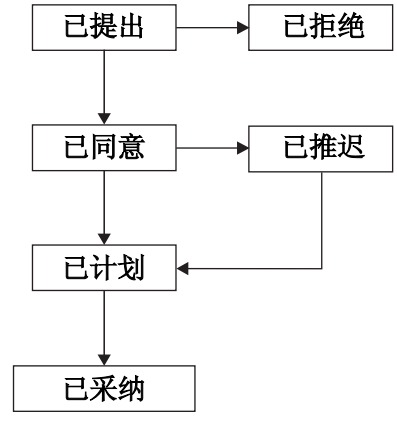


图8.3 变更控制的状态转换图

定义一组变更状态，并采用状态转移图或状态图表示，这样做总是很有用。图 8.3 给出了一个例子。

确定是否要改变变更提议的主体需求状态以表示这种变更也很重要。关于这一点至少有两个学派。一部分人认为变更和需求之间的依赖关系掌握在变更中，因此没有必要修改需求的状态。另一部分人认为，当决定采纳变更建议时，就意味着需求是变更的主体，说明其评审状态已经改变。（这就是本书第 2 章的观点。）不管持什么观点，都需要确定变更建议的状态，并确定这些状态是否对受影响需求的评审状态产生影响。

总之，采办机构主要考虑产生 stakeholder 需求。这是一种创造性过程，最初很难限定。但是随着工作的推进，随着 stakeholder 的数量和场景已经达成共识，还是可以更准确地作出计划的。

变更控制开始不太正式，但是通过开发、制造和投入运营项目逐渐成熟，变更控制会得到进化。

8.4 供应商机构

供应商机构负责根据客户的要求构建系统或系统组件。在获得构建系统的合同之前，他们必须准备一份建议，说明他们想怎样完成这件工作，包括预计开销和完成时间。通常要求多个供应商机构提出建议进行竞争。因此可以从两方面考虑供应商机构：竞标，以及竞标成功之后的合同执行。

8.4.1 竞标管理

本节研究过程管理问题，以便提出满足客户需求集合的建议。

计划

供应商机构内部的需求管理常常从收到投标邀请（ITT）开始，又叫做征求建议（REP）。这种邀请或征求会包含一组将来交付系统时必须满足的需求。

所收到的需求性质取决于客户（即发出邀请的机构）的机构类型。如果客户是采办机构，则需求很可能是 stakeholder 需求。客户也可能是另一个高层的承包商，准备进行分级向下承包。在这种情况下，需求很可能是包括附带设计约束的系统需求。为了讨论方便，我们把供应商所收到的需求叫做输入需求，不考虑需求实际上是 stakeholder 需求还是系统需求。

不管所收到的输入需求是什么，第一个任务都是进行评估，以便确定输入需求是否：

- 定义清晰，并与纯粹描述信息有区别；
- 无歧义；
- 一致；
- 没有不恰当的设计约束。

简而言之，就是要确定输入需求是否构成竞标的坚实基础。

从计划观点看，标识必须满足的需求数量很重要，因为它可以用作了解要完成工作范围的一种指标。

在输入需求评审期间，一切都必须通过标识具体问题以及建议这些问题的潜在解决方案表示出来。这类解决方案可能包括对需求表达词汇的修改建议，甚至可能包括也许通过现成商品化组件就可以满足的需求，。

一旦进行评审，所标识的问题都必须解决。通常需要与客户沟通以澄清问题，或使所建议变更得到批准。这种沟通的程度取决于邀请所附加的条件。如果邀请是针对单一供应商的，则可以立即进行沟通。

但是，如果邀请是竞争竞标的一部分，则需要更慎重一些。这通常是因为竞争规则要求一个潜在供应商的所有查询，都要复制（连同客户回答一起）给所有其它潜在供应商。因此，有可能通过询问问题，一个供应商将信息泄漏给其它竞争供应商。在这种情况下，更恰当的方式可能是找出问题和观察点，不是带着问题找客户，而是内部讨论并确定如何处理。每个问题的可能处理方法包括：

- 忽略；
- 作出某种假设并形成文档；
- 不管结果如何，决定向客户提出的必要问题。

最后一种行动可能会产生进一步的行动，以便以对竞争对手帮助最小的方式提出请求。

在整理输入需求的同时，必须进行创建建议方案的工作。显然，这种工作的主要输出，是能够提交给客户的投标书。编写投标书有很多不同方法，但是这些方法都要保证恰当地对所有输入需求进行应答。投标经理必须将每个需求分配给个人或小组，由他们写出书名回答。

所有这些回答的一致性是至关重要的，否则投标书最终就会是凌乱、互不关联的碎块。最好的方法是建立一种能够构成解决方案基础的模型。取决于投标书的性质，这种模型可以是能够成

为构建一组系统需求基础的抽象模型，也可以是概要的设计体系结构。这对输入需求提供了可跟踪性也提供了发现不一致的地方的方法。问题永远是，开发解决方案的人要以形成文档的假设，以及对客户实际含义的潜在最佳假设为基础，处理不完备的信息。不过实际情况就是这样！

到竞标阶段结束，提交投标书时，重要的是竞标小组要记录在竞标准备过程中所积累的所有信息。在指定截至日期之前，竞标小组常常面临极大压力来完成投标书的精细加工和提交。他们常常准备歇口气，可能忘记采用能够被开发小组日后使用的格式，妥当地记录所有信息。对于大型投标书，信息量有可能很大，从提交投标书到开始开发的间隔时间也可能很长（例如 6 到 8 个月）。在这些情况下，记录信息更加重要，因为开发小组中可能没有人参加竞标准备，即使有人参加，经过很长一段时间之后，也有可能忘记一些关键假设和推论。

竞标阶段一项更重要的活动是与供应商达成协议。这通常是竞标成功的一个先决条件，但是对开发解决方案的细节程度有影响。供应商机构与其供应商之间达成协议的基础，必须建立在要供应组件的一组需求基础上。竞标阶段所要求的详细程度由有关机构之间的协议决定。这将取决于机构之间的工作关系的性质，取决于经验和信任程度（有关达成共识过程的介绍，请参阅第 2 章 2.6.1 节。）

监视

在编写投标书时度量进展情况是至关重要的，因为时间通常相当有限，且提交日期是没有商讨余地的。提交的投标书最终必须清晰地说明将如何满足每个输入需求。但是，只是说明各个需求如何满足还不够，还需要检查所有说明是否有效。这是评审过程的一个方面，但是通过比较已经进入解决方案模型（进而进入系统需求或设计组件）的输入需求百分比，可以指示进展情况。

通过评估仍有未决问题的输入需求个数，加上尚未提出解决方案的输入需求个数，可以度量尚未完成的工作量。

开发解决方案中的另一个重要里程碑是创建使团队满意的模型。要保证尽快建立这种模型并使团队“买账”，是经理的一个至关重要的任务。

除了所有这些监控手段之外，还必须度量系统需求的质量。所采用的方法与前面介绍的采办机构监控编写 stakeholder 需求质量的方法类似，要定义状态，并将贯穿这些状态的进展与评审准则联系起来。

变更

在准备投标书期间，有三个潜在变更源：

- 客户；
- 供应商；
- 内部。

有读者可能认为，在准备投标书期间不会有客户变更，而在理想情况下也应该是这样。但是，为了安全起见最好还是不要做这种假设。变更的可能性一般与待开发系统（或组件）的规模大致成正比。对于很大的系统，供应商常常根据 RFP 早期草案启动竞标活动，以便使竞标小组能够沿正确方向推进和思考。这期间客户会发布最新版本，有可能包含重大变更。

收到 RFP（或其需求）新版本的第一个任务，就是确定变更的性质和程度。取决于客户和发布 RFP 所使用媒介的不同，变更位置可能突出显示，也可能完全不知道。一旦发现，这些变更必须与已经完成的工作关联，并对必要的新工作和返工进行评估。

来自客户的变更还可以通过对竞标者询问的回答获得。这些变更通常有很强的针对性，可以为评估带来方便。

来自供应商的变更更有可能发生。这些变更可能是对投标书最初请求的响应，说明其不能按所定义的方式满足需求，变更也可能出现在过程后期，供应商发现最初认为可行的内容实际上不可行。

内部变更的原因与供应商变更的原因基本相同。最初的假设实际上无效，因此必须采取其它方法。

不管变更来源是什么，使各种需求基线保持最新状态都是最重要的，即：

- 输入需求；
- 对供应商提出的需求与对输入需求的可跟踪性；
- 竞标小组内部作出的假设和解释。

8.4.2 开发

计划

项目的开始阶段是以向客户提交的，并在谈判期间修改过的投标书为依据而签订的合同为基础的。此外，在竞标过程中还会产生其它信息，但是不一定使写入投标书中的。这些信息包括详

细需求、假设、概要或详细设计信息，以及对与开发工作有关的风险的初始评估。这些信息要在估计开发工作所需时间和成本时使用。

开发阶段所涉及的活动与投标书准备或竞标阶段相比，必须考虑得更全面、更细致得多。它们之间一种很重要的差别是，开发阶段不是产生投标书，而是把以前提交的投标书作为输入需求的一部分。

在开发活动中生成的信息取决于开发性质，但是将不可避免地包括解决方案模型的创建。这个工作分两个阶段完成：首先产生一种抽象模型，然后产生一个或多个潜在的设计解决方案。如果创建了多个解决方案，则需要定义解决方案的比较评估准则，然后确定选择哪个方案。这种比较评估最终要给出选项，以及需求综合平衡的可能性。这种综合平衡可以完全在供应商机构内部进行，也可以吸收客户和供应商参加。

需要必要的活动来保证合同规格说明中的所有输入需求都被说明，保证所建议的解决方案已经嵌入到系统需求和设计中。细节程度通常需要提高，以保证在最详细层次上不会产生遗漏。

在开发阶段，重要的是要保证理解每个需求的测试手段（或演示满足每个需求的方法），并形成文档。

第一步是审计已有的信息来确定其范围和质量。在理想情况下，竞标小组产生的所有信息都应该被收集在一起并归档，可供开发过程直接使用。很多时候并不是这种情况，很多重要信息可能已丢失。在竞标小组意图和开发团队实际完成的工作之间，可能存在重大的不连续性，而这又会使机构业务面临风险。

审计完成之后，项目经理必须通过比较所提交的投标书与合同，确定从提交投标书以后出现了哪些变更。下一步要确定这些变更会带来的影响，并计划实施活动，对系统需求、设计和组件规格说明作相应的变更。

任何未确定的假设和说明都必须与客户协商，尽管在理想情况下，这些问题在合同谈判期间都应该已经解决。

在计划开发时经常出现的另一个问题是，在系统交付时是一次性供应所有功能，还是供应一系列版本，逐渐增加功能，最终交付完整版本。供应一系列版本可以尽快向客户供应初步能力，在系统的可使用性还不太确定的软件开发中，这种方法很常见。

从需求管理的观点看，必须以每个版本要实现的需求集合为基础计划版本。这种决策可以通过在每个需求上补充版本属性来记录。这种属性可以是枚举列表，也可以是布尔值。枚举列表的

一组可能取值可以是：

{TBD，版本 1，版本 2，版本 3}

其中 TBD 表示“待决定”，通常是属性的默认值。

在使用布尔属性时，每个属性的值为真或假，并为每个版本创建一个属性。

监控

监控开发中的进展应该关注对当前范围和待生成输出信息质量的评估。了解已经花费多少时间和工作量也至关重要。根据这些信息，可以估计是否能够在计划所允许的工作量和时间限度内完成任务。这种估计必须考虑经理获得信息的时间，或预期获得信息输出的速率。

如果经理发现进展落后于计划，则可以采取适当的纠正行动。这些行动不可避免地会改变计划，例如调整现有活动的时间或资源，或补充额外的活动。

监视活动必须保证项目信息是最新的。重要的一点是，输入需求和供应商需求要根据达成共识的变更依次修改，这种修改可以依据从输入需求到供应商需求及通过所建议的解决方案所存在的可跟踪性链来进行。

变更

开发阶段也有与竞标阶段一样的三个变更源。开发期间的客户变更程度一般要比竞标阶段轻得多，内部和供应商变更大致没有变化。标识任何变更的性质和结果过程都是一样的，但是这时出现的变更后果要严重得多。小变更可以在客户合同或供应商协议内部解决。但是更严重的变更可能会涉及对合同条款和条件的修改。在开发期间引入的变更通常会既影响开发进度也影响成本。一旦确定了后果，需要作出商务决策，要么接受成本和时间惩罚，要么与客户和供应商谈判。

如果变更是对采用多版本开发的产品提出的，变更管理的作用就是确定这种变更要在哪个版本中实现。

总之，供应商机构要通过准备提出一种建议来应对客户请求，如果建议成功，则可以继续开发系统。

要保证客户提出的需求是开发的坚实基础，这一点非常重要。随着变更的引入，使输入需求一直保持最新，可保证项目的基础牢固。从输入需求到建议解决方案，到供应商需求，到测试信息的可跟踪性，可保证变更的影响能够被评估，保证机构任何时候都掌握开发状态。

8.5 产品机构

产品机构定义 stakeholder 需求，并开发产品满足这些需求，因此具有采办和供应商机构的很多特征。重要差别是，供应链顶层的客户-供应商约定是在整个机构内部完成的，尽管通常由不同部门承担定义 stakeholder 需求和开发产品来满足它们。

8.5.1 计划

单一产品版本

单一产品单一版本的计划包括与采办和供应商机构一样的活动。竞标和开发阶段之间的差别可能仍然存在。例如，在开始一个新产品时，公司可能要有构建新产品涉及内容的初步设想。为此，需要提炼 stakeholder 需求并产生概要设计。

stakeholder 需求的产生与采办机构采用的方式非常类似，需要标识 stakeholder 和用户场景。但是，实际上不是与真实 stakeholder 面谈，而是有人志愿（或被邀请）担当“代理”stakeholder。这意味着他们要担当 stakeholder，从这个角度来看他们承担了定义 stakeholder 与 stakeholder 需求的角色。从计划的角度来看，这似乎没有什么差别，仍然必须标识和面谈，也必须提取需求，恰当地构成并嵌入到达成共识的结构中。最后，必须评审需求并估计其质量。

概要设计的产生与编写投标书的工作非常类似。主要差别是可以直接接触提出需求的人，因此能够进行更具有交互性的开发，可以修改 stakeholder 需求使实现更容易一些、缩短产品上市时间并降低成本。如果需求模糊或矛盾，得到澄清显然也容易得多。这听起来可能相当不正式，在有些情况下也确实是这样。不过在展开工作之前，应该就正式程度达成共识。

产生了达成共识的 stakeholder 需求和概要设计并经过产品机构评审之后，可能决定不再继续开发，也可能决定追加投资，完成更详细的设计，或开发一种早期原型。因此可以看出，产品开发可以通过一组阶段完成，每个阶段都在前一个阶段的基础上进行。每个阶段都有指定的预算和一组目标。每个阶段结束时要进行评审，对照预算和目标进行评估。这个过程可以采用如图 8.4 所示的阶段门的概念描述。

在初始门（阶段门 0）处，定义了一组目标、预算和进度。这些都要输入到计划过程中，确定必须产生的信息，以便达到该阶段的目标，以及在预算之内达到所需状态的工作计划。初始目标可以只是检验概念和市场规模的一些初步估计等。在阶段结束时，要针对目标对所完成的工作进行评审，确定是继续产品开发工作还是停止开发。这种评审还应该考虑当前业务目标，这些目

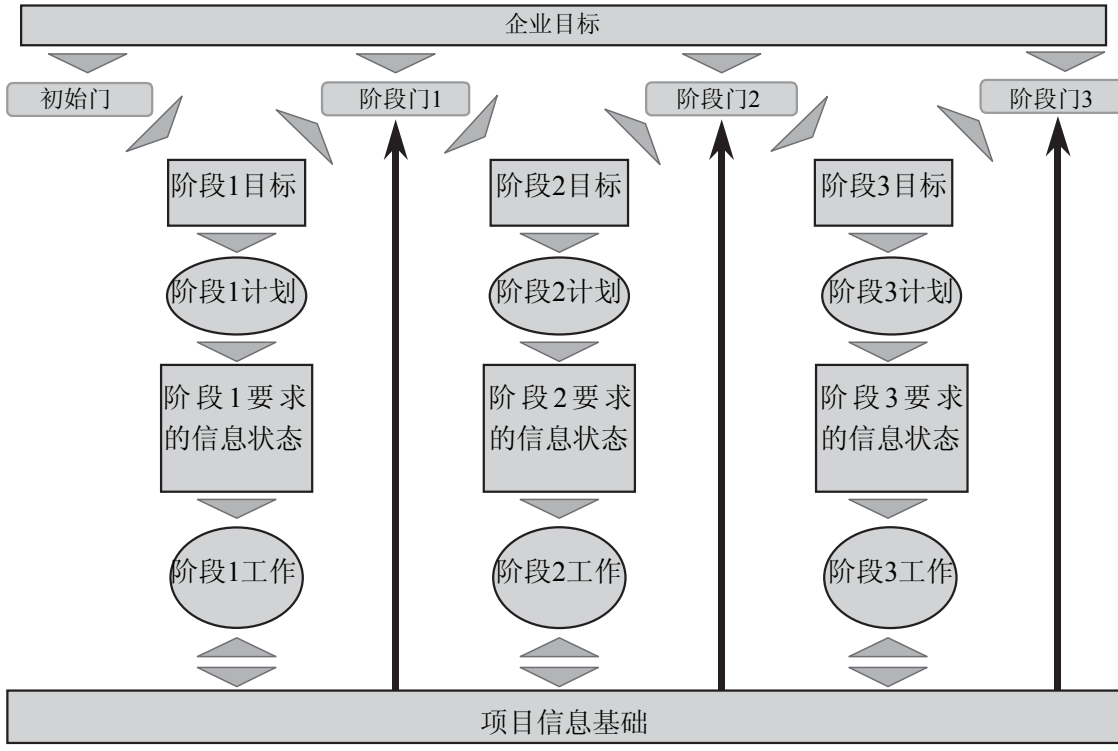


图8.4 阶段门与项目工作

标在该阶段内有可能改变或进化。

如果要继续项目开发，则需要进一步就预算、进度和目标达成共识。对于第二个阶段，可能要决定开发前面已经讨论过的成本很高的方案，并更详细地研究市场条件。阶段门评审要检查并对比所估计的成本和预期收益，很自然地作出取消或投入补充经费决定。如果是后者，则需要确定开发工作要走多远，例如：

- 对开发和产品成本作进一步调查；
- 开发原型；
- 小批量生产，并让真实客户试用；
- 投入全面生产；
- 等等。

因此，阶段门过程可以一次完成一个阶段，逐步投入经费和资源，使机构可以控制其投入策略，并关注投资的潜在回报。

多产品与版本

产品机构可以有同一个产品在不同进化阶段上的多个版本。在典型情况下，会有多个产品版本同时被购买了这些版本的用户使用，还有些版本正在开发，有些版本还在定义。从计划观点看，每个版本都可以看作是独立“项目”，要经过其自己的一组阶段和门。但是，也需要对产品不同版本有近期的计划。什么时候当前使用的版本退出使用，由更新的模型取代，作出这种计划很重要。这些问题也可以归入阶段门过程，以便能够举行一组阶段门评审，同时决定最佳投入策略，以保持或提高市场份额。

在这个方面还有针对不同市场开发不同版本的问题。例如，需要开发支持不同自然语言的不同用户界面，以便在不同国家销售。

为了应对这种差别，需要引入“变种”概念，表示“在形式或细节上与主版本有差别”。因此，可以开发用户界面是英语的产品“主版本”（也许用“核心产品”这个词表示更好），以及针对法语、德语和西班牙语市场的变种。每个变种都可以有自己的版本，使得每个版本都可以在前一个版本的基础上改进。

图 8.5 说明的是同一个产品在不同进化阶段上的平行版本和变种。字母 S、D 和 U 分别表示产品正在被描述、开发或使用。每一个状态对应于阶段门生命周期中的一个或多个阶段。

从需求管理的角度看，每个变种都会有很多与核心产品相同的需求，但是也有一些变种特有的需求，因此具有不同于其它变种的特征。另一方面，变种的每个版本可能没有不同需求，因为每个版本都试图满足同一组需求（希望随版本的提高不断改进）。

前一节曾经使用了“发布”一词，读者会对发布和版本感到不解。两者的差别是，发布是交付给客户的版本，而并不是所有版本都要交付给客户。

计划每个产品的变种及其版本的进化是进一步的组织任务，也可以使用阶段门机制进行控制。这些变种及其版本的开发在时间上可能重叠，在变种正在实际使用时，每个变种需要支持至少一个版本。

编写规格说明和开发所涉及的活动，与前面介绍过的采办和供应商机构活动非常类似。主要差别是，如果同一个产品有不同版本和变种，则会有在多种背景下都使用的公共信息。这使需求管理变得更复杂，使得理解每个版本和变种需求基线如何重叠变得非常重要。如果公共需求覆盖多个版本和变种，使变更管理（稍后讨论）进一步复杂化，则这种重叠尤其重要。

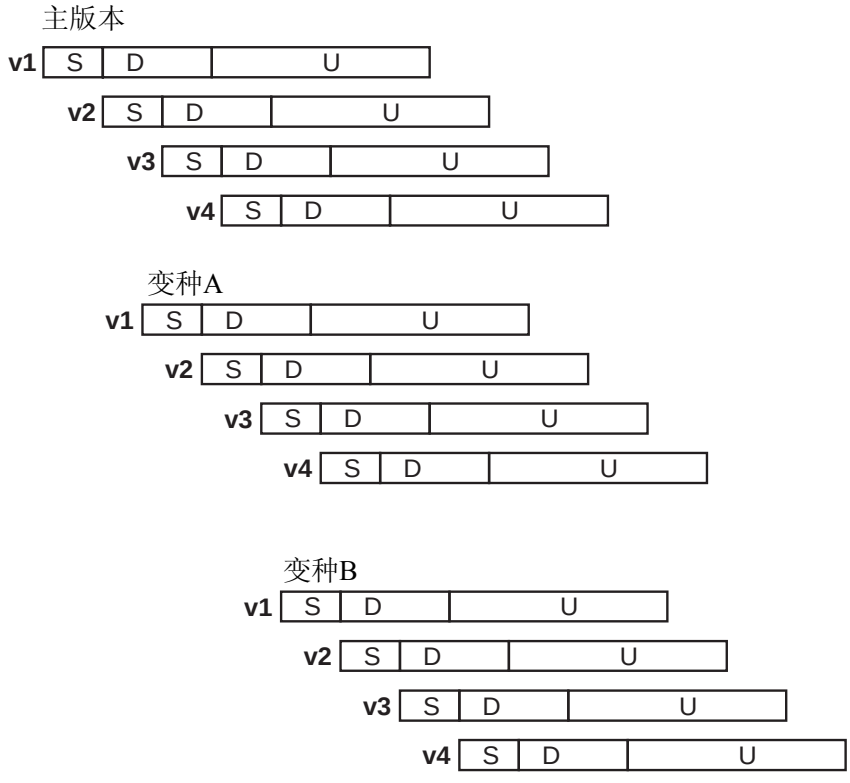


图8.5 版本与变种

8.5.2 监视

在产品机构中的进行监视所采用的机制与其它机构完全相同。如果使用阶段门作为机构决策的基础，则计划过程包括标识每个阶段结束时必须存在的数据状态。然后可以根据所需状态所达到的范围度量进展情况。一般来说，这类状态可以通过以下方式度量：

- 是否存在将成为可跟踪性链的目标的新对象（例如响应 stakeholder 需求的解决方案对象，或响应系统需求的设计对象）；
- 是否存在属性值；
- 是否存在所要求的评审状态；
- 从一个数据集到其它数据集，是否存在可跟踪性链（例如从 stakeholder 需求到系统需求，从系统需求到设计，从所有这些到测试策略和可能的测试结果）。

度量采用当前达到的所要求的数据质量百分比表示，为数据质量和一个阶段内的进展提供有用的指标。

8.5.3 变更

正如前面已经提到的，产品机构中变更管理主要增加的因素是产品的多个变种具有相同的需求，变更方案会对这种通用需求的一个或多个提出。必须回答的问题包括：

- 是否所有变种都要进行变更？
- 什么时候进行变更？

答案常常是所有变种都要进行变更，不过并不是同时进行！这要求在变更处理状态转移图中引入一个额外的状态（如图 8.6 所示），因为在变更完成之前，每个变种都必须进行变更。

图 8.6 还说明，每个变种都有已计划、已推迟和已变更状态。只有当所有变种都分别达到“已变更”状态时，变更才会达到完成状态。

总之，产品机构执行与采办机构和供应商机构类似的任务。此外，产品机构还必须控制其产品族，以便作出合适程度的投入，使在商业上的总体付出是可接受的。

8.6 小结

以下按计划、监视和变更顺序，依次归纳正文主要讨论的问题。

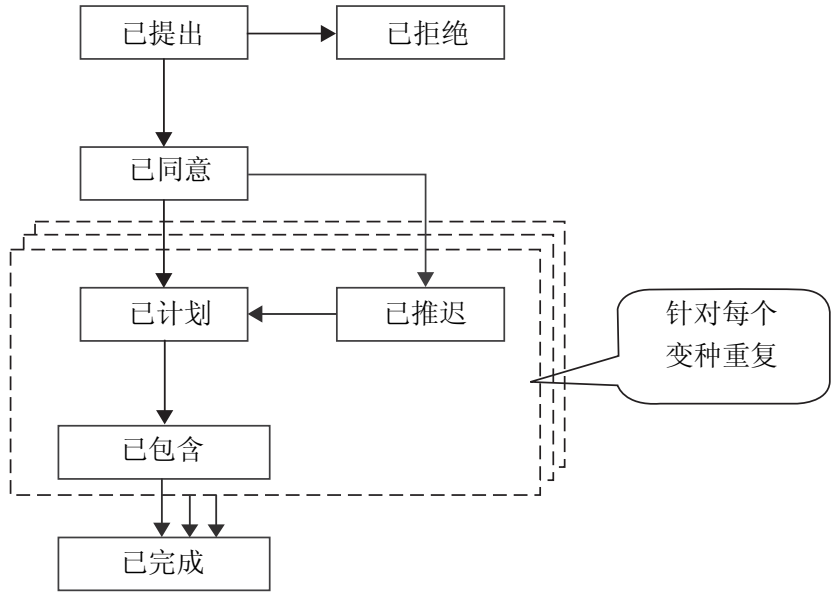


图8.6 针对变种变更管理的经过修改的状态转换图

8.6.1 计划

计划应该由必须产生的输出驱动，然后引入创建所需输出的活动。输出可以分为：

- 信息对象的类型（例如 stakeholder、stakeholder 需求、系统需求、设计或解决方案对象等;
- 与信息对象关联的属性；
- 信息对象之间的链，建立可跟踪性的满足依据、测试策略等；
- 确定信息和相关属性所需质量的评审准则；
- 可能通过一系列阶段（例如通过评审）渐进达到的特定状态。

在能够开始任何工作之前，都必须由承担该工作的团队所在机构对工作授权。对于采办和产品机构来说，采用某种像阶段门这样的机制是合适的，可以控制投入力度以及所能承担的经济和商业上的付出。在供应商机构中，必须有一个方案准备领导机构，通常还会确定预算限制。开发进展的要求通常包含在与客户签署的合同中。

可以考虑采用渐进式开发方法，特别是以前没有开发过的系统。这自然会产生发布、版本和变种概念。

8.6.2 监视

通过调查所需输出的当前状态来度量进展情况至关重要。通过这种方式进行的进展情况度量并结合与计划进行对比的工作量和时间投入，使作出可发展的计划成为可能。不考虑输出，只度量所耗费的时间和工作量，会产生不实际的扭曲观点。

8.6.3 变更

变更最为关键的问题是对将被开发系统的影响，以及由此对开发计划的影响。只有在获取了（项目）输出的当前状态的最新信息后，才能得到对影响的认识。提供从输入信息到导出信息之间的可跟踪性链，在这里具有特别重要的意义。

确定什么时候可以或应该实现变更，通常会影响计划，根据变更的范围大小，也可能产生严重的计划返工。变更还会导致引入额外的发布、版本或变种。

第9章 DOORS：用于管理需求的工具 9

并没有什么不同寻常的，所要做的，只不过就是在合适的时间敲击合适的琴键，琴自己会发出声音。

（Johannn Sebastian Bach，作曲家，1685 - 1750）

9.1 引言

系统工程师和管理人员需要合适的工具来辅助完成需求管理过程。现在已经有各种各样的工具问世。本章概要介绍这些工具中的一个，即 DOORS（第 6.0a 版）。DOORS（面向动态对象的需求系统）是一种领先的需求管理工具，已经被全世界数以万计的工程师使用。这个工具最初是由 Oxford 的 QSS 公司开发的，现在由 Telelogic 公司负责产品开发和市场推广。

DOORS 是一种跨平台的、企业级的需求管理工具，用于捕获、连接、跟踪、分析和管理各类信息，以确保项目符合所描述需求和标准。DOORS 为满足业务需要的沟通提供了手段，使不同功能小组可以协同开发项目，使用户可以检验所构建系统是否正确及是否是以合适的方式构建的。DOORS 在屏幕上提供的视图户提供了强大且易用的漫游机制。

9.2 需求管理案例

今天的系统工程师都需要有效的需求管理，以便提供解决方案。需求管理是捕获取、跟踪和管理 stakeholder 需求，以及在项目整个生命周期中出现的变更的过程。随着产品变得越来越复杂，已经几乎没有人能够理解整个产品，也没有人能够理解全部的组成部件。结构化目前是组织需求的最佳方式，使需求的信息遗漏和冗余具有更好的可管理性。因此，需求管理也就是沟通。所以需求的正确沟通是很重要的，这样可以加强团队协同，降低项目风险，使产品满足业务目标的要求。如果能够很好地管理需求，就可以及时、在预算限度内、按规格说明向市场提供合适的产品。

9.3 DOORS 的体系结构

对于任何应用系统来说，需求和相关信息都可以存储在 DOORS 的中心数据库中。这种数据库可以通过各种方式访问，并在应用系统的整个生命周期内存在。DOORS 数据库中的信息存储在模块中。通过文件夹和项目可以在数据库中组织模块。项目是一种特殊的文件夹，包含特定项

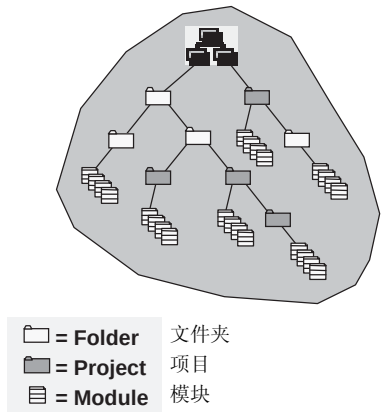


图9.1 DOORS数据库结构

目的所有数据（如图 9.1 所示）。

DOORS 的文件夹用于组织数据，就像计算机文件存储中的文件夹一样。文件夹可以包含其它文件夹、项目或模块。文件夹具有名称和相应的描述，用户观察和操作文件夹中数据的能力可以通过对访问权限的控制来限制。

DOORS 的项目是供团队人员管理与团队工作量有关的数据的。项目应该包含与应用系统的需求、设计、开发、测试、生产和维护有关的所有数据。项目提供管理用户及其对项目中数据的访问的能力，提供备份数据和将一部分数据分布到其它 DOORS 数据库的能力。

DOORS 的模块是数据集的容器。一共有三类模块：

- 正式模块——最常使用的模块类型，用于存储类似信息的结构化集合。
- 描述模块——非结构化源信息（信件或面谈记录）。
- 连接模块——包含其它数据元素之间的关系。

用户界面非常类似 Windows Explorer，使用户可以在数据库中漫游。

9.4 项目、模块与对象

9.4.1 DOORS 数据库窗口

DOORS 数据库窗口使用户能够观察和管理 DOORS 数据的组织方式。图 9.2 给出了数据库窗口，左侧是数据库浏览器，右侧是所选文件夹的内容列表。

DOORS 提供了变更现有文件夹和项目的名称或描述的能力。如果需要重新组织或修改数据库的结构，还可以移动文件夹和项目。还可以在数据库中剪裁、复制或粘贴文件夹和项目，以重新组织或复制数据库的一部分。

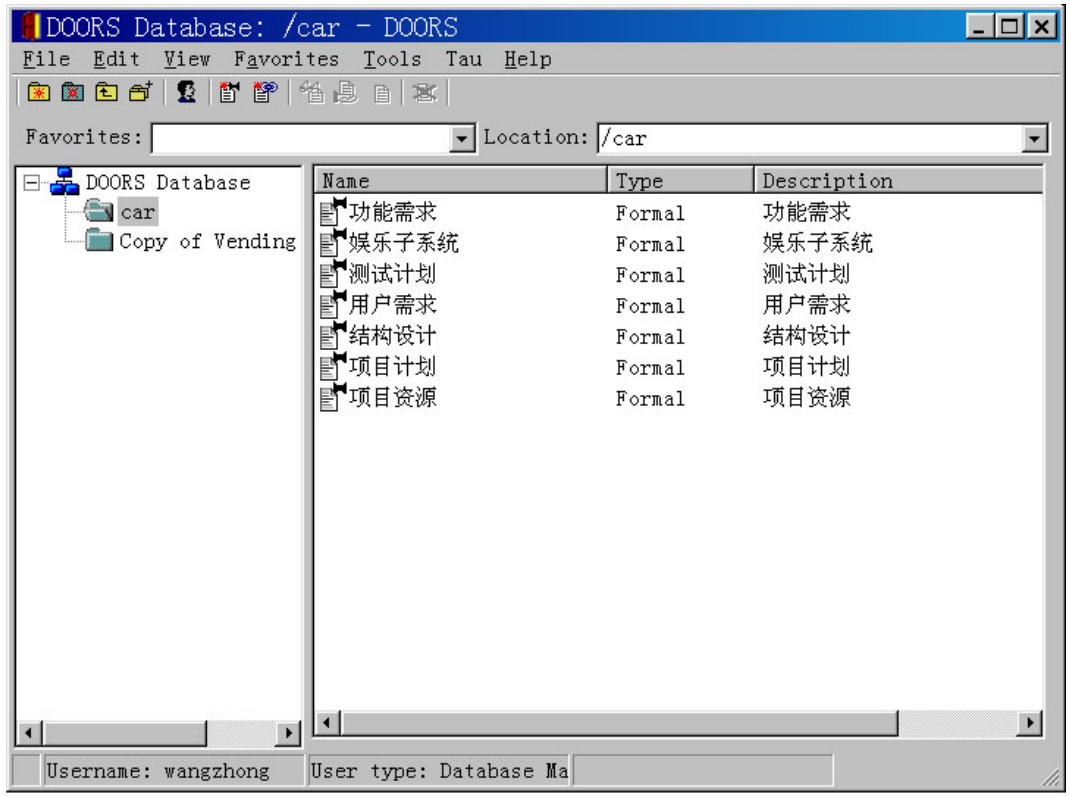


图9.2 项目视图

9.4.2 正式模块

使用 DOORS 数据库窗口，可以通过菜单 **File → New → Formal Module** 创建新的正式模块，如图 9. 3 所示。可以输入新模块的名称及其描述。模块中为对象创建的唯一标识符是采用合法编号系统命名的数字。

这个数字可以加前缀，以反映模块的内容，例如用 PR 代表产品需求。通过为每个模块定义唯一前缀，可建立 DOORS 项目中所有信息在项目范围内的唯一标识符。这可以提供一种方便的引用。

当正式模块被打开时，默认状态是左侧显示模块浏览器，右侧显示模块数据，如图 9.4 所示。

模块浏览器可以很方便地移到文档的特定位置，并且还可以显示模块中的信息结构。信息节可以与 Windows Explorer 一样进行展开或收缩。

右侧部分显示的是模块数据。默认显示有两列，即“ID”列和“text”列，其标题是模块描述。ID 是在创建对象时，由 DOORS 指派的唯一标识符。DOORS 使用这种标识符跟踪对象，以及与对象关联的任何其它信息，例如属性和链。文本列像文档那样地显示数据，显示组合标题数字、标题本身和与每个与需求关联的文本。

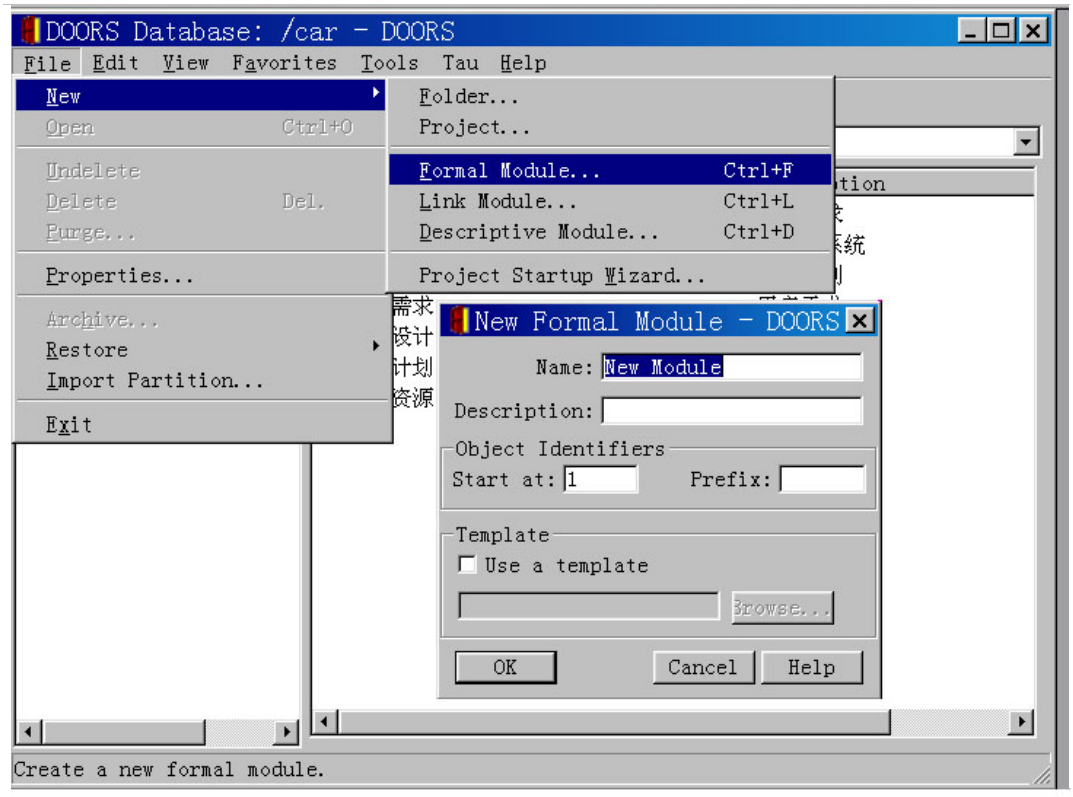


图9.3 创建新的形式模块

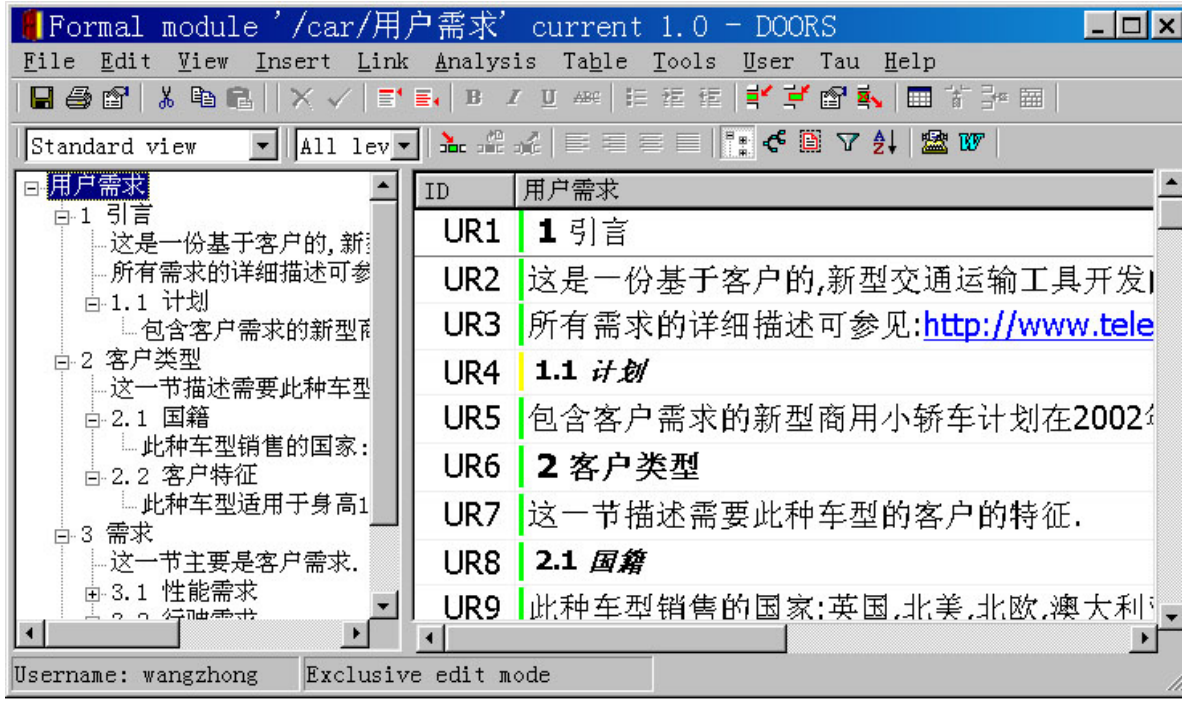


图9.4 形式模块的默认显示

DOORS 提供了多种正式模块的显示选项，如图 9.5 所示。在“标准视图”中，所有层次的对象都采用“文档”格式显示。用户可以限制显示层次。例如，“大纲”只显示标题，隐藏所有其它对象细节。这种结果类似于典型文档的“目录”。前面已经介绍过，“浏览器视图”对于观察模块结构，漫游模块中的特定对象很有用。

另一方面，“图形”模式采用树的形式显示，有助于漫游大型数据集。图形模式中的对象名称以“对象标题”和“对象文本”的压缩版本为基础。如图 9.6 所示。

9.4.3 对象

正如上一节所讨论的那样，在正式模块内部，数据是存储在对象中的。对象可以是文本块、图形图像，甚至可以是使用其它软件包创建的电子表格。正式模块的标准视图包括两列，以及一些以下要讨论的可视标识。

如图 9.7 所示，第一列显示由 DOORS 指派的“对象标识符”。对象标识符由两部分组成：

- 前缀（一般是需求集合的缩写）；
- 由 DOORS 提供的绝对编号。

绝对编号是按顺序指派的一个整数（1、2、3 等），用作每个对象的关键字，在模块内部是唯一的。

第二列叫做“主列”或“文本”列，根据内容的不同，可合成表示三个属性：

- 对象编号（例如 1、2.1、3.2.3），表示对象在模块结构中的位置；
- 对象标题提供对象的名称；
- 对象文本对对象进行完整描述。

对于指派了对象标题的对象，对象编号是唯一要显示的。

对象上下的黑线表示当前对象。很多与 DOORS 模块中对象相关的功能，例如插入新对象、粘贴对象、移动对象，都要相对当前对象执行。

文本列的左侧显示绿色、黄色和红色“变更条”。绿色变更条表示自从最新模块基线以来没有发生变化，黄色变更条表示自基线以来保存了变更，红色变更条表示在当前会话中作出的没有保存的变更。

茶色和橙色“连接指示器”显示在对象的右侧，与其它对象有关系。指向左侧的橙色三角形表示输入链，指向右侧的茶色三角形表示输出链。

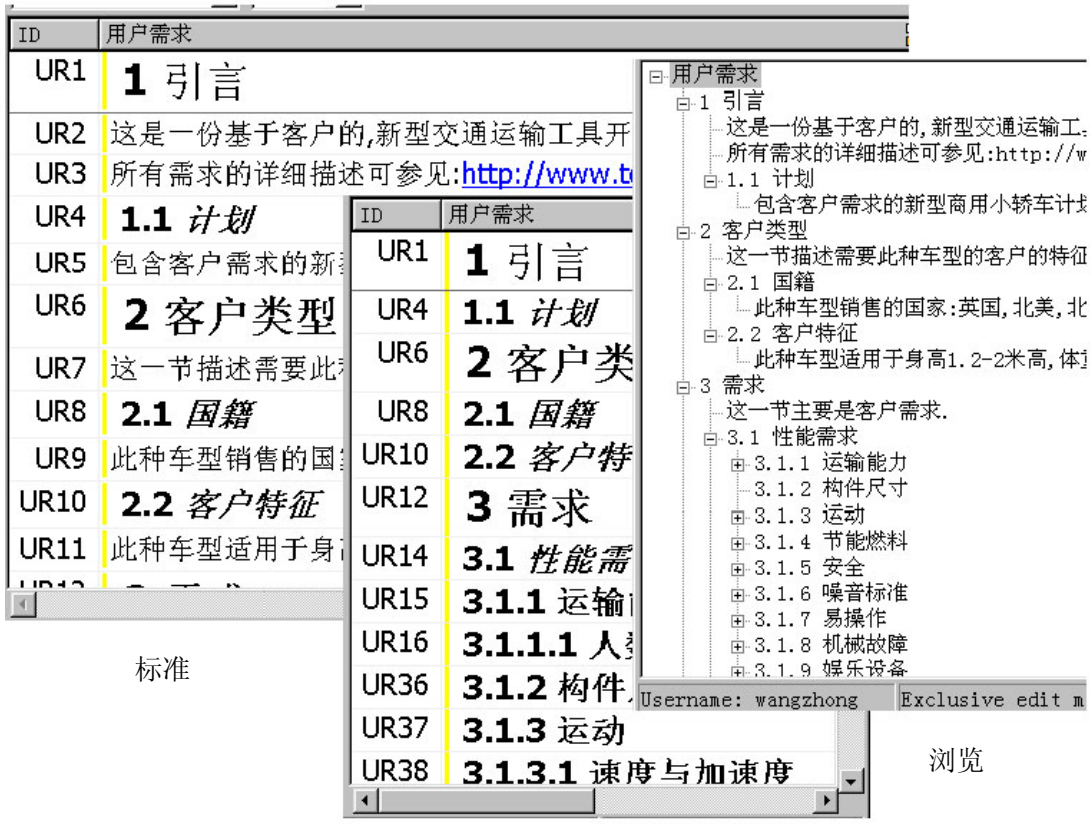


图9.5 形式模块显示模式

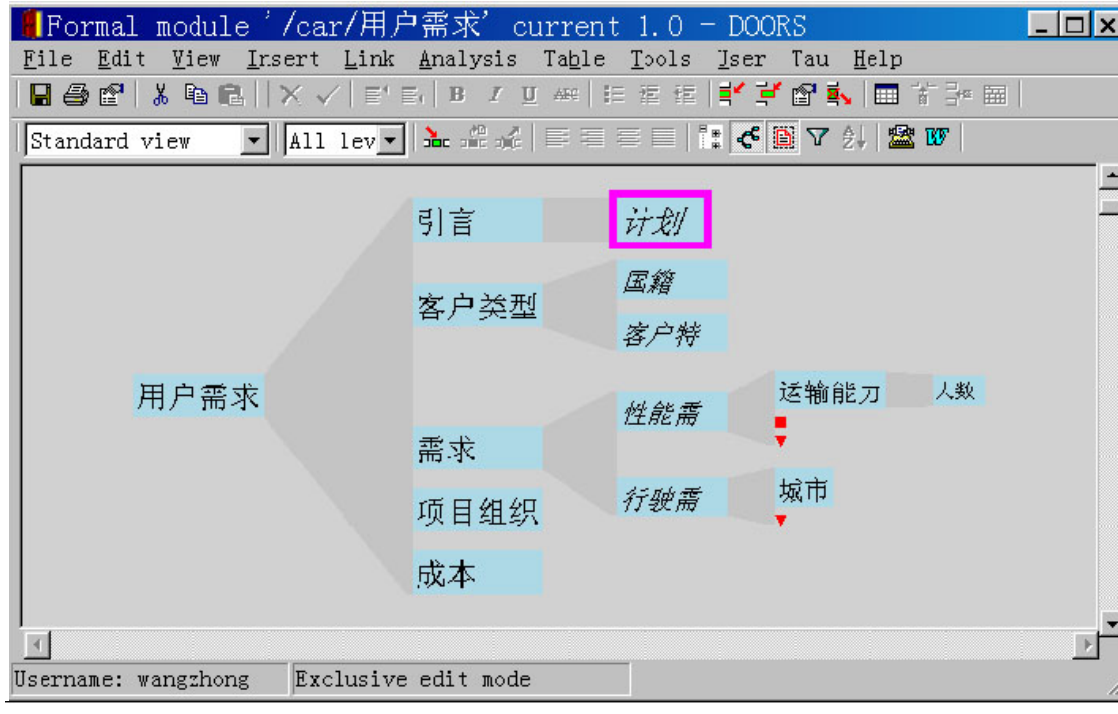


图9.6 图形模式

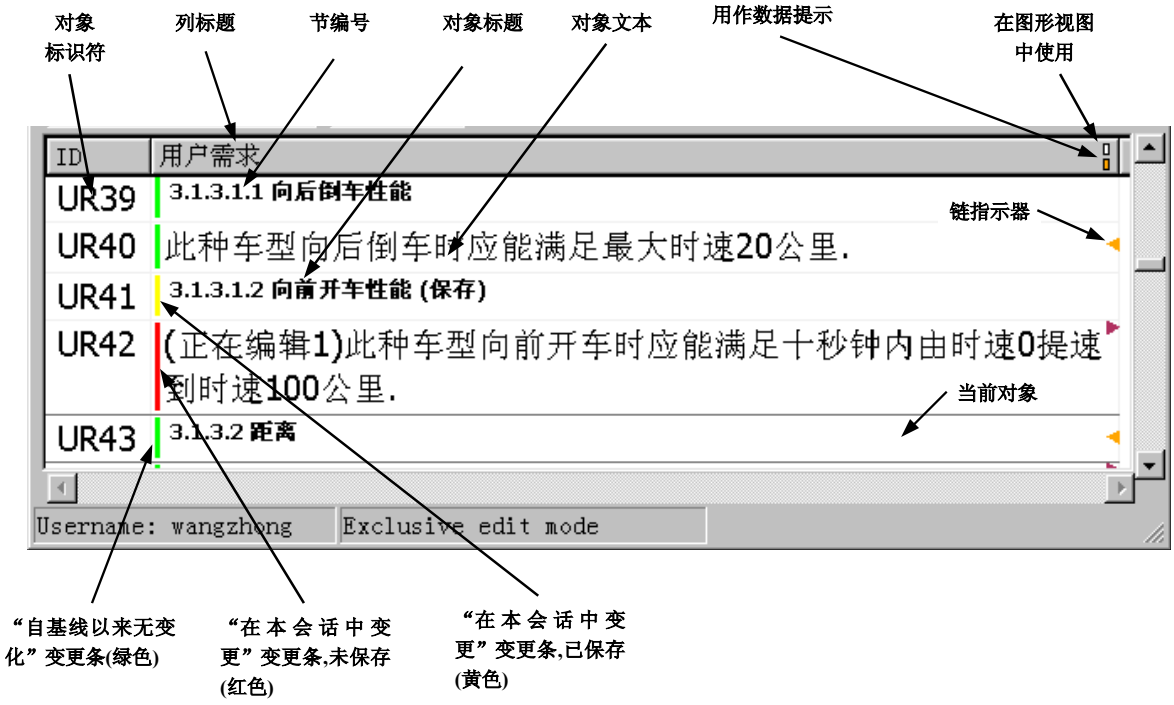


图 9.7 所显示的信息

DOORS 正式模块树的结构提供了简单但功能很强的需求编写方法。由于需求常常分层组织，所以图形模式是一种很有用的视图。

在 DOORS 中创建新对象很简单，新对象放在与当前对象相关的两个位置之一：

- 采用 **Insert → Object**，在当前对象的同级创建新对象；
- 采用 **Insert → Object Below**，在当前对象的下级创建新对象。

这种情况如图 9.8 所示。

编辑对象的层次也很方便。例如，通过剪裁和粘贴功能可以修改 DOORS 树。剪裁操作从模块显示中删除当前对象及其所有子对象。这会导致对象层次结构的重新排列，将层次结构树压缩

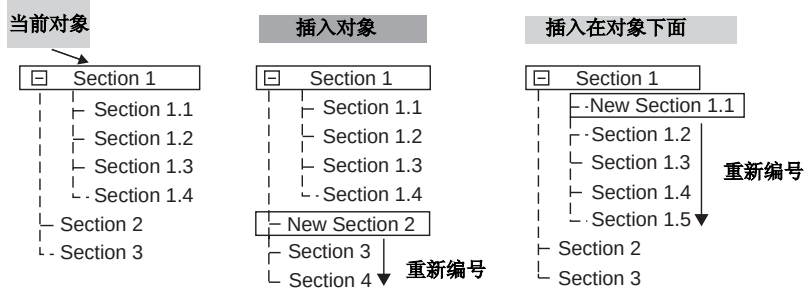


图9.8 创建对象

到由删除对象空出的洞中，使剩余后继对象重新编号。然后标识被删除对象的插入点。取决于对象层次结构的性质，只有两种可能性。对象作为当前对象的同级对象安排，或作为当前对象下一级的第一个子对象。第一种情况如图 9.9 所示。

9.4.4 图片

在 DOORS 中，正式对象可以包含图片。每个 DOORS 项目都可以有一个图片对象库，每个图片都可以插入到一个或多个 DOORS 的正式模块中。DOORS 可以识别的图片格式取决于系统平台。

在一个或多个正式模块中插入图片对象是一种简单过程。图片应该在外部编辑器中准备，然后输入到 DOORS 项目中，如图 9.10 所示。

9.4.5 表格

在很多情况下，需求或与需求关联的信息都以表格形式表示。表格可以在现有对象之后或之下创建，或在空模块的第一层上创建。通过描述所要求的行和列数可以做到这一点。然后，新表格可以插入到正式模块中，如图 9.11 所示。只要没有到其它单元格对象或表格对象的链，就可以删除表格。

9.5 历史与版本控制

9.5.1 历史

DOORS 维护对所有模块内容、模块对象和属性的模块及对象级修改的历史日志。

所记录的变更包括作出变更的人、变更时间以及对象及其属性的变更前后状态。模块历史可以用来跟踪给定模块寿命中的所有事件。可以通过正式模块窗口中的变更条访问对象历史，也可以通过主菜单启动。图 9.12 给出了历史窗口的一个例子。

9.5.2 确定基线

基线是模块的冻结拷贝，一般在项目的重要阶段创建。例如，一组需求通常在紧接评审之前建立基线，并在紧接评审变更被采纳之后建立基线。这使得需求文档的各种状态能够随时并且很容易地重新产生。可以在 DOORS 中对基线进行编号和标注。

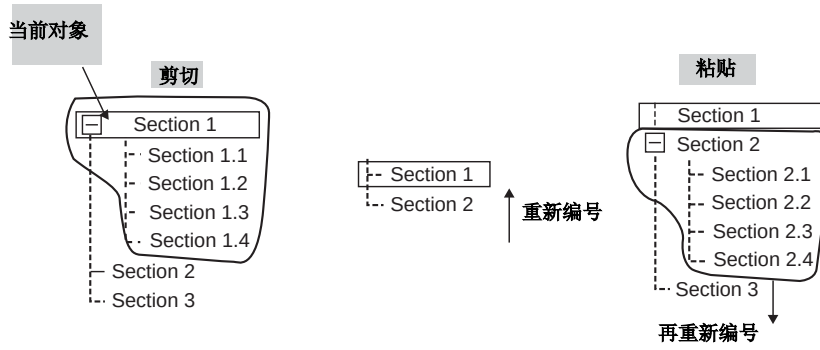


图9.9 剪贴与粘贴对象

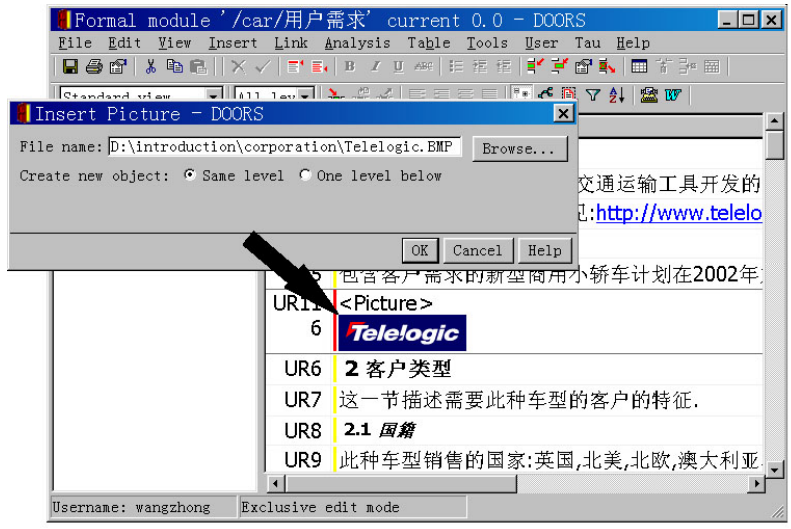


图9.10 图片处理

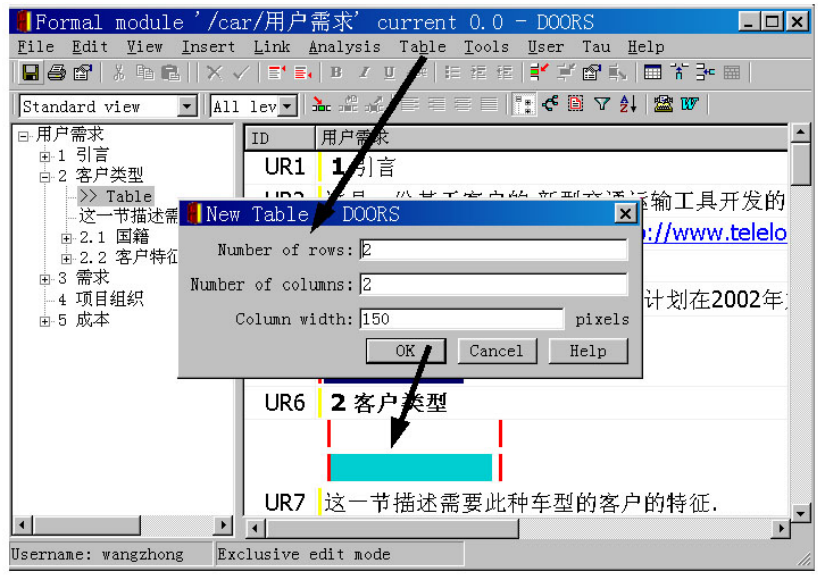


图9.11 创建表格

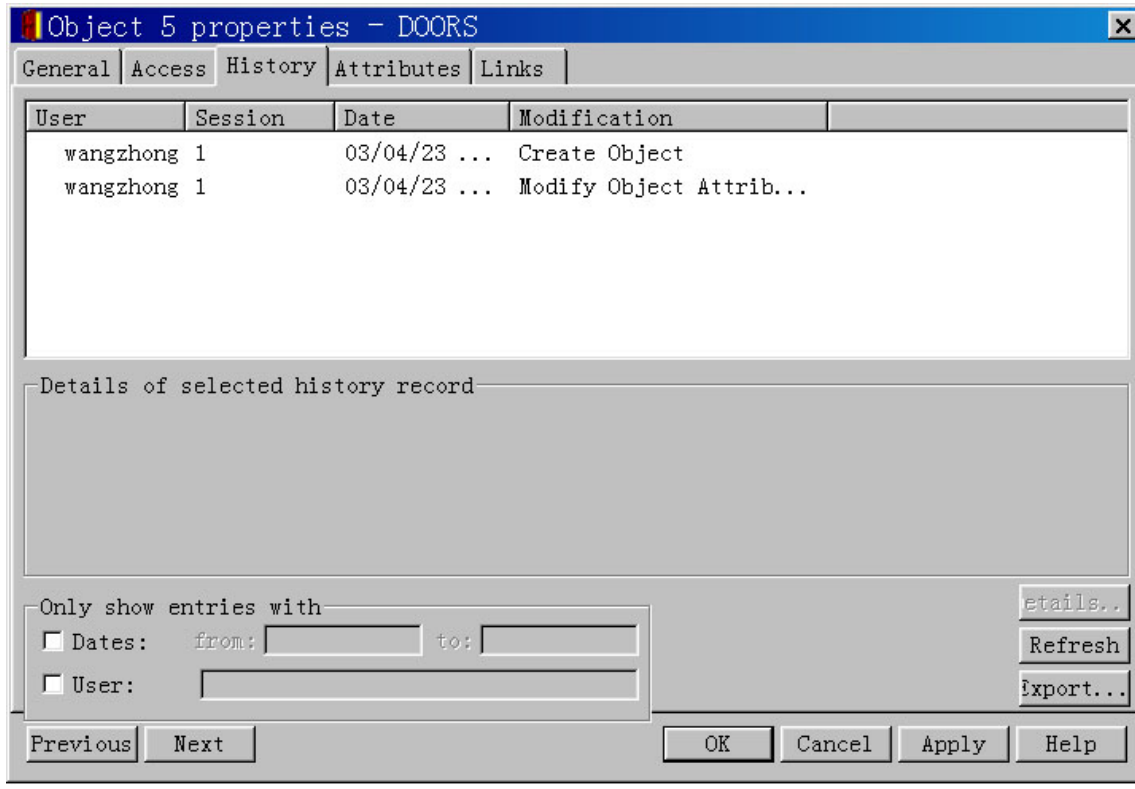


图9.12 历史窗口

基线是正式模块的只读拷贝，不能编辑。如果模块被建立基线，则自从前一个基线以来的所有历史都与新创建的基线一起存储，当前版本的历史被清除。因此，模块的生命历史要跨一系列基线存储。

9.6 属性与视图

9.6.1 属性

属性为模块和对象的相关信息的注释提供了手段。模块属性被用来获取有关模块本身的信息，例如模块拥有者、文档控制编号、评审状态等。对象属性被用来获取单个对象的信息。属性可能是系统定义的，也可能是用户定义的。系统定义的属性自动维护有关模块或对象的关键信息，例如创建时间和创建者; 用户定义的属性可以被用来获取支持用户需求管理过程所需的所有信息。

DOORS 所提供的各种标准属性类型叫做基本类型，通过这些基本类型可以定义属性，例如整数、实数、日期、字符串、文本、用户姓名。也可以有用户定义的属性类型。

通过创建列，可以直接观察和编辑属性信息。通过这种方式，可以直接生成屏幕报告和打印报告。虽然对象可以包含很多属性，但是用户一般一次只对观察这些属性的一部分感兴趣。可以

创建列，只显示所需的那一部分属性，以免用户被信息淹没。直接通过拖拽列标题可以重新确定列的位置。

9.6.2 视图

DOORS 提供“视图”用于以很多不同方式观察相同信息。视图与模块一起存储，通过一个项目的数据可以创建很多视图。在创建视图时，需要指定要显示的对象和属性。例如，你可以要创建一个视图，以便只看到“优先级”属性取值为“高”的模块中的对象。以后视图像表格一样地出现，每行包含一个对象以及已选的对象属性。

9.7 可跟踪性

DOORS 中的可跟踪性通过对象之间的链管理。

9.7.1 链

DOORS 链是两个对象之间的连接。链的一个属性是方向性；所有链都有一个方向，从源到目标。为了表示数据关系需要创建链，因此使用户能够将信息可视化为网络，而不只是树。虽然链具有方向性，但是 DOORS 提供在链的两个对象之间所创建的路径的双向漫游。因此，可以跟踪一个文档的变更对另一个文档的影响，也可以反向进行跟踪，说明决策背后的原始考虑。

DOORS 提供各种方法创建和维护链。可以通过在两个对象（通常在不同的模块中）之间拖拽建立单个链。整个链集合可以采用其它方式创建。例如，“复制与连接”功能可以复制整个对象集合，并将每个拷贝连接到原始对象集合上。

在正式模块的标准视图中，链通过三角形链图标沿主列的右侧表示。指向左侧的图标表示输入链，指向右侧的图标表示输出链。

9.7.2 可跟踪性报告

在 DOORS 中，有多种方式可以在屏幕和书面上创建可跟踪性报告。最简单的屏幕可跟踪性工具是使用 **Analysis → Traceability Explorer**，通过 Windows 风格界面，使用户能够在单个窗口中观察多个文档中的可跟踪性，如图 9.13 所示。

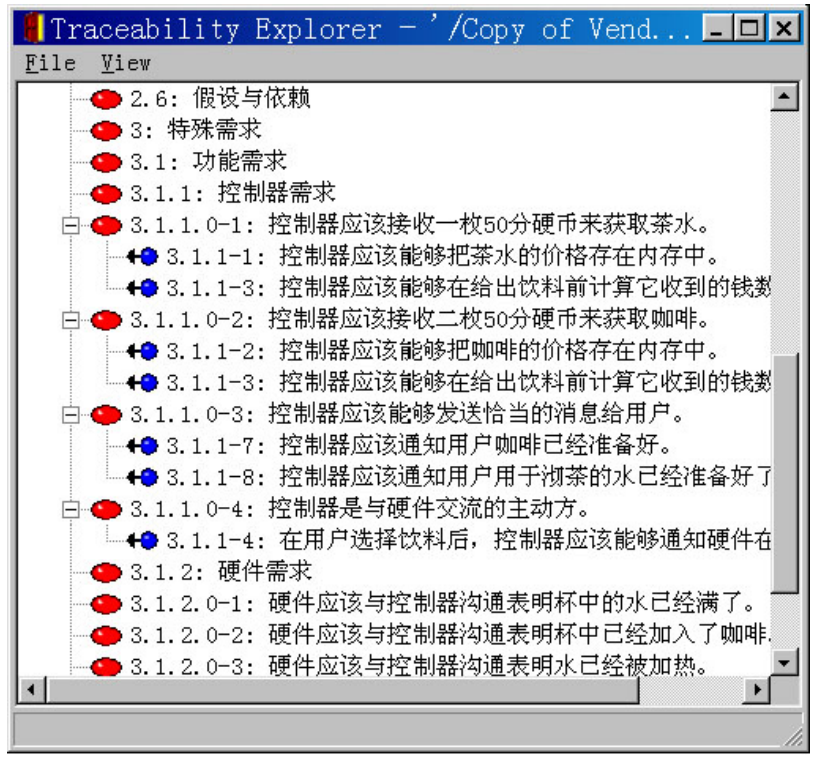


图9.13 可跟踪性浏览器

生成屏幕报告（以后还可以打印出来）的另一种方式，是在视图中添加可跟踪性列。这些列可以显示来自其它文档的被连接对象的数据。可跟踪性列采用 **Analysis → Wizard** 创建，用户选择要显示的链和被连接对象的属性来形成显示报告。可跟踪性列完全是动态的，在创建新链或远程变更数据时更新。通过这种方法，来自多个文档的数据可以集成到一起，形成单一的报告，可以是屏幕报告，也可以是书面报告。

图 9.14 给出了包含一个可跟踪性列的视图例子。这个视图是在当前模块中，是 stakeholder 需求，可跟踪性列通过输入链显示来自系统需求模块的数据。在这个例子中使用了丰富可跟踪性，包括以下可跟踪性列：

1 stakeholder 需求标识符（当前模块）。

2 显示 stakeholder 需求标题和文本（当前模块）的主列。

3 丰富可跟踪性合并器（当前模块中 stakeholder 需求的一个属性）。

4 满足参数（当前模块中 stakeholder 需求的一个属性）。

5 标题为“Contributing Requirements”的可跟踪性列，显示链到 stakeholder 需求的系统需求多个属性。系统需求的对象标识符采用粗体在方括号中显示。此外，每个系统需求的节标题给出系统需求文档中的基本内容。

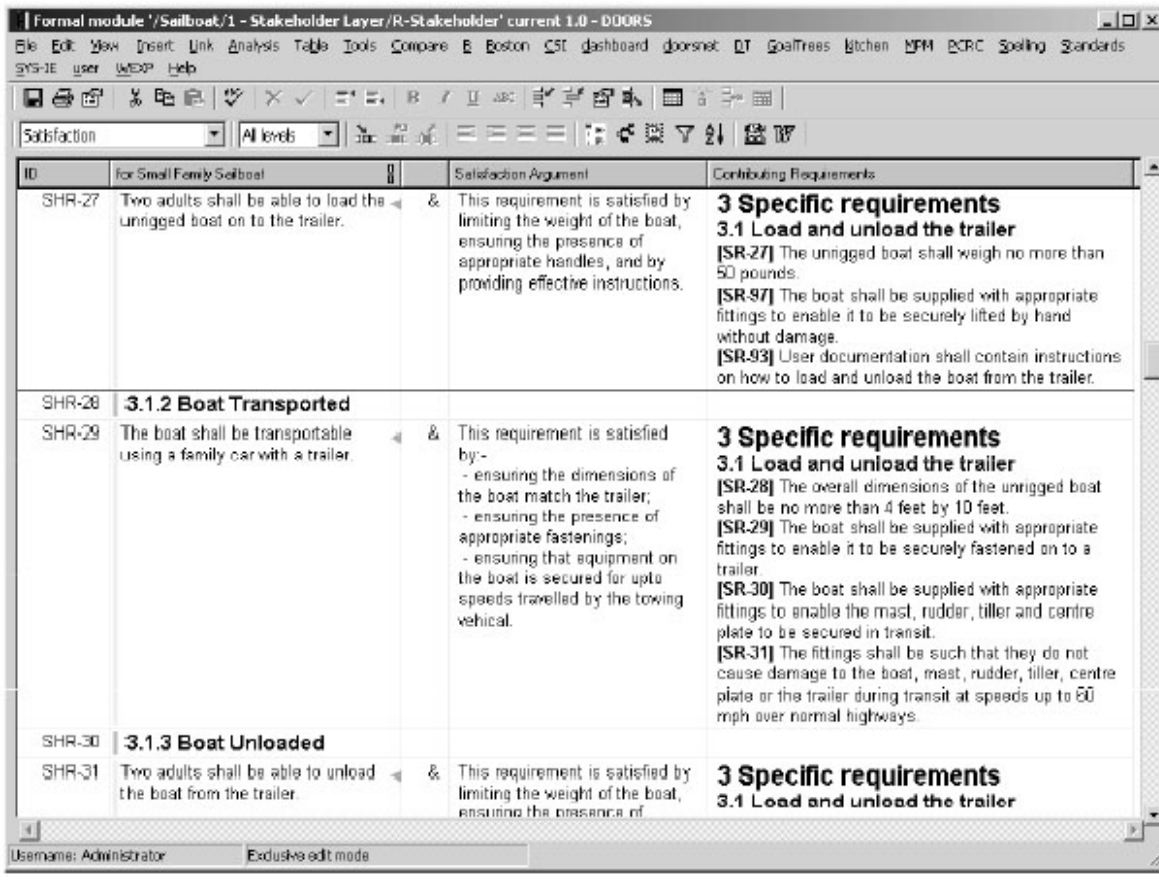


图9.14 显示需求满足情况的可跟踪性列

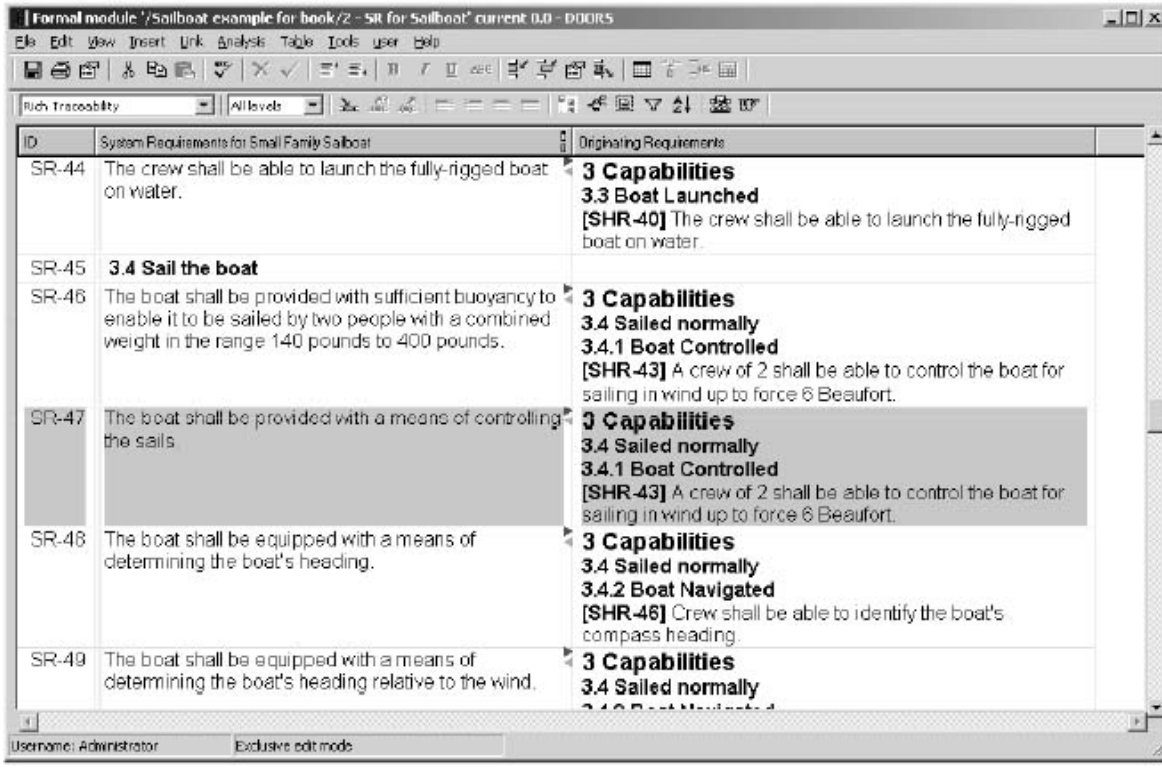


图9.15 流出链的可跟踪性列

图 9.15 给出同一链的另一端的可跟踪性列，即从系统需求文档返回 stakeholder 需求。在这种情况下，输出链被反过来，信息在标题为“Originating Requirements”的列中显示。没有满足参数列。

需求文档常常包含显示列之间关系的可跟踪性矩阵。通过视图中的可跟踪性列，DOORS 不需要手工创建和维护这种矩阵。

9.8 输入与输出

DOORS 和其它工具之间的信息交换能力是非常需要的。包括将遗留信息输入到 DOORS 中，以及将 DOORS 信息输出到外部工具，以进行出版或其它目的。

在项目开发中，有效、可靠地输出和组织大量信息的能力常常是必要的任务。但是，存储格式和平台的多样性及数据结构的不一致性，常常是一种挑战。DOORS 提供了各种输入工具以支持这种活动，尤其是与文档、表格和数据库有关的活动。例如，图 9.16 给出的是如何从 Word 输入到 DOORS 中。通过打开 Word 文档，使用 Export to DOORS 按键，可将 Word 文档输出给 DOORS。在从 Word 输出文件，并输入到 DOORS 之前，需要提供模块名称和描述。

输入到 DOORS 的文档采用与 Word 大纲显示相同的结构，因此标题 1 的文本成为 DOORS 第一层的一个对象。使用段分割符切分每个对象的内容。

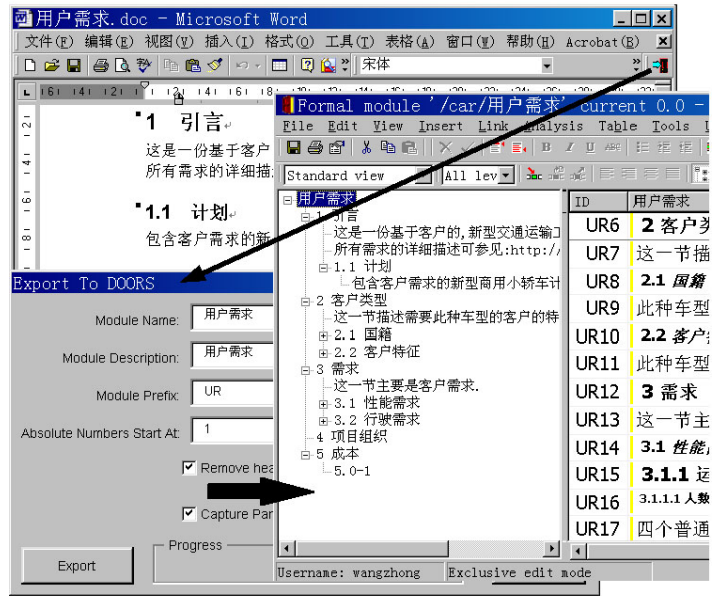


图9.16 从Word向DOORS输出

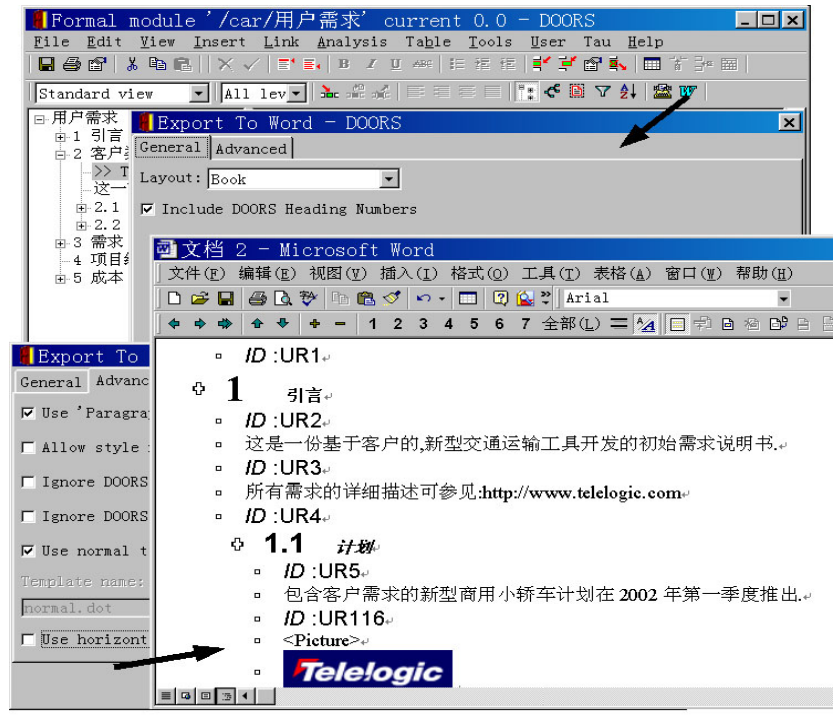


图9.17 从DOORS输出到Word

类似地，DOORS 支持很多输出格式，提供将 DOORS 数据转换到其它桌面工具的方便方法。作为一个例子，可考虑如图 9.17 所示的从 DOORS 到 Word 的输出。

这是前一种操作的逆操作。Word 文档具有与正式模块相同的结构，即对象标题 1 成为第 1 层，标题成为 Word 风格的“标题 1”，等等。文本可采用正常风格显示。

DOORS 提供很多工具和格式的输入、输出能力，包括：RTF、Word、Excel、Access、普通文本、HTML、PowerPoint、MS Project、Outlook 等。

9.9 小结

本章概要介绍了需求管理工具 DOORS。本章通过例子说明本书所介绍的原理的使用方法，例如分层、丰富可跟踪性的通用过程实例等。

同样原理也可用于其它需求管理工具，并在其它需求管理工具中实现。即使只使用字处理工具，本书所介绍的原理也会很有帮助。

参考文献

Alderson A, Hull MEC, Jackson K and Griffiths LE (1998) Method Engineering for Industrial Real-Time and Embedded Systems. *Information and Software Technology*, 40, 443–454.

Andriole SJ (1996) *Managing Systems Requirements: Methods, Tools and Cases*. New York, McGraw-Hill.

Babich W (1986) *Software Configuration Management: Coordination for Team Productivity*. Addison-Wesley.

Bernstein P (1996) *Against the Gods: The Remarkable Story of Risk*. John Wiley.

Boehm B (1981) *Software Engineering Economics*. Prentice-Hall.

Booch G (1994) *Object-oriented Design with Applications*. Redwood City, CA, Benjamin Cummins.

Brown AW, Earl AN et al. (1992) *Software Engineering Environments*. London, McGraw-Hill.

Budgen D (1994) *Software Design*. Addison-Wesley.
Clark KB and Fujimoto T (1991) *Product Development Performance*. Harvard

Business School.
Coad P and Yourdon E (1991a) *Object-Oriented Analysis*. Prentice-Hall.
Coad P and Yourdon E (1991b) *Object-Oriented Design*. Prentice-Hall.

Cooper RG (1993) *Winning at New Products*. Addison Wesley.

Crosby PB (1979) *Quality is Free*. New York, McGraw-Hill.

Crosby PB (1984) *Quality without Tears*. New American Library.

Davis AM (1993) *Software Requirements: Objects, Functions and States*. Englewood Cliffs, NJ, Prentice-Hall.

DeGrace P (1993) *The Olduvai Imperative: CASE and the State of Software Engineering Practice*. Prentice-Hall International.

DeMarco T (1978) *Structured Analysis and System Specification*. Yourdon Inc.

DeMarco T (1982) *Controlling Software Projects*. Yourdon Press.

DeMarco T and Lister T (1987) *Peopleware: Productive Projects and Teams*. Dorset House.

Easterbrook S and Nuseibeh B (1996) Using Viewpoints for Inconsistency Management. *Software Engineering Journal*, 11(1), 31–43.

Finkelstein A, Kramer J, Nuseibeh B and Goedicke M (1992) Viewpoints: A Framework for Integrating Multiple Perspectives in Systems

Development. *International Journal of Software Engineering and Knowledge Engineering*, 2(10), 31–58.

Fowler M and Scott K (1997) *UML Distilled: Applying the Standard Object Modeling Language*. Reading, MA, Addison-Wesley.

Gilb T (1988) *Principles of Software Engineering Management*. Addison-Wesley.

Gorchels L (1997) *The Product Manager's Handbook*. NTC Business Books.

Gotel OCZ and Finkelstein ACW (1995) Contribution Structures. *Proceedings RE'95*, York, UK, IEEE Press.

Harel D (1987) Statecharts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8, 231–274.

Humphrey WM (1989) *Managing the Software Process*. Addison-Wesley.

Jackson M (1995) *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices.* Addison-Wesley.

Jacobsen I, Christerson M et al. (1993) *Object-Oriented Software Engineering*. Wokingham, Addison-Wesley.

Kotonya G and Sommerville I (1996) Requirements Engineering with Viewpoints. *Software Engineering*, 1(1), 5–18.

Kotonya G and Sommerville I (1998) *Requirements Engineering: Processes and Techniques*. Chichester, John Wiley & Sons.

Leite JCP and Freeman PA (1991) Requirements Validation through Viewpoint Resolution. *Transactions of Software Engineering*, 12(2), 1253–1269.

Loucopulos P and Karakostas V (1995) *Systems Requirements Engineering*. McGraw-Hill.

Mazza C et al. (1994) *ESA: Software Engineering Standards*. Prentice-Hall.

Monroe RT, Kompanek A, Metlon R, and Garlan D (1997) Architectural Styles, Design Patterns, and Objects. *IEEE Software*.

Mumford E (1989) User Participation in a Changing Environment: Why We Need IT. In K Knight (ed.) *Participation in Systems Development*.

London, Kogan Page.

Nuseibeh B, Kramer J and Finkelstein A (1994) A Framework for Expressing the Relationships between Multiple Views in Requirements Specification. *Transactions of Software Engineering*, 20(10), 760–773.

Oliver DW, Kelliher TP and Keegan JG (1997) *Engineering Complex Systems with Models and Objects*. New York, McGraw-Hill.

Page-Jones M (1980) *The Practical Guide to Structured Systems*. Yourdon Press.

Perrow C (1984) *Normal Accidents*. Basic Books.

Petroski H (1982) *To Engineer Is Human: The Role of Failure in Successful Design*. St Martin's Press.

Petroski H (1996) *Invention by Design: How Engineers Get from Thought to Thing*. Harvard University Press.

Poots C, Takahashi K et al. (1994) Inquiry-based Requirements Analysis, *IEEE Software* 11(2), 21–32.

Pressman RS (1997) *Software Engineering: A Practitioner's Approach*. McGraw-Hill.

Ross DT (1977) Structured Analysis (SA): A Language for Communicating Ideas. *IEEE Transactions on Software Engineering*, 3(1), 16–34.

Ross DT (1985) Applications and Extensions of SADT. *IEEE Computer*, 18(4), 25–34.

Ross DT and Schoman KE (1977) Structured Analysis for Requirements Definition. *IEEE Transactions on Software Engineering*, 3(1), 6–15.

Rumbaugh J, Blaha M et al. (1991a) *Object Modeling and Design*. Englewood Cliffs, NJ, Prentice-Hall.

Rumbaugh J, Blaha M, Premerlani W, Eddy F and Lorenzen W (1991b) *Object-Oriented Modeling and Design*. Prentice-Hall.

Shlaer S and Mellor SJ (1991) *Object Life cycles:Modeling the World in States*. Yourdon Press.

Shlaer S and Mellor SJ (1998) *Object-oriented Systems Analysis*. Englewood Cliffs, NJ, Prentice-Hall.

Software Engineering Institute (Carnegie-Mellon) (1991) *Capability Maturity Model for Software*. Tech. Report CMU/SEI-91-TR-24.

Sommervile I (1996) *Software Engineering*. Wokingham, Addison-Wesley.

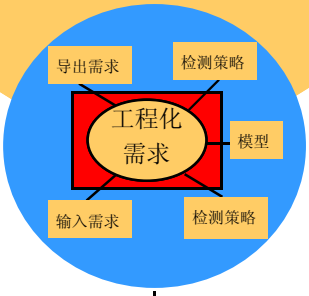
Sommervile I and Sawyer P (1997) *Requirements Engineering: A Good Practice Guide*. Chichester, John Wiley & Sons.

Spivey JM (1989) *The Z Notation: A Reference Manual*. Prentice-Hall.

Stevens R, Brook P, Jackson K and Arnold S (1998) *Systems Engineering: Coping with Complexity*. Prentice-Hall Europe.

Yourdon EN (1990) *Modern Structured Analysis*. Prentice-Hall.

系统，工程，流程



“近年来，我们一直感到缺乏在需求工程方面很有能力的工程师。部分原因可能是需求管理工具供应商一直在使管理层相信，采用其漂亮的工具会解决需求工程问题。当然，工具只能使理解需求工程的工程师更好地完成工作。本书使用大量篇幅介绍需求工程的一些基本技能，以便明智地使用当今各种功能很强的工具。最重要的是，本书将软件需求放在系统背景下，讨论处理这种敏感关系的各种方法。这是一本很重要的书。我认为它对公司特别有价值，使需求工程师和其内部客户能够对可以和应该得到什么，有一种现实的共同理解。”

波音公司高级研究员

Byron Purves

作者简介：

Elizabeth Hull 是英国 Ulster 大学信息学院计算科学系教授。

Ken Jackson 和 Jeremy Dick 是 Telelogic 英国公司的首席顾问。

需求工程

本书针对的是渴望丰富需求工程过程知识的读者，既包括实际工作者，也包括学生。

本书运用最新研究成果，由业界实践经验驱动，在如何编写和结构化需求方面会对实际工作者带来很有用的启发。

- 解释系统工程的重要性，创建针对问题的有效解决方案；
- 介绍系统建模使用的基本表示法，包括数据流图、状态图、面向对象的方法；
- 介绍了一种通用的多层次的需求过程；
- 讨论有效需求管理的关键要素；
- 本书中的一章由丰富可跟踪性的开发人员之一编写；
- 概要介绍了推动需求管理过程的软件工具 DOORS。