第14章　需求质量验证

大多数软件开发者都经历过在开发阶段后期或在交付产品之后才发现需求的问题。当以原来需求为基础的工作完成以后，要修补（fix）需求错误就需要作大量的工作。有研究表明：比起在需求开发阶段由客户发现的一个错误，然后更正这一错误需要多花　68~110倍的时间。另外一个研究发现，在需求开发阶段发现的一个错误，平均仅需要花　30分钟修复，但是在系统测试时发现的错误需要花5~17个小时来修复（Kelly, Sherif and Hops 1992）。检测需求规格说明中的错误所采取的任何措施都将为你节省相当多的时间和金钱。

在许多项目中，包括使用典型的瀑布型生存周期法的项目，测试是一项后期的开发活动。与需求相关的问题总是依附在产品之中，直到通过昂贵并且耗时的系统测试或由客户才可最终发现它们。如果你在开发过程的早期阶段就开始制订测试计划和进行测试用例的开发，就可以在发生错误时立即检测到并纠正它。这样可以防止这些错误进一步产生危害，并且可以减少你的测试和维护费用。

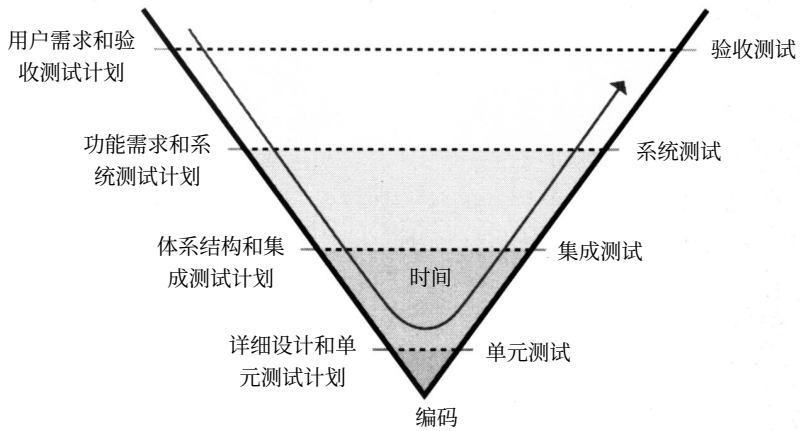

图14-1　软件开发的V字模型

图14-1描绘了软件开发的 V字模型，它表明了测试活动总是与开发活动平行发展的（Davis 1993）。这个模型指明了验收测试是以用户需求为基础的，系统测试是以功能需求为基础的，而集成测试是以系统的体系结构为基础的。在相应的开发阶段，必须规划测试活动并为每一种测试设计测试用例。不可能在需求开发阶段真正进行任何测试，因为还没有可执行的软件。然而，你可以在开发组编写代码之前，以需求为基础建立概念性测试用例，并使用它们发现软件需求规格说明中的错误、二义性和遗漏，还可以进行模型分析。

需求验证是需求开发的第四部分（其余三个为获取、分析和编写规格说明）[⊖]。需求验证

⊖ 一些作者在需求工程这阶段中使用“确认”这一术语（Thayer和Dorfman1997）。验证决定了开发成的产品是否能满足开始时所确定的需求（即正确完成任务）。确认只评估了过渡产品或最终产品是否能真正满足最高层次的特定需求（即完成特定任务）。对于软件需求，这两个术语的差别是微妙的并且是有争议的，所以需求工程阶段，我采用IEEE的术语“验证”。

所包括的活动是为了确定以下几方面的内容：

- 软件需求规格说明正确描述了预期的系统行为和特征。
- 从系统需求或其它来源中得到软件需求。
- 需求是完整的和高质量的。
- 所有对需求的看法是一致的。
- 需求为继续进行产品设计、构造和测试提供了足够的基础。

需求验证确保了需求符合需求陈述（requirement statement）的良好特征（完整的、正确的、灵活的、必要的、具有优先级的、无二义行及可验证的）并且符合需求规格说明的良好特性（完整的、一致的、易修改的、可跟踪的）。当然，你只能验证那些已编写成文档的需求，而那些存在于用户或开发者思维中的没有表露的、含蓄的需求则不予验证。

在收集需求并编写成需求文档后，你所进行的需求验证并不仅仅是一个独立的阶段。一些验证活动，例如对渐增型软件需求规格说明的反复评审，将贯穿着反复获取需求、分析和编写规格说明的整个过程。其它的验证步骤，例如软件需求规格说明的正式审查，是在正式确定软件需求规格说明基线之前对需求分析质量进行的最后一次有用的质量过滤。当你的项目计划或实际工作中的独立任务破坏了结构性时，就要结合进行需求验证活动，并且为随后出现的返工预先安排一段时间，这通常会在质量控制活动之后进行。

有时，项目的参与者不愿意在评审和测试软件需求规格说明上花费时间。虽然在计划安排中插入一段时间来提高需求质量似乎相应地把交付日期延迟了一段时间，但是这种想法是建立在假设验证需求上的投资将不产生效果的基础上的。实际上，这种投资可以减少返工并加快系统测试，从而真正缩短了开发时间。 Capers Jones提出：为防止错误而花费 1美元将可以为你修补错误节省 3~10美元（Jones 1994）。更好的需求将会带来更好的产品质量和客户更大的满意程度，这可以降低产品生存期中的维护、增强和客户支持的费用。在需求质量上的投资可以使你节省更多的钱。

使用不同的技术有助于你验证需求的正确性及其质量（ Wallace and Ippolito 1997）。本章将重点介绍两种最重要的验证技术：正式和非正式的需求评审，还有从使用实例和功能需求中开发出来的概念性测试用例。

14.1 需求评审

通常，总是由一些非软件开发人员进行产品检查以发现产品所存在的问题，这就是技术评审。需求文档的评审是一项精益求精的技术，它可以发现那些二义性的或不确定的需求、那些由于定义不清而不能作为设计基础的需求，还有那些实际上是设计规格说明的所谓的“需求”。

需求评审也为风险承担者们提供了在特定问题上达成共识的方法。我的同事 Barry曾经主持了一个包括来自四个用户代表的软件需求规格说明的评审工作。一个用户提出了一个灾难性的问题：它将使需求做重大更改。会后，需求分析员和项目经理很恼火，因为在前两个月的定义需求会议上，该用户也在场，但她却没有提出这一问题。经过一些调查之后才发现该用户已经反复提出了这个问题，但都被忽略了。在评审过程中，当许多用户一致认为这是一个严重的问题时，分析员和项目经理意识到，他们再也不能忽略这一问题了。

不同种类的技术评审具有不同的称谓。非正式评审的方法包括把工作产品分发给许多其

它的开发人员粗略看一看和走过场似地检查一遍 (walkthrough)。同时执行者描述产品，且征求意见。非正式评审对于培养其他人对产品的认识并获得非结构化的反馈是有利的，但非正式评审是非系统化的，不彻底的，或者在实施过程中具有不一致性。非正式评审不需要记录备案。

非正式评审可以根据个人爱好的方式进行评审，而正式评审则遵循预先定义好的一系列步骤过程。正式评审内容需要记录在案，它包括确定材料、评审员、评审小组对产品是否完整或是否需要进一步工作的判定，以及对所发现的错误和所提出的问题的总结。正式评审小组的成员对评审的质量负责，而开发者则最终对他们所开发的产品的质量负责（ Freedman and Weinberg 1990）

正式技术评审的最好类型叫作审查（ inspection）（Ebenau and Strauss 1994; Gilb and Graham 1993）。对需求文档的审查是可利用的最高级软件质量技术。一些公司已经认识到：在审查需求文档或其它软件产品上花费一个小时，可节省十个小时的工作时间（ Grady1994）。我尚不知道有哪些其它的软件开发或质量评估可以产生十倍的回收投资比。

如果你对提高软件的质量持有认真的态度，那么就审查所编写需求文档的每一行。虽然对大型的需求文档进行详细审查很无聊并且也很费时，但是我所知道的采用需求审查的人都一致认为他们所花的每一分钟都是值得的。如果你认为没有时间详细审查每个方面，那么就使用简单的风险分析模型来区分需求文档哪些部分是需要详细审查的和那些不重要部分只要用非正式评审就能满足质量要求。

在“化学制品跟踪系统”中，在每次获取需求的专题讨论会之后代表不同用户类的小组对渐增性软件需求规格说明进行非正式评审，立刻就发现了许多错误。在需求获取完成以后，由一个系统分析员把来自不同用户类的软件需求规格说明归纳在一起组成一份大约有 50页的文档，并加上许多附录。两个分析员、一个开发人员、三个产品代表者、项目经理以及一个测试人员一起在三次长达两个小时的审查会上对软件需求规格说明进行审查。审查小组发现了 223个错误，其中包括几十个重大缺陷。所有的审查员一致认为在审查会上，他们在软件需求规格说明上所花的时间（一次一个需求），从长远目标来看，节省了项目开发小组大量的时间。

14.1.1 审查过程

在十九世纪七十年代中叶， Michael Fagan 在IBM制定出了审查的过程（Fagan 1976），该过程被认为是软件业最好的实践（Brown 1996）。人们可以审查任何一种软件工作产品，包括需求和设计文档、源代码、测试文档及项目计划等等。审查定义为多阶段过程，涉及到由受过培训的参与者组成的小组，他们把重点放在查找工作产品缺陷上。审查提供了一个质量关卡（quality gate），文档在最终确定以前，必须通过该关卡的检查。虽然，对于 Fagan的方法是否最有效并且是不是最有效的审查的形式还存在争议 (Glass, 1999)，但是审查是强有力的质量技术，这是毫无疑问的。

1. 参与者

审查参与者必须代表三个方面的观点：

a. 产品的开发者及其可能的同组成员 —— 编写需求文档的分析员提供这方面观点。

b. 先前产品的开发者或正在评审的项目的规格说明编写者 —— 这可能是一个系统工程师或系统构造师，他们可以检查软件需求规格说明，以获得系统说明中每个需求的正确可跟踪

能力。由于没有高层次需求文档，审查工作必须包括真正的客户，以保证软件需求规格说明能正确并完整地描述了他们的需求。

c. 要根据正在审查的文档来开展工作的人们——对于一个软件需求规格说明，你可能需要包括一个开发人员、一个测试人员、一个项目经理和一个用户文档编写人员，他们的工作基础都是软件需求规格说明。这些审查人员将会发现不同类型的问题。一个测试人员很可能会发现一个不明确的需求，而一个开发人员将会发现一个技术上不可实现的需求。

审查组中的审查人员应限制在7个人左右或者更少。如果审查人员太多则很容易在边际讨论、解决问题和关于某些事是对还是错的争论上造成混乱，从而在审查过程中降低分析问题的速度并且会增加发现每个错误所花的费用。

2. 审查中每个成员扮演的角色

一些审查组中的成员在审查期间扮演着特定的角色；这些角色随着不同的审查过程而不同，但其所起的作用是相似的。

作者 作者创建或维护正在被审查的产品。软件需求规格说明的作者通常是收集用客需求并编写文档的分析员。在诸如“一包到底”的非正式审查中，作者经常主持讨论。然而，作者在审查中却起着被动的作用，不应该充当下列任一角色：调解者、读者或记录员。由于作者在审查中不起积极作用，因此，他只能听取其它审查员的评论，思考回答他们所提出的问题，但他并不参与讨论。作者经常可以发现其它审查员没有觉察到的错误。

调解者 调解者（moderator）或者审查主持者所做的是：与作者一起为审查制订计划，协调各种活动，并且推进审查会的进行。调解者在审查会开始前几天就把待审查的材料分发到各个审查员，按时召开会议，从审查员那获得审查结果，并且使会议集中在发现错误上，而不是解决提出的问题。向经理或者那些收集审查数据的工作人员汇报审查结果也是调解者的责任。调解者最后的角色是督促作者对规格说明做出建议性的更改，以保证向执行者明确说明在审查过程中提出的问题和缺陷。

读者 读者的角色由审查员扮演。在审查会进行期间，读者一次审查规格说明中的一块内容，并做出解释，而且允许其它审查员在审查时提出问题。对于一份需求规格说明，审查员每次必须对需求给出注解或一个简短评论。通过用自己的话来陈述，读者可能做出与其它审查员不同的解释，这将有利于发现二义性或可能的错误。

记录员 记录员，或书记员，用标准化的形式记录在审查会中提出的问题和缺陷。记录员必须仔细审查所写的材料以确保记录的正确性。其它的审查员必须用有说服力的方式帮助记录员抓住每个问题的本质，这一方法也使作者清楚地认识到问题的所在和本质。

3. 审查阶段

正如图14-2所示，审查是一个多步骤事件（从 Ebenau and Strauss 1994 改编而来）。每一个审查过程阶段的目的简要地总结如下：

规划（planning） 作者和调解者协同对审查进行规划，以决定谁该参加审查，审查员在召开审查会之前应收到什么材料并且需要召开几次审查会。审查速度对能发现多少错误影响甚大（Gilb and Graham 1993）。如图14-3所示，审查软件需求规格说明越慢，就能发现越多的错误（对这一现象的另一种解释是如果遇到太多的错误，就会引起审查速度的下降。）由于不能把太多的时间用于需求审查，所以应根据忽略重大缺陷的风险来选择一种合理的速度。每小时审查4~6页是合理的，但应根据如下因素调整审查速度：

- 一页中的文字数量
- 规格说明的复杂性
- 待审查的材料对项目成功的重要程度
- 以前的审查数据
- 软件需求规格说明作者的经验水平

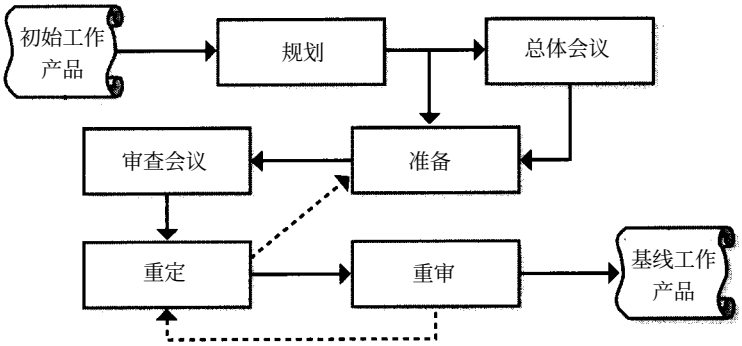

图14-2 审查过程阶段

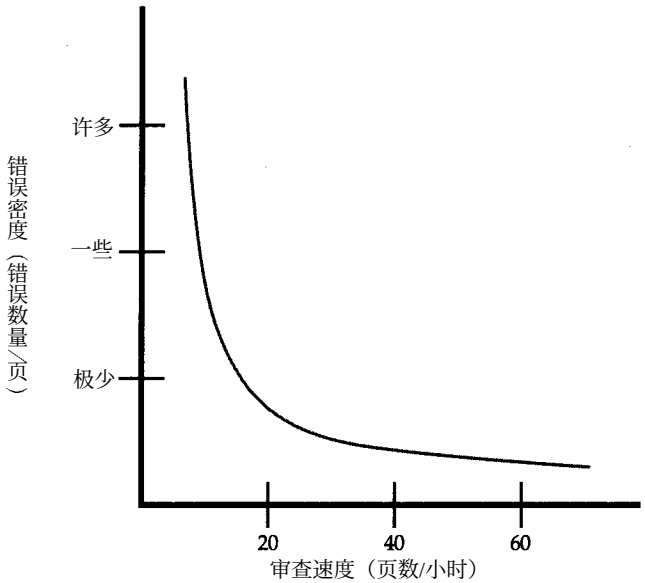

图14-3 审查速度与发现的错误数量之间的关系

总体会议（overview meeting） 总体会议可以为审查员提供了解会议的信息，包括他们要审查的材料的背景，作者所作的假设和作者的特定审查目标。如果所有的审查员对要审查的项目都很熟悉，那么你就可以省略本次会议。

准备（preparation） 在正式审查的准备阶段，每个审查员以典型缺陷（ defect）清单（在本章的后面部分介绍）为指导，检查产品可能出现的错误，并提出问题。审查员所发现的错误中有高达75%的错误是在准备阶段发现的，所以这一步骤不能省略。如果审查员准备不充分，将会使审查会变得低效，并可能作出错误的结论，此时，审查就是一种时间的浪费。记住，你花在检查同僚工作的时间是对提高整体产品质量的一种投资，并且其他组员也会以

同样的方式帮助你提高产品。

审查会议（inspection meeting） 在审查会进行过程中，读者通过软件需求规格说明指导审查小组，一次解释一个需求。当审查员提出可能的错误或其它问题时，记录员就记录这些内容，其形式可以成为需求作者的工作项列表。会议的目的是尽可能多地发现需求规格说明中的重大缺陷。审查员很容易提出肤浅的和表面的问题，或者偏离到讨论一个问题是否是一个错误，讨论项目范围的问题，并且集体研讨提出问题的解决方案。这些活动是有益的，但是他们偏离了寻找重要错误以及提高发现错误几率的中心目标。审查会不应该超过两个小时；如果你需要更多的时间，就另外再安排一次会次。在会议的总结中，审查小组将决定是否可以接受需求文档、经过少量的修改后可接受或者由于需要大量的修改重审而不被接受。

一些研究者认为审查会议是劳动密集型的，以至于很难说清它们的价值。然而，我发现审查能发现审查员在进行个人准备时没有发现的错误。即使有这些审查质量的活动，在继续进行设计和构造软件时，应该根据风险，对为了提高需求质量需要投入多少精力作出决策。

重写（rework） 我所观察到的几乎每一个质量控制活动都可能发现一些需求缺陷。因此，作者必须在审查会之后，安排一段时间用于重写文档。如果把不正确的需求拖延到以后修改，那将十分费时，所以现在正是解决二义性、消除模糊性，并且为成功开发一个项目打下坚实的基础。如果你不打算纠正缺陷，那么进行需求审查将是无意义的。

重审（follow-up） 这是审查工作的最后一步，调解者或指派人单独重审由作者重写的需求规格说明。重审确保了所有提出的问题都能得到解决，并且正确修改了需求的错误。重审结束了审查的全过程并且可以使调解者做出判断：是否已满足审查的退出标准。

4. 进入和退出审查的标准

当软件需求文档满足特定的前提条件时，你就可以进行需求审查了。在审查的准备阶段，进入审查的标准为作者设定了前进的方向。这些标准还可以使审查小组避免把时间浪费在审查之前就应该解决的问题上。调解者在决定进行审查之前，可以把进入审查的标准作为一种清单，并以此作为判断的标准。下面是一些关于需求文档的进入审查的标准：

- 文档符合标准模板。
- 文档已经做过拼写检查和语法检查。
- 作者已经检查了文档在版面安排上所存在的错误。
- 已经获得了审查员所需要的先前或参考文档，例如系统需求规格说明。
- 在文档中打印了行序号以方便在审查中对特定位置的查阅。
- 所有未解决的问题都被标记为TBD（待确定）。
- 包括了文档中使用到的术语词汇表。

相似地，在调解者宣布审查结束之前，你应该定义所满足的退出审查的标准。这里有一些关于需求文档的退出标准：

- 已经明确阐述了审查员提出的所有问题。
- 已经正确修改了文档。
- 修订过的文档已经进行了拼写检查和语法检查。
- 所有TBD的问题已经全部解决，或者已经记录下每个待确定问题的解决过程，目标日期和提出问题的人。
- 文档已经登记入项目的配置管理系统。

• 检查是否已将审查过的资料送到有关收集处。

5. 需求审查清单

为了使审查员警惕他们所审查的产品中的习惯性错误，对你的公司所创建的每一类型的需求文档建立一份清单。这些清单可以提醒审查员以前经常发生的需求问题。图 14-4描述了审查软件需求规格说明时的清单，图 14-5包含了一个使用实例的审查清单。

没有人可以记住冗长清单中的所有项目。削减每一份清单以满足公司的需要，并修改这些清单使其能反映需求中最常发现的错误。可以让不同的审查员使用完整清单的不同子集来查找错误。一个人可以检查出所有文档内部的交叉引用是正确的，而另一个人可以判断这些需求是否可以作为设计的基础，并且第三个人可以专门评价可验证性。一些研究结果表明：赋予审查员特定的查错责任，向他们提供结构化思维过程或情节以帮助他们寻找特定类型的错误，这比仅仅向审查员发放一份清单所产生的效果要好得多（ Porter, Votta and Basili 1995）。

14.1.2 需求评审的困难

那些审查需求文档的公司面临着许多困难（ challenge）。下面是一些普遍的问题，并附有解决的方案。

1) 大型的需求文档　审查一份几百页的软件需求规格说明是令人畏惧的。你很可能完全忽略整个审查过程，并继续进行软件的构造开发——这不是一个好的选择。即使是一份中型的软件需求规格说明，审查员们可能会认真地检查第一部分，一些意志坚定的人可以检查到中间部分，但没有一个人可能检查到最后一部分。为了避免使审查小组感到不安，只在把软件需求规格说明作为基线时才进行审查，在审查全部的文档之前，在你开发软件需求规格说明时，可以采用非正式的、渐增式的审查。让一些审查员从文档的不同位置开始检查，以确保认真地检查其中的每一页。如果你有足够的审查员，可以分成几个小组分别审查材料的不同部分。

2) 庞大的审查小组　许多项目参与者和客户都与需求有关系，所以你可能要为需求审查的参与者制作一张冗长的名单列表。然而，庞大的审查小组将导致难于安排会议，并且在审查会上经常引发题外话，在许多问题上也难于达成一致意见。我曾参加过一个有 14名审查员的审查会。14个人对是否离开一个燃烧的房子意见不一，更不用说在判断一个特定的需求是否正确上达成一致意见了。尝试用以下的方法处理庞大的审查小组：

- 确保每个参与者都是为了寻找错误，而不是为了解软件需求规格说明中的内容或者为了维护行政上的位置。如果一些参与者只是想大概了解审查的内容，那么就邀请他们去参加总体会议，而不是参加审查会。
- 理解审查员所代表的观点（例如客户、开发者或测试者），并且委婉地拒绝以相同的观点看待问题的参与者。在准备阶段，你可能要收集持有同样观点的反馈人的信息，但只要派其中的一个作为代表参加会议。
- 把审查组分成若干小组并行地审查软件需求规格说明，并把他们发现的错误集中起来，剔除重复的部分。研究表明：多个审查小组比起单一的大组而言，可以发现需求文档中更多的错误（Martin and Tsai 1990; Schneider, Martin and Tsai 1992; Kosman 1997）。审查小组总是发现错误的不同子集，所以并行审查的结果是追加的，而不是冗余的。

3) 审查员在地域上的分散　越来越多的公司正通过地域上分散的开发组进行合作开发产

品。地域上的分散性使需求审查更加困难。视频会议是一种有效的解决方案，但在电话会议中，你无法知道对方赋予形体的语言以及脸部的表情，其效果比面对面的会议要差。比起面对面的会议，这两种远程会议更难于进行调节（控制）。

组织和完整性

- 所有对其它需求的内部交叉引用是否正确？
- 所有需求的编写在细节上是否都一致或者合适？
- 需求是否能为设计提供足够的基础？
- 是否包括了每个需求的实现优先级？
- 是否定义了所有外部硬件、软件和通信接口？
- 是否定义了功能需求内在的算法？
- 软件需求规格说明中是否包括了所有客户代表或系统的需求？
- 是否在需求中遗漏了必要的信息？如果有的话，就把它们标记为待确定的问题。
- 是否记录了所有可能的错误条件所产生的系统行为？

正确性

- 是否有需求与其它需求相冲突或重复？
- 是否简明、简洁、无二义性地表达每个需求的？
- 是否每个需求都能通过测试、演示、审查得以验证或分析？
- 是否每个需求都在项目的范围内？
- 是否每个需求都没有内容上和语法上的错误？
- 在现有的资源限制内，是否能实现所有的需求？
- 是否任一个特定的错误信息都具有唯一性和明确的意义？

质量属性

- 是否合理地确定了性能目标？
- 是否合理地确定了安全与保密方面的考虑？
- 在确定了合理的折衷情况下，是否详实地记录了其它相关的质量属性？

可跟踪性

- 是否每个需求都具有唯一性并且可以正确地识别它？
- 是否可以根据高层需求（如系统需求或使用实例）跟踪到软件功能需求？

特殊的问题

- 是否所有的需求都是名副其实的需求而不是设计或实现方案？
- 是否确定了对时间要求很高的功能并且定义了它们的时间标准？
- 是否已经明确地阐述了国际化问题？

图14-4 软件需求规格说明的审查清单

- 使用实例是否是独立的分散任务？
- 使用实例的目标或价值度量是否明确？
- 使用实例给操作者带来的益处是否明确？
- 使用实例是否处于抽象级别上，而不具有详细的情节？
- 使用实例中是否不包含设计和实现的细节？
- 是否记录了所有可能的可选过程？
- 是否记录了所有可能的例外条件？
- 是否存在一些普通的动作序列可以分解成独立的使用实例？
- 是否简明书写、无二义性和完整地记录了每个过程的对话？
- 使用实例中的每个操作和步骤是否都与所执行的任务相关？
- 使用实例中定义的每个过程是否都可行？
- 使用实例中定义的每个过程是否都可验证？

图14-5 使用实例文档审查清单

在共享网络文件夹中的电子文件进行文档评审改变了传统的评审会议。在这一方法中，评审员利用字处理软件的特性，在他所审查的文档中插入评论。每个审查员的评论都做上初始标记，这样每个审查员就能看见先前审查员所写的评论。基于 Web的聊天工具可以进行实时的远程讨论，但他们只提供了很窄的通信带宽。基于 Web的嵌入式协作软件工具，就像软件开发技术公司所提供的Review Pro(http://www.sdtcorp.com)也有助于进行远程讨论。如果你不想通过审查会进行审查，那么必须认识到审查效率将下降约 25%。

14.2 测试需求

通过阅读软件需求规格说明，通常很难想像在特定环境下的系统行为。以功能需求为基础或者从使用实例派生出来的测试用例可以使项目参与者看清系统的行为。虽然没有在运行系统上执行测试用例，但是设计测试用例的简单动作可以解释需求的许多问题（ Beizer 1990）。如果你在部分需求稳定时就开始开发测试用例，那么就可以及早发现问题并以较少的费用解决这些问题。

编写关于黑盒子或功能上的测试用例可以明确在特定条件下系统运行的任务。因为你无法描述可能的系统响应，在你面前将会出现一些模糊的和二义性的需求。当分析员、开发人员和客户通过测试用例进行研究时，他们将对产品如何运行的问题有更清晰的认识。

在开发过程的早期阶段，可以从使用实例中获得概念上的功能测试用例（ Ambler 1995; Collard 1999）。然后，你就可以利用测试用例来验证文本需求规格说明和分析模型（例如对话图）并评价原型。这些基于模仿使用的测试用例可以作为客户验收测试的基础。在正式的系统测试中，可以把它们详述成测试用例和过程（ Hsia, Kung and Sell 1997）。在客户定义他们验收的标准时，你询问客户的基本问题是：“ 如果开发出你们所期望的软件，你是怎么来判断开发出的软件是你真正所需要的？ ”如果他们不能回答关于每个特性或使用实例的这种问题，他们就必须澄清需求。

在前面的章节中，我提到过一种情况：我让开发组中的 UNIX脚本专家Charlie为我们正在使用的“ 商业错误跟踪系统 ”编写一个简单的电子邮件接口扩展。我写了许多需求， Charlie觉得这对他很有帮助；因为他以前编写脚本时，别人从未给他提出需求。不幸的是，在我为电子邮件功能编写测试用例之前却延误了两个星期。后来，我找到了错误，那是因为二十多个测试用例中代表的我对电子邮件功能的认识与我提出的需求格格不入。在 Charlie完成他的脚本之前我纠正了错误的需求，所以在他完成脚本的编写时，这些脚本是正确无误的。

需求测试的含义最初对你来说可能看起来比较抽象。可以用一个例子把这个概念描述得更清楚，所以让我们看一下“ 化学制品跟踪系统 ”的开发组是如何把需求规格说明、分析模型和早期创建的测试用例结合在一起。下面列出了与请求化学制品这一任务相关的一个业务需求、使用实例、功能需求、部分对话图和一个测试用例。

1) 业务需求 该系统的一个主要业务动机可以用如下的需求来描述：

“ 化学制品跟踪系统 ”通过鼓励重复使用公司中可用的那些化学制品容器以降低购买费用。

2) 使用实例 与这个业务需求相一致的一个使用实例是“ 请求一种化学制品 ”，它包括允许用户请求化学制品仓库中已有的化学制品的路径。以下是这个使用实例的描述（详见图8-3）：

请求者通过输入化学制品的ID号或从化学制品绘图工具导入（import）化学结构来

请求一种化学制品。系统则通过向请求者提供来自化学制品仓库的一个新的或已用过的化学制品容器或者让请求者向外部供应商发送订单，从而满足请求者的要求。

3) 功能需求 以下是关于让用户选择可用的化学制品的一些功能，而不是向外部供应商发送订单：

如果请求化学制品仓库中的容器，系统将显示可用容器的列表，用户就可以选择一个容器或要求向外部供应商订购一个新容器。

4) 对话图（dialog map） 图14-6描述了“请求一种化学制品”使用实例中关于这一功能的部分对话图。对话图中的矩形框表示了用户和系统之间概念上的对话元素，箭头则代表了对话元素之间可能的导航路径。

5) 测试用例（test case） 由于这个使用实例有许多可能的执行路径，所以你可以想出许多测试用例来阐明普通过程、可选过程和例外。以下只是一个测试用例，该测试用例是以向用户显示化学制品仓库中可用容器的列表为基础的。这个测试用例是从该用户任务的使用实例说明和图14-6描述的对话图中派生出来的：

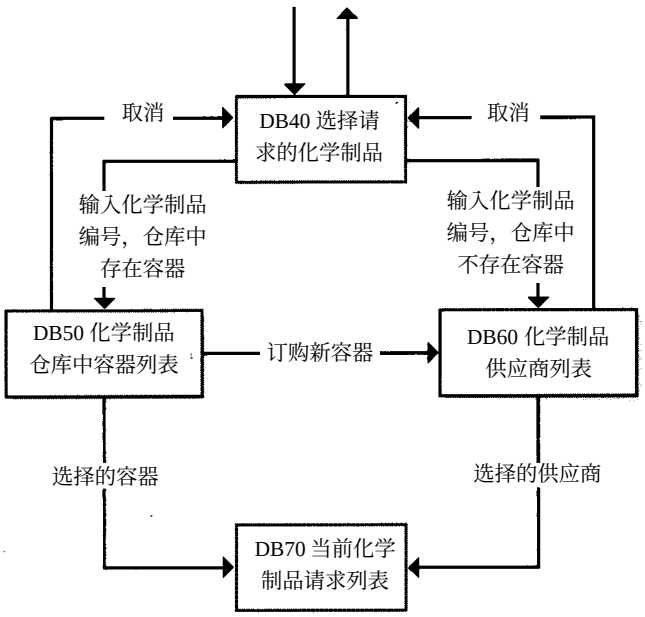

图14-6 在“请求化学制品”使用实例的部分对话图

在DB40对话框中，输入一个合法的化学制品ID号；化学制品仓库中有两个这种化学制品的容器。此时出现了DB50对话框，并带有两个容器号码。选择第二种容器。关闭DB50，此时容器2被加入DB70对话框中当前化学制品请求列表的底部。

Ramesh是“化学制品跟踪系统”的测试主持人，根据他对用户如何与系统交互来请求一种化学制品的理解，编写了许多这样的测试用例。他把每个测试用例映射到相应的功能需求上，以保证现有的需求集合可以“执行”每个测试用例，并且至少使每个测试用例覆盖每个功能需求。下一步，Ramesh用高亮度的笔跟踪对话图中每个测试用例的执行路径。图 14-7中的阴影线描绘了上面的测试用例样本是如何跟踪进入对话图的。

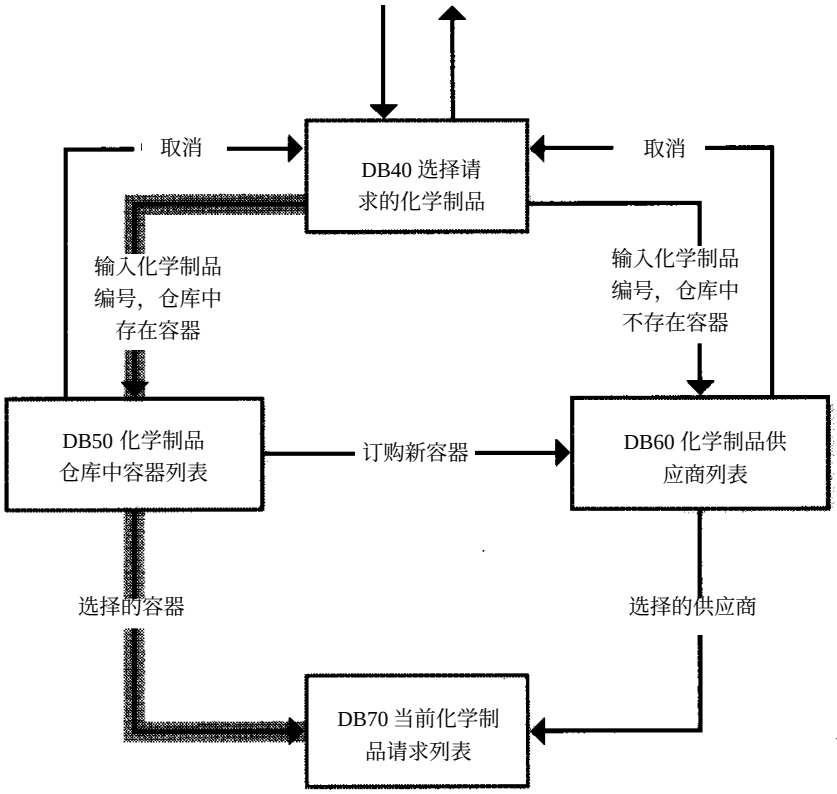

图14-7 在“请求化学制品”使用实例中跟踪一个测试用例进入对话图

通过跟踪每个测试用例的可执行路径，你可以发现不正确和遗漏的需求，并在对话图中纠正错误，精化测试用例。例如，假设以这种方式执行完所有测试用例后，对话图中从 DB50到DB60之间标有“订购新容器”的导航线未被加亮。可能有两种解释：

1) 该导航是一个非法的系统行为。这条线必须从对话图中移去，并且如果软件需求规格说明中包含有这样过渡的需求，那么也应该移去这一需求。

2) 该导航是合法的系统行为，但是遗漏了验证这一系统行为的测试用例。

类似地，假设一个测试用例说明了用户可以采取一些措施从 DB40直接移到DB70。然而，图14-6对话图中没有包含这样的导航线，所以测试用例不能以现有的需求来执行。因此，又存在两种解释，要判断哪一个是对的：

1) 从DB40到DB70的导航是一个非法的系统行为，所以测试用例是错误的。

2) 从DB40到DB70的导航是一个合法的系统行为，则可能是对话图或可能是软件需求规格说明遗漏了用于执行测试用例的需求。

在这些例子中，分析员和测试人员在编写代码以前把需求、分析模型和测试用例结合在一起来检测遗漏、错误和不必要的需求。软件需求在概念上的测试是通过在开发早期的阶段寻找需求错误，从而成为一种在控制项目费用和进度上的强有力的技术。

收集需求并编写需求文档是软件项目设计成功的很好起点。但还需要保证需求的正确性，使需求能体现出良好需求说明的全部特性。如果你能把早期的黑盒子测试设计、非正式需求评审、软件需求规格说明审查和其它需求验证技术相结合，你将花比以前更少的时间、更低的费用来构造质量更高的系统。

下一步：

- 从你的项目软件需求规格说明中任意选择一页功能需求。召集代表不同风险承担者观点的一个小组并仔细审查一页需求以寻找与好的需求说明特性相偏离的那些需求。
- 如果从随机评审中发现太多问题，以至使评审员对整个需求质量感到担忧，那么就要让用户和开发人员代表审查整个软件需求规格说明。在审查过程中要培训审查小组中的成员，使之发挥最高效率。
- 为使用实例或软件需求规格说明中未编码的部分，定义概念上的测试用例。判断风险承担者们是否对测试用例所反映的预期的系统行为持有一致的意见。确保你已经定义了允许测试用例“执行”的所有功能需求，并且不存在多余的需求。