高级软件需求分析师

Senior Software Requirement Analysis Professionals

中科院计算所培训中心
2014 年 6 月

目录

前言 5

第一章 系统思考：高质量软件需求工程概述 6

1.1 需求工程：需求开发与需求管理 6
一、软件需求工程模型 6
二、良好的需求应该具有的特征 7
1.2 项目风险与需求开发 10
一、需求不确定性风险的化解方法 10
二、需求开发的过程框架 15

第二章 需求规划：产品的目标与轮廓 18

2.1 理解客户：从宏观的角度理解客户需要 18
一、关键问题：客户为什么要做这个项目 18
二、流程再造：需求的来源与基础 22
三、业务流程再造的基本工具 25
四、从业务创新的角度理解客户 28
2.2 产品创新：新产品规划的需求方法论 28
一、新产品构思的创新方法 29
二、适合创新的软件工程方法 33
三、通过用户体验评估发现问题 39
四、用户期望与效果的比较分析 52
2.3 产品的轮廓：未来产品的素描 58
一、清晰的表述产品的目标 58
二、定义解决方案的边界 60
三、确定解决方案将受的约束 66
四、总结归纳：项目的陈述 67

第三章 面向客户：如何开发客户需求 68

3.1 建模分析：让复杂变得简单 68
一、概念模型：发现业务的共性及其关系 68
二、理解特征：概念的共性和变化性 75
3.2 行为分析：发现业务功能 81
一、关注源头：业务用例与业务事件 81
二、理解特征：行为的共性和变化性 84
三、变化模型：发现变化与理解变化 87
四、发现功能：在建模中发现功能需求 91
3.3 原型分析：沟通的手段 91
一、原型是“什么”和“为什么”要原型 92
二、原型的各种形式 92
三、通过原型挖掘需求 94
3.4 沟通技巧：理解涉众的需要 96
一、良好沟通需要关注的问题 96
二、培养和锻炼面谈的技巧 96
三、沟通的方法论 98

3.5 产品边界的最后确定.....102
一、最终确定产品的价值与范围.....102
二、客户需求说明书参考模板.....103
3.6 需求获取问题的进一步讨论.....106
一、需求获取的指导方针.....106
二、需求获取中的挑战.....108
第四章 面向产品：如何开发产品需求.....109
4.1 复杂系统的需求分解.....109
4.2 用例分析：描述产品部件的场景.....112
一、用例的完整概念.....112
二、用例是规范行为的契约.....114
三、用例模型及其创建.....116
4.3 用例结构化：应对复杂性的手段.....119
一、包含、扩展与泛化.....119
二、包含的场景描述.....120
三、扩展的场景描述.....121
四、用例的泛化关系及场景描述.....122
五、利用用例描述需求要注意的问题.....123
第五章 深入分析：如何分析与确认需求.....128
5.1 功能性需求：产品应该如何工作？.....128
一、关注细节：事务与功能需求.....128
二、避免误解：如何减少二义性.....131
5.2 非功能性需求：产品的质量特征.....132
一、关注质量：产品的特征与独到之处.....132
二、抓住重点：明确关键质量属性.....135
三、避免冲突：质量属性的取舍.....139
5.3 验收标准：可测量的的需求.....140
一、验收需要标准的原因.....141
二、测量的尺度.....141
三、明确理由.....141
四、非功能需求的验收标准.....142
五、功能性需求的验收标准.....144
5.4 设定优先级：哪些需求是最重要的？.....144
一、为什么要设定需求的优先级.....144
二、从多个角度考虑设定优先级.....144
第六章 总结归纳：编写需求规格说明.....149
6.1 需求规格说明书模板.....149
6.2 项目驱动与问题描述.....150
一、项目目标.....150
二、客户、顾客和其它利益相关方.....151
三、产品的用户.....151
6.3 产品限制条件的确定.....151

6.4 功能性和非功能性需求的描述.....153
一、工作的范围.....153
二、产品的范围.....153
三、功能性需求和数据需求.....153
四、非功能性需求.....154
6.5 阐述项目问题.....154
6.6 需求文档编写的若干建议.....155
一、产品需求规格说明书参考模板.....155
二、善于书写良好的文档.....159

第七章 质量控制：需求的管理、验证与确认.....163

7.1 需求管理的目的与任务.....163
7.2 获得对需求一致的理解.....163
一、建立利益相关方理解需求的渠道.....164
二、获取对需求的承诺.....165
7.3 需求跟踪.....166
一、需求跟踪的动机与方法.....166
二、需求跟踪中的管理活动.....168
三、查找和消除不一致.....168
7.4 需求变更控制.....169
一、确定需求变更类型.....169
二、审批变更申请.....169
三、管理变更请求.....169
7.5 验证与确认的基本概念.....171
7.6 需求验证测试的步骤.....173
7.7 需求确认与正式评审方法.....178
一、正式评审过程.....179
二、评审前复查规格说明.....182
三、需求评审的问题分离技术.....183
四、需求评审的困难.....187
7.8 结语：执著的追求卓越.....188

附录 用例点项目估算方法.....190

一、用例点估算的基本思路.....190
二、确定末调整用例点数.....192
三、计算复杂度因子及开发工作量.....196

高质量软件需求工程

Excellent-quality Software Requirements Engineering

前言

在高质量软件项目中，需求工程的作用举足轻重。统计表明，软件缺陷一半以上的原因来自于需求分析中的问题。仅凭这个数字，就足以告诉我们提升需求开发水平是多么重要，这正是在软件项目中，我们需要对需求分析下功夫的最大原因，本课程的主要思想如下：

1，软件开发是一种高风险的创造性活动，大多数项目风险都与需求密切相关，而这一类风险的控制又十分困难，单靠推断来进行风险控制是不合适也是不可能的。因此，如何通过适当的工程模型消除需求不确定与变更风险，就成为项目成功的根本保障。

2，信息技术不是把传统工作流程搬到信息平台上，必须要根据信息技术扁平、共享的特点，对工作流程进行梳理后重构。分析师需要利用自己独特的对于业务和技术两方面融合的知识水平，以及长期工作中总结归纳出来的经验，为提升组织战略能力提供重要支撑。

3，从广义上看软件需求可以分成两类，一类是有明确单一客户的需求，另一类是面向市场的创新产品需求。这两类需求有什么共性？又有什么个性？如何才能把需求做得更好？分析师应该具有其他人所没有的思想、眼光和感知世界的方法，突破已有的思维模式，规划出有特色、有新意、有创造力的新产品，那我们该如何去做呢？

4，很多人认为需求的变化是不利的，但我们能不能把变化由敌人转化为朋友？一般化的分析很难做到深入，抓住变化才是深入分析之本。通过对业务共性和变化性进行分析，往往能更加透彻的理解需求，并且从中发现潜在需求。分析的基本工具是建模，分析师必须对于概念、行为、变化这三大模型娴熟的使用，并且用它更好的表达分析思想。

5，在产品需求的定义中，利用场景描述行为是减少需求二义性的有效手段。分析师需要通过编写恰当的用例场景和专业的需求文档，准确对系统行为进行详细描述，从而完整定义产品功能性需求。在需求规格说明中，还需要以可测试的方式描述质量属性与验收条件等非功能性需求，这都是高质量需求分析的重要控制点。

6，作为一个完整的工程方法，除了高水平需求开发以外，还需要严谨与规范的需求管理。如何保证各方对需求有一致的理解？如何实现需求跟踪？如何实施需求变更策略？特别是随着近年来项目越来越复杂，需要有更加合理的需求管理方法，为高质量软件开发提供关键支撑。

7，在研究软件需求分析的时候，如果只关注一些具体的技巧和方法，并不可能真正理解需求分析的精髓。如果把眼光放得更高一些，把问题抽象出来进行分析，发现它们之间的逻辑，往往更容易看到本质，也更容易适应各种变化的环境。

本课程汇集了许多专家多年来理论和实践的总结，课程既有理论高度，又能提升实践技巧，使理论与实践完美结合。在授课过程中强调了知其然更要知其所以然，从根本上解决问题。从而避免死板僵化毫无生气的分析模式，代之以生动活泼富有创意的分析过程。通过课程的教学，希望组织的项目开发达到一个新的水平。

中科院计算所培训中心 谢新华

2014 年 6 月 北京

第一章　系统思考：高质量软件需求工程概述

需求工程是软件工程的重要组成部分，它属于软件工程技术范畴，其方法论与现代软件工程方法的发展密不可分。因此为了更全面的理解需求开发方法，首先需要对需求工程有个宏观而全面的了解，并且通过缜密思考，形成完整的需求工程方法论。

1.1 需求工程：需求开发与需求管理

软件项目开发中一般都会面临两大难题：一个是在计划之前需要确定软件的需求，也就是项目的目标，第二是在实施阶段判定目前离目标还有多远（可见性问题），也就是需要确定剩余的工作量。这些在实际工作中遇到的问题、带给我们的困惑，或多或少都与需求有关。那么，什么是需求呢？

产品为用户所做的事情，以及产品在这种上下文背景中必须满足的约束，就是产品的需求。

这个定义似乎就是我们平时所做的工作，但经验告诉我们，事情并不像表面看起来那么简单。如果在研究这些问题的时候过于关注眼前的事情，往往就不能发现问题的本质。我们需要更深入的提炼出一些东西。凡事都有逻辑，我们需要理解一种体系和内在的逻辑关系，需要提炼出一套思想和方法论。一句话，我们需要大思维。

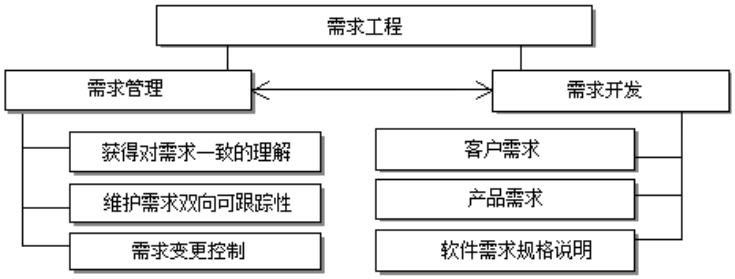
一、软件需求工程模型

为什么软件需求开发需要一套完整的工程方法呢？

程序设计的有效性依赖于规格说明，规格说明的有效性依赖于对问题域的理解。没有好的需求，就没有办法验证程序设计的正确性，也就没有办法把程序与客户的期望用一种逻辑性的方法联系起来。特别是当项目比较大需要很多人共同工作的时候，良好的需求可以保证整个开发团队沿着规定的目标一起前行。

大部分在软件开发中遇到的问题，都是由于收集、编写、协商、修改产品需求过程中的手续和作法（方法）失误带来的。出现的问题包括：非正式信息的收集，未确定的或不明确的功能，未发现或未经沟通的假设，不完善的需求文档，以及突发的需求变更要求。这一切都告诉我们，必须深入地研究和建立良好的需求工程方法。

从宏观上说，我们可以把需求工程分为需求开发和需求管理两部分，如下图所示。



需求开发的目的：是产生并分析客户、产品和产品部件的需求，其过程建立了一套需求收集、建模、分析、和描述的顺序、方法和模版，使需求可以以一种有序而深入的方式进行。所谓需求分析，是把软件系统的功能表示成单一的信息变换过程，需求分析也是一个分解和求精的过程。

需求管理的目的：是管理项目的产品和产品部件的需求，并标识这些需求与项目的计划和工作产品之间的不一致性。由于需求在开发过程中会发生若干变化，这就进一步提升了需求管理的重要性，所以必须建立一套规范的方法来从事需求管理活动。 需求开发与需求管理是相辅相成的两类活动，它们共同构成完整的需求工程体系。

需求工程是软件工程的子工程，由于软件工程是一种独特的智力密集型特征，致使需求工程并不可能按约定俗成的概念和知识，用削足适履的方式，用某些固定的方法到处套用就可以成功的。相反，它体现为一种观察问题和解决问题的方法，最终体现为理解和实施的能力，因此这一章的内容就显得举足轻重。如果在一开始就站到一个宏观把握时空优化理念的高度，那么当进入后面细节讨论的时候，就会觉得心有灵犀、游刃有余了。

二、良好的需求应该具有的特征

1，为什么需要良好的需求

要说清楚什么是良好的需求，先要说清楚为什么需要良好的需求。这个问题看似很普通，但细细想想并不那么简单，长篇大论说某某怎么说的其实没什么意义。我们可以通过一个小例子来，从内心里把这个问题搞通，理不通则道则不行。

假定有两个人，一个人精通业务，另一个人精通编码，他们需要共同来开发一款软件。

第一种方法，可以把软件分成两部分，一个人完成一部分。结果，一个人必须学习编码，另一个必须学习业务，两个人都在自己不熟悉的方面耗费了过多的时间，总的效率非常低下。这是非常典型的小农经济工作方式，低效率是它的特征。

第二种方法，以一个人搞定业务，另一个人搞定编码。他们都在自己熟悉的领域工作，总体效率大幅度上升。这就是工业化的典型特征。通过专业分工，不但提高了效率，也减低了人员培养的难度，毕竟高水平的通才很难得，这叫做降低机会成本。

但是强调专业分工有个缺点，就是在两个专业领域交汇处会存在歧义，理解上也有难度。专业分工增加了相互沟通的难度，这称之为沟通成本。当沟通成本达到一定量值时，相反会降低生产力。例如在沟通上会耗用更多的时间，有歧义的编码必然带来返工，结果大幅度降低了效率。沟通成本的提高使专业分工的好处大打折扣。

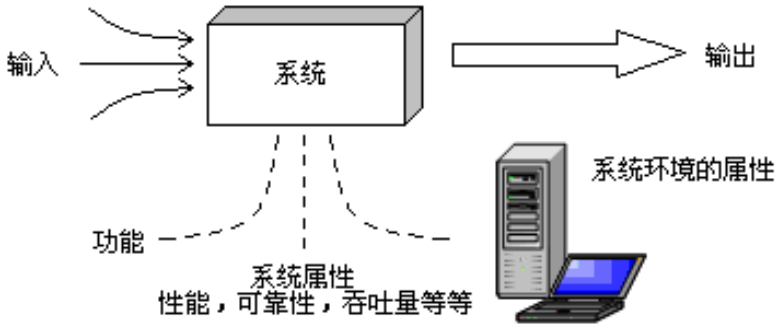
解决的办法就是在两个专业领域的接口处加强沟通，写出无歧义的专业文件。通过这个文件明确产品是什么样的，这就是规格说明，也称为需求。良好需求的特点，就是可以在降低机会成本的同时减少沟通成本，使总的效率提高。

但是事情远没有那么简单，有的情况下，沟通成本会远远大于机会成本。例如在开发创新产品时，需求是在开发过程或用户使用过程中涌现，需求本身就是不断变化的，这时沟通成本会大大提高。为了解决这个问题，职责稍稍模糊的，以合作为特征的方式，往往可以使效率最大化。

所以，什么是好的需求是和环境相关的，甚至和团队成员的水平也是相关的。我们下面描述什么是好的需求，主要考虑在降低机会成本的同时降低沟通成本这个环境下的要求。这是分析师的一个好习惯，描述一个问题之前，先把可能造成歧义的地方展现出来，并进行明确定义。

2，站在产品外部描述产品

需求是一种站在产品外部描述产品的手段，站在产品外部来看，任何软件系统都可以从五个方面完整的描述，如下图所示。

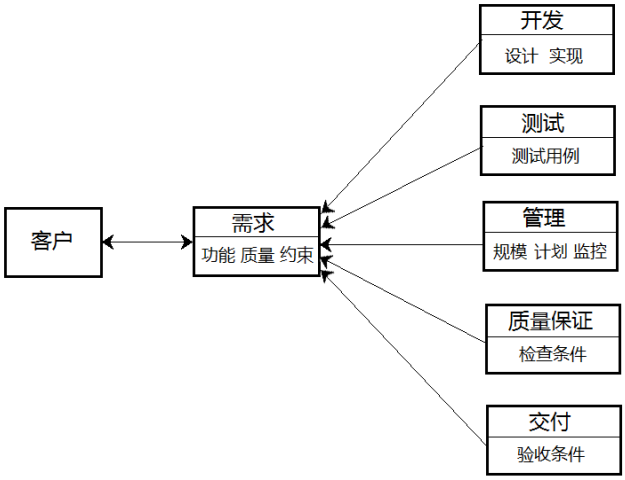


- **系统的输入**：不仅仅是指输入内容，还有输入设备、形式、外观和感觉等必要的细节。很多开发人员都发现，在这个领域可能包括大量的细节，并且具有易变性，尤其是 GUI、多媒体或互联网的产品。
- **系统的输出**：对输出的描述，如语音输出或可视化显示等必需的支持，以及系统所产生信息的协议和格式。
- **系统的功能**：把输入映射到输出的动作，以及它们不同的组合。
- **系统的属性**：典型的非功能需求包括：可靠性、可维护性、可得性以及吞吐量等开发人员必须考虑的问题。
- **系统的环境属性**：附加的非功能性需求，如系统在不同的操作系统、负载下的操作能力等等。

这几种划分已经使用了很多年，而且十分有效，它有助于一致和完整地考虑需求，我们可以通过定义这五类需求形成一个完整的软件需求集。另外，我们也可以根据这些指导原则来判断一个“事物”是不是需求。

我们应该注意到，在开发软件系统时，最为困难的部分就是准确说明要开发什么。最为困难的概念性工作便是编写出详细的规格说明。如果最初的需求一旦出错，最终会给系统带来极大损害，并且以后再对它进行修改也极为困难。

需求是一个被软件工程所有活动所依赖的工程成果，无论管理、设计、实现、测试、质量保证还是交付活动，都依赖于正确的需求。如果需求发生改变，那么所有的工程活动都会相应发生改变，这种影响力覆盖面之广，是其它工程活动所无法比拟的，如下图所示：



3，良好的需求描述的特征

在“需求规格说明”中，每一项需求都是一个单句写成的（在功能需求中，需要带有一个动词），并且需求必须是可以测试的。它应该具备如下特征，这也是后期检查需求书写质量的依据：

- **完整性**：每一项需求都必须将所要实现的功能描述清楚，以使开发人员获得设计和实现这些功能所需的所有必要信息。
- **正确性**：每一项需求都必须准确地陈述其要开发的功能。若软件需求与对应的系统需求相抵触则是不正确的。
- **可行性**：每一项需求都必须是在已知系统和环境的限制范围内可以实施的。在获取需求过程中最好始终有一位软件工程小组的组员负责检查技术可行性。
- **必要性**：每一项需求都应把客户真正所需要的和最终系统所需遵从的标准记录下来。“必要性”也可以理解为每项需求都是用来授权你编写文档的“根源”。
- **划分优先级**：给每项需求、特性或用例分配一个实施优先级，以指明它在特定产品中所

占的分量。如果把所有的需求都看作同样重要，那么项目管理者在开发或节省预算或调度中就丧失控制自由度。

- **无二义性：**对所有需求说明的读者都只能有一个明确统一的解释，避免二义性的有效方法包括对需求文档的正规审查，编写测试用例，开发原型等。
- **可验证性：**检查一下每项需求是否能通过设计测试用例，来确定产品是否确实按需求实现了。如果需求不可验证，则可确定其是主观臆断，而非客观分析了。一份前后矛盾，不可行或有二义性的需求也是不可验证的。

4，良好需求规格说明的特征

把所有的需求集成起来，就成为“需求规格说明”。从整体文档编写的角度上看，好的“需求规格说明”还应该具备如下特点：

- **完整性：**不能遗漏任何必要的需求信息。注重用户的任务而不是系统的功能将有助于你避免不完整性。
- **一致性：**一致性是指与其它软件需求或高层（系统，业务）需求不相矛盾。在开发前必须解决所有需求间的不一致部分。
- **可修改性：**在必要时应维护与修订“需求规格说明”。这就要求每项需求要独立标出，并与别的需求区别开来。每项需求只应在“需求规格说明”中出现一次。这样更改时易于保持一致性。
- **可跟踪性：**应能在每项软件需求与它的业务和设计元素、源代码、测试用例之间建立起而不是大段大段的叙述。

对于严肃的工程活动而言，良好的需求不但是必须的，也是项目成功的根本保障。

1.2 项目风险与需求开发

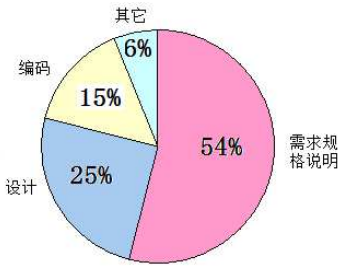
软件开发是一种高风险的创造性活动，很多情况下前期不可能完全预测后面会遇到的风险，完全靠推断来进行风险控制是不合适，也是不可能的。因此，如何发现主要风险并通过适当的工程模型消除这些风险，就成为项目成功的根本保障。

一、需求不确定性风险的化解方法

经验告诉我们，大多数项目风险都与需求有密切关系，而这一类风险与控制又十分困难，这就促使人们来认真思考，降低项目风险的基本措施是什么呢？

1，需求缺陷是最大的风险

软件质量相当程度上是以缺陷（defect）的形式出现的。软件缺陷是由很多原因造成的，但从多个项目的统计结果来看，我们会意外的发现由需求规格说明造成软件缺陷是最大的原因，如下图所示。



产生这种情况的原因如下：

- 客户一般是非计算机专业人员，软件开发人员与客户沟通存在着比较大的困难，对要开发的产品功能理解也不一致。
- 由于软件产品还没有设计、开发，完全靠想象去描述系统的实现结果，所以有些特性还不够清晰。
- 客户的需求总是在不断的变化，容易引起前后文、上下文的矛盾和需求描述的不一致。
- 需求分析没有得到足够的重视，在规格说明书的设计和写作上投入的人力、时间都不足。
- 没有在整个开发队伍中进行充分的沟通，造成人们对问题理解的不一致。

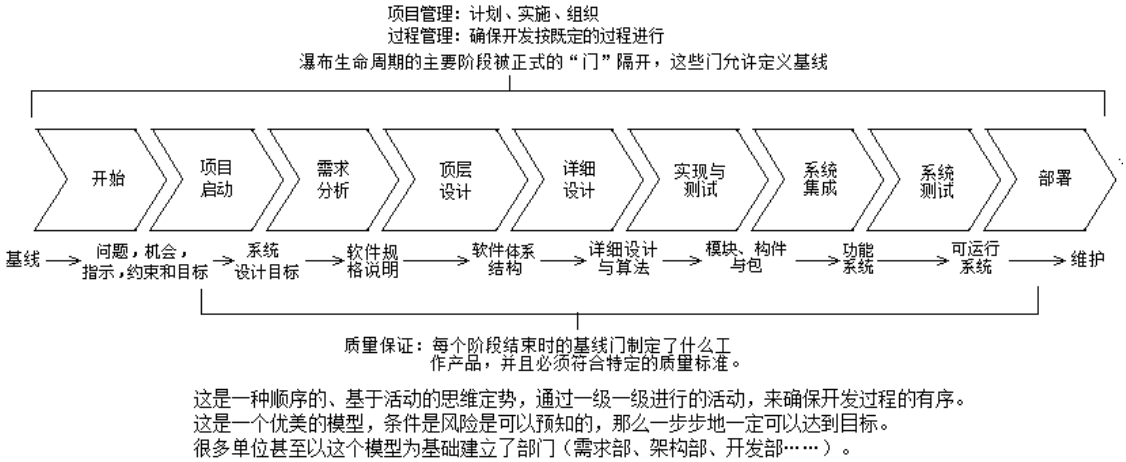
实践表明，在整个项目的进展过程中，需求的变更不可避免，其影响力相当大。需求的变更不可避免的会与各个要素产生互动影响，而且大多数情况下处在一个主动位置：

- 好的需求有助于开发小组对问题的认识一致，需求的变化会引起重新设计，进而影响计划、时间、成本和质量。在出现问题的时候，重新编制代码的代价远远超过重写一份需求文档的代价。
- 正确的需求是测试的基础，需求的变化会影响确认和验证，质量标准的变化会影响工期和成本。
- 需求是交付的依据，由于对需求理解的不一致带来交付中的问题屡见不鲜。

需求不是独立存在的，它在项目开发的几乎每一个节点上都在发挥作用。因此，探讨化解项目风险特别是需求风险，必须在项目过程的大背景下来研究。

2，瀑布式的顺序方法

正是由于人们对于通过专业分工从而降低机会成本的追求，早在上个世纪 70 年代，为了应对软件项目中的风险，人们提出了一种瀑布式的顺序方法，如下图所示：



这种模型的基本思想，是通过严格的专业分工，降低人员知识上的门槛，从而提高了总体效率。为了降低沟通成本，在各个阶段定义了正式的“门”，对上一阶段的交付物提出了严格的要求和质量标准。人们认为，如果每一阶段的质量都是好的，那总体上项目一定会走向成功。咋看起来，这个模型很完美，可惜这仅仅是一厢情愿，因为在这个支离破碎的世界上，完美是不存在的。

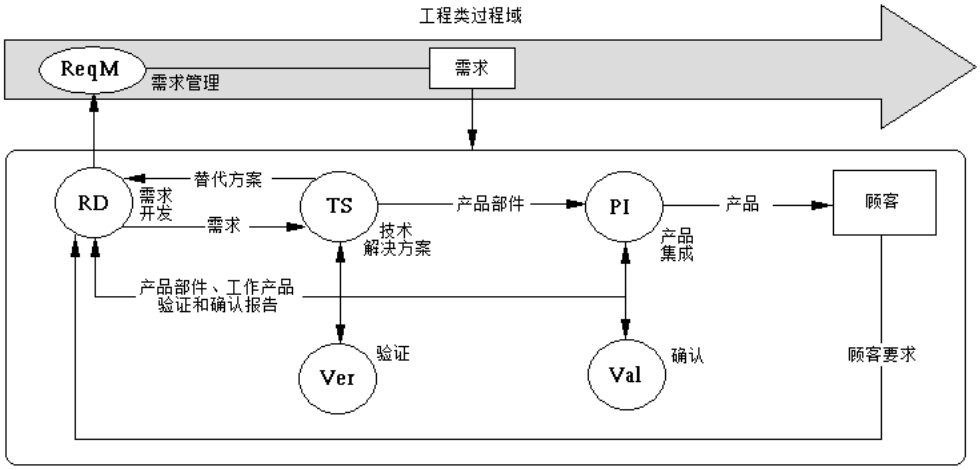
由于在项目初期产品还没有设计、开发，完全靠想象去描述系统的实现结果，必然造成有些特性还不够清晰。这时候即使进行了正规的需求评审，仍然不能保证需求就是正确和清晰的。由于没有预料到的需求风险时常会在后期发生（需求缺陷、新的想法、需求变化），结果就使人们无奈的打破了这种顺序结构，成为一种头疼医头脚疼医脚应变方法，使整个项目失去控制。

正是人们对于风险的规律认识越来越深刻，才发现仅仅依靠前期的需求分析、书写完善的规格说明是不够的。成功的软件工程模型必须建立有效的反馈系统，把变化作为重要的要素来考虑，在过程中不断发现问题和解决问题。只有这种在整个项目过程中建立的完善反馈系统，才能够在

实践中不断发现与消除需求的不确定性，从而逐步消除风险，并且把项目引向成功。

3，设计对于需求的反作用

正是基于这样的背景，现代软件工程过程都强调设计对于需求的反作用，例如，在 GJB 5000A-2008 标准中，明确要求软件工程各个过程域之间的关系不再是一个线性的关系，而是一种复杂的反馈关系，通过有计划的加入各种反馈、控制和修正，引导着项目走向成功。特别是在完成了前期需求进入设计阶段以后，人们往往比较容易的发现初期需求的不确切和含糊之处。这时，由设计方提出问题，需求方据此通过进行补充收集和确认，明确这些含糊之处，从而使需求进一步完善，如下图所示。



这样一来，需求开发就不仅仅是一个前期行为，也不再认为需求开发的结果是被冻结的，而是在产品整个生存周期的各阶段，都可能会标识和精炼需求。另一方面，需求的来源不仅仅是客户，也来自于开发团队和其他两利益相关方，是一种各方面共同沟通的结果。在工程规范中将更强调在整个开发生命周期过程中对需求的协调、收集、反馈和精化。这些变化对于软件工程方法论的影响十分关键。

4，大型项目开发的基本策略

从一般意义上说，项目规模越大，失败的可能性就越大，一般来说，规模扩大一倍，失败的可能性往往会增加十倍。但是常见的情况是，我们所遇到的项目正是由于其规模和复杂性才具有价值，哪些策略可以帮助我们控制项目的规模呢？

- **抓住真正的需求：**软件项目的交付目标表现为一组需求，需求定义了软件的功能和功能的质量。不能为客户创造价值的需求应该受到质疑，很多情况下这两类需求应该考虑推迟或者放弃。
- **分而治之：**寻找机会把大项目分成小项目，比起由大量相互依赖的庞大功能组成的大项目，几个项目独立的小项目更容易管理。
- **设置优先级：**业务环境瞬息万变。大型项目在完工之前，需求会改变很多，有些需求会随着业务的变化甚至被取消，但是关键需求通常会维持不变。理清需求的优先级，优先实现最关键的需求。
- **尽快交付：**在看到演示产品之前，多数客户都不知道自己想要什么。由于不同的人对需求不同的理解，往往会把需求想得过于复杂，很可能与真正的需求不沾边。让客户试用演示的产品，没准会发现其实事情没那么复杂，可能会有更简单的解决方案。首先实现最重要的需求，尽快获得客户反馈，越快越好。

越复杂的项目越难成功实现，缩小项目规模通常会降低设计与实现的复杂性，这是提高成功

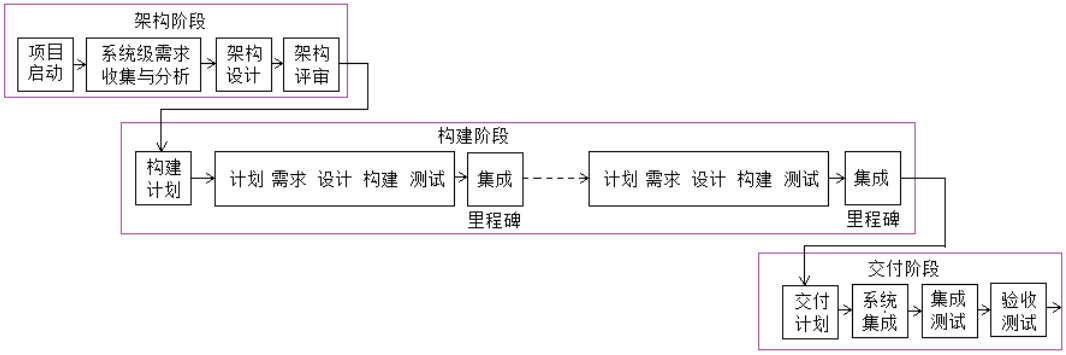
率最有效的途径。

4，建立合理的软件工程模型

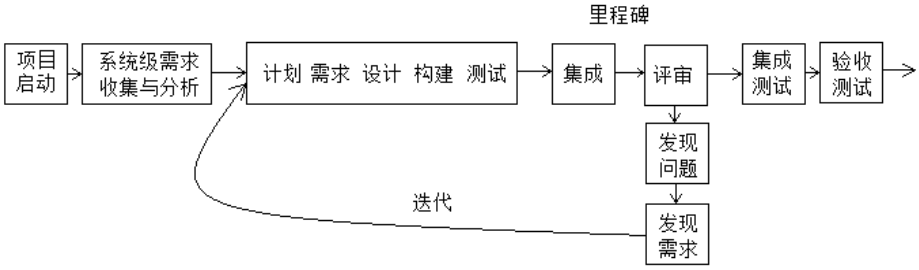
如何建立合理的软件工程模型？一般可以分成两种情况：

第一种情况是，如果项目是以两个组织（企业）间合同清晰界定的明确交付，在这种情况下需求必须十分清楚。但是人的特点是：范围越大考虑问题越宏观，范围越小考虑问题越细致。为避免由于项目规模比较大，造成了项目开发的困难，可以把一个大项目分成若干相对独立能够持续交付的子部分，每个里程碑必须进行一次集成和交付，这称作增量模型。

由于限制了每个阶段的规模和复杂度，这样就简化了问题，提高了效率，降低了机会成本。要注意到，项目初期系统级的需求分析还是应该清晰而稳定的。分阶段交付虽然稍稍提高了一些沟通成本，但是机会成本的降低更大。需求的开发需要支持这种分阶段处理问题的方式。



另一种情况是，项目是以创新为特点和价值的，初期的需求并不太清楚，很多需求希望后期不断发生和涌现。可以使用下面具有强迭代的逐步求精的模型。



在这种模型下，强烈的变更使沟通成本大幅度提高，成为项目的主要矛盾。此时除了应该限制项目规模以外，对人员的要求更强调多面手，知识结构上要有所覆盖。由于专业性被拆散了，反而提高了对人员的要求，增加了机会成本，对需求的要求和组织方法上也会发生相应变化。

这两种工程方法代表着两种极端情况，它们共同特点就是通过分解，把大问题变为小问题。在具体项目中，就需要考虑项目的特点，仔细分析一下机会成本和沟通成本这一对矛盾，看在项目中哪一个是主要矛盾，这才能做成正确的决策。

实践中大部分项目处于这两者之间，形成一种混合体。这种通过可演示的结果来评估自己目前身在何处，也使我们对于项目的实际进展更加有把握，对需求的理解也更透彻。迭代开发的评估比传统方法会更频繁，客户的参与程度会更高，而不是把问题全部堆积到项目后期来解决，这样就降低了项目的风险，通过早期发现需求得不确切之处，引领项目达到一个更加成功的目标。

5，应对需求变化的增量迭代模型

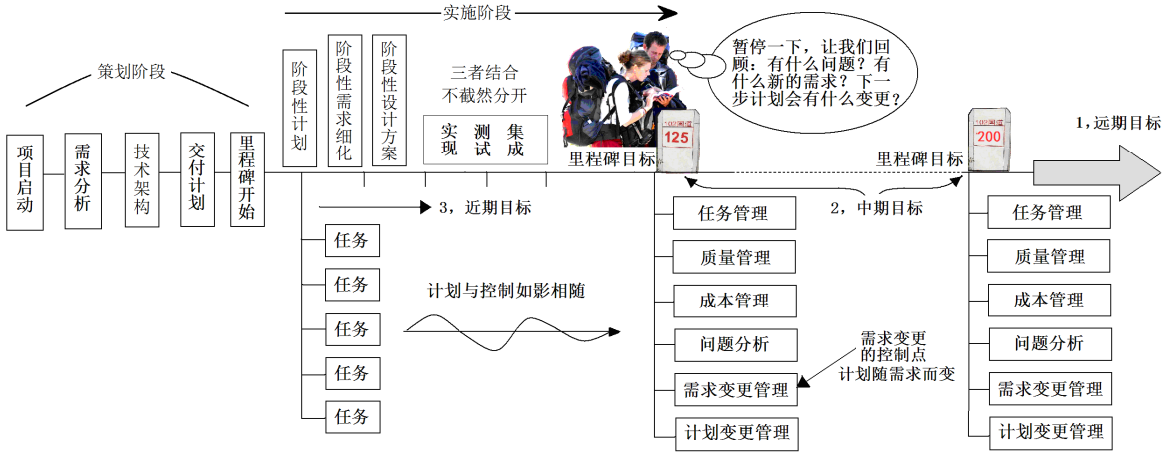
下面举个例子来说明在实践中我们是如何确定合理的开发模型的。

1）基本过程框架

假定有一个由合同界定的项目，规模庞大而且业务本身比较敏感，对项目开发有比较严格的

规范要求（包括计划、监控、质量保证、配置管理等）。项目前期已经做了比较好的系统级需求分析和系统架构设计，从需求大的方面来说还是基本稳定的。但是由于项目庞大，很多细节前期并没有很好的定义，在项目开发过程中，功能级的需求需要进一步细化，而且需求变更不可避免。

根据这样的特征，我们采用了分段线性化的增量迭代模型，通过分阶段持续集成来减少每一部分的规模，以提高效率和质量。并且在两个方面作了改进：一个方面是在开发模型中强化了反馈系统；另一个方面是在开发模型中强调了人的作用和团队参与，形成了一种“增量迭代”过程框架，如下图所示。



为了减少需求变更对于计划的不利影响，防止随着需求不断变更而计划也不断随需而变，我们可以制定如下的规则：

首先：以每个里程碑为一个阶段单元，建议一个阶段的时间长度为项目整个生命周期的 1/12 左右，每个阶段是一个完整的计划、实施与控制过程，其任务就是交付一定数量的产品。

其次：在一个阶段内需求是不能变更的（如果客户有变更要求可以说：现在离里程碑点已经不远了，先记下来，到那个时候一起讨论），所有团队成员全力以赴达到里程碑目标。

最后：在每个里程碑点上，除了传统的检查评审以外，还需要加上一项，就是与客户一起主动寻求新的需求，进而制定新的计划，化被动为主动。这样一来，需求管理与项目管理之间的集成关系就变得清晰了，如下图所示。

从顺序上来说，计划制定上分三个层次来考虑问题：

首先考虑远期目标，先有目标分析，确定最终成功交付的一个稍稍模糊但是具有宏观指导意义的目标。远期计划不能没有，不然会形成一个目光短浅随需而变的开发；也不能太详细，否则会形成靠猜测形成的“玻璃柜计划”，既无法更新，也无法应对需求变更。

然后是中期目标，也就是里程碑目标，根据需求的优先级，设定每个里程碑需要交付的功能，也就是说，阶段计划关注的是该点的结果，而不是活动。

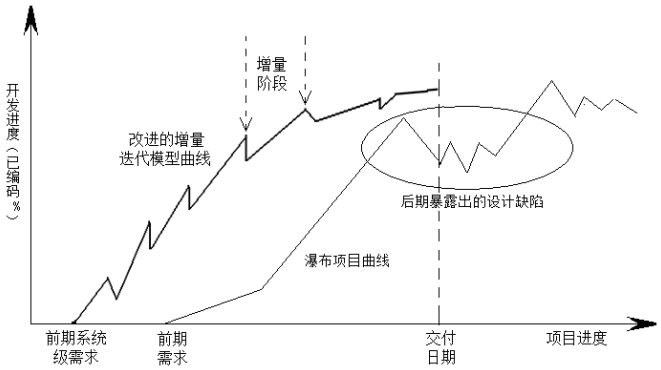
最后才是近期目标，以目前的阶段（很多情况下还包括下一个阶段）为界，进行详细计划，包括这个阶段的任务和活动，其中还需要对需求进行细化分析，这就可以根据现有的状况来应变，使计划可以落实。

从宏观上说，一个阶段的实施是动态的，而里程碑评审是静态的。对于变化而言，动态过程中实施变化要比静态过程实施变化难得多，所以坚持了这个原则，就有可能在动态实施与静态变化之间实现了良好平衡。

2）增量迭代模型所解决的问题

上述增量迭代模型实施了动态与静态的平衡。由于增量阶段的实施是动态的，而里程碑评审是静态的。对于变化而言，动态过程中实施变化要比静态过程实施变化难得多，所以坚持了这个规则，就有可能在动态实施与静态变化之间实现了良好平衡，很好地在反馈中消除风险。

很多企业的工作现实都告诉我们这样的场景：初期为了收集需求耗用了大量时间，严重推迟了项目真正开始的时间。当项目做到最后开始集成的时候，各个组件的接口和行为的不一致才被发现，开发团队被迫修修补补（重新设计根本没可能）。为了对这些不一致的修补进行测试，又耗用了更长的时间。结果最后交付的是一个迟到的、超预算的、脆弱的、维护起来极其昂贵的系统。我们需要问一下，过去我们是不是经常发生这样的情况？如果是，那采取什么方案来解决？下图表示了上述改进的增量迭代模型和瀑布模型在开发进度上的统计区别。

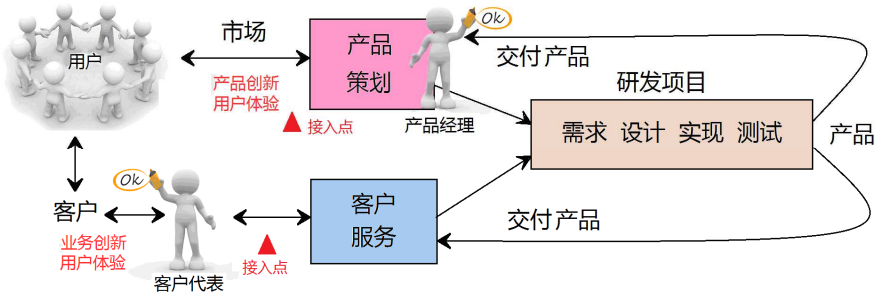


而增量迭代模型是一个受控反馈过程，通过每个增量周期的反馈、调整，就好像在瞄准运动中的目标，在早期就不断修正开发的方向，避免了后期大规模的返工。

二、需求开发的过程框架

1，是项目还是产品

有的企业为了管理方便，把单一客户和非单一客户分称为项目和产品，作为狭义表述不是不可以，但是站在更高一点来看可能会看得更清楚，如下图所示。



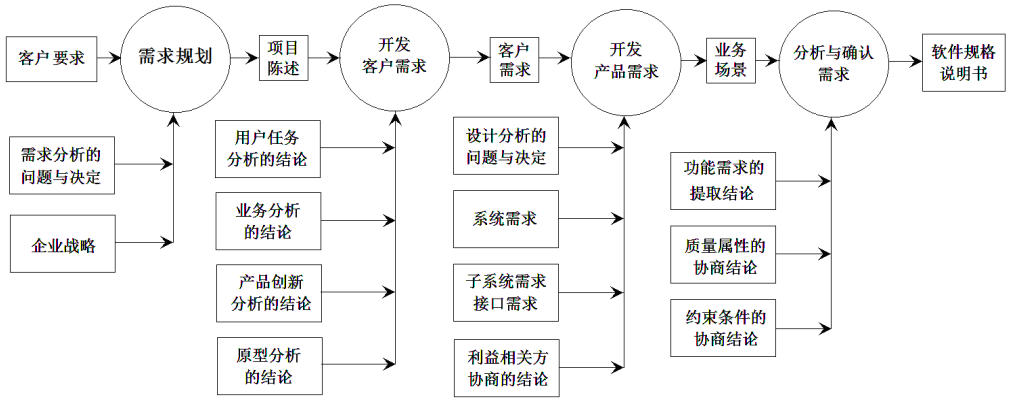
从产业链的角度来看，作为产品部门，需要用创新精神来策划面向市场的产品（产品创新、用户体验），并且用产品经理作为和开发方的接口；作为有明确需求的项目，客户方同样需要用创新精神来进行产品策划（业务创新、用户体验），并且用客户代表作为与开发方的接口；所以这两者只是接入位置不同而已，并没有本质的区别。

需求工程的关注点，是指进入项目研发阶段的需求，对于研发项目的需求分析人员来说，当然会有所侧重，但能力要求也并没有什么不同。在需求工程里所谓客户需求，指的是理解并明确客户需要；所谓产品需求，指的是定义研发项目要交付的产品需求（这个用语源自 GJB 5000A），并且作为后期测试和交付的依据。需求分析的基础是理解客户的思维，所以本课程会花一定的篇幅，讨论产品策划的需求收集有关问题。

我们所做的每一项工作，都是产业链的一个环节，下游为上游服务。从某种意义上说，服务对于上游来说，专业性更强（看一下：理财、基金、银行、保险、法律），但又严重受上游需求的影响。这个世界很有意思：下游服务受制于上游需求，但上游也会受下游服务规则的制约。有时候下游会很强势，这取决于下游的专业能力和服务的价值，整个世界就这么实现了平衡。理解这点很重要，本课程的基本观点，就是依据这个模型形成的。

2，需求开发的顶层过程框架

通过对需求开发总体目标的理解，我们可以建立需求开发的顶层过程框架。这个过程框架是需求开发行为的整体轮廓，也给我们的课程提供了一个总的思路。如下图所示。



1）需求规划

任何项目的开始之初，最重要的事情就是对需求进行整体上的规划：也就是站在客户的角度，明确产品的目标，对项目描绘出一个清晰的轮廓，并且力争在整个项目开发周期中，这个轮廓是稳定的。这个站在整个系统角度的相对抽象的需求，也为前期的架构设计打下了基础，并为后期细化需求建立了一个初始的框架。

2）开发客户需求

在这个阶段，需要对所确定的业务领域进行的工作进行学习，理解他们正在做以及他们希望做的事情，理解了才可能沟通。需要分析团队向用户（包括易用性专家、安全专家、操作人员等）咨询，使我们知道产品将来工作是什么样子的。

在上述沟通的基础上，站在客户的角度形成客户需求。这个需求反映了组织机构或客户对系统、产品高层次的目标要求。它描述了诸如范围、要求、应该完成哪些工作以及约束这些高层次问题。客户需求一般表述比较抽象，缺乏很多必要的细节。

3）开发产品需求

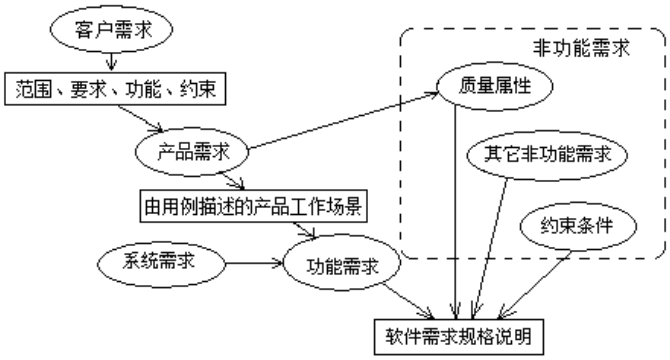
站在产品本身的角度，描述为了达到客户需求，产品应该是什么样的？这种需求需要考虑产品的运行方案，并且详细描述每个产品部件的工作场景，在大多数情况下，这可以在用例（use case）文档或方案脚本（scenario）说明中予以说明。

4）分析与确认需求

分析每一个工作场景，消除需求的模糊之处，形成在开发、测试、质量保证、项目管理以及在验收中都起关键作用的“软件规格说明（SRS）”。这个规格说明包括如下几个内容：

- **产品目标**：描述产品所解决的问题和必须达到的目标。
- **功能需求**：从产品部件的工作场景中抽取出来，清晰而无歧义的定义开发人员必须实现的软件功能，使得用户能完成他们的任务，从而满足了客户需求。包括：产品需要哪些功能？这些功能是为解决哪些问题？功能与问题之间的追踪是什么样的？功能需求如何测试？如何验收？等。
- **非功能需求**：描述了产品必须遵从的标准、规范和合约；质量要求；设计或实现的约束条件及质量属性。所谓约束是指对开发人员在软件产品设计和构造上的限制。质量属性是通过多种角度对产品的特点进行描述，包括：产品需要遵守什么样的质量标准？易使用性、易扩展性、性能是什么样的要求？如何度量？如何验收？等。

依据这个需求开发过程，就可以保证需求便于阅读而且具备很强的层次感，在不同阶段表达问题的重点也不同，并且能够在发展过程中不断演进。其各部分关系如下图所示。



3，编写便于阅读的需求

文档既是沟通的工具又是沟通的成果。无论需求是表示成用例集、功能列表、特性列表或者是其它的形式，都必须形成文档。一个由多人参与的工作不可避免的存在沟通问题，这就需要获取一份能够被复审和批准，大家都认可的，作为约定的文档。对于任何大型复杂的项目，不可能没有需求规格说明就盲目的开始项目开发，关键是这种规格说明的粒度要合理。

完成项目的障碍往往来自于对文档的理解。由于利益相关方的专业背景、知识结构以及所关注的问题各不相同，文档的编写就需要考虑到这个情况。我们必须使需求演化具有可视性，以模型和文字相配合来描述需求将会减少人们沟通的障碍。需求的描述应该是平易近人的，以一种与技术无关的方式表达，这样做的目的是为了便于理解和沟通，使利益相关方的参与成为可能。

需求的编写应该便于阅读并且与阶段性的关注点保持一致。长篇大论事无巨细汇集到一起的需求不容易阅读、不便于开发、更不能发挥应有的作用。整个需求编写过程有点像编写书的提纲，一本书有部、章、节三级提纲，后面的二级提纲会随着一级大纲的推进而逐步展开，而三级提纲将随着二级提纲的推进而逐步细化。

需求的编写与汇集应该进行合理的分解，清晰的分离开如下三个层次：宏观层面的系统级需求、聚集层面的子系统需求（包括子系统接口）、细节层面的产品部件需求（包括模块接口）。不同层面的需求应该分开书写，并且自上而下逐步分解和求精。需求的层次关系要很清楚，要注意下层需求对于上层目标的支持关系。这种把不确定性因素逐步明朗、由粗变细、循序展开的方法，可以同时兼顾需求的刚性和弹性、前瞻性、反馈性、准确性等诸多要求。

第二章　需求规划：产品的目标与轮廓

在研究软件需求分析的时候，如果仅仅关注一些具体的技巧，并不可能真正的理解需求分析的精髓，如果把眼光放得更高一些，更关注一下与之相关的方方面面，往往更容易看到本质。任何项目的开始的时候，需求分析最重要的就是对项目规划一个清晰的轮廓，并且力争在整个项目开发周期中，这个轮廓是稳定的。在这个阶段的分析中要注意避免的是过分关注细节，只见树木不见森林，造成出现不应有的漏洞。

2.1 理解客户：从宏观的角度理解客户需要

如何才能沟通？关键是要立足自己的专业性，站在对方的角度想问题。因此，需求分析人员最本质的工作就是理解客户的需要，并且建立从客户到实现之间的桥梁。这个桥梁如此重要，以至于需求分析成为软件是否成功的关键。但是，客户所提出的需要真是需要吗？

大多数情况下客户提出的要求都是含糊不清的。为了理解客户为什么要提出这些要求，就需要站在更宏观的角度，理解客户所遇到的问题，因为只有解决问题的产品才是好的产品，这就自然把问题分析看成需求开发的第一步。

一、关键问题：客户为什么要做这个项目

任何研发项目要取得成功，需求分析人员首先要做的就是理解客户。而理解客户的首要问题，就是先理解他们所做项目背后的“为什么”。当理解了“为什么”后，团队就能够：

- 做出更好、更加明智的决策；
- 更好地平衡各种矛盾，并做出更好的取舍；
- 由于他们被授权后可以独立思考问题，所以能够做出更好、更具创造性的方案来。

但是为了解决“为什么要做这个项目”的背后，是客户希望解决什么问题。

1，一切需求都开始于问题

对于客户来说，需求分析人员的价值在于可以基于技术的背景下，站在客户的角度思考问题。任何项目的动因都来自于对问题的解决，而项目的价值也来自于对问题解决的有效程度。如果一个项目所针对的问题不是十分清晰，则项目的价值就会受到质疑，这就是问题分析的任务。

1）问题中往往蕴含着机遇

在问题分析阶段，分析人员需要充分研究和理解问题域中存在的问题、机会和约束条件。从某种意义上说，问题和机遇是一个问题的两个方面，只有发现了问题，在解决问题中就可以获得新的机遇。但是，表面上呈现的问题往往并不是真正的问题，仅仅罗列问题并不可能带来问题的有效解决方案。为了有效分析问题，就需要分析、理解和归纳相关问题领域。要知道通常一个问题可能会有很多种解决方案，我们的工作就是找出最合适的问题解法。

2）问题往往来自于用户的感觉

问题分析的源泉来自于哪里？事实上问题往往来自于用户自己的感觉，如果觉得什么事物是个问题，那它就是一个问题，值得我们花时间去研究。但是感觉并不能代替解决方案，分析师的任务，就是帮助用户分析问题，并且提供解决这些问题的方案，这中间很可能需要改变用户本来觉得划算的办法。例如：用户本来觉得需要构建一个系统，但是您很可能告诉他通过一次培训就可能解决这些问题。

通过分析可以缩短期望的和感觉之间的差距，分析团队应该尽早介入问题分析阶段，以期给项目的相关人员提供足够的、正确的信息。问题分析阶段的目标是在具体立项之前就对要解决的

问题有一个更好的理解。就像做任何复杂练习题一样，我们必须从心中的目标开始。

3）研究问题领域

人们的通病是，总是隔着窗户想象我们的邻居到底是什么，这种想象又太多的夹杂着自己的背景和生活经验。所谓站在客户的角度想问题，那就需要把自己的思维空间向前方移一步，实实在在的转移到客户端，深入进去，看看客户到底发生了什么？这不但加深了理解，增加了沟通的机会，也避免了以主观想象代替客观现实。

罗列问题是在问题定义上达成共识的方法，就是把问题写下来，看看是不是每个人都能同意。在这个过程中，理解某些问题解决方案的优点通常很有用，用户对优点的描述通常可以给问题提供额外的背景，从用户的角度理解了解决方案的优点以后，我们也就更好的理解了相关人员对问题的看法。这个阶段通常包含如下任务：

分析人员可以通过系统所有者、用户、和其他有关人员的参与，从不同的侧面，不同的详细程度，不同的表述方式，不同的感知，不同的观点详细研究这个领域的业务、机会、指示和约束。不要吝啬开会，在白板上讨论并且用数码相机拍下讨论的过程十分重要，初期的准备是必要的，但讨论的结果和初期的准备会很不相同。作为一个严肃的项目，需要正规的项目初步分析文档送管理层讨论，要注意到管理层所关注的是对业务的支持，以及由此带来的经济效益，而不是要人头疼的技术手段。

4）分析问题和机会

关键性的话题：在什么情况下项目才算成功？问题陈述与分析也是管理层最关心的内容，主要包括：

- 当前组织业务方法上存在什么问题？
- 这些问题是当前的关键问题吗？
- 这些问题是如何影响业务的发展的？
- 有没有可度量的数据支持这个观点？
- 这些问题是不是一定需要某种信息系统来解决？
- 解决了这些问题对业务发展的意义何在？
- 加入了信息系统对业务流程会有什么样的影响？
- 信息系统的加入会如何改变目前的工作方式？
- 系统能提高组织的工作效率吗？提高率是多少？
- 信息系统对业务的贡献度是多少？如何度量？

问题分析是站在客户的角度，而不是技术的角度思考问题。可度量的问题收集和分析过程，将会为下一步构思某种解决方案打下了坚实的基础。

5）理解根本原因：分析问题背后的问题

对问题进行分析的时候，我们经常会发现人们往往把病症当成病根。所以问题分析很重要的一点就是理解问题的根本原因，找出问题背后的问题。一定要注意的是不要把表面存在的现象当成问题，我们必须关注真正的问题到底在哪里，以期找到业务上的问题症结所在。

通过对关键业务过程的分析可以帮助我们理解问题，但这种分析应该避免对信息技术本身问题的关注，可以提出这样的问题：“如果目前的流程是完美的，还需要这个信息系统吗？”

客户或最终用户提出的需求，通常是他们心目中可行的解决方案，并不是问题唯一的解决途径。有时候各方面表面上的不一致并不一定是真正的不一致。

F-16 战斗机设计师哈里.希拉克尔（Harry Hillaker）就给出过一个经典案例：

军方最初对 F-16 提的需求是 M=2～2.5 之间的低成本轻型战斗机。要知道速度从 M=1 到 M=2 时，空气阻力会提高 4 倍。考虑到对动力、机身重量等的苛刻条件，即使到今天，这样的需求也不是容易的事。

真的需要如此高的速度吗？设计团队追问军方，得到的回答是：为了迅速撤离战场。

为了解决这个问题，是不是有其他的解决方案？例如：降低一些速度，提升推理重量比，改善战斗机的加速性和机动性，能不能以灵巧型取代最初对速度的要求？这种考虑造就了 F-16，并形成整个第三代战斗机的设计理念。

软件开发也可以借鉴这条经验，需求开发只是客户要什么就是什么，并不能形成真正对开发有帮助的需求。通过询问客户，分析客户要求的功能真正的意义，来定位真正的问题，从而提出比客户原有建议更好，成本更低的解决方案。通过关注问题的真正含义，理解需求的轻重缓急，把最有价值的需求摆在第一位。

2，现场体验与领导意图

无论从任何方面来说，没有调查就没有发言权都是正确的。仅仅依靠道听途说或者客户的介绍，能和客户一起深入探讨问题吗？很多情况下，能够进行现场体验，是一个很可贵的机会。

1）能不能到现场去体验

为什么很多情况下开发方与客户的沟通会存在障碍？关键是双方掌握的信息存在差距。无论在会议桌上如何交流，毕竟还是凭想象在理解。凭想像去理解为什么这么做是一回事，但彻底搞清楚又完全是另外一回事。为了能真正地了解客户的需求，只要有机会，深人工作现场是一个很好的办法。

深人工作现场意味着要让团队离开工作间，到事件的发生地去实地考察。

举个例子，如果团队正在为矿山的施工单位设计安全工作许可证系统，客户怎么解释可能都结束不清楚，那能不能去矿山现场看看？如果与安全负责人混在一起，见识一下拖车，观察一下狭窄的工作条件、古怪的网络连接以及有限的客户工作空间。花上一天时间在场地内与那些将要每天都使用你系统的员工一起工作。参与进来，提出问题，变身为你的客户。这样的结果，需求人员往往更有发言权，也更容易拉近讨论的距离。

有这样一个经典案例：

丰田希望为北美市场设计一款新的车型，负责重新设计的首席工程师希望针对北美市场的特点改进设计。初期的需求很模糊，也很难看出在北美销售的车型到底应该有什么特点。

为了体验北美人如何利用车辆来生活、工作及娱乐的，项目首席工程师开着一辆原型车走遍了美国、加拿大和墨西哥所有的州、省。下面是他的发现。

- 北美的司机在车里的吃喝频率要比日本司机高（在日本，驱车距离一般要短一些）。正是由于这个原因，在新的车型中，需要设置一个中央托盘和 14 个杯槽。
- 相比美国，加拿大道路上的路拱（道路中间较高）要更高一些，因此在驾驶过程中如何控制“漂移”就非常重要。
- 在加拿大的安大略省，侧风非常严重，因此侧风稳定性要重点考虑。在任何强侧风地带，驾驶新的车型都要做到更稳定，更容易操控。

尽管这位首席工程师也可能会从市场报告上了解到这些问题，但如果没有亲力亲为，他就无法取得像现在这样深刻的理解和判断。任何事情的成功都离不开亲力亲为，我们现在需求分析中的很多问题，都和自己坐在办公室中凭空想象有关。

2）发现领导意图

当有一群相左意见时，谁能作最后的决定？这就需要在需求分析的前期发现领导的意图。领导意图是一种简单的表述或者声明，也是对项目的问题、目的或使命的一种总结。在产品开发的紧要关头，这种声明可以帮你做出重要的决定。

领导人对项目的意图可能并不是非常远大，也可以非常简单。但可以帮助我们理清为什么要讨论这个问题，然后向客户求证是否抓到了要点。领导者考虑问题的角度可能比具体实施人员看得更高，对问题的看法也可能更确切，不要忽略领导意图，这往往是项目成功的关键。

3，从行业和产品两个视角分析问题

很多情况下仅仅就事论事的分析，往往并不容易看清问题的本质，如果站的高一些远一些，你会发现现在的问题在过去可能并不是个问题。想想看银行业这 30 年来到底发生了什么？再想想军事领域这 30 年来到底发生了什么？我们就更容易看到问题的本质。

1）注意行业先于客户

用户的需要往往和他们所处的行业发展走势有关。当我们对产品构思一片迷茫的时候，我们可以通过三个方法，站在巨人的肩膀上登高望远，把出路看透彻：

- 比较行业历史来发现规律；
- 用经验加综合分析推演未来；
- 从竞争战略的角度去归纳未来的大事。

当我们面对客户含糊不清的需求表述的时候，我们心中的那种困惑可能会促使我们来进一步思考，到底什么才是真正的产品之源？

对行业的理解，对于我们思考应该开发什么样的产品是非常有意义的。针对我们需要为其开发软件系统的产业，我们从过去，看到现在，再看到未来，我们就会知道我们所做的每一件事情的意义，而不是仅仅蒙着头用户怎么说我们就怎么做。

第一流的分析师往往善于引导客户需求，而不是仅仅询问客户需要哪一种产品。他们要做的是依靠教育大众、与大众沟通、设法创造这种产品的市场。这些不断推出的新产品将会极大的影响了人们的生活和社会观念。

需求分析师不应该只是客户的记录员，更不应该以山寨版为荣，而是需要有创新思维，力争创造伟大的产品。为此，我们首先需要深刻理解行业，以洞悉产品演化大势。然后再调查市场来体会需求，在行业价值链、商业模式、以及战略竞争层级上，构思出一个让客户震惊的伟大产品。

2）产品是一个战略

除了关注行业以外，还要注意到新产品的构思实际上是一项战略考虑，软件产品一旦完成，它的生命周期将希望比较长，并通过不断的升级并获得更多的客户认可来创造品牌效应。任何一个公司开发新的软件系统需要回答四个重要的战略问题：

- 是卖产品还是卖服务？
- 是面向大众市场还是面向专用市场？
- 面向的是水平市场还是垂直市场？
- 如何持续获得商业利益？是什么样的盈利模式？

所谓战略竞争就是创造差异性，以创造一种独特的价值组合。我们可以从三个视角来思考这个问题：

第一视角：定位

企业常说的话是定位者生，无定位者死，善定位者王。例如我们是泛泛的做一切项目，还是根据目前情况，专注于某各方面？这种选择往往决定了企业的生死。

第二视角：取舍

要在竞争中选择不做哪些事情。例如，到 1985 年，尽管英特尔公司已经把 CPU 用于 IBM PC，但他们的主要业务还在存储器上，但来自日本的低价存储器压力使他们步履维艰，1985 年安迪.格鲁夫力排众议，坚决砍掉了存储器生产，全面转型微处理器，这种战略取舍使公司获得了今天的成功。

第三视角：适配

企业各项运行活动之间适配，以创造最核心的竞争优势。

所谓战略实际上就是对如何开展竞争的问题作出清晰的选择。上述所有这些思考可以帮助我们站在更加宏观的角度，思考这款软件应该具有什么样的特性，形成具有创新意识的产品轮廓。

需求分析的成功关键是合作，而只有理解了合作才有可能产生。理解了“为什么”，才有可能

进一步回答“怎么来做”，所以这是分析的关键一步。

二、流程再造：需求的来源与基础

良好的需求收集过程首先需要站在对方的角度想问题。从行业的角度理解客户为什么要做这个产品，就要看到 IT 需求更深层次的动因，是来自于技术进步引发的新一轮竞争环境的需要。这种竞争环境促使人们反过来审视已有的业务方法，重新梳理业务流程，以保证企业的经营战略、组织结构、业务流程、IT 技术架构保持战略上的一致性，从而提升竞争力，这就是业务创新。

1，BPR 的产生

企业之所以取得成功，是因为具有从头到尾比竞争对手做得更好的“战略素质和能力”，而信息技术是提升企业战略能力的有效手段，它实现了流程中各点能力的提升。在初始宏观分析的时候，分析师如果能站在这个角度提出问题、分析问题，将会把问题看得更深入和确切。

BPR（Business Process Reengineering，业务流程再造）是美国麻省理工学院计算机教授迈克尔哈默和 CSC 管理公司的董事长钱皮，依据提升企业的战略能力的关键点，于 20 世纪 90 年代提出来的。

BPR 是最初的动力，是 1980 年代世界经济的萧条促使了群体反思。人们认为：为了飞跃性地改善成本、质量、速度等现代企业主要运营基础，必须对流程进行根本性改革。世界进入了 21 世纪，随着信息技术的蓬勃发展和普及，社会环境和企业商业现实发生了翻天幅度的变化。信息技术的加入，给流程再造带来了环境和动力，它深刻影响了企业的业务方法和业务流程。

BPR 是对企业僵化、官僚主义的彻底改革，在美国也被称为“恢复美国竞争力的唯一途径”，并已经取得了良好的成果。

2，BPR 包含的内容与特点

BPR 主要包含以下几方面内容：

1）企业流程再造原则

这些基本原则指导了业务流程再造，包括：

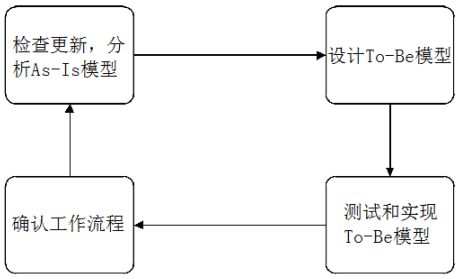
- 组织领导能力；
- 顾客至上；
- 面向流程；
- 以人为本。

2）企业流程再造的过程

通过一系列的活动和它们的内部关系，描述了如何进行流程再造，包括：

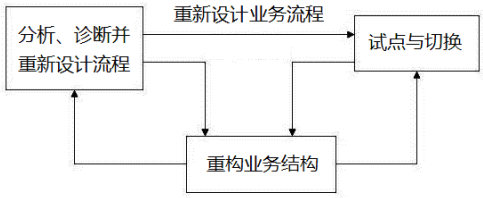
- 确认企业流程再造目标；
- 组建流程再造团队；
- 获得现有流程系统描述；
- 识别再造的机会；
- 完成模拟分析；
- 制定转变计划；
- 完成新建设计流程系统；
- 维护系统。

BPR 过程的总体其循环流程如下图所示：



BPR流程循环图

首先梳理当前业务（As-Is）模型，分析这个模型的缺点，考虑在信息化的条件下，这个模型可以怎样改变，从而生成未来（To-Be）业务模型。提出与这个模型相适应的 IT 需求，测试和实现这个 To-Be 模型，在验证的基础上确认工作流程。根据流程应用的状况，发现新的问题，再继续改进。也就是说，业务流程再造是基于实践的，可能会在每个节点上都会进行多次反复，而且是个不断改进的过程，如下图所示。



当然对于需求分析师来说，并不一定要成为业务流程再造专家，但是，如果分析师在这个方面的知识积淀是深厚的，对于其中牵涉到的一些概念和方法是清楚的，那么就可以帮助我们更好的与客户沟通，以减少与客户之间的知识壁垒。通过和客户一起进行更高水平的讨论，双方共同梳理思想，就可以使未来产品的轮廓变得更加清晰。

3，从一个案例看 IT 支持下的流程再造

下面通过福特公司的北美汽车公司进行的一次成功流程再造案例分析，可以使我们对 BPR 有个概貌的了解。

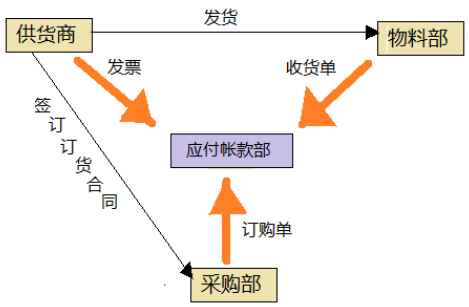
1）案例背景

上个世纪 80 年代初，福特像许多美国大企业一样面临着日本竞争对手的挑战，正在想方设法削减管理费和各种行政开支。促使福特公司认真考虑“应付帐款”工作的是日本马自达汽车公司。

马自达公司是福特公司参股的一家公司，尽管规模远小于福特公司，但毕竟有一定的规模了。根据瓶颈分析，他们发现本公司负责应付帐款工作共有 500 人，效率非常低下。而马自达公司负责应付帐款工作的只有 5 个职员。5：500，这个比例让福特公司经理再也无法泰然处之。

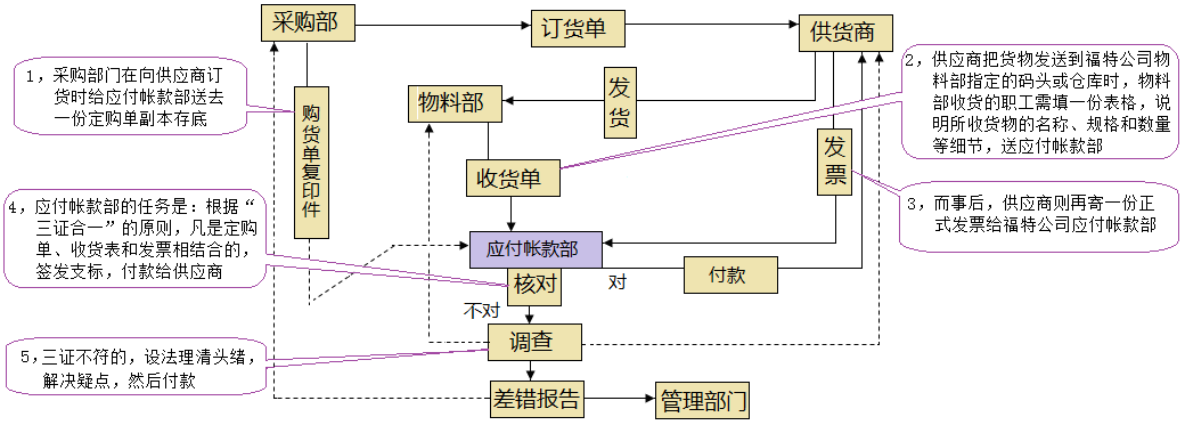
2）当前业务（As-Is）模型

福特汽车公司首先梳理当前业务（As-Is）模型，分析这个模型的缺点，原来的采购付款原部门职责如下图所示：



再造前应付帐款部的采购流程为：

该部有 500 多名员工（北美），负责审核并签发供应商供货帐单的应付款项，其业务流程图如下图所示。



说明：

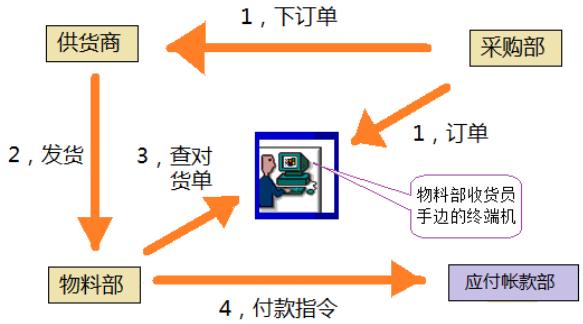
- 采购部门在向供应商订货时给应付帐款部送去一份定购单副本存底；
- 供应商把货物发送到福特公司物料部指定的码头或仓库时，物料部收货的职工需填一份表格，说明所收货物的名称、规格和数量等细节，送应付帐款部；
- 而事后，供应商则再寄一份正式发票给福特公司应付帐款部。
- 应付帐款部的任务是：根据“三证合一”的原则，凡是定购单、收货表和发票相结合的，签发支标，付款给供应商。
- 凡是三证不符的，设法理清头绪，解决疑点，然后付款。

分析表明，福特公司应付帐款部最大的工作量是解决“三证不符”应付帐款问题，应付帐款部是整个材料供应效率的瓶颈的，其工作完全符合帕累托 80/20 规则，即，其 80%的工作量全是由于 20%的“三证不符”帐款造成的。

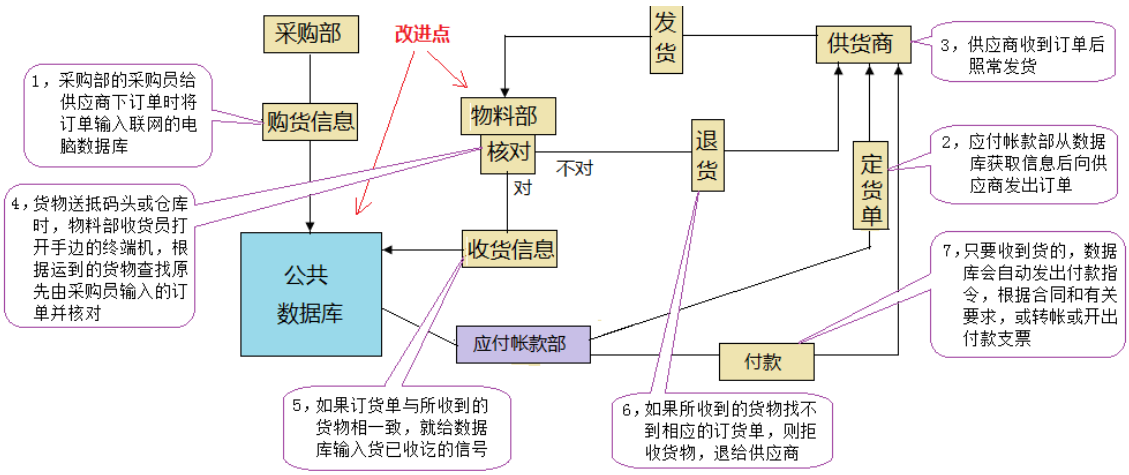
3）第一次未来（To-Be）业务模型

建立当前业务模型（As-Is Model）不是目的，只是研究业务问题过程中的一个环节。我们的目的是在分析的基础上，建立适应新产品的业务流程，被称之为未来业务模型（To-Be Model）。

采购付款流程的改进在于利用了信息技术，实现了通过数据库实现了信息共享。为了简化流程，改变了付账部的工作职责，合并工作组，把核对的职责由“付账部”转移到了“物料部”，提高了效率，其职责如下图所示。



具体再造的业务流程如下图所示。



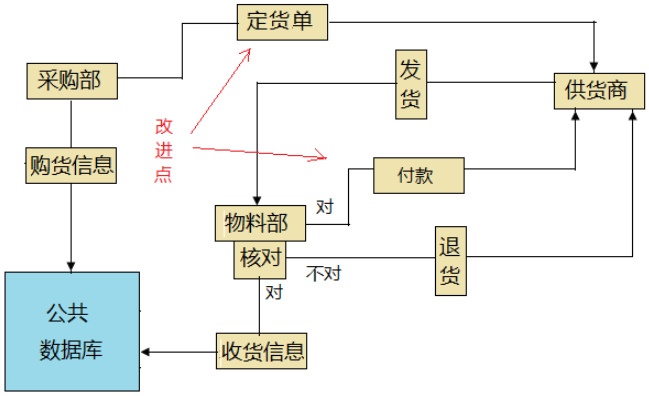
说明：

- 采购部的采购员给供应商下订单时将订单输入联网的电脑数据库。
- 应付帐款部从数据库获取信息后向供应商发出订单。
- 供应商收到订单后照常发货。
- 货物送抵码头或仓库时，物料部收货员不再填写表格了，而是打开手边的终端机，根据运到的货物查找原先由采购员输入的订单并核对。
- 收货员的任务很简单，订货单与所收到的货物相一致，就给数据库输入货已收讫的信号。
- 如果收货员所收到的货物找不到相应的订货单，则拒收货物，退给供应商。
- 只要收到货的，数据库会自动发出付款指令，根据合同和有关要求，或转帐或开出付款支票。

由于转移了职责，流程再造后应付帐款部由原来 500 人减到 25 人，而且不再负责应付帐款的付款授权。

4）第二次未来（To-Be）业务模型

在业务运行过程中，人们发现付账部的工作量越来越少，相应工作人员完全可以并入物料部，组成一个专职工作组来承担。这就导致了第二次流程再造：取消付账部，进一步简化管理层级和流程，致使工作效率大幅度提高。最后，得到了业务流程再造的最后的结果，如下图所示。



这个例子非常深刻的反映了信息技术的发展，对于传统组织设计和流程设计的冲击。

很多企业的业务部门在梳理业务流程的时候，总是希望不改变现状，只是用软件来复制当前的业务流程，问题是这样的“信息系统”除了给企业带来麻烦还能带来什么呢？

三、业务流程再造的基本工具

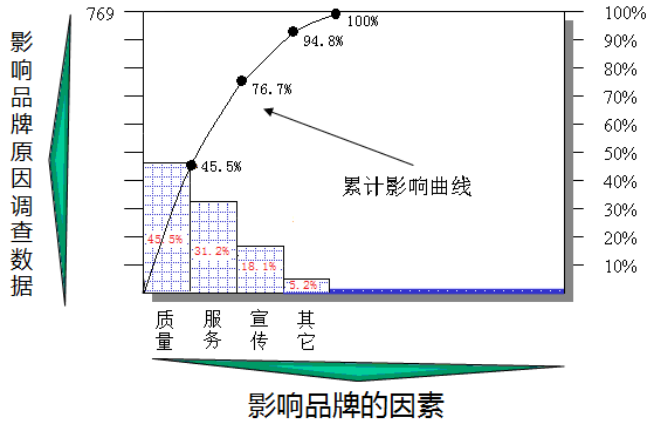
自上个世纪 90 年代到 2000 年以后，由于信息技术的发展，推动了各类企业流程再造的进程，

并且形成了一些行之有效的方法和工具。主要有如下几种方法综合应用。

1，Pareto（帕累托）分析

在业务流程再造的时候，经常需要发现影响事物的主要方面，这可以使用 Pareto（帕累托）分析方法。Pareto（帕累托）分析是一种统计科学方法，可以用于筛选在业务流程中对整体效果产生显著效果的有限数量的任务。它采用著名的二八原则（80%的任务可以通过 20%的工作来完成），在质量的改善和优化当中，大部分的问题（80%）是由几个关键的原因（20%）导致的。这就是“关键的少数、次要的多数”原理。

Pareto 图按事件发生的频率排序而成，它显示由于各种原因引起的问题数量或不一致的排列顺序，从而找出影响质量的主要因素，这才能有的放矢，采取正确的措施。



影响主要因素通常分为 3 类：A 类为累计百分数在 80%范围内的因素，它是主要的影响因素。B 类是除 A 类之外累计百分数在 80%-90%范围内的因素，是次要因素。C 类为除 A、B 两类之外百分比在 90%-100%范围的因素。因此 Pareto 图又叫 ABC 分析图法。

Pareto（帕累托）分析的目的是在大量可能的行为中筛选出少数具有重大意义的，能够增强提升效果的部分，通过优先级的分析来安排任务权重比，最终决定在实际任务进程中哪些问题最需要被考虑到。

2，瓶颈分析法

依据生活中的体验，我们可以发现整个工作系统发生超负荷运行（例如堵车、春节期间购票系统堵塞），并不是处处都超负荷的，而大部分问题是出在几个节点上，这就是瓶颈。发现这些瓶颈形成的原因主要有以下几种：

1）流程在执行过程中资源分配不均衡

流程中的某一个环节由于人员数量投入不足使其操作的速率低于其他流程环节的速率，那么它就会影响到整条流程的执行。解决的办法就是纵观整条流程，重新规划、分配对整条流程的资源的投入。

2）由于流程之间资源抢占冲突而产生的瓶颈

如果一条流程为下游的若干个流程提供输出结果，下游的流程之间就很容易为抢占上游的流程输出结果而产生不必要的冲突，并形成瓶颈。那么，新的 IT 流程是如何克服这些瓶颈的？克服了这些瓶颈真的能进一步提高效率吗？它是怎么做到的？

3，SWOT 分析

SWOT 分析方法是一种企业战略的分析方法，就是根据企业自身的既定内在的条件进行分析，找出企业的优势、劣势以及核心竞争力所在。其中：

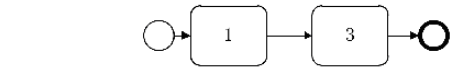
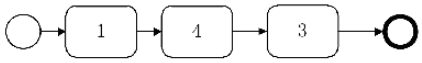
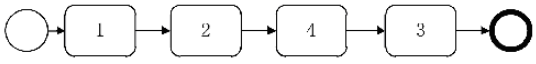
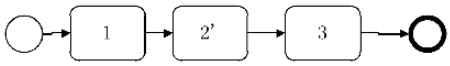
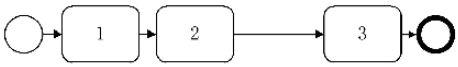
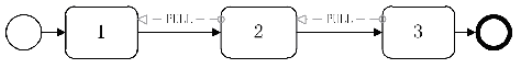
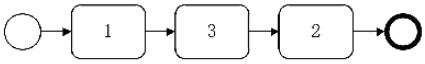
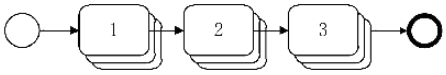
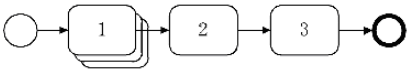
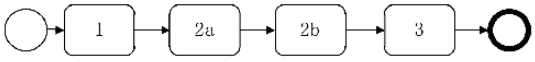
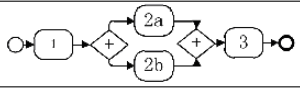
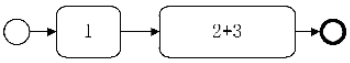
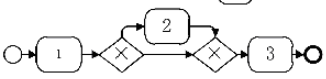
- S 代表 strength（优势）；
- W 代表 weakness（劣势）；
- O 代表 opportunity（机会）；
- T 代表 threat（威胁）。

在这之中，S、W 是内部因素，O、T 是外部因素。按照企业竞争战略的完整概念，战略应该是一个企业“能够做的”和“可能做的”之间的有机组合。进行 SWOT 分析的时候需要注意以下原则：必须对企业的优势和劣势有客观的认识，必须区分企业的现状和前景，必须考虑全面，必须与竞争对手进行比较，哪些方面优于或者是劣于你的对手。在分析的时候，需要保持 SWOT 分析的简洁化，避免过度复杂和过度分析，具体分析时因人而异，因项目而异。

4，流程改进模式

流程改进模式（Process Enhancement Patterns，PEP）是业务流程再造与优化理论当中重要的组成部分，下图展示了 15 种改进模式是当前被利用的用于改善 As-Is 模型的主要方法。

常用15种流程改进模式

名称	图示	描述
消除		去除没有价值的额外工作
替换		用更加合适的任务替换现有环节
插入		插入新的任务
改进		对单个任务进行改进
延迟		对单个任务进行延迟处理
回拉替换		实施回拉原则
重排序		改变流程顺序
巩固		用众多实例巩固到每个环节
单个巩固		用众多实例巩固到单个环节
水平解耦		将单个环节水平拆分
垂直解耦		将单个环节垂直拆分
合并		将多个环节合并
细化		为单个环节提供选择
并列化		将不同环节并行化
可选择化		将一个环节变成可选

流程改进模式（PEP）是最重要的进行业务再造和优化的手段和方法，PEP 主要用来对流程模型进行潜在的变更，对系统进行优化，以便展现出更好的效果。

四、从业务创新的角度理解客户

很多人往往关心怎么来写需求，其实写需求在整个需求分析中只是处在最后水到渠成的位置。就好比写文章，写作技巧有用但不是最重要的，最重要的是生活的体验与理解。如果没有这些理解，除了能写出某些神剧的东西来，还能写出什么有分量的东西来吗？因此，需求分析人员最根本的能力，是理解客户。

如何理解客户呢？我们会发现，从开发一个具体项目的角度来看，需求可能是相对确切的，但是如果把这个项目放在客户的大背景下来看，就可以发现这只是企业业务创新的一个环节。站在业务创新的角度来看，我们也才能理解为什么要做这个项目，以及为什么后期需求会变。

这里又回到一个根本的问题：是业务影响技术还是技术影响业务？

在需求分析中，业务是根本，技术是为业务服务的。但是技术手段影响业务的例子比比皆是，拿军事来说：冷兵器时代有冷兵器时代的组织方法，机械化时代有机械化时代的组织方法，而信息时代有信息时代的组织方法。无数历史事实证明，如果不考虑技术对业务的影响，在机械化时代套用冷兵器时代的组织方法，没有不打败仗的。

信息技术已经引发了一场革命，原来的业务模型是适应当时环境和技术背景下的模型，当技术环境发生变化的时候，业务模型必然要发生适应性改变。这个世界的一切都有可能因为信息技术的发展而变，这是一个不以人的意志为转移，而且不可阻挡的潮流。

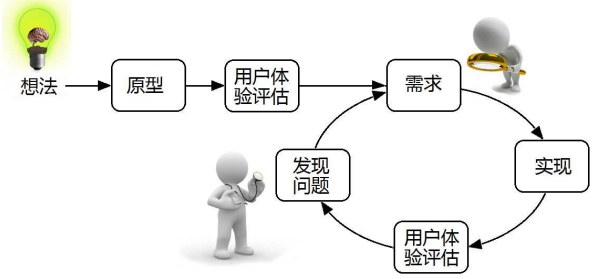
时代正在发生变化，我们的业务思想也要随之发生变化。问题在于这是一个不断发展的变革过程，同时也会引起组织结构和职责的变化。在这个过程中，如果软件设计是按照某个流程固态的完成的，那么当流程发生改变的时候，势必引起软件产品的重新开发。

如果软件是不容易改变的，结果本来应该是流程再造驱动力的软件系统，反而成为了业务创新的阻力，这是我们不希望的。这就要求在我们的分析中，必须把变化作为一个重要的要素来考虑。如果软件能够随需应变地搭配不同功能的需求，无需额外的工作能去响应业务条件的变化，企业就可以在市场上具备巨大的竞争优势，这一切都使需求分析的关注点发生了引人注目的变化。

2.2 产品创新：新产品规划的需求方法论

有一类产品并没有明确单一的客户，而是面对广大客户群的创新产品，这类产品往往还伴随着同类产品的剧烈竞争。这时需求的来源与收集方法就会有许多新特点。但是，与前面已经讨论过的基本原理并没有本质上的不同。我们同样需要把我们的视野向前移一步，站在客户（用户群）角度，认真调查研究一下这中间到底发生了什么？他们有什么困惑？他们喜欢什么？不喜欢什么？什么样的产品才能给他们带来更大的利益呢？

从产业链的角度来看，相对于单一客户的接入点而言，产品策划的接入点向前移了一步。而相对于产品研发来说，产品策划等同于客户。基于人们认识事物的螺旋上升法则，其关键之处在于整个过程并不是一次完成，而是一个不断发现与改进的螺旋迭代过程，如下图所示。



也就是说，在开发出最初的原型以后，需要把这个初期产品提供给一定数量选定的目标人群使用，并且收集反馈，发现关键问题，并从中提出新的产品研发方案。如此不断循环，使创新产品提升市场竞争力。那么，我们如何规划出有特色、有新意、有创造力的新产品呢？

一、新产品构思的创新方法

任何产品的创意都来自于构思，在创新产品构思的过程中，需求分析师应该具有其他人所没有的思想、眼光和感知世界的方法，必须突破已有的思维模式和行为模式，突破长期束缚自己的思维瓶颈，达到自己从未达到过的高度，这是产品创新的源泉所在。

1，产品创新的几个关注点

为了构思创新产品，在需求中考虑的问题不能仅仅为了技术而技术，技术上先进并不等于产品先进。我们设计任何软件产品都是为了解决客户问题，当我们面对产品设计问题的时候，需要不断的问自己三个 W：

- 为谁（Who）？
- 解决什么问题（What）？
- 为什么要解决这个问题（Why）？

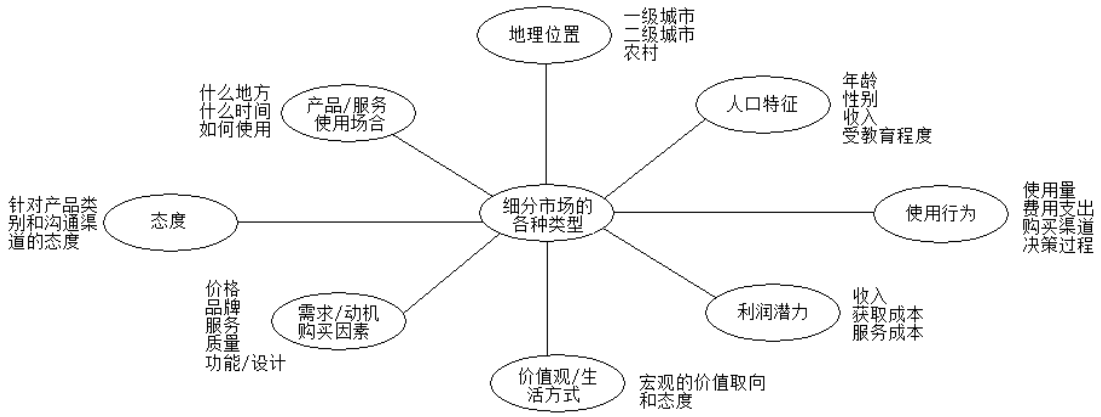
由此可能引发一个极具创新的产品概念。在这个过程中，分析师需要进一步延展自己的思维空间，站在客户的角度，利用创新的思维，仔细构思一下未来产品能不能做得更好呢？怎么才能做的更好呢？能不能为用户提供更多更好的服务呢？我们需要改变已有的仅仅为了完成任务的思维习惯，去和客户一起创作一个崭新的工作方式，有几个关注点需要我们把握。

1）理解客户思维

任何软件产品都是为人服务的，所以构思产品就需要对客户思维方式有更深的理解，这样才可能继续后面更深层次与更细致的业务研究。我们需要问自己的问题是：什么样的产品才是用户满意的产品呢？用户为什么会满意这样的产品呢？这就需要我们必须以市场为导向细分客户，市场思维要优先于客户思维。这里需要考虑三大选择：

- 满足哪些客户的需求？选择一个细分的市场；
- 满足客户的哪些需求？分析客户心理和行为模式，通过市场调研获取消费者需求、购买决策驱动因素、使用行为、满意度因子等信息，从而确定需求。
- 如何满足客户的需求？分析我们有哪些竞争优势，如何通过差异化产品设计及其服务组合，有效组织系统化创新。

在 IT 行业，使用最多的是麦肯锡的细分八法，特别是其中按照价值观和生活方式的细分，是当前最有效的细分方法，如下图所示。



细分完市场和目标客户以后，就需要仔细考虑客户的心理与行为模式，这里的客户包括个人的

或者是机构的，我们需要研究他们购买软件或者服务的决策过程，以及决定客户满意度的各种因素。

2）引导客户需求才是高手之道

一般认为：好的产品的特点是能鲜明而独特地满足用户需求，这称之为有 Focus（专注）；而伟大产品的特点是能引领或者创造客户需求，这称之为有 Vision（愿景）。伟大产品的设计最重要的是树立前瞻愿景，让我们产品的目标客户离不开它。为了达到这个目的，我们软件设计的目标也应该由输出产品变成输出行为方式。换句话说：仅仅满足客户需求是平庸公司所为，引导客户需求才是高手之道，因为这样才可能构建出名垂青史的伟大产品。

3）关注应用层面的创新

我们如果仔细观察最近几年软件世界都发生了什么，就会发现人们并不是把一个又一个新技术抛到客户面前，而是尽力利用原有的技术，反复精化优化，在应用层面提供丰富多彩的新概念，其中十分重要的改变就是软件设计要充分关注用户体验，因为用户体验是产品之魂，也是一个伟大产品最重要的要素。

研究用户体验很复杂也很重要，而且真正能做好这一点的企业并不很多。用户体验不是靠着震撼性技术创新来达到的，而是尽可能把众多不被重视的细节做好，就是说“重要的不是你能实现什么，而是你将怎么样来实现它”。

4）如何做一个和别人不一样的产品呢？

分析师需要时时问自己的问题是：到底什么是好产品，什么是伟大的产品呢？我们即将构建的 IT 产品，如何最大限度的满足用户需要呢？我们将如何做一个和别人不一样的产品呢？

好的产品构想已经不仅仅考虑如何实现功能，而是尽力创作一种新的应用方式，如何使用户以更简单更快乐的方式来操作这些功能？新的产品能不能简化操作？产品的操作能不能极其自然、便利、充满美感和快感？这些对于用户体验的考虑往往成为优秀产品的灵魂。

2，用户体验为什么重要

大部分人都会同意这样的说法：重要的是使用的产品能正常工作及使用起来也没什么困惑不解的地方。创新产品的特征与用户体验有很大关系，从来没有人抱怨某项东西使用起来太容易了！每个人都会有一些所喜爱的故事，这些故事涉及的多是某个东西特别好用或异常难用。

1）用户体验不仅仅是前端设计问题

很多人以为用户体验仅仅是一个前端设计问题，这是一个误解。产品设计中处处有用户的体验问题，即使是一个底层框架，它的上层用户使用起来方不方便？有没有可能出现误解的地方？这个框架是不是易于学习？这些都决定了这个框架是不是受用户欢迎。两个框架功能上是一样的，但是更容易使用、更少出错的框架往往比较容易推广。

有时用户体验可以意味着生与死的差别。比如，打开炉子上的标示灯或安装一个新的照明设备等。如果这些说明书被误解或误读，则很容易就能导致财产损失、人员受伤甚至死亡。居室之外常常能发生的类似情况如：让人困惑的街道标识，设计过于复杂或简单的汽车仪表板，或者容易引起人分神的设备（如汽车、导航系统或者手机等）。

2）用户体验决定了产品的成败

在商业环节中倡导用户体验往往与增加收入和（或）降低成本相关联。不少企业由于新产品的用户体验差而造成了损失；而有的企业则因为产品的一个关键卖点的易用性做得好而盈利。

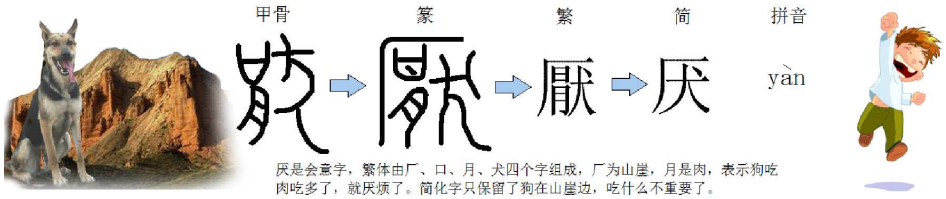
用户体验在我们生活中所起到的实际作用要远远比大多数人所认识的作用更广泛，而不仅仅只限于使用某个网站、某个软件或最新技术等。用户体验每天都在影响着我们每个人，它与文化、年龄、性别和经济阶层等变量交织在一起。

3）使用人群越复杂，用户体验越重要

随着产品越来越复杂，用户体验在我们的生活中起着持续上升的作用，当技术发展和成熟时，

它们就会被越来越趋于多样化的人群所使用。而使用人群越复杂，用户体验就越重要。

比如我们回顾一下文字的革新。在古代，文字是非常复杂的的东西，只有极少数精英才能使用或阅读它们。随着文字逐步为大众所使用，人们越来越需要更容易书写和辨识的文字，文字开始变得越来越简单。直至现代，有了简化字和拼音文字，文字变得更易于书写和阅读，它穿越不同的年代，一直沿着这个方向演变。



如今通过现代技术，文字已经变成了一种大众广泛使用的工具，成为越来越重要的文化交流载体。通过这个文字的故事，我们对于用户体验悟出了什么呢？

4）看似无关的想法组合在一起

站在用户的角度，有时候把看似无关的一些想法组合在一起，可以创造一个新想法。例如：把电话和照相机组合在一起，或者是把带有 GPS 功能的电话与语音驾驶行走向导组合在一起等，都创造了大量的新产品和新概念。

5）转换性思考

转换性思考是把一个想法应用到另外的领域，例如，把传统的拍卖销售转换为基于 Web 的拍卖销售。还有一个方法是消除约束，通过消除约束来探讨得到新的、不同的、也许有用的思想的可能性。由其它行业的实践触发的想法作为跳板，可以实现对原来熟悉的、显而易见的工作方式的超越。

3，用户体验的三个层次

让我们进一步研究一下用户的体验，我们会发现用户体验会往往成为一种品牌特征而存在。而用户体验又可以分为三种层次：基于物质的体验、基于情感的体验与基于环境的体验：

1）基于物质的体验

物质体验是基于一个产品和服务的核心功能或核心物理价值而产生，对于软件产品来说，包括价格、开发过程的标准化、产品和服务的功能与性能优势、产品的质量属性、技术优势、相关配套产品的成熟度、安装与售后服务等。

大部分用户将依靠这些信息做出理性的决策，所追求的是产品的物质利益。而且在这个范畴大部分用户从价格上来说都有趋低趋势，也就是追求价格最低而质量最好的同类产品，因为这些产品功能基本上都是同质化的，没有什么本质差别。也就是说物质体验决定了这个产品所必须具备的基本特质，但并不会给开发方带来很大的价格优势。

2）基于情感的体验

情感体验是基于产品引起的精神愉悦及情感方面的价值，包括人性化需求、对员工工作方法和工作模式的关注；应用这个产品所带来的附加感觉；对于企业品牌和消费层次的强调与表达；对于探索未知、挑战自我、追求完美的追求等。

这种对于产品带来感觉的追求，形成了用户体验的第二层次，这些内容也是产品最具差异性的东西，或者说这方面做得越好，产品的用户满意度就可能越高，产品的价格优势就越大。

3）基于环境的体验

环境体验是基于产品环境及外界刺激、购买产品时的整体体验、互动的个性化体验功能与体验价值，使用户产生蒙太奇般升华的兴奋感。这种体验是当一个人达到情绪、体力、智力甚至是精神的某一特定水平时，他意识中产生的美好感觉。它是主体用户对客体环境的刺激产生的内在反应，具有很大的个体性、主观性和不确定性。

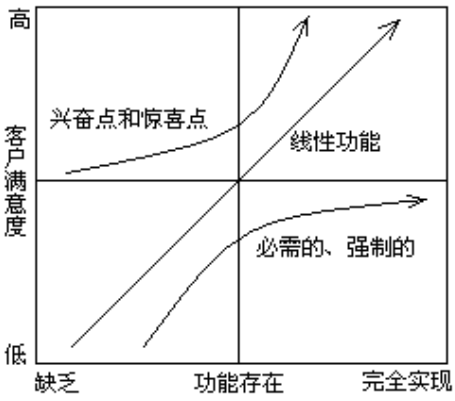
这种对格调和环境的追求，形成了用户体验的第三层次。例如，我们会发现很多顾客愿意花更多的钱购买一种优雅的格调、浪漫的氛围，而并不关注功能本身的内在价值。这种用户体验上的兴奋感，将会给产品提供无可比拟的竞争优势，关注这方面的用户体验将会给产品带来巨大的价格优势。

4）客户满意度的 Kano 模型

产品的价值很大程度上来自于客户满意度，因此需要研究产品功能与客户满意度的关系。最先提出产品功能与客户满意度规律的，是日本管理学家狩野纪昭（Noriaki Kano），他的方法是把能力分成 3 类：

- 作为阈值的功能，或者说必需的功能。
- 线性功能。
- 兴奋点和惊喜点。

这几种功能与客户满意度的关系见下图，这个图称之为 Kano 模型。



阈值功能是产品要成功必须具备的那些功能，也常常也称之为必需的功能。改善阈值功能和增加阈值功能的数量对客户满意度并没有多大的影响。由于产品要有那些必需功能的才能在市场上生存，应该强调必须开发所有那些阈值功能。但并不是一定要优先开发所有产品必须功能，但是由于客户把这些功能看成强制性的，他们必须要在发布产品之前变成可用。从图上可以看出来，有的时候只需要满足必须功能的部分实现就可以了，因为对必须功能一定程度的支持以后，客户满意度的上升幅度就趋于平缓。

线性功能是一个处于“越多越好”状态的能力。这里客户满意度会随着这部分内容的上升而直线上升，而产品的价格也与线性特性相关。

最后，兴奋点和惊喜点是那些提供了很高的满意度，并常常可以为产品增加额外价格的那些功能。但是缺少兴奋点和惊喜点并不会让客户满意度降到中性以下。兴奋点的特点是，客户会喜欢它，但如果没有它客户也不会拒绝这个产品。实际上兴奋点和惊喜点也被称之为未被了解的需求，因为在大家没有很好的研究它的时候，并不知道自己需要它。

4，关注顾客的心理变化

构思软件产品需要对人和人性有深刻的理解，如果一个分析师对用户没有爱，我不相信会构思出真正被用户欢迎的产品来的。在创新问题上，我们来看看顾客的心理有什么变化。

1）服务和方便性

今天的顾客比以前知道的要多，有办法得到以前得不到的消息，在因特网上可以找到最好的价格，并且在几乎所有的产品和服务之间作选择，地理位置不再重要，顾客可以在几乎世界上任何地方订购产品。顾客忠诚度曾经是传统公司的保命路线，现在正在消失，取而代之的顾客对服务和方便性的要求。因此，先产品的构建就必须考虑这个现实，产品是不是为组织或者顾客提供了更好的服务和更大的方便性呢？如果没有，所构建的产品就很可能很快地被抛弃掉。

2）文化与心理

我们应该注意到，不同的社会和文化背景中用户需求是不一样的，也就是说我们还必须理解用户心理，并且在理解用户心理的基础上来创建伟大的产品，下面还是以我们常见的手机作为例子来思考：

手机在中国意味着什么？中国手机的核心词是“时尚”。中国的厂商，利用外来的技术，只是改变时尚的面孔来迎合消费者就取得了成功。但是我们发现在欧美市场，手机只是上班用的通讯工具而已。当我们看到欧美消费者手上拿着大而土的手机时，你不要奇怪，因为下班以后人们一般不用手机的。正是在不同文化背景下对同一种产品的不同需求，使手机在不同地域的设计重点会有很不相同，这就需要在构思产品的时候仔细思索、细细品味。

3）技术与应用

谈到创新，我们现在很多时候还是把思考点放在使用什么技术上，或者称之为技术创新，毫无疑义技术创新是重要的。但是，如果认为研发出了一种新技术就给产品注入了新的生命，那就有失偏颇了。

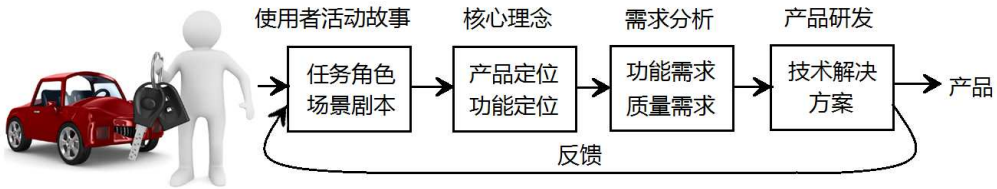
让我们看看周围的世界，并不是每个人要活着都需要自己种粮食吃（小农经济思维恐怕应该过时了），也不是所有从事科学技术的人都需要研究一套与牛顿-莱布尼兹不一样的微积分学，我们一直在使用别人的成果。所以，我们认为软件分析人员充分使用人类的已有知识和经验，来进行应用层面的发明创造是一件很了不起的事情。真正高明的设计其实是尽力去创造那些用户需要但表达不出来的东西，这靠的是一种思考力。所以一说到创新，就感觉这好像是高不可攀阳春白雪，但事实上我们国内目前最缺的，就是那种在应用层面创新的能力和意识。

毫无疑问，创新是企业的生命，但是并不等于说有了创新就一定能构建出满足市场需求的产品，过度的创新和对质量需求的不恰当追求并不总是可取的。微软的例子是发人深省的，他们总是构建一个更简单的能够完成对手 80%能力的产品，最终的取胜者往往是微软（回想一下网景和 IE 浏览器的例子）。这就告诉我们创新不是一切，我们还需要从更广阔的思想寻求新和旧的平衡，恰当的创新需要从全面分析和感觉两个方面寻找正确的答案。

二、适合创新的软件工程方法

由创新产品研发的特点，催生了新的软件工程方法和需求开发方法。创新不是某个人灵机一动的产物，而是一个庞大、极其复杂的系统工程，它需要集体的智慧，也需要激发这种智慧的形式。那么我们该怎么做呢？

新方法的关键之处是站在用户的角度而不是站在技术的角度看问题。目前比较流行的方法，是针对产品的特点，通过任务角色和场景剧本（使用者活动故事）来挖掘产品定位和功能定位，由此形成产品的核心理念，再由这个理念延伸到功能结构的工程方法，如下图所示。



那么我们如何做才能保证成功呢？

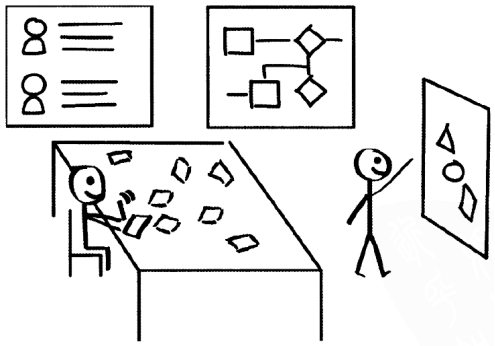
1，形成讨论和构思的人文环境

在这个过程中讨论和构思非常重要，应该有一种机制和环境来激发群体的想象力

1）要有一个开放的大空间

在这个空间里，我们可以随意站起来溜达，可以将图画粘在墙上，也可以将收集到的卡片放在大桌子上，只要能够发现设计问题，怎么做都行。良好的构思是一个思维和互动活动，开放的

环境更容易触发想象力。



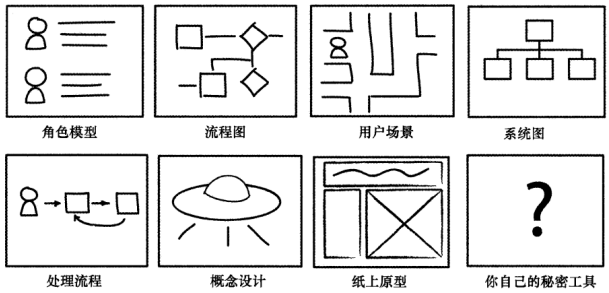
在这个过程中，需要将希望软件具有的特性都列出来。通过主持一个创新研讨会：开发团队与相关人员聚集在一起，将系统所要实现的场景写下来。研讨会的目的是取其广度。你要撒开大网，发现尽可能多的特性。

这样做不是由于必须要开发所有提出来的这些特性，而是我们希望将所有东西都摆到桌面上来，以纵览全局。研讨会通常需要相关人员坐下来一起画出一些图画，一边讨论系统一边将问题、对策、解决方案写在卡片上，大的空间更容易保留已有的信息和画图，而不是写了就擦。

2）要画很多图画

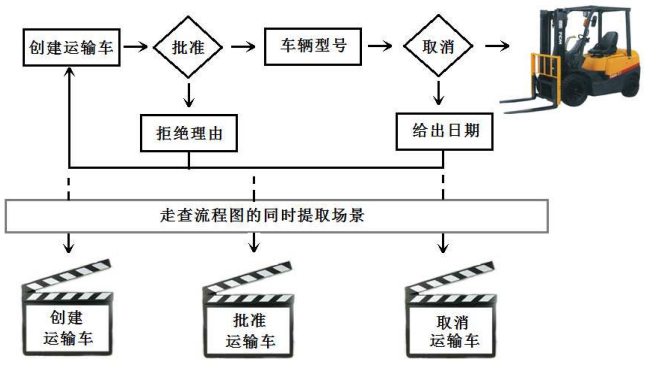
画图是一种非常棒的集体讨论系统的方式。角色模型（描述将要使用系统的人）对于了解客户会很有好处。流程图、处理流程和用户场景对于角色的行为分析很有作用，能够让人深刻理解系统的工作方式。系统图和信息架构图帮我们对工作进行组织和划分。

概念设计和纸上原型是两种很廉价的试验方式，可以发现哪些方法有效果。记住，现在我们追求的是广度。因此，不要太纠结于细枝末节，也就是说不要钻研过深．但是，一旦有了一些好的图画，并理解了系统如何运转，你就可以准备开始具体的需求定义工作了。



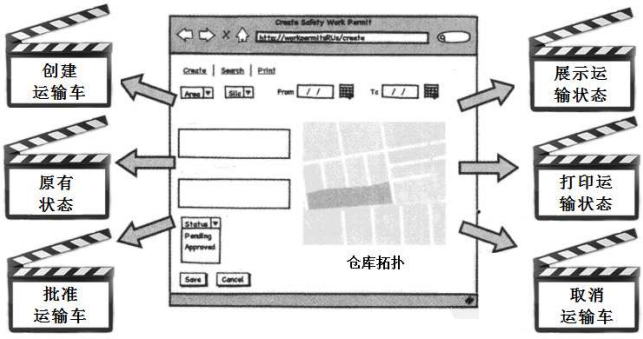
3）编写大量用户场景

利用这些图，大家一起走查一遍，同时描述用户场景。



这样的流程图上可以画一些低保真的纸上原型，项目的大多数核心场景都可以从这样的低保真原型中得出。如果应用系统连一个或两个界面都放不下，那就将这些界面分成一些更小的功能

块。



在选取用户场景时，要选择那些小的、离散的和完整的领域对象。但是，初期对象比较大也问题不大，一旦开始实现就要将其分成小块（通常 3- 5 天的工作量）。前期主要关注的是广度而不是深度。因此，不要潜得太深，否则被水草缠住就麻烦了。

2，最先的场景是使用者活动故事

编写场景是个好方法，但如果我们构思的是一个全新的产品，那最先的场景并不是我做了个什么东西，用户会怎么用。而是站在用户的角度，描述用户是怎么活动的？这中间会发生什么问题？有什么需要？

举个例子：

假定某公司在办公自动化由桌面向移动转移的大背景下，希望针对企业内部移动社交通讯应用的需求，从企业云入手，对社交通讯进行企业化改造，要求设计一个企业内部移动社交通讯应用软件，以解决员工间无缝沟通，并且能保证企业信息安全。

1）人物角色

该产品主要用户为企业内部员工。该企业员工约有一半以上从事和软件相关的产品、设计、开发、测试、运营等一线工作。这一群体定位为产品的首要目标用户，它决定了产品的特点（人口特征、地理位置、价值观、生活方式）。

2）场景剧本

场景剧本是产品使用者与活动的故事，可以帮助我们定义需求。根据既定的人物角色，团队想象了人物角色一天中的可能需要用到产品的场景，再通过场景剧本细化需求。

人物角色：

晓昂是企业的一个软件发分析师，他需要快捷的办公，降低沟通成本，让自己工作得更高效。

情境剧本：

- 早晨，晓昂在地铁上，利用空余时间收发电子邮件，在这种拥挤的活动空间中，能不能更方便的做这件事情呢？
- 在路上，晓昂接到一个来自未存号码的电话，他可能不会接。但如果他能够知道该来电来自于同事，就会毫不犹豫接起了电话。
- 来到公司，晓昂看到网站上有新一季度的研发活动，他想参加开发一款移动应用，却没有合适的产品经理（PM）与他合作，如果他能有“寻他搜索”功能搜索公司其他部门的 PM，这就可能方便得多。
- 中午，晓昂和大家一起去吃饭，却忘记了叫刚来没多久的小张一起去，但手机上没有存小张的电话号码。如果他能打开系统搜索小张的电话，打电话叫他一起来午餐，就可能达到目的。
- 下午，晓昂在去首创大厦开会的路上，但忘了具体的开会地点，能不能更方便的询问同事开会地点呢？

- 晚上，晓昂在外面和朋友吃饭，随手打开手机邮箱看看有没有人回复他的邮件。但在吃饭期间，如果耗用过多的时间，是不是对朋友不礼貌呢？

3）核心理念

根据产品定位，该系统有了一个初步目标。系统应该是一个方便、安全、可靠的企业级社交通讯工具，在通讯应用的基础上接入企业用户统一认证平台，增强隐私性和安全性，为用户提供安全的沟通以及企业化的个性服务。

从技术要求的角度分析，作为一个企业及系统，安全非常重要。系统首先要保证用户认证的可靠性，减少漏洞、降低风险。其次，由于该系统作为一个移动应用，是面向用户的，移动应用对用户体验的要求高于其他应用设备，因此还要保证应用的易用性，切实考虑用户需求，做好用户行为分析，最大化的提高用户体验。

最后，还要保证应用的先进性，快速响应用户行为，增强用户验证的易用性。从开发规范角度分析，该应用应遵循公司统一的开发规范，架构上遵循高内聚低耦合的原则，降低后期维护和拓展成本，为接下来的功能扩充预留必要接口。

从发展规划角度分析，需求分析中应该有为二期应用准备的功能考虑，必须能够建立一种可持续发展机制，保证高效、稳定的运行环境。

3，创建用户代表

在上述活动故事的场景中事实上创建了一个虚拟的用户代表（晓昂），这个代表意义很大。事实上一个面向市场的创新产品面对的用户很多，不可能逐个进行访谈，而建立一个虚拟的角色，就可以替代广义的用户来帮助我们思考。

但是上面例子中的代表是有缺陷的，他真能代表我们将要面对的客户群吗？他是不是一个典型呢？或者说只是一个个案？

1）如何让用户代表成为典型？

尽管用户代表是想象出来的，但如果赋予他与产品与其用户相符的属性，就可以成为许多人的一个准确的典型，就可以帮助我们构思出更合适的产品。

用户喜欢什么样的软件与用户特征有关，你不可能设计出所有的人都喜欢都适用的软件，你的软件使用范围是一个特定的群体，而这个群体的特征可以用用户代表抽象出来。事实上，在准备这门课程讲义的时候就有一个虚拟的用户代表：

晓昂，女，32 岁，软件工程硕士学位，为某大型银行 IT 部门工作，初期是在开发部门作为编码员，后来又做了 3 年的需求分析师。她的阅读量很大，对自己的工作很有兴趣，也希望把工作做得更好。她不喜欢人们用长篇大论说明一件事情，喜欢电影和音乐，最近正在考察汽车市场情况准备更换新的汽车。

我不可能认识所有的学员，但是我相信这个虚拟的用户代表会有大家的共同特性，我把她的照片贴在墙上，在思维受阻的时候，我总是问自己，晓昂对这个主题到底想了解什么？她在工作中有什么困惑？她如何才能把自己的工作做得更好？

如果大家觉得这门课程很有意思，是因为大家与晓昂有类似的知识、背景、领导力、思维方

式和好奇心，同时也有与她相似的对工作的兴趣与进取精神。

2）大项目可以创建多个用户代表

当项目比较大的时候，可以创造多个用户代表，以帮助我们搞清楚不同背景的人对功能的不同看法。拥有用户代表意味着在项目的早期就设定优先级，在全面描述用户代表以后，需求活动就集中于满足这个用户代表，结果就产生清晰的产品目标，范围变得更清楚，可用性需求更明确，也更容易知道哪些需求必须注意，而哪些需求可以安全的略过。这样一来，我们就可以列出一些重要的小组，对不同的用户群进行分类，正确地对用户群进行分类是成功的需求分析的第一步。

4，头脑风暴会议

很多资料都谈到了头脑风暴这个技术，事实上头脑风暴是一个会议技术，通过收集会议成员的即兴想法，为具体问题找到解决方案。这种技术的立论依据是：“驳倒轻率的思想容易，提出新的设想难，不管这些想法多么离奇，也不应该一直不理会它，不加以考虑和研究。”

1）仅仅会执行的组织是没有创造力的

一个死板教条的组织往往习惯于领导做指示，下属按部就班的执行，事实证明，这样的组织是没有生命力的，也是没有创造力的。

头脑风暴采用团队工作来产生想法和解决问题，这是一种群体决策技术，它的要点是，不进行评价或者解释说明，而是快速产生设想，任何设想都可以：离奇的、疯狂的或者不切实际的，设想产生后，再进行评价，而且不断地进行评价。

2）前期生成的需求越多后期发生的不愉快就越少

从需求的角度，前期生成的需求越多，后期发生的不愉快就越少，即使不能一次性的完成这些需求，在将来系统升级或者在设计中预留升级接口都是非常有意义的。从许多需求中选择，远比从头开始要好，这样一来，我们就可以从长长的清单中选择、分类、排序和删减。

3）一个设想可能以另一个设想为基础

在一个群体中，人们提出设想，一个设想可能以另一个设想为基础，要欢迎所有的设想，不指责任何设想，更多数量的原创设想会产生更多数量的有用设想，单独一个人很少会有这样多的创造性设想。当减少限制以后，人们将更愿意提出有用的设想。

这种跳出思维框框的原创性技术是革命性的，头脑风暴现在几乎被世界上所有的大公司所采用，它已经成为“创造性设想”的同义词。头脑风暴使团队成员从社会角色的限制中解脱出来，因而能够产生尽量多的设想，鼓励所有的参与者说出任何设想，不管它显得多么“愚笨”都不会受到指责，因为参与者的任务是提出设想，而不是检查它们。

4）头脑风暴会议的准备

- 确定并邀请参与者、记录员和会议领导。发布有关地点、时间、位置和会议预期长度等信息。一般预期的时间大约 2 个小时。
- 确定具体要开发的软件系统或者子系统，发布合适的文档（如概念研究总结，面谈总结，项目计划目标）给所有的团队成员。
- 定义头脑风暴会议的规则，发布给参与者。
- 保证室内有舒适的椅子，桌子，白板，投影仪，便签以及食物和饮料等，还需要给记录员准备数码相机，以记录白板上的实时讨论信息。

5）领导规则

- 允许设想以口头形式（首选）和书面形式表达。
- 适当的时候允许沉默。
- 从所有参与者那里尽可能多地收集设想，产生设想的时候，不要指责或者判断。
- 说明会议程序。
- 不允许互相之间指责、讨论或者争论。

- 不允许嘲笑，也不要过于严肃。
- 鼓励异想天开或者夸张的设想。
- 确立目标和主题。
- 保持快速的节奏，以减少限制和评价。
- 帮助参与者变换和组合设想。
- 帮助参与者产生设想。
- 设置时间限制。
- 充分利用所分配的时间。

6）记录员规则

- 不要具体描述设想，只需要捕捉他的精华。
- 如果需要，可以简单解释。
- 采用说话者的原话。
- 用简短的单词或者短语，记录下所陈述的设想。
- 必要的时候可以用数码相机记录白板上即时所画的图形。

7）参与者规则

- 倾听并理解别人的设想，“站在别人的肩膀上”产生新的设想。
- 完全投入到过程中，并自由思考，充分展开想象的翅膀。
- 耐心，必要的时候保持沉默。
- 一次只提出一个设想。
- 不要怕麻烦，要提供解释，语言要简短干练。
- 不要评价他人的设想，不要指责和判断。
- 不要发出一些可能被别人理解为指责的暗示性语言。
- 不要打断别人，轮流发言。
- 不要太重视任何一种设想。
- 产生尽可能多的设想，短时间内，先数量后质量。
- 每一设想都要保持简洁，不能用简洁的语言表达的，往往是没有思考清楚的。
- 写下你的设想，并交给记录员。

8）头脑风暴总体规则

- 欢迎所有的设想，没有错误的、愚蠢的、麻木的或者离题太远的设想。
- 提出的每一个设想都属于群体，而不是属于提出设想的人。
- 每个设想都有价值。
- 会议中要有适当的人数。
- 在头脑风暴完成之后再讨论设想。

9）头脑风暴会议的后续工作

首先，建立一个设想清单，如果会议团队有时间，可以提供简短的匿名建议设想清单。

然后，去掉无用的和不完整的的设想，注意，不是扔掉，文档中是有记录的，只是从考虑问题的清单中去掉。

最后，把剩下的设想分成 3 类：

- **有效的**：看起来是需要的，可测试的需求。
- **可能的**：可以是需求。
- **似乎不是**：可能不是有效的需求。

记录发现的需求，并根据重要性，依赖性以及风险进行综合排序，这对于描述完整的需求是十分重要的。

三、通过用户体验评估发现问题

创新产品的螺旋迭代特征，决定了每个迭代周期都有一个收集需求和确认需求的过程。那么，这个需求从哪里来？无论我们当初做了多么好的构思，最终产品是否受欢迎还是用户说了算。但是对于用户群来说，就不可能以某个用户的评价为基准，漫无目的的收集用户的反映也没什么价值的，只有通过有计划的测试和对信息进行统计分析，才能帮助我们发现潜藏的问题，发现问题的重点，从而形成对上一循环产品的改进方案。

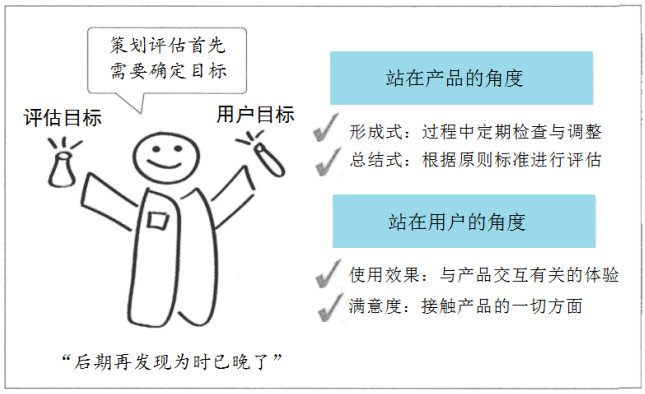
因此，创新产品研发在每个迭代阶段，都需要通过分析和评估用户的体验，可以尽早发现问题。在这个过程中，娴熟应用 SWOT 分析，找出产品的优势、劣势、机会、威胁，发现新产品的核心竞争力所在，发现产品的改进点和创新点，对于形成新产品研发的基本理念很有帮助。

1，如何策划用户体验评估

用户体验评估的目的是把猜测从决策中剔除出去。一个面向大众的创新产品，靠猜测决定需求是个风险很大的做法。这就需要一种涉及大众的评估方法，尽早发现潜藏在数据中的规律，这不可避免的需要度量和测量。

在很长的时间里，许多决策都是凭“直觉”做出的，这无形中增加了产品的风险。度量的优美之处在于，可以让数据把众多的猜测从用户体验决策中剔除出去。有时正确的设计方案是违反直觉的，直觉固然重要，但数据更好。

在策划任何用户体验评估时，首先要确定评估的目标和方法，如下图所示。



1）确定评估目标

要决定是数据如何在产品开发生命周期中使用。本质上包括使用方式：

- **形成式：**为在过程中定期检查，并做出对最终结果有积极影响的调整。事实上，越早进行形成式用户体验评估，用户体验评估对设计的影响就越大。
- **总结式：**评估一个产品或者一项功能与其目标结合得有多好。总结式测试也可以用于对多款产品的比较研究。形成式测试关注发现改进产品的方式或途径，而总结式测试侧重的是根据一系列的原则标准进行评估。

2）确定用户目标

计划一个用户体验评估，还需要了解用户和他们使用产品的目的。

例如，用户是否因为工作要求被迫每天使用该产品？还是很长时间才使用此产品一次？了解用户在意什么，这是至关重要的。用户是否在意完成任务的效率？用户是否在意产品设计是否美观？所有这些问题归结到测量用户体验的两个主要方面：

- **使用效果：**与用户使用产品、与产品发生交互所做的所有工作有关，它包括测量用户能成功完成一个任务（或一系列任务）的程度，包括完成每个任务的时间、完成每个任务所付出的努力、所犯错误的次数以及成为熟练用户所需的时间（易学性）。

- **满意度：**与用户接触和使用某产品时所说和所想的一切有关。用户也许会报告说，产品使用起来很容易、很让人迷惑或者超出了他的预期。用户也许会觉得，产品视觉上很吸引人或不太可信赖。用户的满意度对用户有很多使用选择的那些产品来说是非常重要的。

3）选择正确的度量

在为评估选择度量时，需要考虑很多方面的问题，包括评估目标和用户目标、现有收集和分析数据的方法技术、预算以及交付结果的时间。在选择度量的时候需要关注下面一些问题：

- **是否顺畅：**业务进行得尽可能顺畅是很重要的用户体验，一次业务通常有清晰界定的开始和结束。
- **比较产品：**与竞争对手产品或上一版产品相比，你的产品表现如何？通过进行比较，你能确定你产品的优点和缺点，以及这一版产品与上一版相比是否有进步。
- **频繁使用：**许多产品是用来被频繁使用的。这些产品使用起来需要既简单又高效，完成这些任务所需的努力应该尽量最少，大多数人都没有时间和耐性去使用那些难用的产品。
- **导航：**许多评估关注于改进产品的导航。这对网站、软件程序或消费类电子产品来说可能最为常见。这可以包括确保用户可以快速和轻易地找到他们需要的内容、轻松地在产品模块间进行切换等。
- **非凡的用户体验：**有些产品努力争取创造一种非凡的用户体验，而不是简单的好用。这些产品需要吸引人、激发情感、有趣，甚至还要能有点上瘾。

满意度也许是最常见的自我报告式的度量，但不一定总是最好的。感觉“满意”通常是不够的。最有价值的自我报告度量之一是用户期望。最佳的用户体验是那些超越用户期望的产品。当参加者说一个产品使用起来比期望更简便、更高效或更愉快，你就会知道这个产品有门儿了。

4）确定数据收集和整理的方案

首先考虑一下数据将被如何采集。你应该提前就计划好如何获取评估所需要的所有数据，这些决定会对之后进行数据分析所需的工作量有重大影响。

就较小样本的实验室测试来说，用 Excel 软件收集数据很可能就足够了。更大的数据可能需要专用统计软件或者开发基于数据库的在线数据存储程序。

另一个要考虑的问题是数据如何整理。通常需要让你的数据以一种便于快速且容易分析的形式呈现。数据整理可以包含以下步骤：

- **筛选数据。**在数据中检查是否有极端值出现。最有可能出现极端值的是任务完成时间（就在线评估而言）。有些用户会在评估中途出去吃午饭，这样他们的任务时间会异乎寻常的大。还需要过滤掉那些不符合目标用户或有其他外界因素会干扰结果的数据。
- **创建新变量：**在原始数据上创建新变量十分有用。例如，你也许需要为自我报告式评分创建一个得分排名处于前两位的新变量，即计算出这两项最高得分之一的用户有多少。也许你会将所有成功数据合计为一个可以表示所有任务的总体成功平均值。
- **检验应答：**在某些情况下（尤其是在线评估），可能需要检验参加者的应答。例如，如果你注意到，有很大比例的参加者都给出相同的错误答案，这就需要对此进行调查一下了。
- **检查一致性：**一致性检查可以包括对任务完成时间和成功与自我报告式度量进行比较。如果许多参加者在相对短的时间内成功地完成了任务，但是针对这个任务却给出一个很低的评分，那么要么数据采集时有问题，要么参加者对问题不理解。

对于只考虑了两三个度量的很简单的用户体验评估来说，整理起来是很快的。很显然，处理的度量越多，占用的时间将会越多。

2，对任务时间进行统计分析

完成一项任务所用的时间可以说是用户体验最重要的方面了。缺少任务，也就不可能有完成任务的时间。如果用户只是漫无目标地体验某款软件，你也不能测量软件的效果。在几乎所有的

情境中，参加者完成某项任务越快，其体验就越好。一个任务由同一个参加者操作得越频繁，效率就变得越重要。

1）对测量的数据进行分析

概率论与数理统计是一门非常重要的工程数学，进行用户体验的数据统计分析，当然离不开数理统计。不过在工程实践中，重要的不是令人烦扰的公式和繁琐的理论推导，而是充分利用已有工具帮助我们统计分析。我们要做的事就是理解，理解每一种统计量的意义以及适用场景，下面是在一个用户体验评估中，12 名被试者完成任务的时间（单位：秒）分析。

	A	B	C	D	E
1	参加者	工作时间		工作时间	
2	P1	34			
3	P2	33		平均	35.08333333
4	P3	28		标准误差	3.246112671
5	P4	44		中位数	33.5
6	P5	46		众数	22
7	P6	21		标准差	11.24486415
8	P7	22		方差	126.4469697
9	P8	53		峰度	-1.321525965
10	P9	22		偏度	0.251441718
11	P10	29		区域	32
12	P11	39		最小值	21
13	P12	50		最大值	53
14				求和	421
15				观测数	12
16				最大(1)	53
17				最小(1)	21
18				置信度(95.0%)	7.144645813

为了使用 Excel 分析数据。首先，点击“工具”→“数据分析”（注意：如果在”工具”菜单中没有见到”数据分析”选项，你需要先通过选择“工具” →“加载宏” →“分析工具库”进行设置）。

在“数据分析”窗口中，从分析选项列表中选择“描述统计”。然后，确定你要进行描述统计分析的数据区域。在我们的例子中，我们将数据输入范围定为从 B 列的第 1 行至第 13 行（注意：数据表中数据的第一行为标签行，所以我们选择“标签在第一行”复选框，是否包含标签是可选择的，但不是必需的，选择它能够帮助你使数据具有更好的组织性）。接着，选择输出区域（选中你想要呈现结果的数据表区域的左上角）。最后，我们可以表示出我们想看到的统计结果和 95%的置信区间结果，（95%置信区间是 Excel 中的默认设量）。

仔细研究这张表，可以看到很多从数据中透出的信息。

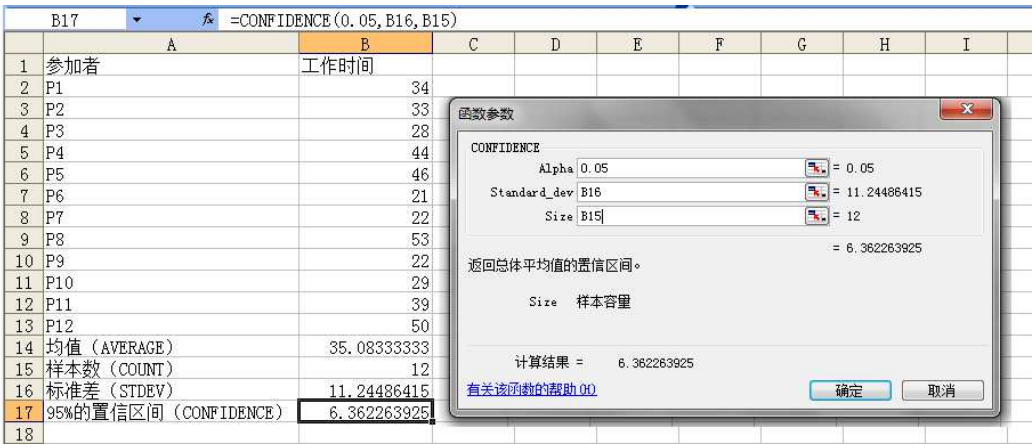
- **平均数：**在上图中，平均数是 35.1，即表示完成任务的平均时间刚刚超过 35 秒。大多数用户体验度量的平均数都是非常有用的，也是用户体验报告中最常引用的统计值。
- **方差：**方差是另一种常见且重要的变异性测量，它说明了数据相对于平均数或均值的离散程度。计算方差的公式：首先，求各数据点与平均数的差，然后求其平方，得到的值再求和，最后用样本数量减 1 之后的差值去除该求和之后的结果。在上图中，方差是 126.4。
- **标准差：**标准差是最常用的变异测量，一旦知道了方差，其平方根就是标准差。上图所示的这个例子中的标准差为 11 秒。理解标准差比理解方差稍显容易，因为它的单位与原始数据的单位是相同的（在这个例子中为秒）。
- **置信区间：**置信区间是一个范围，用来估计某统计值的总体实际值。例如，假设你需要估计整个总体的平均数，且希望估计正确的概率达到 95%。上图中 95%的置信区间在 7 秒左右。这意味着你可以确信有 95%的概率总体平均数为 35 秒加或减 7 秒，即 28 秒至 42 秒。

也可以使用 Excel 的“CONFIDENCE”函数快速地计算置信区间。计算公式非常容易建立：

= confidence(α 系数，标准差，样本大小)

α 值是你设定的显著性水平，典型的值是 5%或 0.05 。标准差可以通过 Excel 的“STDEV”函数能够很容易地计算出来。样本大小是你要检验的案例数，或数据点的数目，通过“COUNT”

函数能够容易地计算出这个值。下图是一个示例。



	A	B
1	参加者	工作时间
2	P1	34
3	P2	33
4	P3	28
5	P4	44
6	P5	46
7	P6	21
8	P7	22
9	P8	53
10	P9	22
11	P10	29
12	P11	39
13	P12	50
14	均值（AVERAGE）	35.08333333
15	样本数（COUNT）	12
16	标准差（STDEV）	11.24486415
17	95%的置信区间（CONFIDENCE）	6.362263925
18		

Excel 中的“CONFIDENCE”函数与描述统计函数中的置信水平计算有些差别。然而，请记住，这一函数是基于总体标准差的，但在大多数情况下，它是未知的。因此，上面两张图的置信区间并非完全一致。我们建议你采用一个更保守的观点，即使用描述统计中的置信水平。因为这种计算方法没有对总体的标准差进行任何假设。最终，你选择哪种计算方法不是一个重大问题。随着你研究样本大小的增加，两种计算方法之间的差异会逐渐减小。真正重要的问题是你需要使用其中的一种。对于计算和呈现度量的置信水平来说，无论我们怎样强调其重要性都不为过。

2）比较平均数

比较平均数有多种方法，但是，在进行统计之前，你应该首先回答以下问题：

第一个问题：谁与谁比较？

是同一组参加者内的比较还是不同参加者组之间的比较？例如，如果比较来自于男性和女性参加者的数据，这很可能是不同参加者组之间的比较。像这样比较的不同样本被称为独立样本。但是，如果你比较的是同一组参加者在两种不同产品或设计上的数据（组内评估），则需使用重复测量分析或配对样本比较。

第二个问题：样本量有多大？

如果样本量小于 30 ，可以使用 t 检验；如果样本量等于或大于 30 ，则使用 z 检验。

说明：t 检验与 z 检验

z 检验一般用于大样本（即样本容量大于 30）平均值差异性检验的方法，比较两个平均数的差异是否显著。

t 检验主要用于样本含量较小的平均值差异性检验的方法。t 检验是用 t 分布理论来推论差异发生的概率，从而比较两个平均数的差异是否显著。

进一步说，当总体标准差已知时，采用 z 检验；总体标准差未知时，但 n 较大（n>50）或虽 n 较小，但样本来自正态分布时，采用 t 检验；

一般情况下，对于大样本，两个均数的比较可以用 z 检验，也可以用 t 检验，二者结果接近；而对于小样本，两个均数的比较应该用 t 检验而不应该用 z 检验，因后者会把 P 值估计得过小以至于把原来可能无统计学意义的资料解释为有统计学意义

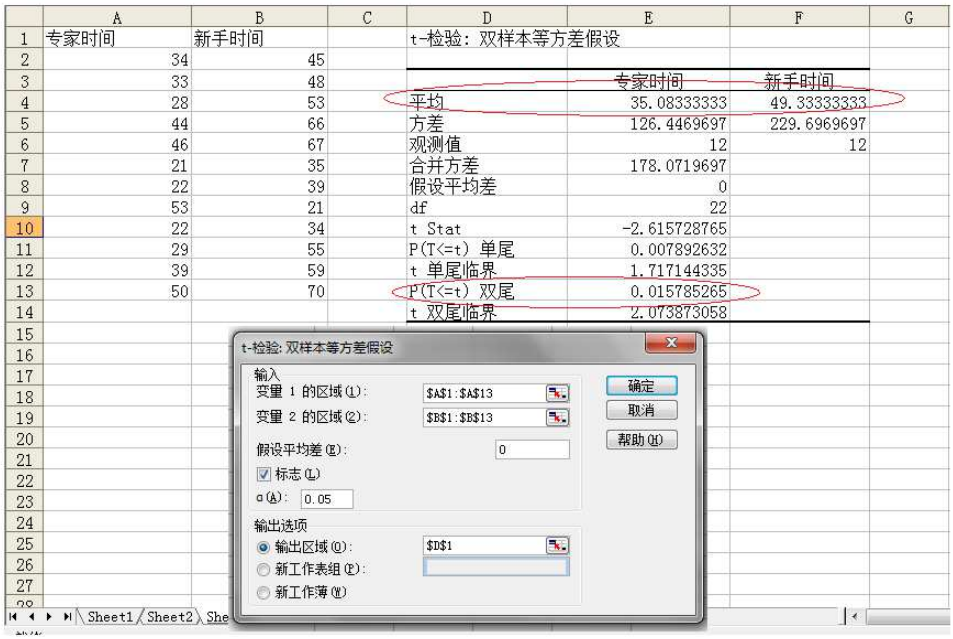
第三个问题：有多少样本需要进行比较？

如果比较的是两个样本，可以使用 t 检验；如果比较三个或更多的样本，则使用方差分析。

3）独立样本分析

在用户体验评估中，经常会比较基于独立样本的平均数，这仅仅意味着这些组别是不同的。例如，你感兴趣的是在满意度评价中专家级参加者与新手参加者之间的比较，检验这两个群体是

否存在显著差异。这种比较在 Excel 中可以轻松地实现，如下图所示。



首先，点击“工具”→“数据分析” →“t 检验：双样本等方差假设”选项。然后，输入两个变量的数据。数据来自 Excel 数据表的 A 列和 B 列。我们决定包括每行的标题，所以选择“包含标题”选项。

因为我们假设平均数之间没有差异，所以在这里我们输入 0（你也许听过这叫做“虚无假设”）。这意味着我们要检验两个变量之间是否存在差异。

我们选择 α 水平为 0.05 。这表示我们对自己的结果有 95%的把握。也就是说，当两个群体没有差异时，我们仍然有 5%的概率做出“它们之间存在差异”的错误推断。

分析这个结果。最先注意的事情可能是专家与新手之间平均数的差异。

- **谁更快？**专家（35 秒）要比新手（49 秒）快。
- **差异显著吗？**结果中一个重要指标是 p 值。因为事先我们对谁较快（专家或新手）没有做出任何假设，所以我们要看双尾分布的 p 值。这个 p 值约为 0.016，小于 0.05 的阈限。

  因此，我们可以说专家与新手的完成时间在 0.05 水平上具有统计意义上的显着差异。

请记住重要的一点：要说明 α 水平，因为它表明你可接受的错误概率有多大。

说明：什么是 p 值

P 值就是当原假设为真时所得到的样本观察结果出现的概率。如果 P 值很小，说明原假设情况的发生的概率很小，而如果出现了，根据小概率原理，我们就有理由拒绝原假设，P 值越小，我们拒绝原假设的理由越充分。

总之，P 值越小，表明结果越显著。但是检验的结果究竟是“显著的”、“中度显著的”还是“高度显著的”需要我们自己根据 P 值的大小和实际问题来解决。

什么是原假设？统计上把传统的观点作为原假设，新颖的、感兴趣的、想去论证的观点作为备择假设。就好比一个犯罪嫌疑人，在没有确凿的证据前都只能以他无罪为原假设，因为一个人“无罪判有罪”比“有罪判无罪”的后果严重的多，大家都不愿被冤枉。

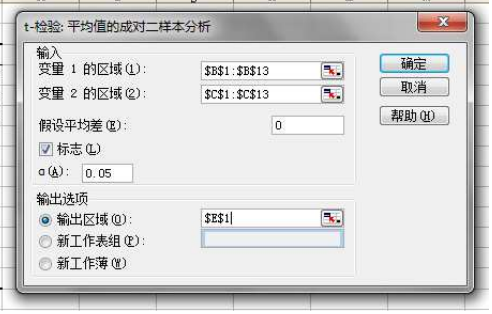
所以推广开来，你想证明 A 设计比 B 设计好，原假设就设为 A 设计与 B 设计效果相同，备择假设就设为 A 设计比 B 设计好，若得出的 P 值较小，一般以 0.05 作为临界值，比 0.05 小就可以接受 A 设计比 B 设计好的事实，若比 0.05 大，就说明没有足够证据证明 A 设计比 B 设计好，

保险起见拒绝备择假设，接受原假设。

4）配对样本

当需要比较同一组参加者的平均数时，应使用配对样本 t 检验。例如，你感兴趣的是两个原型设计由同一组测试者使用是否存在差异：你可以让同一组参加者先使用原型 A 完成任务、然后再使用原型 B 完成类似的任务，并且测量的变量是主观报告的易用程度和时间，则使用配对样本 t 检验。在 Excel 中能够轻松地进行这种检验，如下图所示。

	A	B	C	D	E	F	G
1	参加者	设计A_SUS	设计B_SUS		t-检验：成对双样本均值分析		
2	P1	80	48				
3	P2	88	55			设计A_SUS	设计B_SUS
4	P3	76	53		平均	77.75	57.0833333
5	P4	90	80		方差	125.4772727	153.719697
6	P5	93	81		观测值	12	12
7	P6	67	51		泊松相关系数	0.652121253	
8	P7	68	61		假设平均差	0	
9	P8	55	41		df	11	
10	P9	77	55		t Stat	7.22959171	
11	P10	71	57		P(T<=t) 单尾	8.43746E-06	
12	P11	88	59		t 单尾临界	1.795884814	
13	P12	80	44		P(T<=t) 双尾	1.68749E-05	
14					t 双尾临界	2.200985159	
15							



在主菜单上，选择“工具”→“数据分析”→“t 检验：平均值的成对二样本分析”选项。然后，选择要比较的两列数据。数据来自要比较的是 B 列和 C 列，从第 2 行到第 13 行。

接下来，确定“假设平均差”的值，在这个例子中，我们选择 0 ，因为我们假设 B 列和 C 列的平均数没有差异。然后，我们设 α 值为 0.05 ，表明我们对结果有 95%的把握。输出选项是你想在哪里呈现结果。这个对话框的设置完全类似于独立样本 t 检验。当然，主要的差别在于比较来自于相同参加者内部而非不同参加者之间。

分析输出的结果：

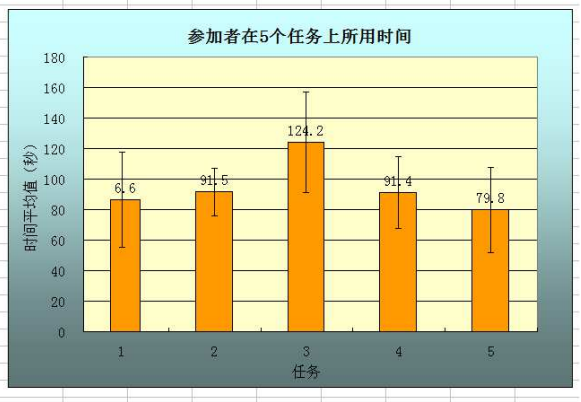
与独立样本结果输出一样，重点要看平均数和标准差。p 值为 0.0000169 表示两种设计之间有显著的差异，因为这一数值远远小于 0.05。

在配对样本检验中要注意的是，所比较的来自于两个分布的样本量要相等（虽然可能会有缺失值）。在独立样本条件下，样本量无需相等。其中某一组的参加者可以比另一个组的参加者多。

5）分析和呈现数据

可以用多种不同的方法分析和呈现任务时间数据。其中，最常用的方法可能是：通过以任务来平均每个参加者的所用时间，这是一种直接报告任务数据的方法。下表呈现了测量 20 个参加者完成某个任务的数据，还包括总结性的数据，包括平均数、中位数、标准差、每个任务的置信区间，并且用柱状图表示了出来。

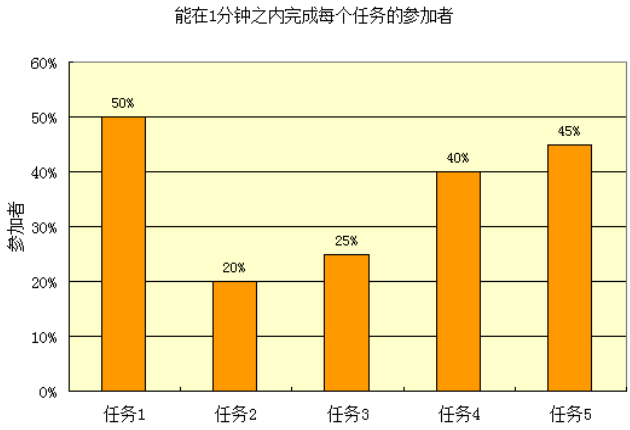
	A	B	C	D	E	F
1	20个参加者在5个任务上所用时间					
2	参加者	任务1	任务2	任务3	任务4	任务5
3	P1	259	112	135	58	8
4	P2	253	64	278	160	22
5	P3	42	51	60	57	26
6	P4	38	108	115	146	26
7	P5	33	142	66	47	38
8	P6	33	54	261	26	42
9	P7	36	152	53	22	44
10	P8	112	65	171	133	46
11	P9	29	92	147	56	56
12	P10	158	113	136	83	54
13	P11	24	69	119	25	68
14	P12	108	50	145	15	75
15	P13	110	128	97	97	78
16	P14	37	66	105	83	80
17	P15	116	78	40	163	100
18	P16	129	152	67	168	109
19	P17	31	51	51	119	116
20	P18	33	97	44	81	127
21	P19	75	124	286	103	236
22	P20	76	62	108	185	245
23	样本数（COUNT）	20.0	20.0	20.0	20.0	20.0
24	均值（AVERAGE）	86.6	91.5	124.2	91.4	79.8
25	中位数（MEDIAN）	58.5	85.0	111.5	83.0	62.0
26	标准差（STDEV）	70.9	35.2	75.4	53.9	64.0
27	95%置信区间（CONFIDENCE）	31.1	15.4	33.1	23.6	28.0



重要的是在图中用误差线标示了置信区间。为什么一定要报告置信区间呢？这是因为统计平均的风险是参加者中存在潜在的差异。比入，如果有几个参加者花了过长的时间才完成了某个任

务，这会大幅度地增加均值。通过报告置信区间，显示任务数据中的变异性，进而确定任务之间是否存在统计上的显著性差异。

另一个分析任务时间数据的有效方法是使用阈值。许多情况下，唯一重要的事情是关注用户能否在一个可接受的时间范围内完成某些特定的任务。在许多方面，均值是不重要的，研究的主要目标是减少需要过长时间才能完成任务的用户数量。



3，对任务成功率进行统计分析

任务成功是最常用的用户体验度量，它几乎是一个通用的度量，只要用户可以操作一个定义好的任务，你就可以测量其操作成功的程度。如果你的参加者不能完成他们的任务，那么你就知道有些事情出问题了，对于有些方面需要进行修改而言，这是最为恰当的并带有强迫性的理由。

1）以二分式标示任务成功

为了测量任务成功，要求参加者操作的每个任务都必须有一个清晰的结束状态，你需要知道什么构成了成功，因此应该在收集数据之前就给每个任务定义成功标准。

二分式成功是测量任务成功的最为简单和常用的方法。参加者要么成功完成了任务，要么没有成功。接近成功不管用，唯一重要的是用户能成功地完成他们的任务。

用户每操作一个任务，都应给予一个“成功（1）”或“失败（0）”的得分。有了数字得分，就可以很容易地计算出操作的平均值及其他需要的统计值，如下图所示。

	A	B	C	D	E	F
4	P3	1	1	1	1	1
5	P4	1	1	1	1	1
6	P5	0	0	1	1	1
7	P6	1	0	0	1	1
8	P7	0	1	1	1	1
9	P8	0	0	1	1	0
10	P9	1	0	1	0	1
11	P10	1	1	1	1	1
12	P11	0	1	1	1	1
13	P12	1	0	1	1	1
14	均值	67%	42%	92%	75%	83%
15						

2）非参数检验

当我们进行 t 检验和相关分析时，我们假设数据为正态分布而且数据的方差接近齐性（控制变量不同水平下观测变量总体方差无显著差异）。但是，有些数据的分布并不是正态的。例如，就任务成功（成功、失败二分式）来说，数据基于二项式分布，取值只有两种可能性，因此，我们不能对数据做出同样的假设，这就需要采用非参数检验（有人喜欢将非参数检验称为“分布无关性”检验）。

非参数检验对数据做出的假设，与我们先前介绍的平均数比较和描述变量之间关系的统计方法所做出的假设不同，它用于分析称名数据和顺序数据。例如，你可能需要知道，在特定任务的

成功和失败上，男性与女性之间是否存在显著差异。或者，你有兴趣了解专家，中等水平，新手三者之间在使用产品某个功能的时候，是否存在差异等。

非参数检验包括多种不同的类型，最经常用到的是卡方检验（x^2检验）。

卡方检验是以x^2分布为基础的一种常用假设检验方法，它的无效假设 H_0 是：观察频数与期望频数没有差别，以此用来比较类别（或称名）数据。

例如，我们需要分析在任务成功率上，三个不同组别（新手、中等水平和专家）之间是否存在显著差异。

你共测试了 60 名参加者，每组 20 人，记录了他们在某一任务中的成功或失败。

你还计算了每组参加者中成功的人数。新手组中，有 6 名参加者成功；中等水平组中，有 12 名参加者成功；专家组中，有 18 名参加者成功。你需要知道不同组在任务成功率上是否存在统计意义上的显若差异。

在 Excel 中，你可以使用 CHITEST 函数来进行卡方检验，这一函数计算的是，观测值与期望值之间的差异是否仅由随机因素所致，如下图所示。

	A	B	C
1	分组	观察值	预期值
2	新手	6	11
3	中等	12	11
4	专家	18	11
5	合计	33	33
6			
7		卡方检验值	0.033071251

这一函数的使用起来很简单= CHITEST（实际范围，预期范围）。

这里为=CHITEST(B2:B4,C2:C4)。

- 实际范围（观测值）：指每组参加者中成功完成任务的参加者人数。
- 预期范围（期望值）：指参加者之间没有任何差异的条件下你所预期的值。由于总完成量为 33，如果没有显著差异，则希望把这个数值平均分到各组，也就是每组 11。

分析这个数据：

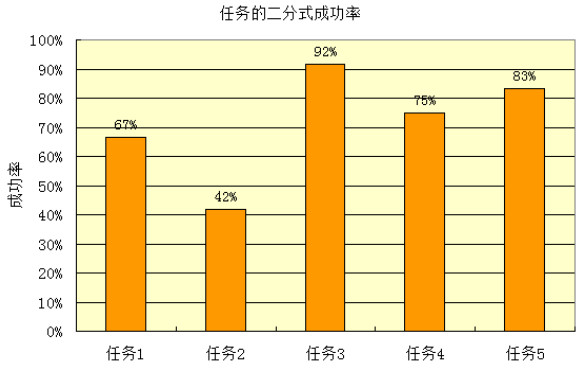
数据分布因随机水平而致可能性约为 3.3%（0.03307）。因为这一数值小于 0.05（95%置信水平），我们可以合理地推论：三组参加者的成功率之间存在显著差异。

3）分析和呈现数据

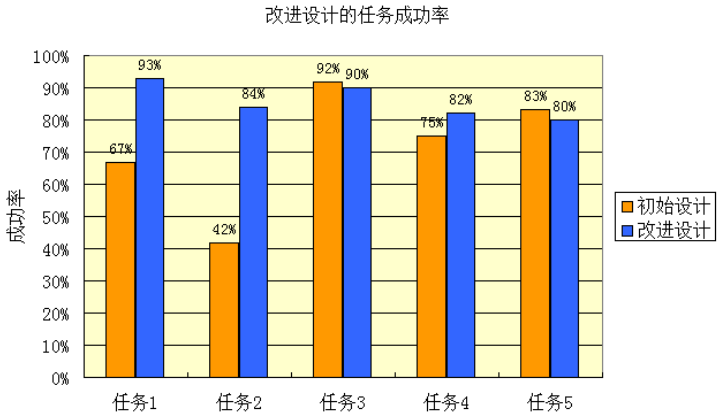
从用户角度查看，二分式成功数据的主要价值在于：你可以区别不同组别的用户，他们操作的方式不同或碰到不同类别的问题。这里有一些区分不同参加者的常用方法：

- 使用频率（经常使用的用户和不经常使用的用户）
- 使用产品的已有经验
- 专业领域（专业领域性低的知识和专业领域性高的知识）
- 年龄组

下图是不同任务的成功率相互比较的柱状图，从而找出最需要改进的任务。



也可以对初始设计与改进设计任务成功率进行比较，从而体现新设计的效果，如下图所示。



4，对错误进行统计分析

错误是一个可能的结果。比如，如果用户在使用一个电子商务网站完成某次购买行为中碰到了一个问题，这问题可能是产品上标有困惑性的标签所致。这个错误或者说该问题的结果，则可能是在选择他们想购买的产品时选择了错误的选项。本质上，错误是一些不正确的动作，而这些动作可能会导致任务的失败。

1）何时测量错误

在很多情况下，发现和区分错误比仅仅描述用户体验问题更有帮助。当你想了解某个可能会导致任务失败的具体动作或一组动作时，测量错误是很有用的。比如，某用户可能在网页上做出了错误的选择，出售了一种股票而不是买进更多股票。某用户在医疗器械上可能按了错误的按钮及给病人开了错误的药物。

这两个案例中，重要的是了解犯了什么错误，以及不同的设计元素可能会在多大程度上会增加或减少错误的频次。错误可以告诉你，产品造成了多少误解、产品的哪些方面造成了这些误解。不同的设计可以使错误的类型和频次有多大程度上的不同，以及一般情况下产品真实可用的程度是多大。

2）收集和测量错误

测量错误并不总是那么容易，你需要知道正确的操作应该是什么样子的。比如，如果你正在研究密码重量置表，则需要知道什么是成功重置密码的正确操作序列以及什么不是。对于正确和不正确操作的范围，你定义得越好，就越容易测量错误。

一个重点考虑的问题是，一个既定的任务只存在单个错误机会还是存在多个错误机会。一个错误机会本质上是一次出错的可能性。比如，如果你测量一个登录屏幕的用户体验，就有可能存在两个错误的机会：当输入用户名时出错以及输入密码时出错。一个任务中可能存在多个错误机会，但你只需要关注其中的一个。

整理错误数据的最常用方法是按任务整理，只需要记录每个任务和用户的错误数量。如果只有一个错误机会，则错误数量将为 1 和 0：

0 ＝没有错误

1 ＝一个错误

如果可能有多个错误机会，则错误数量将在 0 和最大的错误机会数量之间变化。错误机会越多，用表格来整理这些数据就越难和越耗时间。你也可以使用自动化或在线的工具收集该数据。

3）按任务归纳错误频率

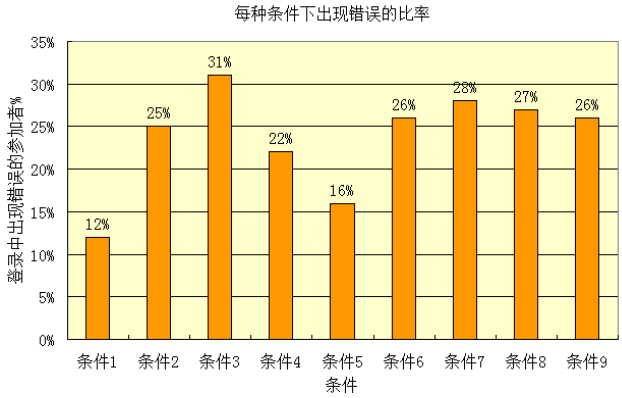
分析错误的第一个途径，是按任务归纳错误频率并绘制出错误数量。这样可以显示出每个任务上的错误数量。

注意，在这类分析中，你没有必要使用置信区间，因为你不想以此推断更为普遍的用户群，

而只对考察哪些任务有最多的错误感兴趣。

对于每个任务，用参加者的总数量去除错误数量，将会告诉你每个任务上出现错误的参加者比例是多大。如果有不同数量的参加者执行了每个任务，这种方法特别有用。

下图是一个例子，在这个例子中，有可能每个任务只有一个错误机会（比如输入的密码不正确）。他们感兴趣的只是：当使用不同类型的应用界面时，碰到错误的参加者百分比。

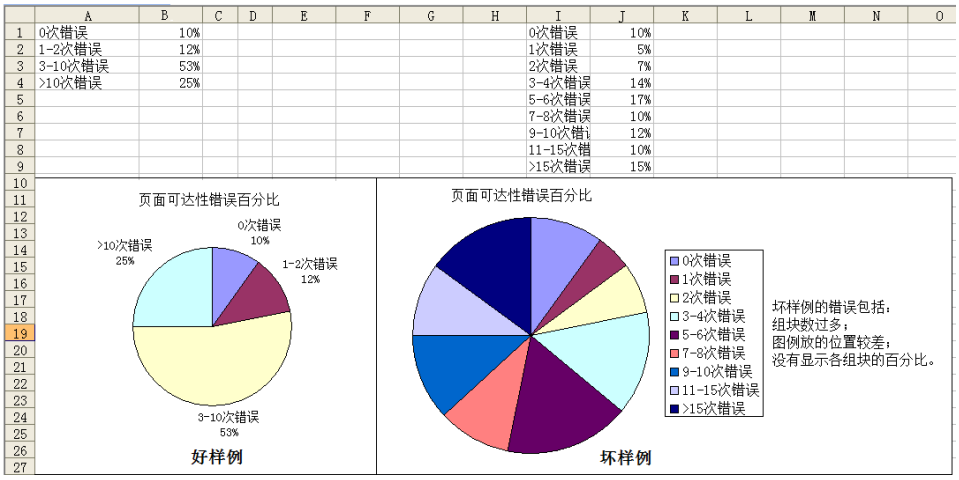


4）错误度

分析错误的第二个途径就是呈现任务错误度，的方法是从总体的角度进行。你可以不必总是关心某个特定的任务，只要关注总体上参加者的操作情况即可。这里有几个可供选择的做法：

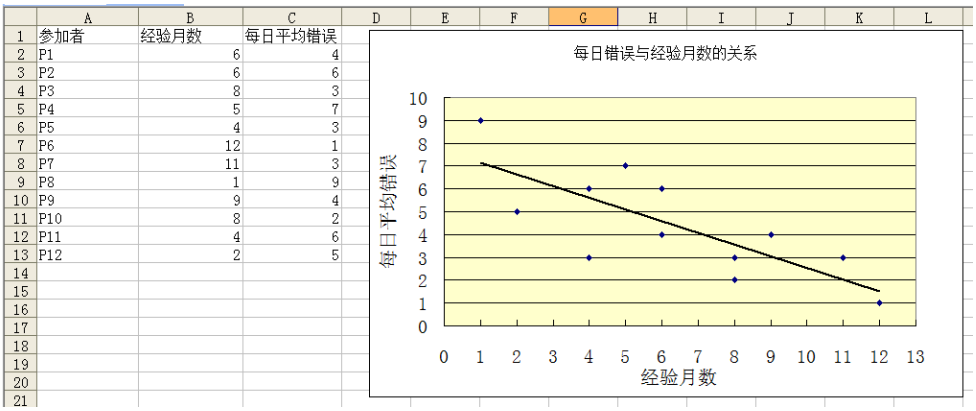
- **总错误率：**把每个任务的错误率平均成一个总的错误率。这就是告诉你该研究的总体错误率的情况。比如，你可以说任务的平均错误率是 25%。对于报告错误而言，这是一种有用的结果导向型度量。
- **平均值：**计算所有任务的平均值，这样就可以获得一个特定的错误数量。比如，如果你正在考察很多任务，你可以报告 50%的任务中存在 10%的错误率或更高。或者你也可以说至少有一名参加者在 80%的任务上出错 1 次。
- **阈值：**对于每个任务，可以确定一个最大的、可以接受的错误率。比如，你可以只关心发现那些错误率在某个特定阈值（比如 10%）之上的任务。接着你可以计算有百分之多少的任务在这个阈值之上或之下。比如，你可以简单地说有 25%的任务超过了某个可以可接受的错误率。

在呈现错误的时候，经常使用饼图。饼图显示了整体的各部分或相应的百分比。当你要显示整体中各部分的相对比例（如在用户体验测试中，有多少参加者在某任务上成功、失败或者直接放弃）时，饼图是非常有用的，如下图所示。



5）利用散点图展现关系

有时候，知道不同变量之间的关系是很重要的，例如，我们可能会关心：错误与经验是不是紧密相关？下图表示了在 Excel 中观察不同的参加者产品体验时间（经验月数）与每日平均错误之间的关系。



	A	B	C
1	参加者	经验月数	每日平均错误
2	P1	6	4
3	P2	6	6
4	P3	8	3
5	P4	5	7
6	P5	4	3
7	P6	12	1
8	P7	11	3
9	P8	1	9
10	P9	9	4
11	P10	8	2
12	P11	4	6
13	P12	2	5

首先，通过工具栏“插入”→“图表”→“散点图”，可以很轻松的得到不同的参加者每日平均错误与使用经验（经验月数）之间的关系。右键任意一个数据点，选择“添加趋势线”，就能轻松地将趋势线添加到散点图上。从图上可以直观地看出，随着产品体验的逐月增加，每日平均错误数逐渐减少。这种关系被称为负相关。

6）相关系数

以散点图的形式显示数据仅仅是第一步，进一步还需要了解变量之间的相关程度，这就需要计算相关系数，如下图所示。

	A	B	C	D	E	F	G
1	参加者	经验月数	每日平均错误			经验月数	每日平均错误
2	P1	6	4		经验月数	1	
3	P2	6	6		每日平均错误	-0.76160463	1
4	P3	8	3				
5	P4	5	7				
6	P5	4	3				
7	P6	12	1				
8	P7	11	3				
9	P8	1	9				
10	P9	9	4				
11	P10	8	2				
12	P11	4	6				
13	P12	2	5				

首先，在“数据分析”工具中选择“相关系数”。

然后，明确数据范围。在图中，数据包括 B 列和 C 列。

我们也选择了第一行，即标题行。

唯一需要做出决定的是选择输出范围（在哪里呈现结果），其输出结果被称为相关系数。

相关系数的取值范围是-1 至+1，是两个变量之间相关程度的测量指标。相关程度越高，这个值越接近于-1 或+1；相关程度越弱，相关系数越接近于 0。

上图表示的经验月数与每日平均错误之间相关系数为-0.76。负相关系数表示一种负的关系（随着体验时间的增加，错误会减少）。这种相关告诉我们，体验时间与错误数之间存在一种特别强的关系。还要注意，在这个矩阵中，自己对自己的相关系数为 1，当然这也是不言而喻的。

5，对易学性进行统计分析

易学性是事物可被学习的程度。它可以通过查看熟练地使用事物需要多少时间和努力而测得。一个每几个月才使用一次的功能，易学性就能成为一个很重要的挑战，因为用户不得不在每次使用时都要重新学习一下。

1）收集和测量易学性数据

收集和测量易学性数据本质上与其他使用效果度量是一样的，但是需要多次收集易学性数据，

每个收集该数据的例子都可被看成是一项施测。可以每 5 分钟、每天或每个月施测一次。两次施测之间的时间（或当你收集该数据的时间）设定基于所预期的使用频率。

首先要决定的是需要使用哪些类型的度量。易学性几乎可以用任何持续性的使用效果度量予以测得，但是最常见的是那些聚集在效率上的度量，如任务时间、错误、操作步骤数量或每分钟任务成功等。随着学习活动的发生，我们期待看到效率能够提升。

在决定了使用哪些度量之后，还需要决定的是两次施测之间得需要有多长的时间。当学习活动发生于一个很长的时间之后，你要做些什么？如果用户每周、每月或甚至一年使用一次产品，你要做些什么？理想情况是每周／每月／每年邀请同样的一些参加者到实验室。但许多情况下，这非常不现实。如果你告诉他们这个评估将要 3 年的时间才能完成，那无论开发人员和商业资助者会非常地不乐意。

更为现实的方法是，邀请同一批参加者在一个比较短的时间间隔内参与研究，并明确说明由此带来的数据收集上的不足。这里有几个备选的做法：

- 在同一个测试单元中的施测。参加者要完成一些任务或几组任务，一个接一个地，中间没有停顿。对执行管理来说，这非常简单，但是这没有考虑明显的记忆衰减。
- 在同一个测试单元中但任务之间有间隔的施测。间隔可以是一个干扰任务或其他可以促使遗忘的事情。这对于执行管理来说非常简单，但是这会使得每一个测试单元延长。
- 不同测试单之间的施测。在多个测试单元（前后两个单元之间至少要隔 1 天）中，参加者要完成相同的任务。如果产品在一个比较长的时间内才被偶尔使用一下，这种做法可能最不符合实际情况，但是确实是最现实可行的。

2）施测

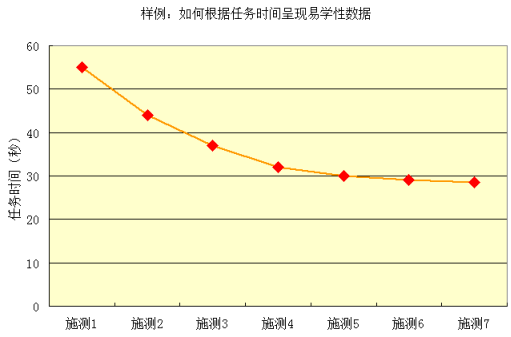
在有些情境下，学习行为是持续的。这意味着用户要非常连续地使用产品，这个过程中没有明显的间断。勿庸量疑，记忆在这样的情境下肯定是个影响因素。学习活动更多地与完成任务过程中不同策略的形成和改变有关。在这种情境下你要做些什么？有一个方法是：在确定好的时间间隔内进行测量。比如，你可能需要每 5 分钟、每 15 分钟或者每小时施测 1 次。

在一项用户体验评估中，我们要评估一套每天会多次使用的新应用软件的易学性。我们开始邀请参加者到实验室进行第一次测试任务，他们均第一次接触该产品。接着他们回到他们常规的工作中并开始使用这个软件完成他们的正常工作。一个月之后，我们再次邀请他们到实验室并让他们再次完成本质上与第一次相同的任务（在细节上有一点细微的变化），同时我们使用同样的使用效果测量。最后，再过了另一个月，我们再次请他们回来并重复之前的测试过程。这样，我们就能够考察两个月的时间内易学性的情况。

你需要多少次施测？很明显至少需要两次，但是在很多情况下至少应该 3 或 4 次。有时很难预测在施测序列中的哪个阶段上发生（或即使将会发生）了最大程度上的学习活动。在这种情况下，你应当偏向采取比你认为达到稳定使用效果所需要的还要多的施测。

3）学习曲线

分析和呈现易学性数据的最常用方法，是通过施测检验每个任务（或合计之后的所有任务）上某个特定的使用效果度量（如任务时间、操作步骤数量或错误数）。这将会显示出使用效果度量如何随着经验的作用而发生变化的，如下图所示。我们可以把所有的任务合计起来，并把数据呈现为一条单一的线，或者也可以把每个任务都单独表示为相应的数据线。这将有助于确定如何比较不同任务的易学性，但是这也会使得绘制出来的图难以解释。



分析这个数据：

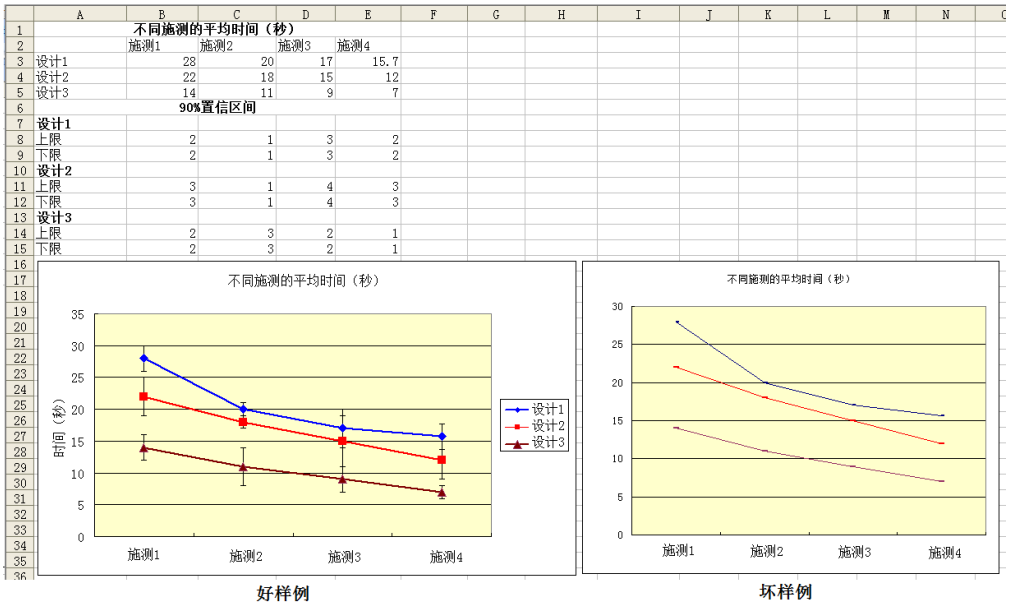
- **斜率：**我们首先注意图表中线的斜率。理想情况下，斜率（有时被称为学习曲线）在 y 轴（就错误、任务时间、操作步骤数或其他任何度量来说，数值小一点都比较好一些）上应相当扁平。如果要确定学习曲线（或斜率）之间是否存在显著的差异，则需要进行方差分析。
- **拐点：**同时还需要注意到渐近线的拐点，或线条从哪儿开始实质性地平滑。在这点上，参加者已经尽其所能学习并且几乎没有提高的空间。多长时间才能使用户达到最大使用效果，项目团队成员对此总是很感兴趣。
- **上下限：**最后，我们应该考察 y 轴上的最高值和最低值。这将会告诉我们要学习多少或多久才能达到最大使用效果。如果差异小，用户很快就能学会使用该产品。如果差异大，用户就需要相当的时间才能变得熟练使用该产品。

分析最高值和最低值之间差异的一个简单方法是考察二者的比率。这里是一个例子：

如果第一次施测中的平均时间是 80 秒而在最后一次施测中是 60 秒，这比率就表示参加者初始使用所用时间为 1.3 倍长。

如果第一次施测中的平均错误数是 2.1 个而在最后一次测试中是 0.3 个，这比率就表示从首次施测到末次施测提高了 7 倍。考察需要多少次施测才能达到最大使用效果也有帮助。对于描述需要多大的学习量才能熟练使用该产品，这是一个好方法。

有时，也会关注为同一个目的不同设计间的易学性比较，如下图所示。



好样例 **坏样例**

折线图的好样例与坏样例（所用的数据相同）。坏样例的错误包括：没有标注纵轴，没有包含图例，没有显示数据点，没有显示置信区间，所用直线磅数太低

四、用户期望与效果的比较分析

什么是好？什么是不好？分析师自己定一个指标来评估用户体验是不是达到了合格标准，这种度量是不是合适？站在用户的角度，事实上的标准应该是用户期望，实际效果是使用结果，这两者的差值往往是更好的分析依据，从而得出未来产品改进的重点。下面我们通过一个案例来说明这种分析方法的使用。分析的目标是：针对目前的产品，哪些功能或特性是下一步改进的重心？以此形成产品改进的目标。

1，问卷调查的过程

以下内容是使用调查方法来评估某产品特性的步骤说明。

1）就评估行动获得相关授权

这一行动最好能从尽可能高的公司管理层取得授权，同时还要经信息系统负责人的批准。

2）建立工作小组

该小组成员应该包括产品部门、开发部门和用户服务部门人员，后者确定评估应包含的内容。

3）设计问卷调查表

运用工作小组获取的信息来开发衡量工具

4）进行试验调查

在面对面的情况下让 5-10 位被调查者完成问卷调查。

5）根据试验调查的反馈信息完善问卷

试验结果会显示出问卷中哪此部分需要改善，哪些部分是与评估无关的。

6）选择被调查人群

选取可以代表目标整体的样本人群，应该包括各个层次的使用者。

7）发放问卷

必须让被调查者明确问卷需要完成的期限。

8）回收问卷

根据样本人群的分布状况，问卷既可以由调查者主动收回，也可以由被调查者还回。

9）分析结果

分析结果可能是十分费时的，但这一过程对于产生有价值的信息是非常必要的。

10）提交报告

应以书面形式概括总结问卷调查的结果，使用尽可能少的技术术语。

2，调查问卷的整体设计

调查使用填空形式的调查问卷，问卷包括四部分。第一部分收集被调查者的背景资料，例如职业、工作 / 学习经历、使用类似产品的时间等。接着是 A、B、C 三个部分。

A 部分和 B 部分使用相同的问题（例如 24 个），这些问题用来获取未来系统各个方面的信息。A 部分测量用户认为的各个属性对于产品的重要程度，并分为 4 个等级，从 1-不相关，到 4-非常关键。

B 部分使用相同的方法，但评估的对象是已经使用产品原型的表现，结果同样分为 4 级，从 1-非常差，到 4-优秀

C 部分的问题是关于以使用产品的总体满意程度。

调查采用互联网形式，被选中的试用者也同时获得了登录这个调查网页的权利。需回收的样本数一般需要超过 100，分析工作则主要依据这些反馈。

1）卷首设计

祝贺您成为这个新产品的第一批试用者，这是关于本产品使用效果的调查问卷，请登陆

http://www.***.***网站，回答并提交相应的问题。

调查问卷分为三部分：

A 部分和 B 部分使用相同的 24 个问题。C 部分是开放型问题。

A 部分的问题是关于您所认为的功能或特征的重要性。

B 部分的 24 个问题则是从已经使用的产品这些功能或特征的工作表现。

最后，在 C 部分，欢迎您根据自己的切身经验发表您个人的看法。

问题均采用 4 级评分制，并且在已登录的页面上直接回答。

第 1 套 24 个问题	第 2 套 24 个问题
关键	优秀
重要	良好
不重要	差
无关	非常差

例如，您可能认为产品某个功能或特征是关键的，这样你对第 1 套问题的评分，可以做如下回答（点击所选择的按钮）：

无关 ○ 不重要 ○ 重要 ○ 关键 ●

如果您认为现有产品某个功能或特征工作表现良好，对第 2 套问题的回答．就可以是：

非常差 ○ 差 ○ 良好 ● 优秀 ○

被调查者提供的任何信息将严格保密。

请在下面的问题后面，填上您的职业等信息：

您的职业？（选择）_____

您的文化程度？（选择）_____

您的年龄？（选择）_____

您的性别？（选择）_____

谢谢您对本次调查的合作。

2）产品功能与特征测评 A 部分 — 重要性（预期）

请就以下 24 种功能或特征对您重要程度在相应的选项后打钩。

1	功能或特征 1 无关○ 不重要○ 重要○ 关键○
2	功能或特征 2 无关○ 不重要○ 重要○ 关键○
3	功能或特征 3 无关○ 不重要○ 重要○ 关键○
4	功能或特征 4 无关○ 不重要○ 重要○ 关键○
5	功能或特征 5 无关○ 不重要○ 重要○ 关键○
6	功能或特征 6 无关○ 不重要○ 重要○ 关键○
7	功能或特征 7 无关○ 不重要○ 重要○ 关键○
8	功能或特征 8 无关○ 不重要○ 重要○ 关键○
9	功能或特征 9 无关○ 不重要○ 重要○ 关键○
10	功能或特征 10 无关○ 不重要○ 重要○ 关键○
11	功能或特征 11 无关○ 不重要○ 重要○ 关键○
12	功能或特征 12 无关○ 不重要○ 重要○ 关键○
13	功能或特征 13 无关○ 不重要○ 重要○ 关键○
14	功能或特征 14 无关○ 不重要○ 重要○ 关键○
15	功能或特征 15 无关○ 不重要○ 重要○ 关键○
16	功能或特征 16 无关○ 不重要○ 重要○ 关键○
17	功能或特征 17 无关○ 不重要○ 重要○ 关键○
18	功能或特征 18 无关○ 不重要○ 重要○ 关键○
19	功能或特征 19 无关○ 不重要○ 重要○ 关键○
20	功能或特征 20

	无关○ 不重要○ 重要○ 关键○
21	功能或特征 21 无关○ 不重要○ 重要○ 关键○
22	功能或特征 22 无关○ 不重要○ 重要○ 关键○
23	功能或特征 23 无关○ 不重要○ 重要○ 关键○
24	功能或特征 24 无关○ 不重要○ 重要○ 关键○

3）产品功能与特征测评 B 部分 － 实际性能

请就以下 24 个方面评估已使用产品的功能或特征的感受，并在相应的选项后面打钩。

1	功能或特征 1 非常差○ 差○ 良好○ 优秀○
2	功能或特征 2 非常差○ 差○ 良好○ 优秀○
3	功能或特征 3 非常差○ 差○ 良好○ 优秀○
4	功能或特征 4 非常差○ 差○ 良好○ 优秀○
5	功能或特征 5 非常差○ 差○ 良好○ 优秀○
6	功能或特征 6 非常差○ 差○ 良好○ 优秀○
7	功能或特征 7 非常差○ 差○ 良好○ 优秀○
8	功能或特征 8 非常差○ 差○ 良好○ 优秀○
9	功能或特征 9 非常差○ 差○ 良好○ 优秀○
10	功能或特征 10 非常差○ 差○ 良好○ 优秀○
11	功能或特征 11 非常差○ 差○ 良好○ 优秀○
12	功能或特征 12 非常差○ 差○ 良好○ 优秀○
13	功能或特征 13 非常差○ 差○ 良好○ 优秀○
14	功能或特征 14 非常差○ 差○ 良好○ 优秀○
15	功能或特征 15 非常差○ 差○ 良好○ 优秀○
16	功能或特征 16 非常差○ 差○ 良好○ 优秀○
17	功能或特征 17 非常差○ 差○ 良好○ 优秀○
18	功能或特征 18 非常差○ 差○ 良好○ 优秀○
19	功能或特征 19 非常差○ 差○ 良好○ 优秀○
20	功能或特征 20 非常差○ 差○ 良好○ 优秀○
21	功能或特征 21 非常差○ 差○ 良好○ 优秀○
22	功能或特征 22 非常差○ 差○ 良好○ 优秀○
23	功能或特征 23 非常差○ 差○ 良好○ 优秀○
24	功能或特征 24 非常差○ 差○ 良好○ 优秀○

4）产品功能或特征测评 C 部分 －整体评价

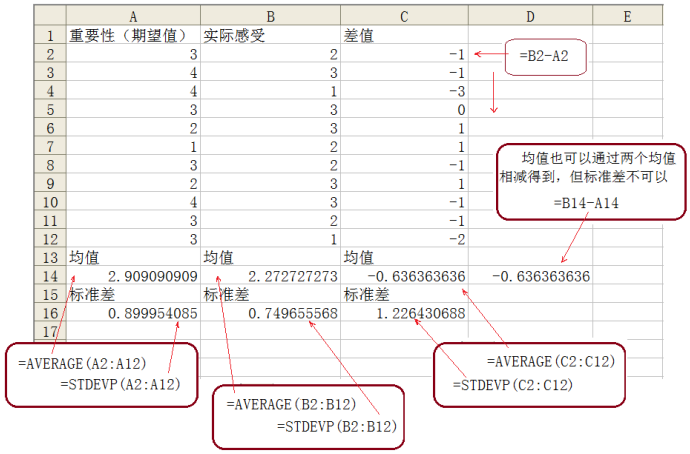
请给出您对已使用产品的整体看法。

非常差○ 差○ 良好○ 优秀○

请对已有产品功能或特征表现发表您个人的意见。

3，基本结果数据处理

对调查问卷的结果进行分析，需要给出每一组数据的期望值（A）和实际性能（B）评分值的平均值和标准差。我们知道，均值是集中趋势，标准差是离散趋势。标准差越大，表示数据高低越分散，反之表示数据越集中。可以用 EXCEL 中的 AVERAGE 函数来计算均值，用 STDEVP 函数来计算标准差。



下表是此次对回收的 112 份样本测评的统计结果。

题号	系统属性	重要程度评分（期望值）			实际性能评分			感受评分		
		评分(以均值排序)	均值	标准差	评分(以均值排序的序号)	均值	标准差	差值(实际减期望的均值差)	标准差	95%置信区间
6	功能或特征 6	1	3.45	0.60	2	2.82	0.53	-0.63	0.73	0.14
11	功能或特征 11	2	3.42	0.50	19	2.20	0.83	-1.22	1.05	0.19
5	功能或特征 5	3	3.39	0.54	10	2.49	0.60	-0.9	0.9	0.17
1	功能或特征 1	4	3.37	0.65	5	2.76	0.59	-0.62	0.92	0.17
13	功能或特征 13	5	3.34	0.53	11	2.47	0.62	-0.87	0.85	0.16
3	功能或特征 3	6	3.33	0.53	1	3.01	0.53	-0.32	0.77	0.14
20	功能或特征 20	7	3.32	0.59	8	2.66	0.64	-0.66	0.81	0.15
15	功能或特征 15	8	3.22	0.69	17	2.22	0.72	-1.00	1.06	0.20
7	功能或特征 7	9	3.20	0.59	11	2.47	0.60	-0.73	0.79	0.15
19	功能或特征 19	10	3.18	0.53	9	2.62	0.59	-0.56	0.81	0.15
16	功能或特征 16	11	3.16	0.49	3	2.78	0.62	-0.38	0.77	0.14
22	功能或特征 22	12	3.11	0.56	22	1.96	0.62	-1.15	0.9	0.17
9	功能或特征 9	13	3.05	0.49	11	2.47	0.64	-0.58	0.77	0.14
12	功能或特征 12	14	3.04	0.72	20	2.14	0.69	-0.90	0.95	0.18
2	功能或特征 2	15	3.01	0.53	14	2.46	0.72	-0.55	0.89	0.16
17	功能或特征 17	15	3.01	0.62	16	2.36	0.56	-0.65	0.84	0.16
8	功能或特征 8	17	2.95	0.56	15	2.37	0.65	-0.58	0.90	0.17
10	功能或特征 10	18	2.80	0.80	3	2.78	0.45	-0.02	0.89	0.16
4	功能或特征 4	19	2.78	0.62	18	2.21	0.55	-0.57	0.79	0.15
23	功能或特征 23	20	2.72	0.69	21	2.12	0.65	-0.60	0.90	0.17
21	功能或特征 21	21	2.63	0.69	6	2.74	0.50	0.11	0.81	0.15
14	功能或特征 14	22	2.47	0.66	22	1.96	0.64	-0.49	0.81	0.15
18	功能或特征 18	23	2.46	0.74	7	2.72	0.51	0.26	0.96	0.18
24	功能或特征 24	24	2.11	0.67	24	1.93	0.57	-0.18	0.77	0.14

4，分析结果并得到启示

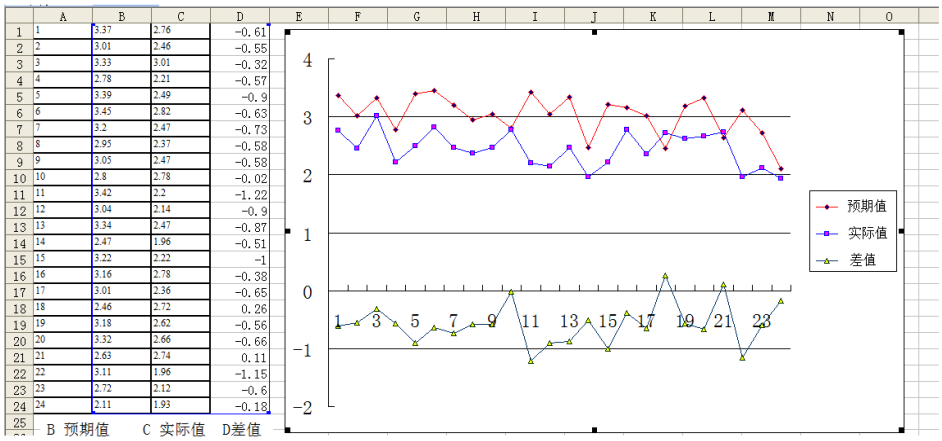
1）浏览表格

- 功能或特征 11 期望位排在第 2 位，但对其实际表现的评分值仅排在第 19 位。

- 功能或特征 24 和功能或特征 23 的期望值和实际表现评分位排名都很低。
- 功能或特征 18、功能或特征 21、功能或特征 10 期望位排名较低，但实际表现评分值排名较高。
- 对系统所有属性的总体评价较差。其中功能或特征 3 平均评分为 3.01，表现较好。
- 24 个差值结果中有 2 个是正值，因此可以认为用户满意度较高。因为差值是实际性能评分减去期望值的结果，所以差值越大，用户满意度就越好。

2）对结果进行综合分析

可以用**蛇型曲线**来表示期望值和实际性能评分之间的差值，同时也显示两套评分之间的差值。在该图中，零值线是非常重要的一条线，如果差值大于零，就意味着产品的实际投人资源多于所需要的数量。另一方而，差值为负，那么产品的表现没有达到期望的效果，也可以说产品在这个功能或特征工作表现不佳。如果差值为零，也就是说，没有差距，则表示产品的用户期望和实际表现非常一致。可以用 Excel 自动生成这套曲线，如下图所示。

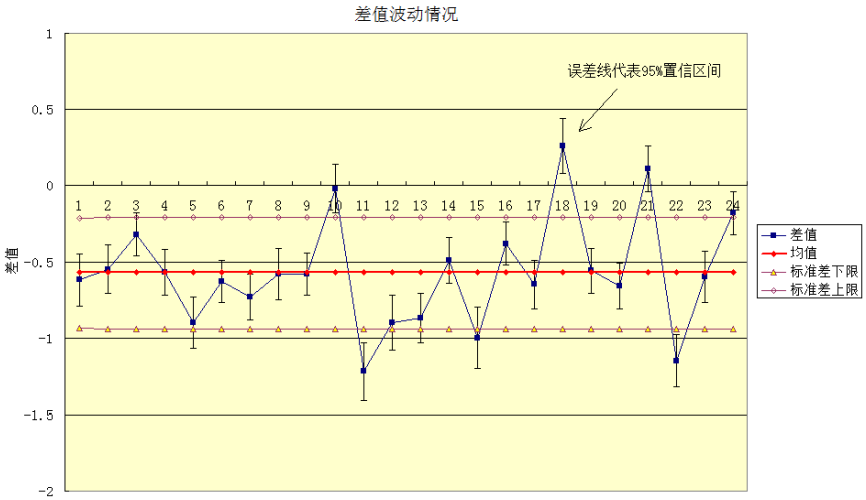


	A	B	C	D
1	1	3.37	2.76	-0.61
2	2	3.01	2.46	-0.55
3	3	3.33	3.01	-0.32
4	4	2.78	2.21	-0.57
5	5	3.39	2.49	-0.9
6	6	3.45	2.82	-0.63
7	7	3.2	2.47	-0.73
8	8	2.95	2.37	-0.58
9	9	3.05	2.47	-0.58
10	10	2.8	2.78	-0.02
11	11	3.42	2.2	-1.22
12	12	3.04	2.14	-0.9
13	13	3.34	2.47	-0.87
14	14	2.47	1.96	-0.51
15	15	3.22	2.22	-1
16	16	3.16	2.78	-0.38
17	17	3.01	2.36	-0.65
18	18	2.46	2.72	0.26
19	19	3.18	2.62	-0.56
20	20	3.32	2.66	-0.66
21	21	2.63	2.74	0.11
22	22	3.11	1.96	-1.15
23	23	2.72	2.12	-0.6
24	24	2.11	1.93	-0.18
25	B 预期值		C 实际值	D差值

从图中可以看出来：基本上系统在各个功能和特征的实际表现都差强人意，存在问题最大的点在：功能或特征 11，功能或特征 22，这需要下一步重点进行改进。另外还需要关注的点包括：功能或特征 5，功能或特征 12，功能或特征 13，功能或特征 15。

3）对差值本身进行分析

从差值的均值为-0.57 来看，用户群整体上并不满意现有产品的表现，但这中间有什么值得注意的情况呢？我们以差值的均值作为中心，以 2 个标准差（0.36）空间作为边界，来分析差值的波动情况。

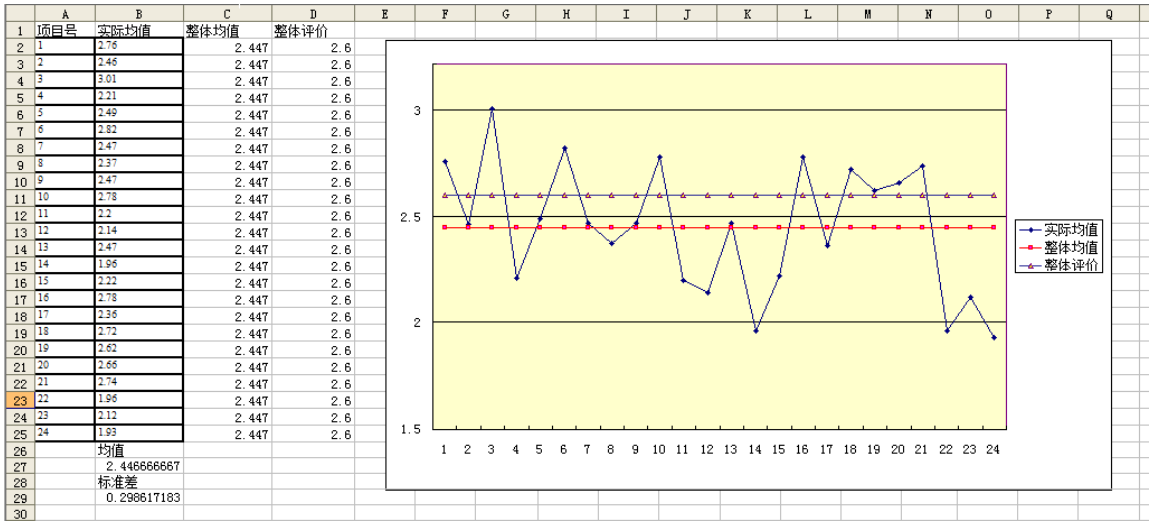


超出上边界为相对表现较好：功能或特征 10，功能或特征 18，功能或特征 21

超出下边界为相对表现较差：功能或特征 11，功能或特征 15，功能或特征 22

4）与整体评价比较

总体评价分数（“请给出您对已有产品的整体看法”）是调查问卷 C 部分问题的平均打分结果，一个有意思的特点是总体评价均位 2.6 高于所有性能表现评分的平均数 2.45 平均位，如下图所示。



项目号	实际均值	整体均值	整体评价
1	2.76	2.447	2.6
2	2.46	2.447	2.6
3	3.01	2.447	2.6
4	2.21	2.447	2.6
5	2.49	2.447	2.6
6	2.82	2.447	2.6
7	2.47	2.447	2.6
8	2.37	2.447	2.6
9	2.47	2.447	2.6
10	2.78	2.447	2.6
11	2.2	2.447	2.6
12	2.14	2.447	2.6
13	2.47	2.447	2.6
14	1.96	2.447	2.6
15	2.22	2.447	2.6
16	2.78	2.447	2.6
17	2.36	2.447	2.6
18	2.72	2.447	2.6
19	2.62	2.447	2.6
20	2.66	2.447	2.6
21	2.74	2.447	2.6
22	1.96	2.447	2.6
23	2.12	2.447	2.6
24	1.93	2.447	2.6
	均值		
	2.446666667		
	标准差		
	0.298617183		

这一差值是非常重要的，它显示了用户群尽管对系统各个属性的评价较低，但对于该系统作为一个整体的产品水平还是给出了较高的评价。这一现象可能是由于宽容的心理，即可能人们虽然知道产品在大多数情况下表现较差，但考虑到投入资金的有限性，认为现有产品还是基本达到了预期。

5）进行分组分析

进一步查看数据分析的结果，我们会发现差值数据点的置信区间比较大，这从另一个方面说明了数据散布比较大，为什么会出现这个情况呢？我们可以试着把被调查者分成组（按年龄、职业、文化程度等）进行类似的分组分析，如果置信区间明显缩小了，那就说明数据特点与分组有明显的相关关系。

我们可以发现某些群体的意见往往和其他几组用户相左，这可能和他们的日常生活、工作息息相关的。这中间有没有共性？有没有个性化的需求？这往往成为新产品独特的亮点。

技术手段是一种工具，而找出需要改进的部位就是一种眼光。有了正确的方向，技术手段就可以用到正确的地方，也为新产品的改进奠定了坚实的基础。使用用户体验来衡量产品有效性已逐渐成为一种趋势。这一整体性方法可以使设计部门、产品部门对产品效果有概括性了解，从而找到改进方向，避免了主观性，产品创新由此而诞生。

差值分析法已广泛应用于这些研究中，而且设计一套合适的问卷也并不困难。然而，对由此产生的统计数据的分析却是一项繁重工作，有时需要统计专家的协助才能完成。

5，分析度量结果决定产品改进方向

通过上述一系列分析，我们找到了产品下一步改进的方法和重点，也就是：

首先要调查功能或特征 5、11、12、13、15、22，尤其是功能或特征 11、15、22，看看问题到底出在什么地方？在第一阶段改进计划中，需要重点加以改进。

而对于功能或特征 10、功能或特征 18 和功能或特征 21，可以不加以关注。

这就有了重点和方向，开拓了视野，也为新的设计提供了想法。

但是也要注意到，仅仅依靠数据是不能作出决定的，数据是一种信息，它帮助我们揭示了问题背后的问题，但数据不是决定，决定还需要靠我们自己思考更多的东西，以及敏锐的洞察力。

分析师需要进一步延展自己的思维空间，站在用户的角度，仔细构思一下未来产品能不能做得更好呢？怎么才能做的更好呢？能不能针对最初解决方案的目的，提供更多更好的服务呢？我们能不能改变已有的仅仅为了完成任务的思维习惯，去和用户一起创作一个崭新的工作方式呢？在明确目标、拥有数据、深入分析的基础之上，我们就可以作出正确的决定。

这一套面对创新产品的构思方法，对于面对单一客户的项目来说也是有所启迪的，这可以为我们的客户创造更好的产品。

2.3 产品的轮廓：未来产品的素描

通过对上述一系列问题的考虑，并且与利益相关方达成一致意见以后，就需要集中精力给未来产品定义一个初始的轮廓，其表现是解决方案的边界和约束。有了这样一个大的轮廓，后期的需求开发将会更好地向着目标前行。因此，这一步非常关键和需要智慧。要注意，在初始阶段的边界定义应该比较宏观而抽象，在后期获得更多知识以后，还可以逐步细化。

一、清晰的表述产品的目标

我们已经花了很大的篇幅讨论如何发现问题，在这个基础上，现在要做的事情就是对产品目标进行清晰的描述。

1，为什么目标十分重要

新产品的诞生首先就来自于一个具有前瞻性的、创新的、具有想象力的产品目标。

什么是目标呢？常见的定义就是“努力去达到的结果或成就”。目标与努力有关，但更重要的是，目标关系到方向。如果在项目前行的过程中出现一些差错，这没什么，修正就是了。但是目标就错了，那就是一个大问题了。目标提供了我们努力的方向和焦点，清晰的定义了我们希望达到的终点，并且为必须要做的事情建立了某种优先秩序。

从项目来说，目标为团队提供了一个公共的框架，形成一个项目团队每个人都通力合作去实现的东西。为了这个目标，我们该做什么？不该做什么？还有什么更好的方法？目标提供了一个是非标杆，避免了无谓的争论，如果有些需求对目标没有意义，那为什么还要劳神去做它呢？

目标也为团队提供了一个粘合剂，一个目标完成之后，团队成员就有了可以共同庆贺的东西，过程很艰苦，但我们完成了，这就有了成就感，每个人都分享到了赞美和荣誉，这种凝聚力成为团队最可宝贵和珍视的东西。而没有共同的目标，团队就仅仅是一个松散的集合，这样的团队怎么可能是高效能的？而这样的项目怎么可能成功？

目标可以消除在开发过程中的恐惧。例如，我们是不是已经达成了想要的结果？我们现在处在开发过程中的什么位置？领先了还是落后了？怎么做才是正确的？这种努力有可能成功吗？所有这些结果、方向、评估和努力，都涉及到项目当初设定的目标。

目标对每个参与项目的人都十分有用，没有目标就不可能去努力，如果不能达成任何目的，那所有的努力事实上就是在浪费时间。没有目标人们就不可能有激情，而激情对于团队或者个人都是十分重要的。目标应该是不能轻易就实现的，正因为如此人们才会为此而奋斗。

目标是人类努力的动力，它给了我们决心，也给我们提供了成就感和满足感。因此，目标无论对于项目、团队以及个人都非常重要。那么我们如何来发现目标呢？

2，简要描述产品的要点

目标是摈弃了一切非关键内容，对产品要点的简要描述。

为什么要做这个项目？我们可以为项目找出很多理由，但我们能不能用最简短的语言（30 秒）把它说清楚？罗嗦也是思路不清晰的表现，在非常短的时间内阐述你的核心思想，对新软件项目

进行简明的表述，对项目会大有裨益。

- 它能够带来清晰的思维。不必面面俱到，简短演讲迫使团队只回答产品是什么、为谁而做这类最尖锐的问题。
- 它能迫使团队从客户角度去考虑问题。通过将注意力集中于产品的功效和理由，团队会深刻地理解产品的魅力所在、客户为什么首先要购买其产品。
- 它能直达要点，不必啰里啰唆阐述很多问题，就好似一束激光，可以穿透并直达项目的核心。这可以帮助设置优先级，更有效地捕捉关键需求。

简短演讲又称之为电梯演讲，所表达的是假定你在电梯里恰巧碰到了客户方的领导，怎么利用短短的电梯上升的 30 秒时间，把自己的观点表达清楚？这是阐述目标的很有效的方法。

讲短话并不容易，现在看一下对简短演讲有所帮助的一个模板。

- 对于[目标客户]来说
- 他们有这样的[需求或机会]
- 我们的[产品名称]
- 是一种[产品类别]的产品
- 我们产品的[主要优势，令人信服的购买原因]。
- 其不同于[主要的竞争产品]
- 在于我们的产品有如下[主要区别]。

简洁并不容易，简短意味着删除了很多无关紧要的东西，只保留了最核心的内容。如果你第一次做得不是很好，也别担心。简洁描述一件事情是一项艰难的工作，不过物有所值。

团队可以有多种方式进行简短演讲。你可以将模板打印出来，让他们都试着填写内容，然后把大家聚在一起交流。或者可以通过大屏幕放映模板，让大家一起填写，再审查模板中的每一个元素。有了简短描述，我们还可以发挥创造能力，思考一下这个产品的亮点在什么地方？

3，描述产品的亮点

没有亮点的产品是平庸的产品，平庸的产品是没有竞争力的。为此，我们需要为项目制作一个包装，把产品最能够吸引人的方面以及产品的潜在优势描述出来，这可以迫使我们考虑如何做一个有特色的产品，对于团队的交付也大有益处。

那么我们如何做到呢？

很多人认为创造力不是谁都具有的，但我们只需三个简单步骤，就可以列出这个亮点。

1）第一步：集思广益，列举出产品优势

不用向客户讲什么产品特性，他们不会关心。人们只会关心产品产品的优势。

举个例子，假设我们正在试图对一个家庭宣讲买辆微型汽车的好处。我们可以为他们列出所有特性，或者向他们展示微型汽车如何提高生活质量。

与仅仅描述技术性能相比，看到差异了吗？因此，发现产品亮点的第一步就是要坐下来，与团队集思广益，列举出人们为什么要使用你的产品。然后挑出最前面的三条理由。

第二步：创建一个广告词

并不一定是在做广告，但好广告词的关键在于使用尽可能少的单词表达产品最靓的部分。例

如，无需告诉你下面这些公司要传递什么理念，他们的广告词就说明了一切。

- 福特：让每个工薪阶层都能拥有自己的汽车。
- 星巴克：奖励自己每天的时光。

能体会到这些广告词所要表达的情感吗？但我们的广告词没必要这么专业。与团队一起进行 10 到 15 分钟的思考，就当是练习一下大脑的创造性。

第三步：设计产品包装

找出了人们被这个产品吸引三个动人理由，再加上吸引人眼球的广告词，现在就可以将其组合在一起了。

非常酷的图片是不是必要？必要的，没有人在枯燥无味的环境中会有想象力，很酷的图片也帮助人们记住广告词。当然，并不是让团队去创作一幅不朽的作品，只是向客户展示出产品优势。有了这个“包装”，目标已经表达出来了，而不是一种人人生厌的官样文章。

在这个过程中还可以得到很多乐趣，这也是一种很好的促进团队建设的方式。即使对于有明确客户的项目，也预示着我们进行了思考，大家展开了想象的翅膀，认真地挖掘了一下软件背后的各种深意。

二、定义解决方案的边界

有了目标，下面要考虑的问题就是为了达到目标，产品应该包含什么。

所谓解决方案的边界指的是解决方案与包围解决方案的现实世界之间的边界，我们必须明确，哪些是解决方案的一部分，哪些是通过系统与外界的接口进行的。换句话说，就是把我们面对的世界分成两部分：我们的系统，以及与我们系统进行交互的事务。最终的范围定义需要在需求获取以后对产品进行进一步分析之后才能确定下来，但是一开始就给出一个初步的范围定义还是有好处的，这可以大致上圈定我们的工作范围。在初步的范围定义阶段我们需要做的事情如下。

1，列出问题和机会：

需要关注以下几个方面的问题：

- 紧急程度（什么时间实现？）；
- 可见性（系统在多大程度上对客户或执行管理层是可见的？）；
- 收益（新方案会增加或者减少多少支出？）；
- 优先权（那些问题是最重要的？如果预算出了问题，需要减掉哪些问题？）；
- 可能方案（用简单的方式表述：如，不改变令人满意的事物、快速修复、对现有系统改进、重新设计现有系统、设计一个新系统等等）。

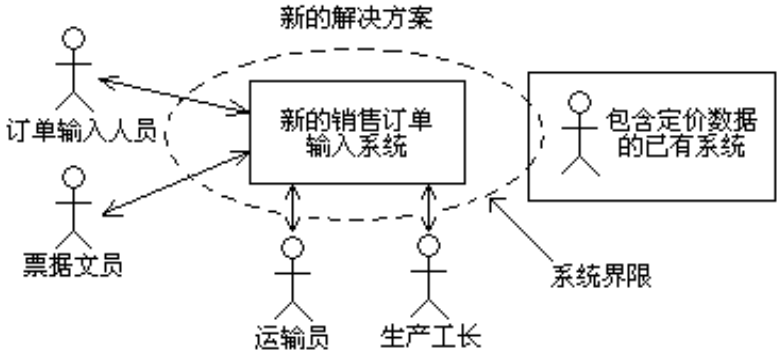
2，发现项目的外部系统

在很多情况下系统的边界都很明显，但是有些情况下却不是那么清晰。比如一个订单输入的例子，这个系统要集成在一个更大的系统中，分析人员必须确定它是否要和其它系统共享数据，新的系统是不是要在不同的主机或客户之间分布的，等等。

在确定范围的时候，一般遵循先外后内、由粗而精的原则。首先需要发现谁在使用和支持系统，也就是把系统看成一个黑箱，集中精力发现系统的外部参与者，这看起来很明显，但是我们如何才能找到参与者呢？下面的问题可以有所帮助：

- 谁会对系统提供信息？谁会在系统中使用信息？谁会从系统中删除信息？
- 谁是系统的操作者？
- 系统将会在哪里被使用？
- 系统从那里得到信息？
- 哪些外部系统要和系统进行交互？

回答了这些问题之后，分析人员就可以得到一个宏观的上下文图，这也是项目启动会议上首先要研究的图，下面是一个简花了的宏观系统上下文图，这样的图不需要面面俱到，但足以表达所有的重要信息。



这样的上下文图实际上是一个系统宏观上的透视，或者是一个体系上的轮廓，并不需要绘制详细的信息流和事无巨细的相邻系统细节。

3，发现项目的内部功能集

在获得谁在使用和支持系统的基础上，再考虑他们要干什么？也就是列出内部功能集。经验可以告诉我们，要判断一个功能集是属于项目内还是项目外并不容易，而且常常会有两种截然相反的观点。这种决策有时还会花去很长的时间。我推荐在初期需求分析会议上利用一下所谓“内/外列表”，这种表共三列，第一列可以是任何内容，每当对某个主题是内还是外有疑问的时候，可以把它加入表格，并询问这属于项目“内”还是“外”？

这时候需要关注：哪些功能集是已有的，能够靠其它系统提供的？哪些功能集可能是人工操作，而不是自动化系统完成的？哪些功能集可以购买外部服务，而不是自己开发？在后期需求分析中，当队内外有疑问的时候，也参考这张表，并随着工作的进度对表格进行修改。下面是为“购买请求跟踪系统”建立的内外列表。

“内/外”列表		
功能集	**内**	**外**
以任意形式开发票		√
产生请求报告（请求可能由买主、分部或某个人发起）	√	
把请求合并成一个队列	√	
部分发货、延迟发货、错误发货	√	
所有新的系统服务、软件	√	
系统中任何非软件部分		√
识别任何可用的已存在软件	√	
申请	√	

项目的范围可以描述成一个简单的列表，但不需要详细定义列表中的功能，也不十分关心精确的需求分析，尤其不关心任何费时的建模或者原型化。问题在于这样作出的范围列表可以稳定下来吗？事实上在我们的实际工作中范围发生变化的情况时有发生，这对软件的开发与质量带来极大的困扰，所以，下一步还需要对范围进行进一步分析，使范围可以稳定下来。

如何进行分析呢？在我们在与客户沟通的实践中为什么会感觉沟通不畅？其根本原因是掌握的信息不足，连讨论的基础都没有，还谈何沟通？因此我们需要进一步获取相关信息。

很重要的两个信息就是：这个项目究竟多大？另一个是它有什么风险？尽管这些信息似乎与需求分析无关，但这些信息使我们对范围的确认有了依据。例如，原来认为可以在这一版解决，但考虑到规模，就放在下一版了。原来感觉在规定的时间内可以完成预定的内容，考虑到了风险，在预定的时间内可能完成不了，这无疑增加了与客户沟通的机会。

3，估算项目的规模

为了判断所列出的范围是否合理，需要简单估算一下项目的规模。这里说的是要尽量计算出在这个范围内的项目是要一个月、三个月还是六个月才能完成。我们无法得知更准确的时间，但是仍需让项目发起人知晓他们的软件何时可以交付，即使只是一个大体上的猜测。下面是对于初步估算步骤的建议：

第一步，分解：

由于估算小块的工作更容易，因此在估计的时候首先把任务进行分解。记住，估算是一种猜测，猜测的范围越大，可能产生的错误也就越多。

第二步，估计每一部分规模：

基于经过确认的模型或历史数据，通过比较来估计规模。在估计的时候，除了与历史数据进行比较以外，还需要把正在估计的任务相互进行比较，以确定更准确的结果。随着我们对产品特性与属性的关系的理解不断加深，确定这些属性的方法也随之而发展。

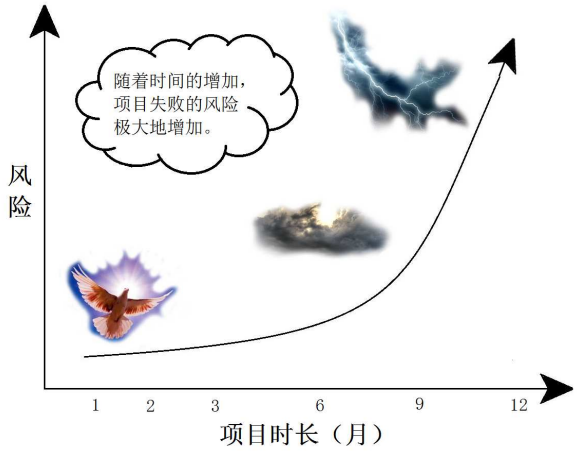
第三步，估算工作量：

根据历史数据，把规模转换为相应的工作量。这里要注意的是估计不要太乐观，一项任务事先没有想到的情况会很多，其规模一般比预想的大。

第四步，估算项目时间

根据估算出的工作量，最后给出为完成这些工作量的任务，大约需要多少时间

一个重要的成功经验就是：坚持开发周期不超过 6 个月，如下图所示。



大型、无限制项目的问题就在于，它们似乎永远都过度承诺但又延迟交付。总要添加更多的东西和加人更多的特性。不久，成本扩大了，项目随着自身不断加大的重量而崩溃。

值得留心的就是项目周期不要超过 6 个月，或者应更短一些。只要超过这个周期，风险就很

大。完成就是完成，集中优势兵力各个歼灭敌人，让每一个子项目成功的经验带到下一个子项目。这比迷茫于一个庞大遥遥无期的项目，更容易给人带来成就感。

这不是说我们希望交付的所有 IT 新方案都可以在 6 个月内建设完毕，而是如果想交付一些大块头项目，就必须要将其分为更小、更易于管理的小块。

4，讨论一下风险

在项目的开始阶段，多数人都避而不谈项目风险。但谈论风险是一种绝佳的方法，能让人们了解到项目的成功都需要哪些要素，这些事情也不一定会发生，我们只是考虑过它，将来在问题发生了的时候不至于手足无措。

谈论风险的最好时机就是在项目的初期。如果你有什么问题或者见到任何瑕疵，最好趁现在将其公布于众。与客户、团队分享并讨论你的恐惧感会有益处，这样大家就有机会结合在一起，从他人的经验中学些东西。初步讨论一下风险，对于确定合理范围也相当有意义。一般来说，考虑风险可以关注如下一些方面的问题：

1）产品规模

项目的风险与产品的规模成正比的，常见风险因素有：

- 产品规模估算是不是受到信任？
- 产品规模估算的平均值偏差是多大？
- 产品开发生命周期是不是太长？
- 可复用的软件要求是不是太多？

2）需求风险

在确定需求时一般都面临着一些不确定性。如果在早期容忍了这些不确定性，并且在项目进展过程当中得不到解决，就会对项目的成功造成威胁，常见的风险因素有：

- 各方是不是对产品缺少清晰的认识？
- 各方是不是对产品需求缺少认同？
- 在需求收集中客户参与度够吗？
- 未来需求会不断变化吗？
- 在需求变化时管理过程合理吗？
- 在需求分析时进行了变化相关分析吗？

4）技术风险

软件技术的飞速发展和经验丰富员工的缺乏，意味着项目团队可能会因为技巧的原因影响项目的成功，常见的风险因素有：

- 团队成员经过了充分的培训吗？
- 团队成员对方法、工具和技术理解足够吗？
- 团队成员应用领域的经验是不是足够？
- 该项目采用了新的技术和开发方法吗？

一般把风险清单的的前十位单独列出来加以关注。随着项目的进展，已经解决的风险可以删去，新发现的风险需要加入进来，也可以画出风险随着项目进展的变化曲线。

而当你很难判断是否值得为一个特殊问题担心时，下面的三条可以帮助我们：

- 接受我们所无法改变的；
- 改变我们所能改变的；
- 用智慧去分辨差异。

你不可能预测到未来的一切，生活中会遇到各种各样的挑战，未来会发生什么你可能知道，也可能不知道，甚至永远也不会知道。但考虑过和没考虑过是不一样的。

5，创建否定清单

在上述两个问题明确之后，现在我们可以坐下来进一步思考最初的范围列表了，要着手解决一个关键性的问题就是：哪些事情可以不做？

也就是说，重要的不是项目要做什么，而是项目不做什么！

这个命题对客户来说可能不那么高兴，但最终获益者也是客户。说到要做什么，人们往往会兴奋的往里面添加自己想要的东西；但说到不做什么，人们就会问：为什么？正是这个为什么，促进了人们的思考。是因为时间？成本？还是风险？

舍弃是一种平衡的艺术。在分析了规模和风险之后，就减少了意气用事，而更多的是一种科学考量，也可能由此引出了项目发展的路线图。

设置项目范围的时候，要做什么和不做什么都要说清楚。通过创建一个否定清单，可以清楚地表明项目范围内外所涵盖的内容。这样不仅可以设置清晰的客户期望值，还能确保你和团队排除杂念，专注于真正重要的事情。

范围内	范围外
创建新许可证 升级 / 读取 / 删除现存许可证 搜索基础报告 打印	与遗留封路系统有接口 离线能力
未解决的	
与后勤记录整合 安全卡刷取系统	

在图中：

- **“范围内”：**包括需要关注的事物。它们可能是些高级特性（比如说，报告），也可能是一般目标（比如说，与亚马逊一样的可扩展性）。
- **“范围外”：**包括的是那些无需担心的事物。可能是那些可以推迟到下一版本再发布的东西，也可能是超出了本项目的范围，现在没必要考虑。
- **“未解决的”：**仍需做出决定的事情。最终，我们希望将所有未解决的事项移到内或者外部分中去。

这样形象化地展示可以马上传递出很多内容。将范围内的候选项放在左边，范围外的放在右边，未解决的放在下面，每个人只要看一眼，就能明白项目的范围。

这个过程中的关键点是：展示范围不是目的，而是沟通的手段。决定不做什么不是分析人员说了算，而是客户在分析人员帮助下掌握了各种情况之后，所做出的决定。如果暂时不能作出决定也没关系，“未解决的”栏目记录了这些内容，我们在后面通过收集客户需求掌握了更多情况之后，还需要回过头来更细腻的探讨范围问题，此时这些问题都可以作出决定。

6，协商

在考虑范围问题的时候，牵涉到一个分析师的重要能力，那就是协商。

需求在很多情况下会牵涉到多个利益相关方，他们所处的领域、职责、目标以及对于需求的理解不同，要求也不同，这就需要协商。能够进行协商的基本条件，是客户对于分析师能力的信任。本质上说，软件开发是一种服务，站在开发的角度来看，分析师相对于客户来说是处在服务方，客户方希望服务方提供的是专业知识（想象一下，人们对于基金、理财、银行、律师、咨询的期望），如下图所示。

如果客户对分析师没有起码的尊重和信任，那协商也就不存在了。一般来说，客户对于服务方越尊重，协商越容易成功。那么尊重和信任从哪里来？是由过去的工作积累和证明的。如果在过去所承担的工作中，分析人员信誓旦旦的说在规定的时间内可以完成，结果交付时间总是拖期；如果分析人员说没有问题，结果总是问题百出，那以后分析人员说什么还有什么分量？如果分析人员不能在与客户交流的时候展现专业性，那客户凭什么信任你？

因此，分析人员需要在工作中提高自己的水平，并且不断提高自己预估的能力。一次不准确不要紧，但要分析原因在什么地方，在下一次预估的时候要进行修正，正是在这种不断的修正中，人的能力在不知不觉中就上升了。

在这个基础之上，下面对于正确的协商提供一些建议。

1）不要模棱两可

需求是一种约束和规则，容不得半点的模糊和多义性。

是就“是”，不是就“不是”。即使很难，也要坚持做正确的事。

含糊的承诺，后期造成项目需求的朝令夕改，对很多项目来说就是一场噩梦。

如果在项目实施以后才发现那个需求的决定真的太糟糕了，应当做出必要的调整而改变需求（但不能经常如此）。此时，应当让受影响的各方知道做了哪些修改，以及他们需要做出或考虑哪些调整。坚持错误的需求决定会对整个项目带来损害，重要的是尽量不要在项目方向上做出改变。

2）倾听你的内心呼唤

如果我们对某个将要做出的需求决定有潜在的保留，就应当把它拿到台面上。

说需求分析师不是记录员，是因为分析师有自己的经验、眼光和判断力。我们要学会表达自己的内心想法，内心的想法也许会促使我们做出某些调查，做一些讨论，在特定地方帮助证实或否决我们的想法。

不要想着马上解决问题。要让对方尽可能多地说话，从而获取尽可能多的信息。必要的话，召集多个利益相关方开协商会，来得到有关此问题的更广泛信息。会谈过程中，应复述对方表达的信息，以确保你理解了别人所说的话。

一旦能够准确地评判问题，并真正理解了别人的背景，就可以由倾听和理解转而寻求解决方案。这时，有必要总结我们的理解并写出来，这样可被共享和确认。

一个需要注意的问题是，提出不同看法的人是否曾有机会自己考虑过这个问题？如果与会者自己没有进行关键性的思考，那么只给出一个答案并不能帮多大忙。将对问题背景做出的总结呈现给他们，请他们仔细考虑一下这个问题，再进行讨论时可能就更容易达成一致。

3）寻找共同点

各种背景的利益相关方走到一起，以某种方法找出他们的共同点，让各方的人都从中受益，这就是需求开发中最经常要做的事情。

当不同利益相关方凑到一起，第一个议题就应当是找到共同目标。共同的目标就是所有人都赞成的目标（换句话说，就是找出需求是什么）。下一步，大家要找出“成功”是如何定义的。也就是说他们如何评判问题已圆满地解决了呢？最后，人们开始着手工作，找出解决问题的办法。

为了成功地完成任务，所有人都要撇开偏见，寻求方法、解决问题。该方法能够达到成功的衡量标准，并且不是只照顾某个方面的好处。

要是始终无法达成一致，就要花时间检查如下这些问题：

- 找出他们立场的根源所在（必要的话，私下做这件事）。
- 找出他们最关心的是什么（你也许能让他们在不太重要的地方做出让步）。
- 找出他们是否有决策的权利（如果没有，你可能需要将有真正决定权的人请到协商桌旁）。

成功解决分歧就意味着做出让步、做出牺牲。在处理问题时，最好的做法是试着站在对方的角度，从他那边看问题。如果你理解了对方的观点和需求，找出共同点就容易多了。

4）如果无法达到一致，就让所有人稍微不满吧

我们的首个目标就是让每个利益相关方满意。但我们必须认识到，这样的结果往往不可能做到，经常需要做的是折衷平衡。

生活中有个出乎意料的事实：一个不同的人一起有效工作的团队，往往是团队中的每个人都稍微不高兴。如果团队中每个人都要用对自己来说舒服的方式行事，团队合作就不存在了，这就需要规范和约束。对每个人来说稍稍有那么一点不舒服，并且舍弃了某些东西，这种妥协才可能让团队能够以合理的花销来成功完成其任务。

决策是一个生态系统，有些情况下，需求并非所有人都一致同意。不同利益相关方对问题的看法会有不同，这就要求所有利益相关方都为了共同的利益，舍弃个人前嫌而共同努力。这种“先舍而得”的结果，就是共赢的局面。这就需要每个方面不得不做出一些妥协，以便让大家都能接受这一个需求决定。

协商，是需求分析中最具艺术特征的东西。协商本身就是一个沟通的艺术，是平衡的艺术，是取舍的艺术，是实现共赢的艺术。学会如何协商对于每个人都是有挑战性的技能。

三、确定解决方案将受的约束

在涉及巨资的系统开发之前，我们必须停下来，考虑一下系统可能受到约束条件调节，比如：

进度：系统必须在 4 月 15 日前运行。

成本：系统成本不能超过 35 万。

技术：所有的新系统必须使用 Oracle 数据库。

政策：新系统必须使用“双倍递减余额”技术。

约束条件是我们提供的解决方案拥有的自由度的限制，大部分的约束可以通过下面的检查表得到合适的解答。

约束源	可能的例子
经济的	● 哪些财政或预算的约束可以应用？ ● 由销售的成本问题需要考虑吗？ ● 存在任何许可问题吗？
行政的	● 有可能影响解决方案的内部或者外部行政问题吗？ ● 有什么部门间的问题吗？
技术的	● 在我们技术选择上有什么限制吗？ ● 我们是否会被限制在已有的平台或者技术上工作？ ● 我们对新技术的应用有限制吗？ ● 我们需要使用购买的软件包吗？
系统的	● 解决方案要集成在现有的系统中吗？ ● 我们必须维护和现有解决方案的兼容性吗？ ● 必须要支持哪些操作系统的环境？
环境的	● 由环境和管理的约束吗？ ● 合法吗？ ● 有安全性需求吗？ ● 我们可能被哪些标准限制？

进度与资源的	● 有进度要求吗？ ● 我们被限制使用在已有的资源上吗？ ● 我们可以使用外埠劳动力吗？ ● 我们可以扩展资源吗？暂时的，还是永久性的？

在确定约束条件之后，有些会成为新的需求（“使用现有的会计系统提供商的 MRP 系统”），其它约束会影响到资源、实现计划以及项目计划。作为问题解决者，我们有责任找出每个特定应用环境的潜在约束，并确定每个约束对问题解决空间的潜在影响。

四、总结归纳：项目的陈述

经过上述所有的分析，我们就可以把所有的考虑集合起来，书写出一个完整的项目陈述，一般这个陈述需要描述以下几个方面的内容：

- **问题**：不期望发生的情况，它妨碍组织完整的实现其任务。
- **机会**：能够改善组织的可能性。
- **指示**：管理层、或者其它外部影响强加的新需求。

具体的问题陈述一般应该包括如下几个部分。

- **项目的目标；**
- **项目概念；**
- **问题陈述；**
- **项目影响范围；**
- **项目构想；**
- **有关的约束：**业务限制；技术限制；

文档化的“项目陈述”非常重要，它描述了项目的整体思想，把项目开发的目标与轮廓用文图形和文字的描述出来，以便于利益相关方阅读和评价。随着项目的进展可能会发生很多变化，但是项目陈述确定了目标，是项目在动态过程中不至于偏离目标。当然，这种项目的概念分析将伴随着项目开发的全过程，是一个不断精化和优化的过程。

第三章　面向客户：如何开发客户需求

通过项目的需求规划，我们已经站在比较高的层面，就项目的目标、问题、概念、范围、目标达成了共识，形成了清晰的目标与轮廓，为获取需求打下了坚实的基础。

现在需求获取的主角转向了产品研发团队。分析人员应该抛开宽泛的思维方式，暂时不去考虑产品应该是什么样的，而集中精力在客户想要什么东西这个更窄内范围开展工作。这样的入口不但更细腻，开展需求收集与分析也更容易。需求收集不仅仅是问答，而是在学习客户业务的过程中去理解，理解了才可能沟通。这需要分析师具有很好的学习力、理解力和沟通能力。

3.1 建模分析：让复杂变得简单

在客户需求阶段，以技术团队为主的需求分析人员需要更好地与客户沟通，发现并梳理客户到底需要什么东西，把软件产品战略构想细化，这就需要有一些新的视角。那么我们从哪里入手呢？一句话，从发现业务的概念入手是个很好的入口点。

一、概念模型：发现业务的共性及其关系

1，为什么需要从概念入手收集需求？

我们在自己的工作中经常会发现这样的情况：很多人为了便捷，直接要客户说明他们需要什么。但是等产品做出来，才发现做出的东西和客户的需要大相庭径，到那个时候再改已经为时已晚了。为什么会发生这个情况？

人类相互间的沟通主要依靠的是语言，语言的核心是概念（用名词表达），概念反映了对一种事物共性的抽象。但是在不同的领域、行业甚至组织（企业）范围内，对同一个名词会有一些习惯性或者专业性的内涵不同，如果双方对同一个概念理解是一致的，就不容易产生歧义。但如果事实上不一致但各方却认为这是一致的呢？这就很容易产生歧义而且不容易发现。

也就是说,，当我们与客户沟通的时候最大的障碍是什么？是双方对概念理解上的不一致！

为了解决这个问题，很多领域对名词做了规范，例如：国家规定了科学用语规范，军队规定了军语规范。但规范并不能涵盖一切，习惯用法比比皆是。更重要的，软件开发本质上是跨行业跨领域的，这种跨界特征本质上会跨越多个概念定义区，如果不从概念入手，想当然的认为双方看法是一致的，后期出现问题不是正常的吗？所以，从概念入手收集客户需求，是一个简捷而高效的手段。

举个例子：

为什么我们课程第一章会花功夫讨论是项目还是产品？

在很多软件企业发展的历程中，会逐步分离出两个不同的部门和职责：项目部门负责开发有明确客户的产品；产品部门用于策划没有明确客户，面对广大客户群的产品。这形成了一种企业内的表述习惯，如果我无视这种习惯，仅仅按照软件工程学统一的规范语言来描述，人们在听课的时候自然会想：客户需求是项目部门的事，产品需求是产品部门的事。这就产生了歧义和理解上的混乱，使课程效果大打折扣。

因此，在课程的第一章主要作用就是统一概念（包括什么是需求工程），而统一的依据是工程领域内对相关概念的定义：

项目（Project）：是一个唯一（临时的）的，具有同一目标，并且在特定时间内、在预算内、按照质量要求完成的活动序列。

产品（Product）：是指提供给市场，被人们使用和消费，并能满足人们某种需求的任何东西，包括有形的物品、无形的服务、组织、观念或它们的组合。

那么这两者是什么关系？项目的特征是活动，它的目标是按要求产生产品。产品是项目的交付物，它的特征是东西，目标是满足需求。那么客户是不是市场？当然是，除非你不收取费用只是朋友间帮忙。把这两个概念讲清楚了，后续展开课程的时候，就不至于产生理解上的歧义和混乱。所以为此耗费的时间是值得的。问题在于这样来展现概念合理吗？

2，如何展现并统一概念呢？

概念的不一致是隐含的，如果事先问问我们概念上有什么不一致呀？你会发现双方都会两眼茫然，谁能知道对方和我有什么不一致？因此需要有一种便捷的方法来展现概念，在过程中发现不一致并且统一概念。因此，理解业务的入口点就是理解客户的业务概念！

面向对象的分析（OOA）强调从概念模型入手分析需求，为我们提供了发现并理解概念的方法。什么是概念？概念是一种抽象，它用类（Class）来表述。但概念不是孤立的，因此必须通过建立静态的概念与概念之间关系的模型，使我们能够迅速理解领域。

什么是模型？

- 模型是一种抽象，它摒弃了细节而专注于关系
- 模型是一个视图，它用图形展现关系，更有效的触发人们思考
- 模型是一个交流的手段，它使交流更快捷有效。

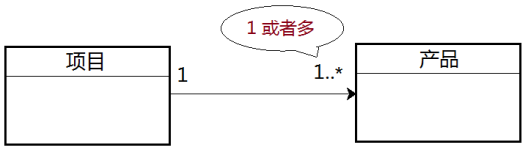
建立模型是把复杂的事情便简单的重要手段，那么如何建立概念模型呢？以我们上面阐述的关于项目和产品的例子来说，如果把所有的概念都用已定义文字的形式来表述，既不现实，也不能为客户所接受。那么我们可以换一种方法。

1）有哪些概念？

先分析一下在我们所关注的领域里（概念域）有哪些概念，把它们用不考虑行为的类图表现出来。例如，我们所关注的领域是项目和产品：

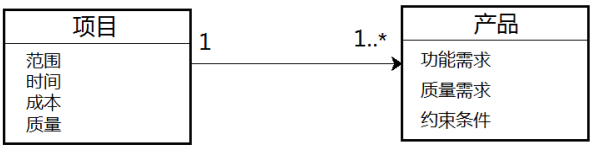
2）概念之间是什么关系？

散乱的概念没什么意义，必须把概念用关系联系起来，才可能形成思维系统。因此需要再考虑一下各个概念之间是什么关系，用关联线表达出来。这种关联也揭示了一种依赖，也就是项目要做的事情依赖于将要开发的产品。



3）概念有什么属性？

上面的模型还是太粗燥，所以在建立了关联之后，再回过头来仔细思考一下每一个概念，这些概念在现实中（对象），用哪些数据可以表征它们的区别？这就是属性。表征属性的时候需要考虑这样几个问题：它们能完整定义这个类的特征吗？它们能用某种信息或数据表达吗？这种信息或数据能够获得吗？



以这三个步骤和客户一起边画图边讨论，就可以把口头描述的东西由最初模模糊糊的状态，逐步地变得清晰了，清晰了也才便于发现问题、达成共识。由于建立这种概念模型是为了帮助我

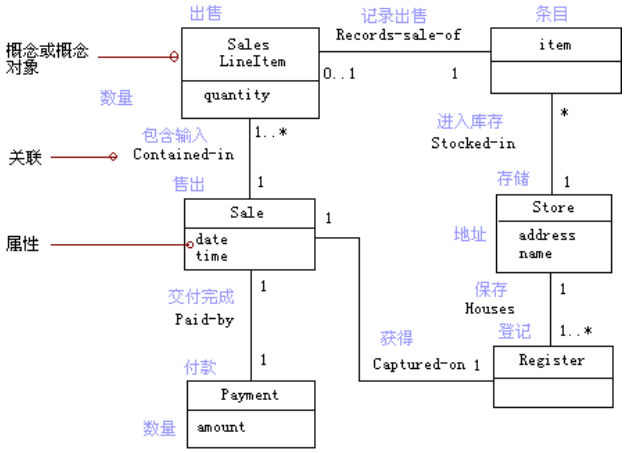
们理解不同领域的问题，有时候也称之为领域建模。

这种分析方法还有一个附带的好处，那就是帮助我们发现数据。

在软件中，任何实体都是用数据的方式展现的。而概念类往往体现了现实中的实体，它们所需要的数据往往在属性中得到体现，而概念类之间的关系，往往可以帮助我们建立数据之间的关系。因此建立概念模型往往成为建立数据模型的基础。

下面再举个简单的例子，说明概念建模是如何帮助我们理解领域问题的。

例如：两个概念类 Payment（支付）Sale（销售）在概念模型中以一种有意义的方式关联。



综上所述：

面向对象的分析首先要进行的就是概念分析。所谓概念是我们感兴趣的现实对象的一种共性抽象。在初期进行需求分析的时候，我们所面对的客户业务概念很多，为了快速理解概念，就需要通过模来达到目的。

在概念建模中，我们用不定义行为（方法）的类来表达概念，用属性表示概念具有的特征，用关连表达它们之间的关系。从而通过可视化的表示来帮助我们展现思维。

仔细考察上面的内容，可以看出概念模型实际上是可视化了领域中的单词或术语，所以它还可以帮助我们进行领域术语分析活动。

通过对我们感兴趣的概念对象的可视化表示，可以帮助我们更深入的讨论和理解问题。“讨论和理解”这个观点很重要，理解业务不是一个分析师把自己关在屋子里冥思苦想得到，而是要充分的讨论和研究，可视化模型表示可以使我们的讨论更加生动活泼，充份激发人们的想象力。

3，概念类的识别

理解了概念建模的意义，真要建模就需要讨论一些细节，首先得问题是如何识别概念类。

我们怎么样来识别概念类呢？首先是发现和领域术语相关的特征。领域术语反映了领域相关人员对领域的理解，是领域知识的一种沉淀。在一个成熟的领域内，人们可以通过领域术语方便地表达自己的思想，并互相沟通。术语名称可以作为概念类的候选名称。同时，领域术语分析也是对领域术语标准化的一个过程，特别是在一个不太成熟或存在同义、近义术语的领域内。

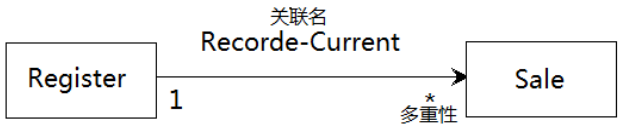
尽管识别概念类的方法很多，但我们还是推荐使用名词短语分析找出概念类的方法，毕竟在一个领域中惯用的词汇都包含了相应的概念。但使用这种方法必须十分小心，从名词机械的映射肯定是不行的，而且自然语言中的单词本来就是模棱两可的。不过，这仍然是灵感的另一种来源。

4，概念类的关系

孤立的概念是没有意义的，我们必须建立概念与概念之间的关系，才能让一个概念有意义，并且把概念的本质研究清楚。

1）找出关联

关联，是类之间有意义或相关连接的一种关系。关联事实上表示是一种“知道”。如果不写箭头，关联的方向一般是“从上到下，从左到右”。



找出关联的指导原则如下：

- 把注意力集中在那些把概念之间的关系信息保持一段时间的关联（“需要知道”型关联）。
- 太多的关联不但不能有效的表示概念模型，分而会使概念模型变的混乱，有的时候发现某些关联很费时间，但带来的好处并不大。
- 避免显示冗余或者导出的关联。

2）角色和多重性

关联的每一端称之为“角色”。多重性表示一个实例，在一个特定的时刻，而不是一段时间内，可以和多个实例发生关联。关联的表示方法如下：

- “*”表示多个。
- “1”表示一个。
- “0..1”表示 1 或者没有，比如一个商品在货架上，可能售出，也可能被丢掉了，这种情况，用“0..1”是合理的。

问题是我们需要关心这样的观点吗？如果是数据库，可能表达这个数据存在，或者损坏。但在概念模型并不表示软件对象，通常我们只对我们有兴趣的内容建模，从这个观点出发，也可能只有“1”或者“*”是合理的。

再一次提醒，发现概念类比发现关联更重要，花费在概念模型创建的大部分时间，应该被用于发现概念类，而不是关联。

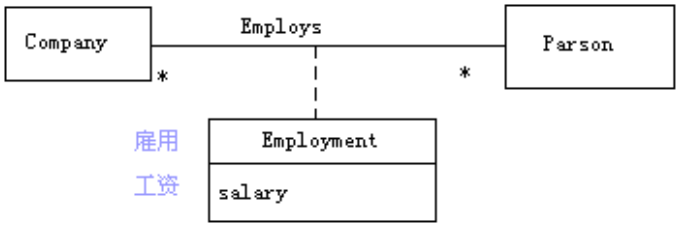
3）关联类

有时候概念类之间牵涉到多对多关系，就需要考虑关联类的问题，例如：

- 在通讯过程中，授权服务为每个商店分配一个店主 ID。
- 每个商店可以支持多个不同的授权服务。

那么，商店的 ID 的属性到地方在什么地方呢？把 merchantID（店主 ID）置于 Store（商店）类中是不正确的，因为一个 Store 类会有多个 merchantID 值。把 merchantID 置于 AuthorizationService（授权服务）类中也是不恰当的，因为它同样也有多个值。

因此，在概念建模中，如果一个类 C 的属性 A 同时具有多个值，那么不要把属性 A 放到类 C 中，而是把属性 A 放到与 C 关联的关联类里面去。下面的例子，表达一个人可能受雇于多个公司的关联类。



说明：组合与聚集

在 UML 中表达关联的时候，还常常用到组合与聚集，用以强调整体与部分的关系。

组合（composite）用于整体—部分关系建模的一种关联，它表达部分的存在全生命周期依赖于组合。比如：手指的生命周期与手一致，用实心菱形表达：

聚集（Aggregation）含义是，部分属于整体，但生命周期可以不一致。比如：森林的存在依赖于树木，但树木的生命周期可以与森林不一致，用空心菱形表示。

在设计模型中，发现和使用组合与聚集相当有意义，其含义也很清楚。它表达某一个对象的创建对于另一组对象的影响。但是在概念模型中，这种整体的创建对部分的影响并不太清楚（一个实际的销售并不可能清晰描述要创建的实际销售商品），所以，花功夫发现这种关系并没有什么实际意义，也不会产生深远影响。很多经验丰富的建模者最终发现，它们在关联的细微含义上浪费了太多的无谓的时间。所以，在概念模型的关联问题上，可以排除这两种表示方式。

5，概念类的属性

属性代表了类中决定对象特征的信息，因此发现和识别概念类的属性，是很有意义的。属性是一个逻辑值，它主要用于保留对象的状态。

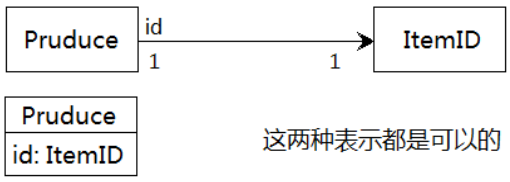
1）有效的属性类型

大部分属性应该是简单数据类型。当然也可以使其它的一些必要的类型，比如：Color（颜色）、Address（地址）、PhoneNumber（电话号码）等。

2）非原始的数据类型类

在概念类中，可以把原始数据类型改成非原始数据类型，请应用下面的指导原则：

- 由分开的段组成数据（电话号码，人名）
- 有些操作和它的数据有关，如分析和验证等（社会安全号码）
- 包含其它属性的数据（促销价格的开始和结束时间）
- 带有单位的数据值（支付金额有一个货币单位）
- 对带有上述性质的一个或多个抽象（商品条目标识符）

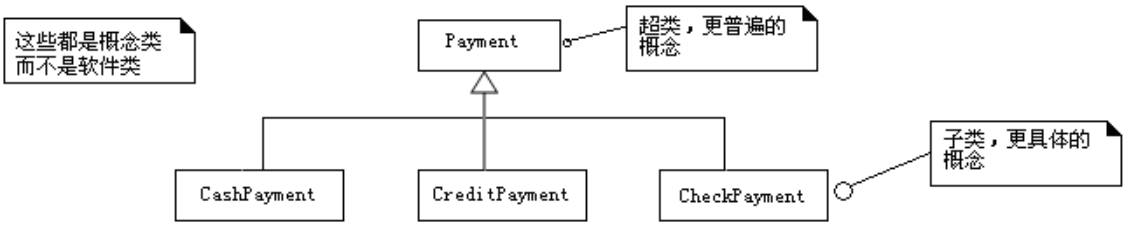


5，概念类的泛化与抽象

泛化建模的目的是发现共性，这往往是软件层次的基本来源，也是面向对象分析中抽象性最重要的表现形式。

1）泛化的基本表现

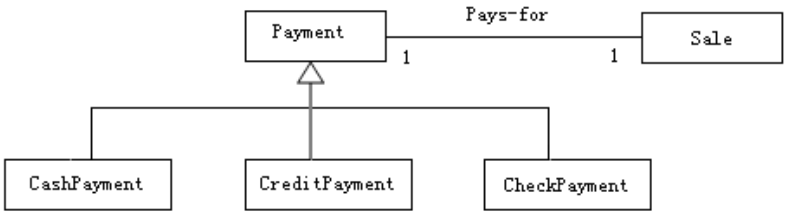
CashPayment（现金支付）、CreditPayment（信用卡支付）和 Check Payment（支票支付）这几个概念非常接近，可以组织成一个泛化的类层次，其中超类 Payment（支付，具有共性）是更普遍的概念，而子类（具有个性）是一个更具体的概念。



泛化（generalization）是在多个概念之间识别共性，定义超类和子类关系的活动。在概念模型中，识别超类和子类极其有价值，因为通过它们，我们就可以用更普遍、更细化和更抽象的方式来理解概念，从而使概念的表达简约，减少概念信息的重复。

2）概念性子类集的一致性

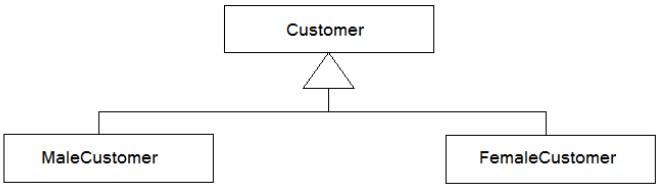
一旦创建了类的层次，有关超类的声明也将适用于子类。一般的说，子类和超类一致性符合“100%规则”，这种一致性包括“属性”和“关联”。



一个概念性子类应该是超类集中的一个成员，这种一致性称之为 is-a 规则。所以，下面这样的陈述是可以的：“信用卡支付是一个支付”（CreditPayment is a Payment）。

3）何时定义一个概念性子类

举个例子，把顾客（Customer）划分为男顾客（MaleCustomer）和女顾客（FemaleCustomer），从正确性来说是可以的，但这样划分有意义吗？



这样划分是没有意义的，因此我们必须讨论动机问题。把一个概念类划分为不同子类的强烈动机为：当满足如下条件之一的时候，为超类创建一个概念性的子类：

- 子类具有额外的相关属性。
- 子类具有额外的相关关联。
- 子类在运行、处理、反应或者操作等相关方式上，与超类或者其它子类不同。
- 子类代表一个活动的事务（例如：动物、机器人），它们与超类的其它子类在相关的行为方式上也不同。

由此看来，把顾客（Customer）划分为男顾客（MaleCustomer）和女顾客（FemaleCustomer）是不恰当的，因为它们没有额外的属性和关联，在运行（服务）方式上也没什么不同。尽管男人和女人的购物习惯不同，但对当前的用例来说不相关。这就是说，规则必须和我们研究的问题相结合。

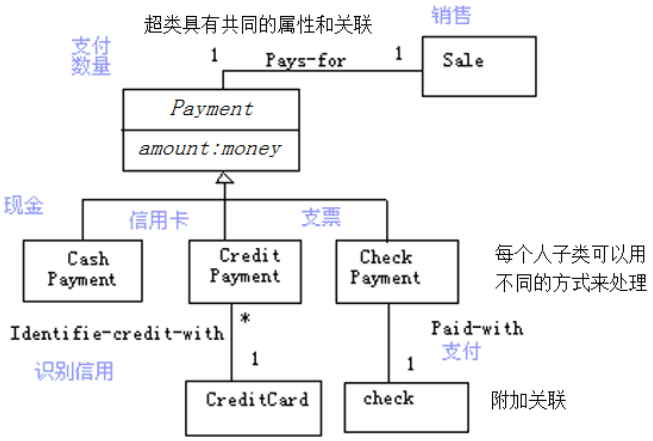
4）何时定义一个概念性超类

在多个潜在的子类之间，一旦发现共同特征，就可以暗示可以泛化得到一个超类。下面是泛化和定义超类的动机：

- 潜在的概念子类代表一个相似概念的变体。

- 子类遵守 100%的 is-a 规则。
- 所有的子类具有共同的属性，可以提取出来并在超类中表示。
- 所有子类具有相同关联，可以提取并与超类相关。

例如：由于个关于支付的概念类（Payment）：



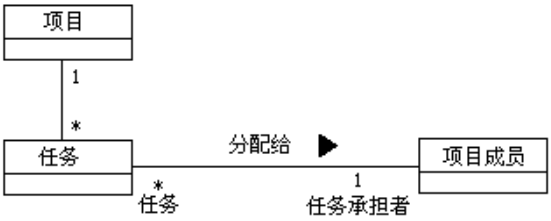
注意，在构造超类的时候，层次不宜太多，关键是表达清晰。事实上，额外的泛化不会增加明显的价值，相反带来很大的负面影响，没有带来好处的复杂性是不可取的。

6，案例：项目管理的概念建模

我们通过一个例子，来看一下概念建模是如何帮助我们理解业务的：假定我们要设计一个“项目管理工具（PMT）”软件，我们如何来理解项目管理这个领域呢？我们可以沿着下面的步骤进行概念建模。

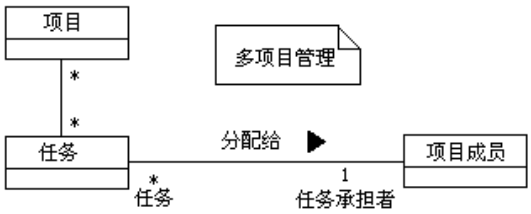
1）项目的开发特点是分解成许多任务，分配下去

2）每个任务只能分配给唯一的人，而一个人可以负责多个任务



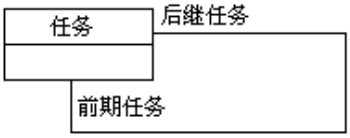
3）现在已经可以实践“多项目管理”，多个项目可以共享某各子任务

例如：很多企业级应用会用到一些与领域无关的模块，例如身份验证、消息机制等。恐怕我们的产品需要支持这样的特性。

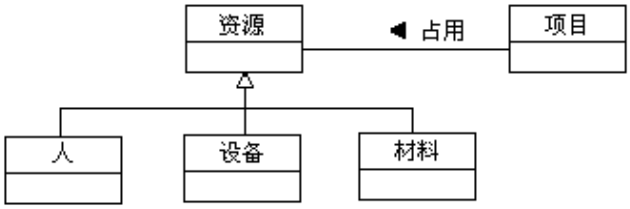


4）项目分解成多个任务以后，需要为任务排定日期

排定日期的时候还要考虑依赖关系。

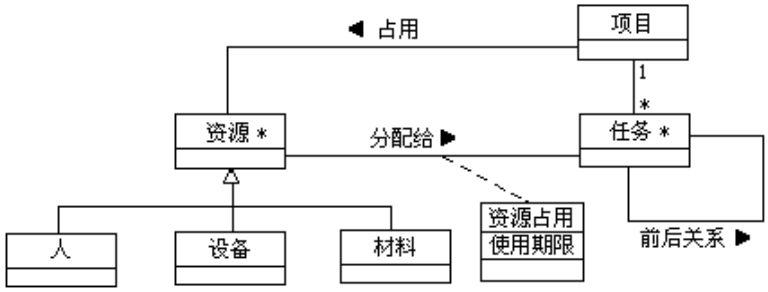


5）我们的项目涉及的资源有多种，比如人、设备、材料等

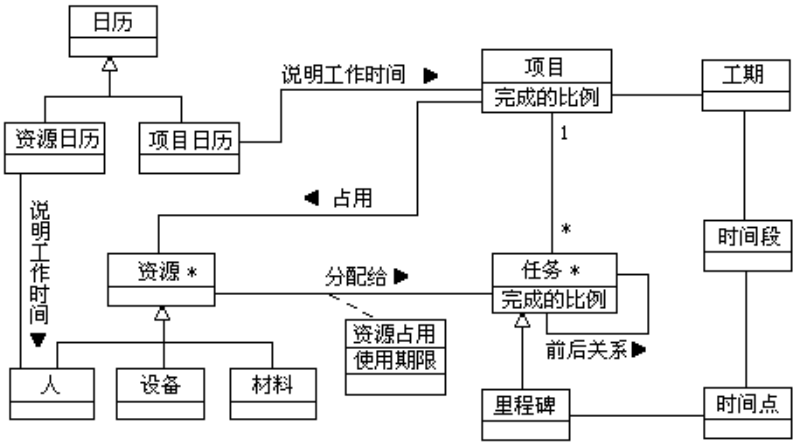


6）从逻辑上看，资源是项目占用的，但对资源的使用要具体分配到任务

我们已经明确了是任务在消耗资源，而且还要明确资源具体的使用期限，这样项目、任务、资源的关系越来越清楚了。



把上面的研究结合起来，这就得到了一个比较清楚的概念模型了，如下图所示。



这个例子告诉我们，在初涉一个业务的时候，概念建模可以很快捷的使我们对业务有一个宏观的理解。

二、理解特征：概念的共性和变化性

建模的目的是为了帮助我们沟通、交流与理解。但是如果形式上是建立了模型，但只是泛泛的客户说什么我画什么，最后的结果要不是漏掉了重要的部分，或者是双方对重要问题的理解还是不一致。那么，这中间到底缺少了什么环节？

任何事物都有特征，事物的特征往往是用共性和变化性来展现的，抓住了特征也就是抓住了事物的根。这种思维方式可以很好的帮助我们分析需求。

1，把变化由敌人转化为朋友

1）需求总是会发生变化

我们已经注意到了，无论我们前期花了多么大的努力来开发需求，需求总是会发生变化。我们的良好愿望是：“需求不仅完整、清晰、易于理解，而且还说明了产品投运后若干年中需要的所有功能!”但实际效果往往是：需求经常是不完整的、需求经常是错误的、需求是容易让人误解的，另外，需求还不会告诉我们全部情况。

需求确实非常重要，从上世纪 70 年代到 90 年代末，人们在致力于一件事情，那就是前期努力做好需求并冻结下来，后期按照需求进行开发。经过 30 年努力到了 2000 年，人们惊异的发现，即便如此重视需求，需求还是在变化，这极大的影响了软件的成功率和质量。既然怎么做都做不好，那其中必有原因，那么需求变化的原因有哪些呢？除了已经阐述的原因之外，还包括：

- 需求是用自然语言描述的，每个人阅读的时候都会加入自己的理解，这就会存在歧义。
- 竞争的剧烈，当对手有新的想法出现的时候，我们自己怎么可能墨守陈规？
- 我们是人，当知道有更好的可能性的时候，为什么不去追求它？这是人的本性使然。

所以，2000 年以后软件工程界的理念就是：与其拒绝变化，不如承认它，管理它，并通过变化为客户创造更好的产品，以此为契机，悄然形成了全新的软件工程理念：

在多年进行软件开发的经历，教会了我们什么呢？我认为最主要的一点就是：需求总在变化。我们应该正视那些自然而然会发生变化的事情，与其抱怨需求总是变化，还不如适应这种变化，尝试用全新的分析方式来支持这种变化，为良好的设计提供更多的支持。那么，需求开发的思想和方法将向什么方向演化呢？

2）通过发现变化来发现需求

当我们仅仅被动的记录客户需求的时候，我们有没有怀疑这种需求是不充分甚至是不正确的呢？当每个人在讲述自己要什么的事时候，我们有没有怀疑他真的知道自己要什么吗？

需求分析本质上是一个认识事物的过程，实践无数次的告诉我们，只是泛泛的说明自己的需要是不可能深入，也是不可靠的。通过发现变化来认识事物，往往更容易发现过去看不到的东西。

任何事物都有特征，而事物的特征是以共性和变化性两个方面来表现的。抓住了共性与变化性分析，就是抓住了认识事物的一把钥匙：共性代表了体系，变化性代表了差异。这个世界没有完全不同的东西，也没有完全相同的东西，只有针对共性和变化性来分析，才能把事物真正看得更透。

3）封装变化：保持关注点分离

既然需求总是会变化，那么作为分析人员来说，我们自然会问：未来产品如何才能适应需求变化呢？关键之处在于：我可能无法知道什么将会变化，但是我能够猜到在哪里会变化。

如果我们在分析中就能注意到这一点，就可以利用面向对象设计的巨大优点，封装这些变化区域，从而更容易地将代码与变化产生的影响隔离开来。

共性和变化性是两类不同的关注点，从分析开始就把这两类关注点分离，就可以减少软件组件相互间的不良影响，从而从根本上减少变更发生的时候的工作量。在一个可培育和发展的软件中，需求总是会发生变化，而且变化量还可能会比较大，因此变化既是个关注点，也是一种职责，需要把变化隔离开来。

这就要求我们，在需求分析的时候，就把变化当成重要的分析主题来看待，从而解决诸如：我们需要分离什么？怎样正确的分离？等等这样一些问题。因此，除了传统的功能分析以外，还要通过对共性和变化性进行分析，使分析更加深入，把变化由敌人转化为朋友。

2，用变化来帮助我们理清思维

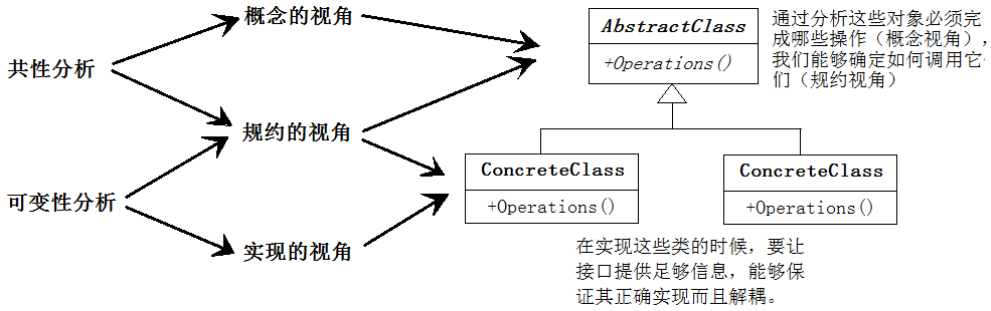
传统上的功能分析可能也会关注哪些是会导致重新设计的原因，但基本的目标还是一个稳定的需求。但是在面向对象分析（OOA）的理念中，一开始就把变化作为重要的设计要素来考虑，

这是软件分析与设计哲学上的一个很大的不同。

如何以变化为主题，通过关注共性和变化性分析，使分析与设计逐步走向深入呢？

下面这张图反映了 OOA 的基本的分析思路：

- 概念代表着共性，抓住了概念也有抓住了事物最基本的规律；
- 具体的实现代表了变化，抓住了变化也就抓住了事物的特点；
- 规约描述了共性和变化性的接口，它使共性和变化性能够联系起来。



从上图可以看到，共性分析与问题领域的概念视角是互相关联的，而变化性分析与（特定情况的）实现是互相关联的。规约视角则位于其中，共性和变化性都要涉及这个视角。规约描述了如何与一组概念上相似的对象通信。这些对象每一个都代表了公共概念的一种变化。在实现层次上这个规约将成为抽象类或者接口。

通过这样的分析往往是浑沌的思维变得更清晰了。有趣的是，基于这种分析方法延伸出的设计，自然地就会遵循“按接口设计”的做法，找出变化之处，把代码封装在定义明确的接口之后，必然使后期生成的代码具有高度内聚（同样的代码不会出现在两个地方）特征而且避免了耦合，这正是消除冗余代码所需要的。强内聚而且“按意图编程”，又使代码具有更好的可读性，这也是优秀代码的一种必需品质。

3，分析矩阵：通过分析变化来发现潜在需求

一个有趣的问题被提出来了：从道理上说抓住共性和变化性确实能够提升人们的认识能力，但在实践中如何帮助我们提高需求分析的成果呢？共性与变化性分析并不是一个人坐在家里冥思苦想的结果，更不是仅凭猜测就可以得出正确的结论。这种分析必须与客户与其他相关人员一起讨论，在讨论中发现问题并得出结论。

多年来与客户打交道的经验使我们懂得，对于非常具体的问题，客户详细的回答一般是可信的，但是他们一般性的回答却不可信。现实世界中问题往往不会秩序井然或者循规蹈矩地出现，在与客户沟通的时候也不可能循着一个问题表一步步的表达。除了最简单的问题，大多数的沟通都是散乱而无规律的。

解决这个问题的一个方法就是建模，但是模型的专业性也可能使我们与客户沟通更加困难。为此我们应该采用客户最为容易理解的模型来与客户沟通，而避免使用专业性的符号。换句话说，我们应该关注模型的思想，而不是模型的形式。

资深人员与初学者能力上最大的区别，在于资深人员强调的是解决问题的能力，而初学者强调的是按照规则行事的能力。在面向对象的分析中，关键是我们用面向对象的方法去思考并引导交流，而不是仅仅画出 UML 图形。只要我们在思想深处对面向对象的分析理解透彻了，在和客户交流的时候完全可以采用更自由的方法，以避免死气沉沉的你问我答最后还是说不清。下面我们举个例子来说明如何使用最简单的矩阵（表格），来展现面向对象的分析思想的。

1）案例的背景

一个美国某国际电子商务公司的订单处理系统，需要处理来自不同国家（地区）的销售订单。

最开始要求很简单：处理美国和加拿大的订单。

系统的需求清单如下：

- 要为加拿大和美国构建一个销售订单系统。
- 根据所在国家计算运费。
- 运费还应该以所在国家（地区）的货币支付。
- 在美国，税额应按当地计算。
- 使用美国邮政规则验证地址。
- 在加拿大，使用联邦快递发货，同时缴纳联邦政府销售税（GST）和地方销售税（PST）。

尽管这个需求已经很清楚，但我们必须通过分析使这个需求变得清晰，分析：

共性：

这是一个销售订单系统，具备国际上任何销售系统所具备的共同特征。

变化性：

这些需求又可以分成两种情况：

- 顾客在美国；
- 顾客在加拿大。

这个问题中存在的变化并不太复杂。光凭观察，就可以显而易见得到下面的表。

因顾客所在地不同的两种情况	
情况	**过程**
美国	根据 UPS 费用计算运费 使用美国邮政规则验证地址 按当地计算销售额和 / 或服务的税额 用美元处理金额
加拿大	用加拿大元处理金额 使用加拿大邮政规则验证地址 通过联邦快递发货到加拿大 按加拿大省的税收规则计算销售额和 / 或服务的税额（使用 GST 和 PST)

这是一个简单的问题，但是我们希望用这个简单的背景来说明分析矩阵的分析过程，而不希望把精力过多地放在案例的背景上。

2）在填写分析矩阵中进行分析

如果情况很复杂，例如有成百上千种情况，几十种变化。我们就可以发现自己当时甚至无法与项目的客户和其它相关人员交谈，因为信息实在是太多了。这就必须提出一种组织海量数据的新方式。使用分析矩阵不仅帮助我们理解问题域，而且有助于实现它。

让我们从观察一种情况开始。首先观察必须实现的每个功能，并标记它所表示的概念。每个功能将分行写出，这一行的左边写上它所表示的概念（共性），然后描述相应的实现（变化性），据此一步一步地给出整个分析剧矩阵。如下表所示。

分析矩阵		
概念	**美国销售（实现 1）**	**加拿大销售（实现 2）**
计算运费	按照 UPS 费率	按照联邦快递费率
验证地址	使用美国邮政规则	使用加拿大邮政规
计算税额	使用美国和当地的税收规则	使用 GST 和 PST
金额	美元	加拿大元
概念是一种抽象类（共性）	实现继承了抽象类，代表了一种实现方式的不同（差异性）。	

当然，事情并非总是如此顺利。在多年与客户沟通的经验都表明：他们通常无法提供完整的需求，因为他们总是按正常情况考虑问题，忽视异常情况（而我们却是必须要处理的）。

3）在分析过程中寻找不完整和不一致

在构建矩阵的过程中，我可能会发现需求中的遗漏。我将使用这些信息扩展分析。这些不一致说明客户提供的信息不完整。也就是说，在某种情况下某个顾客可能提出某种特殊需求，而另一个顾客则不会。

例如，在来自美国市场的需求中，没有提到最大重量的问题，而加拿大市场可能有最重 31.5

千克的限制。通过比较这些需求，我们有必要找到美国客户的联系人，专门询问她重量限制的问题（实际上可能并不存在此限制），然后补上这一漏洞。

随着时间流逝，我们总是需要处理新的情况（例如业务扩展到了德国）。当你发现了某种情况中存在一个新概念时，就扩展分析矩阵，在矩阵中增加一行，即使它并不适用于其他情况。如下表所示。

扩展分析矩阵			
概念	**美国销售**	**加拿大销售**	**德国销售**
计算运费	按照 UPS 费率	按照联邦快递费率	按照德国货运公司费率
验证地址	使用美国邮政规则	使用加拿大邮政规	使用德国邮政规则
计算税额	使用美国和当地的税收规则	使用 GST 和 PST	使用德国增值税
金额	美元	加拿大元	欧元
日期表示	mm/dd/yyyy	mm/dd/yyyy	dd/mm/yyyy
最大重量			30 千克

美国和加拿大有最大重量吗？可能没有，也可能有，但是客户忘了提到这一点。现在，有一个很好的具体问题要问了。我的客户对于回答具体问题是非常擅长的，而对于“还有别的什么吗？这样的问题，他们一般并不擅长。

4）用行来发现规则

现在概念以及其实现上的变化性已经找到，对这些已知信息应该怎么处理呢？

观察上面表中的矩阵。第一行标记为“计算运费”，包括“按照 UPS 费率”、“按照联邦快递费率”和“按照德国货运公司费率”。这一行表示：

- 实现“计算运费费率”的一般规则。
- 必须实现的具体规则集：也就是在不同国家（地区）使用的货运公司。

实际上，每一行都表示实现一个一般规则的特定方式。其中有两行（金额和日期）可以在对象层次上处理。例如，金额可以用包含货币对象的对象来处理。日期可以利用计算机语言库中的不同日期格式。首先确定处理每一行概念的基本方式，把这些设计策略记录下来，很可能发现需求，如下表所示。

扩展分析矩阵				
概念	**美国销售**	**加拿大销售**	**德国销售**	**基本方式**
计算运费	按照 UPS 费率	按照联邦快递费率	按照德国货运公司费率	计算运费费率的各种方式
验证地址	使用美国邮政规则	使用加拿大邮政规	使用德国邮政规则	验证地址的各种方式
计算税额	使用美国和当地的税收规则	使用 GST 和 PST	使用德国增值税	计算应付税额的各种方式
金额	美元	加拿大元	欧元	可以自动兑换货币
日期表示	mm/dd/yyyy	mm/dd/yyyy	dd/mm/yyyy	可以根据顾客所在国家（地区）的要求显示日期
最大重量			30 千克	最大重量的各种方式

5）用列发现实现上的区别

那么列又表示什么呢？它们是针对所表示的情况的特定实现。如下表所示

扩展分析矩阵				
概念	**美国销售**	**加拿大销售**	**德国销售**	**基本方式（行）**
计算运费	按照 UPS 费率	按照联邦快递费率	按照德国货运公司费率	计算运费费率的各种方式
验证地址	使用美国邮政规则	使用加拿大邮政规	使用德国邮政规则	验证地址的各种方式
计算税额	使用美国和当地的税收规则	使用 GST 和 PST	使用德国增值税	计算应付税额的各种方式
金额	美元	加拿大元	欧元	可以自动兑换货币
日期表示	mm/dd/yyyy	mm/dd/yyyy	dd/mm/yyyy	可以根据顾客所在国家（地区）的要求显示日期

最大重量			30 千克	最大重量的各种方式
具体实现规则：列	当有美国顾客时，使用这些实现	当有加拿大顾客时，使用这些实现	当有德国顾客时，使用这些实现	

例如，第一列说明了用于处理美国销售订单的具体实现。

然后我们再通过观察行，分析和确定设计策略：

例如，第一列说明了用于处理美国销售订单的具体实现。应该怎样把这些深入认识转化成设计策略呢？

对上表进行观察（重点是观察行），分析相关概念的特点，每一行都表示实现最左列中概念的特定方式。例如：

- 在“计算运费”行中，“按照 UPS 费率”、“按照联邦快递费率”两项实际上表示“应该怎样计算运费”。所封装的算法是“运费费率计算”。具体的规则将是不同的“UPS 费率”、“加拿大费率”和“德国费率”的计算方法。
- 下两行也是由不同规则及其相关具体实现组成的。
- “金额”和“日期”两行表示可能在整个应用程序中保持一致的类，算法上是可以共享的，但是需要根据国家（地区）的不同表现也不同。

以类似的方式，再来观察列。弄清楚每一列描述了哪种情况用哪一个规则？这些项表示该情况需要的对象系列。每一行代表一个概念，其接口应该是相同的，但具体实现可以各不相同。

6）子矩阵

在实践中，一个简单的矩阵往往是不够的。即使在上面这个简单案例中，也很容易想像，如果一个国家（地区）中有超过一种货运方式会变成什么样子。这就可以在主矩阵中再加入子矩阵，下图给出了这个例子。

内嵌的子矩阵使关系变得复杂，实际上只是在无法直接得到“共性与变化性表”的时候，才使用矩阵中的矩阵。上图中的内嵌矩阵对应的表如下表所示。

内嵌矩阵定义的共性与变化性	
计算运费共性	**取货附加费共性**
UPS 费率	无
USPS 费率	5 美元
联邦快递费率	4 美元
验证地址共性	**最大重量共性**
美国邮政规则	50 磅
美国邮政规则（无邮箱时）	70 磅无
	无最大限制

这种良好需求分析的结果，必然对设计形成更好支持。

7）问题越大，分析矩阵越有用

尽管分析矩阵并不能解决需求分析上的所有问题，但它至少可以用于大多数问题域，特别是

在特殊情况太多，我的脑子无法得到总体视图时最有用。

分析师与客户沟通的时候，当需求不能以非常协调的形式陈述给分析师或开发人员，人们对所涉及的所有东西无法在概念上很好地把握，他们只是谈论一般规则和例外情况，其中显而易见的存在大量遗漏。这时我们该怎么办？是怨天尤人？还是一味的责怪客户？

现在我们可以利用分析矩阵把大问题变成小问题。在这种情况下，对于一个特性，查看最左一列，看这个特性是哪个概念的一种变化。如果找到了这个概念，就将这个概念放入那一行中。如果找不到这样的概念，就说明必须创建新的一行。以此逐步的使混沌逐步走向清晰，这也是领域建模的一种方式。

3.2 行为分析：发现业务功能

通过概念模型分析，我们已经对客户的领域有了一个全面而且正确的理解，但从本质上说概念模型还是一个静态模型，在这个基础上，我们就可以着手发现业务功能。

一、关注源头：业务用例与业务事件

如何发现功能呢？由客户说明我们来记录当然也是一个方法，不过这种方法是不可靠的。其原因在于：一个方面，客户对于功能的理解本身就是含糊的，其中很多含糊之处需要分析人员和他们一起来明确。另一个方面：功能不是独立存在的，它存在于业务流程之中，为整个业务的目标服务。功能之间还存在着承上启下的作用，上一个功能可能会为下一个功能服务，因此只有存在于业务场景下的功能才有意义。

1，业务用例与事件

正是上面的这些原因，我们不应该首先描述功能，而是需要首先集中精力把业务流程描述清楚。问题在于，以主观感受来分离用户场景有一定的先验性，所以更好的方法是利用业务用例来分离场景。那么什么是业务用例呢？

我们定义：在用户（用业务的人或者设备）事件的触发下，产生一个业务流程，它达到了用户的目标，我们说这是一个用例（用户实例）。也就是说，业务用例指的是业务对于业务事件的响应，通过完整的业务流程，达到期望的业务目的。

业务用例是非常好的分析业务行为的工具，但是如何发现用例呢？显然，首要的工作是发现业务事件，并且以这些事件来分离相关业务响应。

所有的系统或者工作都会对外部发生的事件做出响应。例如顾客从书架上取出一本书，然后走到收银台，这样发生一个业务事件：“顾客想买一本书”。店员响应这个事件的做法是：扫描条形码，告知书价，向信用卡公司查询并要求认证，把款项记入收银机，把书放入袋子并递给您。以每个事件为单位描述系统的响应，被称之为“基于事件的分解”。

那么为什么基于事件来分解是更好的方式？

事件是一种非主观方式来划分工作的，也就是确定工作对外界的响应，毕竟这是顾客看待业务的方式。任何工作当前的划分（比如过程分解），都是基于当初在具体情况下策略上的考虑，这些决定在新的自动化系统加入的时候可能不再有效，至少应该质疑它们，避免只是因为它们存在，就认为理应如此。

业务事件指出了哪些东西是在一起配合工作的，从而我们可以提交一些内聚的部分，也可以使这一部分成为详细需求调研的基础。这些部分的依赖性越少，分析师就越有可能关注与这部分的细节，而不需要知道其它部分。

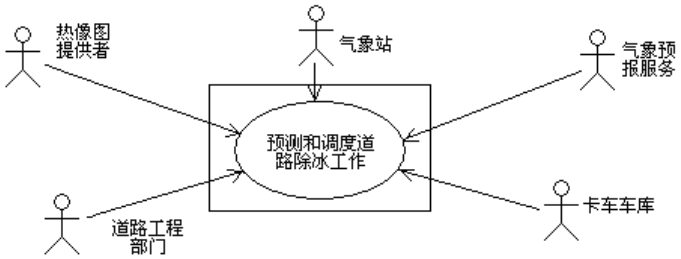
2，如何发现业务事件

那么如何发现业务事件呢？

1）确定业务分析的范围

分析的方法可以总结为由粗而精、自上而下、从外而内。

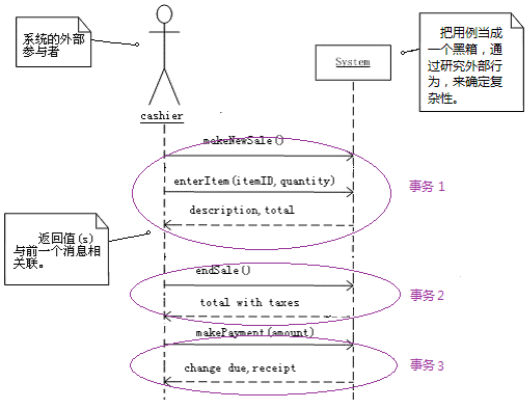
我们已经明确了一个项目的基本范围以及内外关系，如下图所示，为一个“道路除冰系统”的上下文范围。



2）研究参与者如何系统如何交互

依据上面的初始信息，我们首先致力于发现系统事件。

所谓系统事件分析指的是把整个工作系统看成一个黑箱，讨论外部参与者是如何与系统交互的。这个分析过程需要以每个参与者为单位，仔细分析每个参与者与系统的交互方法，如下图所示。



在外部看用例，角色输入一个消息，系统发生反应，然后将处理的结果返回给用户，这就形成一个“环形路线”，每一个环代表一个小目标，称之为事务。对于一个比较大的完整业务目标，它往往由多个事务组成。

在这种一个个参与者精细的交互分析中，事件也就自然的被发现了。不可否认，确实需要一些工作知识才可能弄清楚这些业务事件，我们建议在项目初始阶段就开始确定这些业务事件的过程，那时候利益相关方都在，他们都非常熟悉这些业务事件。

3）通过外部参与者的数据流发现事件

我们也可以更精确的描述外部系统与系统之间数据流动关系，可以通过表格仔细列出每一个参与者与系统之间的数据流动，下表列出了这样的数据流清单。

道路除冰系统数据流清单			
编号	**参与者**	**输入数据流**	**输出数据流**
1	气象站	气象站读数	
2	气象预报服务	区域气象报告	
3	热像图提供者	热像图	
4	道路工程部门	改变的道路	
		新的气象站	
		改变的气象站	
			失效的气象站告警
5	卡车车库	卡车改变	修订的除冰调度计划

			道路除冰调度计划
		已处理的道路	
		卡车故障	修订的除冰调度计划
			对没有处理的道路进行提醒

这样，我们也可以通过研究上下文范围图中的边界数据流来确定业务事件。

如果发生一个外部业务事件，相邻系统就会向工作发送数据流并且从工作得到响应。因此，每一个进入边界的数据流都意味着一个可能的外部业务事件。出去的数据流要么是对外部事件的响应（处理输入数据流并产生输出数据流），要么是一个时间触发的事件的结果（报表、发出邮件、警告等）。大多数事件包括一个单一过程，并且需要说明以下内容：

- 输入及输入来源，来源被描述为外部代理。
- 输出及输出目的地，目的地被描述为外部代理。
- 必须读取记录的任何数据存储都应该被加入到事件图中，事件流应该加入命名。
- 对数据的任何增、删、改、查都应该加入到事件流中，事件流应该加入命名。

例如，根据“道路除冰系统”数据清单，经过分析，我们可以得出相应的事件清单如下。

道路除冰系统事件清单			
编号	**事件名称**	**输入数据流**	**输出数据流**
1	气象站传送数据	气象站读数	
2	气象局预报天气	区域气象报告	
3	道路工程师通知改变的道路	改变的道路	
4	道路工程师安装了新的气象站	新的气象站	
5	道路工程师改变了气象站	改变的气象站	
6	到了测试气象站的时间		失效的气象站告警
7	卡车车库改变了卡车	卡车改变	修订的除冰调度计划
8	到了检查结冰道路的时间		道路除冰调度计划
9	卡车处理了一条道路	已处理的道路	
10	卡车车库报告卡车出问题	卡车故障	修订的除冰调度计划
11	到了监控道路除冰的时间		对没有处理的道路进行提醒

在继续讨论与分析中，凡是被发现的业务事件都应该记录在事件列表中，并且随着讨论的深入，来不断丰富这个事件列表。

3，利用图形描述业务

在明确了事件以后，我们需要把这些事件分分类，与客户代表沟通一下：在客户的组织范围内，谁对某类事件的工作响应可以描述得更清楚？然后约定沟通的时间。在具体和客户一起沟通事件的响应的时候，利用图形描述业务是个好方法，它专注于最主要的关系，摈弃了不必要的细节，并且通过视图使沟通变得容易。

在这个过程中，我们应该仔细思考如何简化图形，而不是把视图搞得像个马蜂窝。在图形的不断简化中，阐明了系统分析中一个很常见的现象：有关问题你思考得越多，它就越简单。一个简单的图形，伴随着简单的文字，用一种让人容易理解的方式包含所有信息，比起看起来事无巨细，充满大量的、次要的、互不相关的细节遮掩了主要实施的那种描述，通常需要更多的脑力劳动。

4，利用事件的响应发现问题

分析师在于客户一起梳理业务过程中，可以利用与客户日益加深的关系更深入的讨论业务需求和设计思想，他可以利用自己独特的对于业务和软件技术两方面融合的知识水平，以及长期工作中总结归纳出来的经验，给客户提供更大的支持。

他可以提醒客户注意 IT 的加入，必然会改变当前业务流程，同时向客户提出一个想法是不是可行？它是否会改进工作？能不能发现一些潜在的需求？这种沟通让客户觉得分析师不仅仅是观察者，而且慢慢的变成了个业务专家，这种尊重的氛围对于进一步建立分析模型是至关重要的。

1）流程为什么是这样？

在建立事件响应业务流程的时候，也可以问一下“流程为什么是这样的”？这样就更便于我们发现问题。如果分析师不敢去问“为什么是这样的？”、“有没有改进的机会？”就可能使“业务化石”从一代产品传到下一代产品。

分析师必须超越那些显然的东西，这意味着理解工作真正的本质。

例如：由于历史的原因，客户的组织往往划分为多个部门，每个部门具有相应的职责和工作方式。但是太沉溺当前的工作方式是危险的，有时候就难以发现业务事件。为了克服这种困难，我们建议事先通过建立事件模型，通过反复沟通和讨论来发现业务事件，并列出相应的事件表。在实际工作流程中事件的响应往往会穿越多个部门，这就发现了内部事件。

2）现行流程是不是完美？

这时候分析师要问的一个问题是：这个工作流程现在是不是完美的？是不是要进行改造？改造的目标是什么？不用计算机系统行不行？如果不行，那最后期望的结果是什么？一定要注意：信息化不是把传统工作流程搬到信息平台上，一定要根据信息化扁平、共享的特点，以提高效率为着眼点，以达到工作结果为目标，对工作流程进行重新梳理后重构。

3）有没有例外和遗漏？

在描述事件响应的时候，应该重点发现例外和遗漏的事件，为什么有这个要求呢？

初步的事件表可能会发生遗漏，我们还需要采用一些方法发现更多的遗漏事件。有些遗漏事件是显而易见的，但有些可能被掩盖在某些事实之下，发现这些遗漏事件一个比较好的方法就是寻找一下“无事件”。

术语“无事件”指的是如果基本事件没有发生将会引起什么事件，从而扩充我们的思维空间，在大多数情况下可能由此发现例外和遗漏的事件。比如一个事件是“客户付款”，无事件指的是客户没有付款将会怎样？是不是引发“跟踪信义不佳的付款者”的另一个事件？或者是引发“向未付款者发送提醒通知”的事件？

仔细的研究自己的业务或者事件清单，问一下“如果事件没有发生又会怎样？”这就可能发现更多的事件。不是所有的事件都有无事件，比如“如果这个事件没有发生将会怎样？”回答是“不会怎样”，出现这种情况不需要介意，因为检查清单致力于找出无事件的目的，是帮助我们发现一些遗漏的事件。请把在这个过程中发现的新事件加入到业务事件清单中，并更新上下文模型中相应的数据流。

二、理解特征：行为的共性和变化性

流程分析可能是软件领域最古老、最传统的方法了。但是如果在行为分析的时候，对行为适当的抽象，针对行为的共性和变化性进行深入分析，这就使分析思维上升到更高的高度，形成了现代领域特征分析方法。

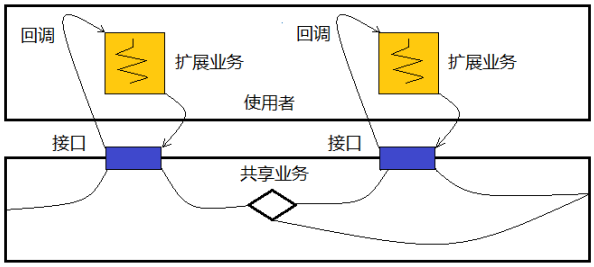
1，什么时候需要进行领域特征分析

在下面几种情况下领域特征分析起到关键的作用

1）设计一个共享的框架

在产品开发上升到一个比较高级的阶段，需要针对一个领域需求的共同点，开发一个共享的框架（Framework），这就会遇到一个比较难处理的问题：

所谓共享的业务指的是存在可以复用的业务共性，这种共享业务越大，复用的价值就越高，但复用的概率就越低。反之，业务规模越小，复用的概率就高，但复用的价值就低。为了解决这个矛盾，就需要仔细分析这种业务的共性与变化性，从而形成正确的设计策略，如下图所示。



共享业务的框架设计的关键点是在需求的时候进行充分的领域特征分析，如果这个分析是不充分或者错误的，那么所开发出的产品就是错误甚至有害的。

2）几个相关客户之间需求的微妙不同

即使在明确客户的产品开发下，有时候也会发生几个相关子客户之间需求的微妙不同。面对这样的问题除了进行协商以外，也可以考虑针对不同用户的需求进行分类处理。那么这些业务有什么共性？有什么不同的个性？我们如何把它们组织起来形成一个共同体？此时进行充分的领域分析是成功的关键。

3）领域行为分析的关注点

领域特征分析主要工作是识别功能具有的行为特点。这个阶段建模首先是发现事件，再考虑在事件触发下业务流程（活动）的描述，以可视化的方式把业务的行为展现出来，这也称之为用例分析，具体包括：

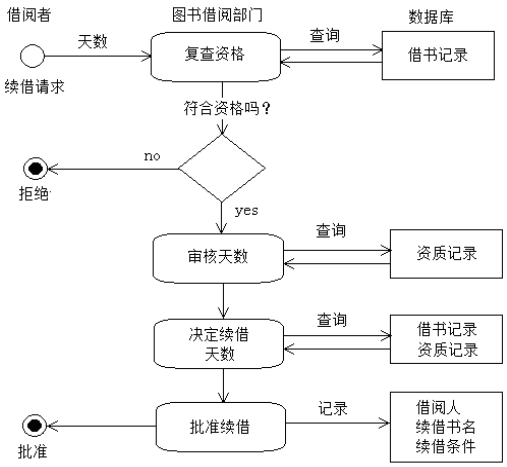
- 分析功能执行的前期行为特征，如功能执行的前置条件、前期准备工作等；
- 分析功能主体行为的特点，发现其具有的显著特点和可能的变化性；
- 分析功能的后期行为特点，如功能执行的后置条件、善后处理工作、功能执行完毕后的控制权转移等。
- 分析与寻找变化点，分离共性的行为和变化的行为，而确定变化行为的位置，并且建立这两种行为之间的约束，成为发现“特征”的关键。

需要为每一种行为特点建立领域内一致的名称和说明，并将其放入特征模型的行为特点层，建立与功能层特征的整体部分关系。

2，对行为的共性进行抽象

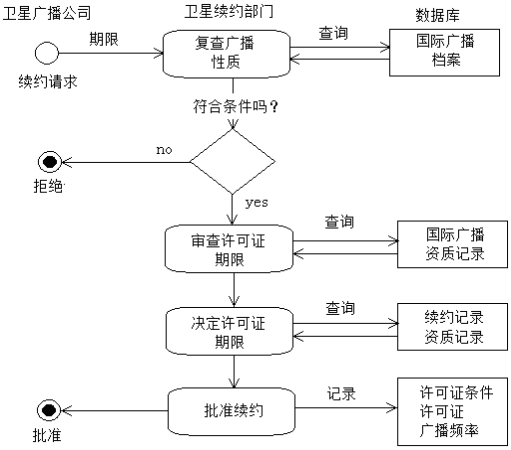
在进行领域行为进行共性和变化性分析的时候，我们首先应该注意在整个领域中寻找业务流程的共性。很多业务流程一旦抽取掉了行业特征，就可以发现其中的共性。

举个例子：假定目前的工作是关于一个图书馆的系统，存在有这样的业务事件：图书馆用户希望续借图书。下图展示了对该事件的系统响应流程。



当一个图书馆用户提交了一个续借请求的时候，系统的响应要么是拒绝续借，要么是批准续借。在图书馆这个行业特征上的工作导致了一份详细的需求规格说明，可用于创建一个特定的产品。这项工作的一项副产品是识别了某些有用的业务流程，这些都可以在图书馆行业的其它项目中复用。但是，复用的范围能不能扩大？

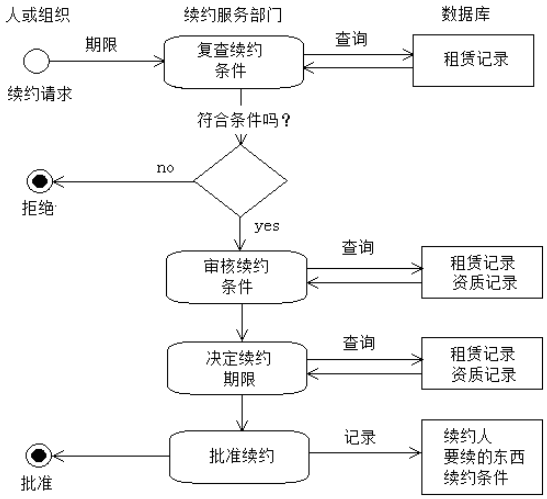
现在想象目前的工作是“卫星广播”这个极不同的行业特征，例如。这个上下文范围中有一个业务事件是“卫星广播希望对许可证续约”。当卫星广播者提交了广播许可证请求之后，系统的响应要么是拒绝该请求，要么是提供新的许可证。



当你站在更高的角度审视这两个不同的行业项目，就会发现在为卫星广播项目与图书馆项目相似的业务流程，有可能在这些相似项目中复用。

第一眼看上去，图书馆续借和卫星广播许可证续约对事件的响应很不一样，它们的不同之处是它们来自于不同的行业。但是再仔细研究它们的响应，就会发现它们之间有比较大的相似之处，如果发现了相似之处，我们就有机会导出更抽象的业务模型，可以在很多不同行业软件中复用，被称之为“续约”。

“续借”和“广播许可证”都是要续约东西，但其中有很大的相似之处：每件要续约的东西都有唯一的标示符、标准的续约周期以及续约费用。下图是我们对这两个业务事件的处理策略进行抽象所得到的结果。我们使用抽象来确定共同的特征，这意味着透过事物的表象来发现有用的相似之处或者分类。它也意味着可以忽略一些特征以发现共同之处。



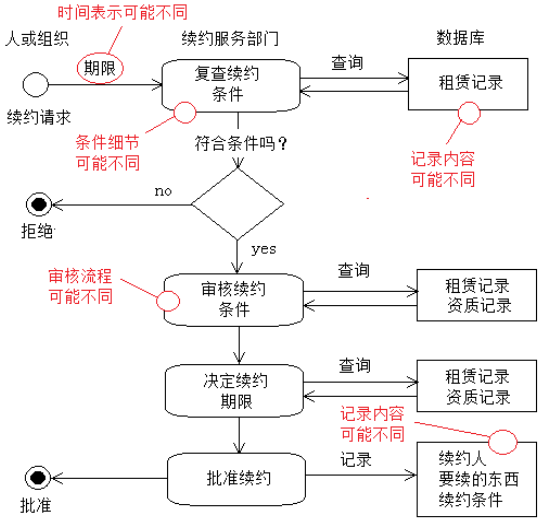
忽略实际物品和主题行业，我们就可以把注意力集中在两个不同系统所执行的底层操作上，我们这样就可以发现相似之处，也就是一种共性，以便于加以利用。

发现和确定共性需要分析师具有一些独特的能力：

- 从不同的抽象层次来看待工作的能力；
- 按不同的方式进行分类的能力；
- 发现望远镜与注满水的玻璃半球都是放大镜的能力；
- 指出显然不同的事情之间相似之处的能力；
- 以抽象的方式来看问题的能力。

3，对行为的个性进行变化分析

进一步想，事实上图书馆借书和卫星广播还是不同的，它们不同的点到底在哪里呢？在这些点上我们如何实现这种变化呢？这些变化部分如何与其共性清晰分离呢？走到这一步，一个特定领域分析已经开始了走向深入了。



通过这个案例，我们进行了领域行为共性与应用个性（变化）的分析，这种分析的翔实准确，使得共性和变化性可以清晰的分离，这就为开发可复用可插拔的软件提供了坚实的基础。

关注共性和变化性，这本身也是个重要的工作方法。这个世界没有什么是完全不同的，总结了共性，就可以减少很多重复劳动，提高理解问题的效率。这个世界也没有什么是完全相同的，总结了个性或者变化性，就可以使我们的产品有特点，理解更深入，更能够解决具体问题。

三、变化模型：发现变化与理解变化

发现了变化点只是分析变化的前提，仅仅知道哪一些点可能变化，并不可能直接形成需求。为了更好地理解变化，就需要对变化性本身进行建模分析，从而知晓变化性，并且有意识的去解决它。通过对变化进行的分析，就可能发现那些在业务分析中没有指明的潜在需求。为了正确地建模，必须定义与模型相关的基本概念，再从基本概念引申到建立分析变化的基础模型。

1，变化主题和变化对象

1）可变性管理

在传统的产品设计中，处理可变性往往是一项辅助性的工作，但在现代软件设计思想中，研究可变性问题成为一项主要和关键性工作。正是这样一个背景，我们的需求分析、项目管理以及设计思想就需要在一个大思维框架下统一起来。我们把分析中识别和描述可变性的活动称为“识别可变性”，而在不同阶段是用“可变性管理”来进行支持的，它包括：

- 支持与定义可变性活动。
- 管理可变性交付物。
- 为解决可变性相关的活动提供支持。

- 收集、存储和管理完成这些任务所必需的追踪信息。

在整个产品设计过程中，都需要导入可变性问题，因此必须以一致的方法保证这些变量被正确的构建。在现实中可变性解决的时刻被称为可变性绑定时间，但绑定时间并不是可变性建模的关注点。为了增加灵活性，也可能会把可变性绑定时间推迟到实现的后期阶段。

2）需要思考的三个基本问题

所谓可变性，指的是变化的能力和变化的趋势。换句话说，我们关注的可变性并不是偶尔发生的，而是有目的地产生的。通过仔细思考下面三个基本问题，有助于我们定义产品线可变性问题：

- **第一个问题是“什么在变”：**回答这个问题就可以准确地识别现实世界中的变化项和属性，这就是变化的主题。
- **第二个问题是“为什么变”：**主题的变化原因很多，例如利益相关者不同的需求、不同国家的法律以及技术原因，而且当变化项存在依赖关系的时候，一个项的变化很可能是由于另外一个项变化引起的。
- **第三个问题是“怎么变”：**这体现了变化的主题的不同形态，从而可以定义变化对象。也就是说，变化对象是变化主题的一个实例。

通过考虑这三个问题，就可以改变人们对可变性的思考方式。

2，变化点与变量

为了正确的进行可变性建模，我们首先需要对模型中的元素进行定义。

1）变化点

变化点代表了领域交付物内部的变化主题（什么在变？），也代表了现实世界中所有可能的变化主题的一个子集。变化点同时也包含了上下文信息，例如：引入变化点的理由等。

2）变量

变量代表了领域交付物内部的变化对象（怎么变？）。

一个变量对应着变化点的一个候选项，并且可以与其它交付物建立联系，这就表明了这个交付物与对应的特定选项有关。

3）定义变化点和变量

为了定义变化点和变量，我们可以通过下面三个步骤来系统性的识别变化：

- **第一步：识别现实世界中的变化项：**例如系统组件中的几种不同的通讯方式，例如有线LAN、无线 LAN、蓝牙等，这就产生了一个变化主题“网络类型”。
- **第二步：在产品上下文环境中定义变化点：**现实世界中的可变性（变化主题）与产品中的可变性（变化点）是不同的。例如，一个变化点意味着有多种需求可供选择，也意味着可能产生不同的应用。例如第一步定义的“网络类型”变化主题，由此形成的变化点是“自定义系统网络”，它表明产品必须适应不同网络类型，但并没有指明究竟是其中哪一种网络类型。
- **第三步：定义各种变量：**为此，需要从已经识别的变化主题中选择一些变化对象，把它定义为变化点的变量。例如对于“自定义系统网络”变化点，把如何用统一的方式处理有线 LAN、无线 LAN、蓝牙等方式，这样一来工作效率要高的多。

3，时间、空间、内部与外部可变性

对可变性的进一步研究，还需要考虑可变性的各种类型。

1）时间可变性

时间可变性指的是一个交付物在不同的时间的不同版本。这是因为软件产品随时间的进化所造成，对于单一系统而言，这主要是靠配置管理来进行版本控制。但是在软件产品线工程中，可

以使用预测与调整把变化点标出一些预先定义的位置，在这些位置上可以相对容易的引入变化。

例如，一个“门锁识别机制”，当时的技术是“磁卡”，但我们预测还会有“指纹扫描”技术出现，这可以把它定义为一个变量，一旦出现这种技术，可以很容易地加入它。

2）空间可变性

空间可变性指的是一个交付物在同一个时间以不同的形式存在。例如，某个系统的变化点“系统访问方式”，可以分成三个变量“浏览器”、“移动电话（短信）”与“固定电话（语音）”，他们需要用一致的方式集成到系统。

时间维度的可变性与软件进化是同义的，而空间维度的可变性问题是产品线架构的主要内容，这也是架构设计的重要主题。

3）外部可变性

外部可变性指的是领域架构对于直接用户可见的可变性。由于是对用户可见的，他们可以直接或者间接的决定每个变量是要还是不要，也可以由产品经理通过选择变量来定义不同的应用，客户可以从这些应用中选择。

4）内部可见性

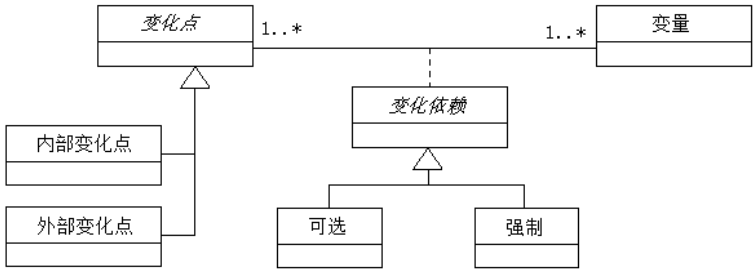
内部可见性是领域架构中隐藏的可变性。这种变量的选择与客户无关，而是由应用开发来决定如何处理可变性问题。例如，通讯系统不同通信协议的转换。客户通常对如此细粒度层次的需求并不感兴趣，也没有必要让客户了解不同的实现方法。

4，变化性建模的模型骨架

对变化建模分析可以通过面向对象分析（OOA）来达到，最常用的是使用概念模型。下面，我们对这种可变性建模方法进行讨论，其目的是建立一种思路性的通用变化模型框架（模式），也称之为变化分析的元模型。在这个框架下，在具体变化问题分析中，可以进一步细化某些内容（比如具体的属性）。

1）模型的骨架

作为这个模型的骨架，所关注的就是：变化点、变量和可变依赖性这三个大问题。首先必须定位变化点与变量之间的关系，也就是建立这两个概念高层的基础模型，被称之为可变性元模型。



2）第一个概念，“变化点（什么在变？）”：

这是一个抽象类，其属性描述了具体的变化点及其它细节。

它被具体化为“内部变化点”和“外部变化点”两个类。这两种“变化点”是互斥的，也就是说对于任何一个具体的变化点，要不就是“内部变化点”，要不就是“外部变化点”。两种变化点具有不同的语义、内涵。内部变化点的变量对于用户不可见，而外部变化点对于用户和应用开发人员都是可见的。

事实上每个类都需要一个文本注释，使得在引入这个元素的时候记录引入的原因，为了简单，这里没有标示出来。

3）第二个概念，“变量（怎么变？）”：

它代表了领域交付物内部的变化对象，其属性描述了变化对象的各种细节。

4）第三个概念，变化点与变量之间依赖：

由于“变化点”与“变量（怎么变？）”之间是多对多关系，这就必须引进一个关联类，称为“变化依赖”，以保证这种关系得以实现。

关联类中至少需要两个属性来建立这种联系，每个属性需要与“变化点”与“变量”变量之间的一个属性相容。这种关联的多样性必须满足下列条件：

- 每个变化点至少与一个变量相关联。
- 每个变量至少与一个变化点相关联。
- 一个变化点可以提供多个变量。
- 一个变量可以与不多个不同的变化点相关联。

5）“必选与可选”：

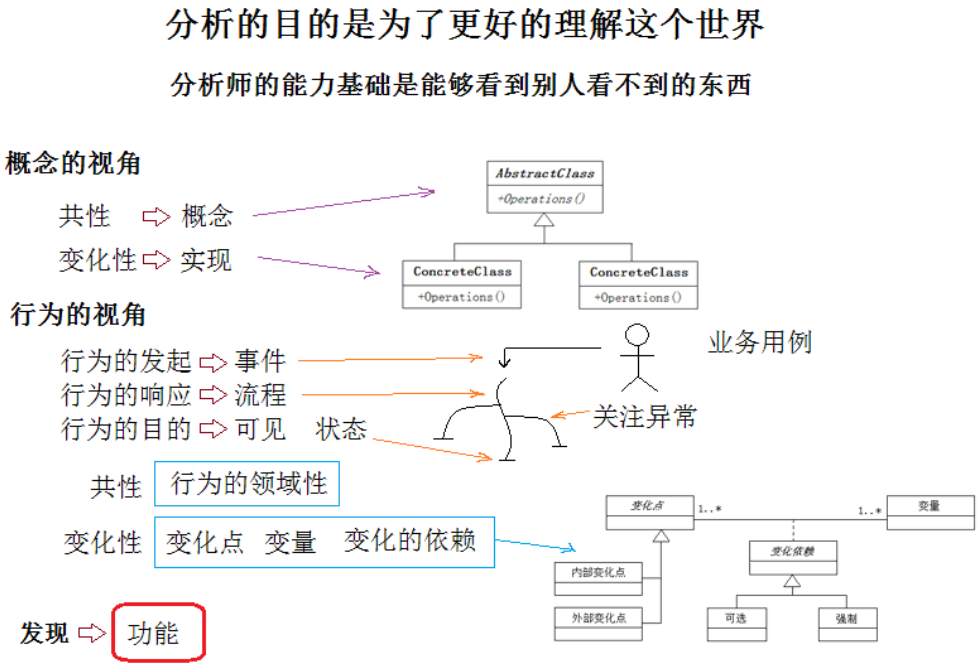
“变化依赖”定义成了一个抽象类，其具体化为一个“必选”关系和一个“可选”关系，这些具体化是完整而且互斥的。

“可选”的变化依赖指的是一个与变化点相关的变量，是某个具体产品的特定部分，但不是必须的部分。例如：“键盘”、“磁卡”、“指纹扫描”是用户可选的应用，但也可以不选。

“强制”的变化依赖指的是一个与变化点相关的变量，这个变量是这个变化点所必需选择的，但并不意味着这个变量必须包含进产品线所有的应用中去。例如：对远程通讯加密，128 位加密是强制变量。同时，也定义了 256 位、512 位、1024 位的可选变量。在用户不做选择的时候，自然认为选择了 128 位加密。但用户选择 512 位加密时，并不等于一定要 128 位，它只是远程加密应用的一部分。

这个模型只是考虑了“变化点”与“变量：之间的一般关系，具体问题还需要仔细考虑每个类的具体属性，以这个模型为基础，就可以分析很多关于变化的具体问题了。

作为变化建模的元模型，本质上也是一个模式，起名为“变化建模”，它提供了一个思路和模型的模板，可以帮助我们在领域架构分析的时候，非常理性的分析变化问题。在具体为变化建模的时候，可以根据情况去掉某些不相关的部分，添加某些属性和说明，甚至也可以添加某些关联。在为每一个预期的变化都作类似的模型分析之后，就可以使我们从变化的混沌中解脱出来，思维变得更加清晰。下图归纳了整个分析方法。



四、发现功能：在建模中发现功能需求

在上述与客户在各个层面的讨论过程中，我们边学习业务过程、边与客户一起明确有关思想、边开展需求收集活动。每一个单独的需求都有一个结构，需求有一些组成部分，每部分都包含某方面的知识，每部分对于理解整个需求都是必要的。

由于希望在和用户交谈的时候，找出一项需求的所有部分以后再找下一项的做法是不切合实际的，所以建议做一些小卡片，把需求框架打印在卡片上，在和用户交谈的时候，发现一项就快速的记录下来，随着对需求理解的加深，逐步地完成这些卡片。在某一次访谈的时候可能谈得不是很清楚，卡片上的空白给我们提示了下一次访谈需要补充提出的问题。下面是这种卡片的模版。

需求编号：唯一 ID	**需求类型**：模版小节编号	**事件/用例编号**：需要这项需求的事件/用例清单
描述：对需求意图的一句话概括		
理由：这项需求的合理性		
来源：提出这项需求的人		
验收标准：这项需求的度量标准，从而可以测试解决方案是否满足了要求		
顾客满意度	**顾客不满意度**	**冲突**
如果这项需求成功实现，利益相关方的高兴程度（评分 1-5，1 为不感兴趣，5 为非常高兴）	如果这项需求没有成为这种产品的一部分，利益相关方的不高兴程度（评分 1-5，1 为无所谓，5 为非常不高兴）	予这项需求冲突的需求
优先级：顾客价值评分		
支持材料：指出说明和解释这项需求的文档		
历史：创建和变更情况		

这种做法给我们带来很大的灵活性，是我们在利益相关方那里收集需求的时候不至于带来很大的障碍。我们可以根据它所属的用例进行分组（见右上角的“事件/用例”编号），这样便于在桌面上查找某张卡片。这样逐步地归纳整理不断完善，就可以形成一个很好的客户需求来。

3.3 原型分析：沟通的手段

有时候分析师会遇到障碍，有些需求没有成型，甚至也没有办法向用户解释这些需求，或者分析师自己也不能理解它们，也许这个产品具有创新性，没有人真正知道需求是什么。在这个时候，原型就是最有效的方法了。

原型，是一个预期系统的小规模的，不完整的，但可以工作的示例。

1）需求收集最大的困难是完全靠想象构思产品功能

需求分析和收集最大的困难，在于完全是靠想象在构思产品功能，客户或者开发团队，通过阅读需求文本或研究分析模型，很难想象软件产品在特定的环境中是如何运行的。即使我们耗费了很大的气力完成了需求获取、分析和编写规格说明，需求中还是有一部分对客户或开发者来说不明确或不清晰。如果不解决这些问题，那么必然在用户和开发者之间对于开发什么产品的理解之间存在期望差距。

2）原型主要用于帮助获取需求

原型是向用户提供一个快速而粗造的实现，以确定用户的需求。有的时候，对于技术上的新框架，或者技术上的新方法，也可以构造原型并通过测试来验证想法。原型可以使新产品实在化，这样，就可以利用原型来减少客户对产品不满意的风险。来自用户的早期反馈也可以使开发小组正确理解需求，并知道如何更好地实现这些需求。原型为分析带来一种生机，并且消除团队在需求理解上的差异。比起阅读一份冗长无味的软件需求规格说明，用户通常更愿意尝试建立有趣的原型。

3）原型仅仅是真实系统的一个模型

原型有多种含义，并且参与建立原型的人不同的期望有关。例如，一个飞机原型实际上应该可以飞翔，它是真实飞机的雏形。相反，一个软件原型通常仅仅是真实系统的一部分或一个模型，并且它可能根本不能完成任何有用的事。

下面我们将讨论原型在需求开发中不同的应用，以及如何使原型成为软件需求开发过程中有效的组成部分

一、原型是“什么”和“为什么”要原型

使用原型有三个主要目的：

- **作为需求工具**：原型作为一种需求工具，它初步实现所理解的系统的一部分。用户通过对原型的评价，可以明确需求中的许多问题，这样一来，在开发真正产品之前，可以用最低的费用来解决这些问题。
- **作为设计工具**：为了探索和选择设计方案，原型可以作为一种设计工具，用它来探索不同的用户界面技术，使系统达到最佳的可用性，并且可以评价可能的技术方案。
- **作为构造项目的工具**：我们可以把原型作为发展最终产品的一种构造工具，原型是产品最初子集的完整功能实现，通过一系列小规模的开发循环，可以逐步完成整个产品的开发。

由于需求的不确定性，早期需求收集确实极为困难，使用原型可以解决下面一些问题：

- **早期需求收集的不确定性**：通过发现这些不确定性，来判断系统中哪一部分需要建立原型，以及希望从用户对原型的评价中获得什么。
- **发现需求中的二义性**：二义性和不完整性使开发者对所开发的产品产生困惑，建立一个原型有助于说明和纠正这些不确定性。
- **使想象具体化**：用户、经理和其他非技术项目利益相关方面对待开发产品时，原型可以使他们的想象更具体化。
- **比术语容易理解**：一个附带的好处是，原型比开发者常用的技术术语更易于理解。

二、原型的各种形式

1，水平原型

1）水平原型是在屏幕上显示用户界面的各种图像

“水平原型”（horizontal prototype）也叫做“行为原型” （behavioral prototype）或“模型”（mock -up ），通常是表达可能的用户界面。它可以帮助我们探索预期系统的一些特定行为，并达到细化需求的目的。当用户在考虑原型中所提出的功能可否使他们完成各自的业务任务时，水平原型使用户所探讨的问题更加具体化。

需要注意的是，水平原型中所提出的功能经常并没有真正地实现。

一个水平原型就像一个电影集。它在屏幕上显示出用户界面的各种图像，也可能允许这些界面之间的一些导航，但是并没有真正实现所有的功能。有一些导航可能起作用，但是用户可能仅看到描述在那一点将真正显示什么内容的信息。

2）数据库查询所响应的可能只是一个固定不变的信息

在这种原型中，数据库查询所响应的信息是假的或者只是一个固定不变的信息，并且报表内容也是固定不变的。虽然有人认为原型应该执行一些有意义的工作，但其实不然。这种模拟足以使用户判断是否有遗漏、错误或不必要的功能。原型代表了开发者对于如何实现一个业务场景的一种观念。用户对原型的评价可以指出这个场景可选过程，遗漏的过程步骤，或原先没有发现的异常情况。

3）也可以使用纸和铅笔来建立水平原型

我们还可以使用不同的屏幕设计工具，甚至使用纸和铅笔来建立水平原型，这要根据具体情

况来选择。当我们建立原型的时候，用户应该把注意力集中在需求和工作流问题上，而不能被精细的外形或屏幕上元素的位置所干扰。在澄清了需求并确定了界面中的框架之后，我们可以建立更详细的原型来探索用户界面的设计。

2，垂直原型

1）垂直原型用于验证构造软件的方法是否完善

“垂直原型”（vertical prototype），也叫做结构化原型或概念的证明，这种原型实现了一部分应用功能。当我们不能确信所提出的构造软件的方法是否完善，或者当我们需要优化算法，评价一个数据库的图表，或测试临界时间需求的时候，就可能需要开发一个垂直原型。垂直原型更常用于软件的设计阶段以减少风险，而不是软件需求开发阶段。

2）垂直原型只实现一部分用户界面和相应的服务器功能

一个垂直原型只实现客户一部分用户界面和相应的服务器功能，可以使我们评估所提出架构的通信组件、性能和可靠性是不是合理。如果实验是成功的，我们基于那一种架构的实施也是成功的。

3，抛弃型原型

在构造一个原型以前，需要充分与客户沟通，并作出一个明确的判断，在评价原型以后，是抛弃掉原型还是把该原型进化为最终产品的一部分。这可能最终导致我们使用两种不同的原型策略。

1）抛弃型原型主要解决需求的不可预测性

抛弃型原型也称为**探索型原型**，它主要解决需求的不可预测性，以此提高需求质量。因为我们打算在原型达到预期目的以后就把它抛弃，所以应该花最小的代价尽快地建立该原型。如果我们在原型上付出的努力太多，到最后可能就不愿意将它抛弃。

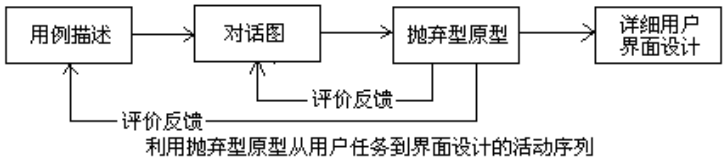
当我们遇到需求中存在大量的不确定性、二义性、不完整性或含糊性的时候，最合适的方法是建立抛弃式模型，这样可以减少在继续开发时存在的种种风险。原型可帮助用户和开发者想象如何实现需求，也可以帮助我们发现需求中的漏洞。它还可以帮助用户判断这些需求是否可以完成必要的业务过程。

2）抛弃型原型中的代码不能移植到正规的产品系统中

当我们建立抛弃型原型的时候，就会忽略很多具体的软件构造技术。正规的软件项目强调在健壮性、可靠性、性能和长期的可维护性。基于这一原因，我们不能把抛弃型原型中的代码移植到正规的产品系统中，除非它达到产品质量代码的标准，否则，你和用户将在软件生存期中遭遇种种麻烦。

3）建立抛弃型原型的步骤

下图描述了在抛弃型原型的帮助下，从用户任务到详细用户界面设计的开发活动序列。



利用抛弃型原型从用户任务到界面设计的活动序列

- 每一个需求的描述包括了一系列操作和系统响应，这些可以用对话图来建立模型以描述一种可能的用户界面机制。
- 抛弃型原型把对话元素细化为特定的屏幕显示、菜单和对话框。当用户评价原型时，他们的反馈可能会引起用例描述的改变（例如发现一个新的可选过程时）并且也会引起相应对话图的改变。
- 一旦确定了需求并勾画出屏幕的大体布局，我们就可以从最佳使用的角度设计每一个用

户界面元素的细节。

- 比起直接从需求的描述跳跃到完整的用户界面的实现，然后在需求中发现重大错误，利用逐步求精的方法所花费的努力将会更小。

4，进化型原型

1）进化型原型是在已经清楚定义了需求的情况下构建的

如果在已经清楚地定义了需求的情况下，也可以构建进化型原型。进化型原型是螺旋式软件开发生存周期模型的一部分，它为开发渐增式产品提供了坚实的构造基础。与抛弃型原型的快速、粗略的特点相比，进化式模型一开始就必须具有健壮性和产品质量级的代码。因此，对于描述相同的功能，建立进化型原型比建立抛弃型原型所花的时间要多。

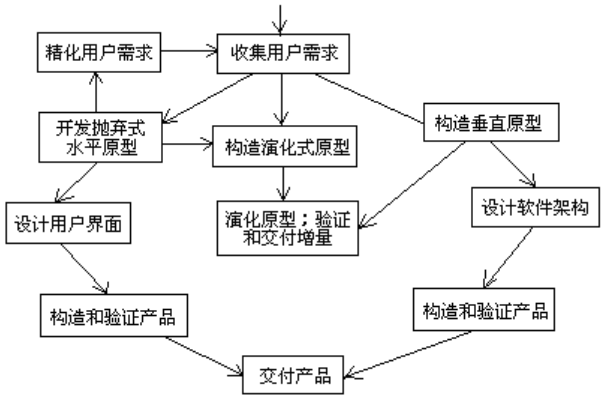
2）进化型原型将是最终产品的一部分

由于进化型原型将是最终产品的一部分，要达到进化型原型的质量要求并没有捷径，我们必须重视软件系统性和完整性的设计原则，把软件设计为易于升级和优化的。

进化型原型应该是整体系统比较关键和风险比较大的一部分，我们应该关注原型的第一次演变，因为它将作为实现需求中易于理解和稳定部分的试验性版本。从测试和首次使用中获得的信息将引起下一次软件原型的更新，正是这样不断增长并更新，使软件才能从一系列进化型原型逐渐发展为实现最终完整的产品。

3）软件开发过程中可以综合使用多种原型方法

原型方法可以帮助我们建立良好的反馈系统，下图描述了在软件开发过程中，如何在合适的时机综合使用多种原型的方法。



软件原型的典型应用		
	抛弃型	**演化型**
水平	● 澄清并精化用例和功能需求 ● 查明遗漏的功能 ● 探索用户界面方法	● 实现核心的用例 ● 根据优先级，实现附加的用例 ● 开发并精化 We b 站点
垂直	● 证明技术的可行性	● 实现并发展核心的客户/服务器功能层和通信层 ● 实现并优化核心算法

例如，我们可以利用从一系列抛弃型原型中获得的知识来精化需求，然后，通过一个进化型原型序列，渐增式地来实现需求。例如，在贯穿于图中的一条可选路径，是在最终设计用户界面之前，使用抛弃式水平原型澄清需求，而与之对应的垂直原型则使核心应用程序算法有效。要特别注意的是，我们所不能实现的：是把一个抛弃型原型固有的低劣性转化为产品系统所要求的可维护性和健壮性。

三、通过原型挖掘需求

1，原型法需要注意的问题

1）原型必须是用户表达最不清晰的部分

泛泛的构造原型效果上实际上并不好，我们构造的原型必须是用户表达最不清晰的部分。为了提高通过原型方式有效的挖掘需求，可以通过建立脚本使用户遵从一系列步骤并且回答一些特定的问题，从而获取所需要的信息。这些活动是对一般的询问“告诉我，你对这个原型的看法如何”的有价值的补充。

2）利用原型帮助询问问题

在每个任务之后，脚本将为评价者提供特定的与任务有关的问题。此外，你可以询问以下一般性的问题：

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

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

3）亲自观察用户是如何使用原型将获得更多信息

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

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

4）希望通过原型突破已经构建的模式

原型实际上已经构建了一个模式，而这个模式是站在设计方的角度构建的，我们希望通过原型达到的，正是突破自己已经构建的模式，使产品真正符合用户的需要。一个作家会问读者：“你们希望什么样的作品？”，这些读者的回答往往是茫然的眼神。一个优秀的作家往往把自己的草稿给一部分读者阅读，读者看到了这些草稿，就有了一个基础，激发了想象力，当这些读者提出各种各样的意见的时候，作家就可以换个思维方式面对自己的草稿，突破自己已经形成的思维模式，这样他就可以写出真正优秀的作品来。

5）把从原型评价中获得的信息编写成文档

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

2，如何使原型法获得成功

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

- 你的项目计划中应包括原型风险。安排好开发、评价和可能的修改原型的时间。
- 计划开发多个原型，因为你很少能一次成功。
- 尽快并且廉价地建立抛弃型原型。用最少的投资开发那些用于回答问题和解决需求的不确定性的原型。不要努力去完善一个抛弃型原型的用户界面。
- 在抛弃型原型中不应含有代码注释、输入数据有效性检查、保护性编码技术，或者错误处理的代码。
- 对于已经理解的需求不要建立原型。
- 不能随意地增加功能。当一个简单的抛弃型原型达到原型目的时，就不应该随便扩充它的功能。
- 不要从水平原型的性能推测最终产品的性能。原型可能没有运行在最终产品所处的特定环境中，并且你开发原型的工具与开发产品的工具在效率上是存在差异的。
- 在原型屏幕显示和报表中使用合理的模拟数据。那些评价原型的用户会受不现实数据的影响而不能把原型看成真正产品的模型。
- 不要期望原型可以代替需求文档。原型只是暗示了许多后台功能，因此必须把这些功能写入软件需求规格说明，使之完善、详细并且可以有案可查。

3.4 沟通技巧：理解涉众的需要

需求收集的过程本质上就是一个沟通的过程，这是需要有一系列方法论支撑的，其基本点在意充分理解客户思维，理解利益相关方的需要，并且调动和综合团队的集体思考。下面提出的一些建议并不一定要处处都用，但可以有选择的在合适的时候使用合适的方法。

一、良好沟通需要关注的问题

软件需求的引出有一系列的方法和共同活动的思想：

- 实际用户需要参与到需求收集的活动中。
- 应该确定所有的相关人员，每类相关人员的代表都应该参与需求收集。
- 模糊的需求应该被确认为原型化的候选者。
- 客户和发起人不一定是所交付的应用程序的使用者，但他们是支付需求工作和最终系统费用的实体。
- 应该确定可能限制系统共能性或者性能的领域约束。
- 每个相关人员都有其特有的偏见，其中包括对开发组织。
- 如果开发的仅仅是软件，就需要定义产品的目标环境（计算机结构、操作系统、通信等）。
- 如果开发的是整个系统（软件和硬件），在软件需求引出来之前，必须存在系统需求。
- 需求过程的输出包括一系列有功能、领域约束、使用案例和原型化（如果可能的话）组成的高层次上也对象。
- 应该记录每一需求的基本原理。
- 在有合适的人员时，引出方法可以最终做出决策。

二、培养和锻炼面谈的技巧

在需求收集时间表中，面谈是一个重要的部分。过去，分析师仅仅与项目相关人员见面，询问他的需要，并希望以这种方式收集所有信息，遗憾的是，经验表明这种方法来发现需求是比较困难的。面谈有一些很重要而且很基本的步骤，这对掌握面谈的技巧是必须的：

1、第一步：准备提出的问题

面谈从一组问题开始，并由此产生有关真正需求的多数知识，我们可以从服务请求、概念研究文档以及软件项目管理计划中，来提示并确立面谈问题的模版。事先准备要提出的问题非常重要，这意味着我们已经对将要进行的会谈进行了准备，也意味着我们不清楚什么问题，抑或我们的会谈将要达到什么目的，毫无准备的会谈将是一个失败的开始。

要注意到，会谈不可能完全按着事先准备的问题板进行，在会谈过程中，用户会提出新的问题，我们也会发现那些事先没有准备，但是在会谈过程中暴露出来的新的感兴趣的话题，但是事先准备问题意味着我们可以控制会谈的范围，不至于把会谈变成一个毫无成果的漫谈。

2、第二步：选择面谈者

多数情况下不可能与所有的相关人员面谈，因此必须选择其中的代表，他们应该有实际知识，比较易于接近，并且能够为问题提供可信的和可靠的答案，主要考虑以下人员。

- **初级人员**：由于他们经验不多，并不能做出很多贡献，但他们可能有新鲜的观点，也可能有意想不到的信息。
- **中级相关人员**：由于经验丰富，这些人员对领域操作性和技术性的问题，能够提供详细的理解，应该经常与项目领导进行面谈。
- **管理人员和其他特殊客户**：CEO 有该领域的知识，并影响着项目的成功，他们无疑是面谈的重要对象，只要有可能，应该经常与执行发起人面谈。
- **理论家**：如果有这样的相关人员，他们能够开阔你的视野。
- **系统典型“用户”**：这样的相关人员是重要的，因为他们将花更多的时间与系统打交道。他们可能是革命精神的思想家，也可能对新系统抱有偏见。
- **“后起之秀”**：这样的人将成为该领域的领导者，可以提供深刻的思想和信息。

人力资源的多样性，有助于验证所收集信息的可靠性。

3、第三步：计划联系方式

如果时间允许，请研究一下想要联系的人对项目的观点。每个人都很忙，所以对面谈的范围要有所控制，仅限于他们感兴趣的话题。最初的联系方式是电话和电子邮件。

4、第四步：进行面谈

面谈可以通过电话、视频会议、MSN 或者 QQ，但最好的方式是通过面对面的谈话，能够获取信息的关键因素是，你和面谈之间建立了良好的氛围，一个舒适而且充满自信的环境。

1）建立氛围

- 请求面谈这允许你在面谈期间记笔记。
- 诚恳的态度：对软件中的利害关系表现做出真正的兴趣。
- 倾听：完全关注对方，对方说话的时候，注意你的眼神不要漂移。
- 从小事入手，再进入敏感的事务。
- 直截了当，表现出诚实。
- 谦虚，没有人愿意和“无所不知”的人交谈。
- 不要跑题，除非你认为分享相关人员的经验很重要。
- 请求去看看将来要使用产品的环境。
- 提供未来的帮助和分项目报告，信息交换是互惠的。

2）提出问题

问题应该简单、简洁，仅包括一个部分，复合的需求说明往往那个很难解释和测试。

3）如果面谈中止

可能面谈者不能提供给你重要的信息，也要感谢他百忙之中参加面谈，应该留下联系电话，以便必要的时候可以联系你。有时面谈者不愿意给你提供信息，这时强调他的专门知识对系统的正面影响，可能会克服这个障碍。也可能需要临时改变方向，应该判断这个方向的意义，如果是有意义的，可能会提供意外的信息。

4）结束会议

应该为下一次面谈打下基础：

- 询问面谈者是否有问题问你。
- 询问是否有其他应该提出的问题。
- 以未来问题或者兴趣点的方式，给你留下未来的目标。
- 询问你是否可以想到其他问题的时候和对方联系。
- 询问是否可以和其他人面谈。
- 联系资料来源，感谢他的参与。

三、沟通的方法论

事实上需求分析师有两个关键能力：首先是对业务的理解力，再一个就是沟通的能力，这两种能力是相互影响又相互支撑的。

沟通能力是一个分析师具有特征性的能力，从根本上说，需求收集的第一大障碍就是如何沟通。很多人以为沟通能力仅仅是个口才问题，这是一个误解，沟通的第一要义是沟通什么？也就是我们自己想表达的思想是不是从逻辑上想清楚了？如果自己思想上是糊涂的，那么沟通本身也就不存在了，所剩下来的就只有记录用户的要求，而这样做出来的所谓“需求”后期风险是相当大的。

为了锻炼这种能力，就需要对周围发生的事情敏感，每件事情都去思考一下，遇到事情多想几个为什么，各行各业实际上是相通的，这样就有了积累。事实上，用户知道越少，后期麻烦越多。用户知道越多，后期麻烦越少。毕竟用户才是这个领域真正的专家以及最终使产品发挥效益的人。需求分析需要有想象力，作为一个分析师的责任，就是激发用户的想象力，再由用户的想象力再激发自己的想象力。正是想象力的发挥，才能够构思一个伟大的产品。

良好的沟通需要一系列方法论来支撑，这些方法从不同的层面提高了沟通的效率和深度，帮助分析师与客户一起讨论，把过去模模糊糊的东西慢慢变得清晰，讨论也会变得比较容易。下面我们讨论一些最重要的引出方法。

1，理解用户的思维过程

沟通的能力，表现在对对方的理解，而不是以自我为核心，是沟通中双方都在修正自己的想法，最后达成风险比较小、目标比较高的共识。在沟通中需要注意以下几个问题。

1）努力听懂对方的话

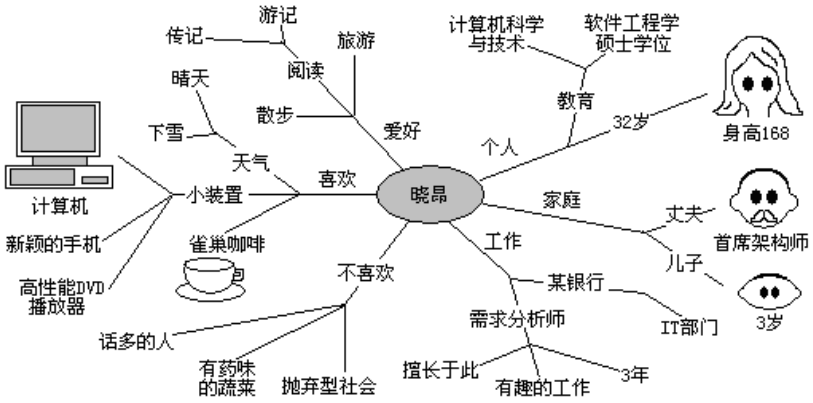
很多失败的需求收集往往是由于双方都不考虑对方在想什么，只是一味的阐述自己的观点，只是固执地站在自己的角度想问题，这样当然也就不存在什么沟通了。我经常听到无论甲方还是乙方，都无奈的说对方听不懂我的话，这里很大的原因，就是听者都没有耐心（或者说能力）来揣摩对方到底想表达什么？因此仔细倾听对方的表达，如果对方观点还有含糊之处，就需要多问几个为什么，来使问题进一步清晰化。当观念出现矛盾的时候，首先考虑一下对方这个观念的本质到底是什么？我现在的想法能不能修改以适应这个观念？双方观点的共性是什么？不同点有哪些？怎么样寻求一致？这样一种沟通过程，将会产生积极而有效的成果。

好的需求分析师需要尽量理解用户的思维过程，并从对用户思维过程的理解思考出更多的东西。我们需要摆脱自己原有的思维定势，站在用户的角度充分研究用户执行任务时作出决策的过程，并提取出潜在的逻辑关系。我们可以用一种思维图在访谈的时候做记录、计划项目和活动、

对研讨会作总结。总的来说，只要我们需要简洁和智能的记录方式，就会用到思维图。思维图也是一种模型，同样具备简洁和值得玩味的特点，同时思维模型也反映了逻辑和讲一个完整的故事，所以是一个很有用的工具。

2）利用思维图展现信息

思维图是绘图和文字的结合，试图按照大脑存储信息的方式来展现信息，我们把每条新的信息与我们已经知道的某些事联系起来，思维图通过线把词和图联系起来的方式，实现了对大脑思维过程的模拟。下图是对我们虚拟用户代表的思维图模拟。



思维图的中间应该是中心主题，中心思想应该是有强烈视觉效果的图像，这一层分解展示了主要概念、话题、思想或者所选择的主题的分解。

思维图非常重要也很常用，很多情况下这是一种讨论的工具，大家聚在一起，在白板上边讨论边绘制，用一到两个词来写这些思想。然后用有颜色的线条来连接这些思想，这样人们更容易记住。就这样，逐步地把不清晰的事情变清晰和条理，也把讨论引向了深入，如下图所示。

这些线条可以用箭头表示方向，但大多数情况下思想之间的联系是双向的，所以不标注箭头可能更好。

3）即使没有联系的内容也可以画上去

思维图并不总是从中心向外扩散的，有时候有一些想法，或者从记笔记中听到一些事情，这些东西与图上已有的东西之间没有联系，但也要画到图上去，因为将来可能会发现联系。最后的图可能不那么漂亮，但对思维是有帮助的。

4）纸和笔更容易展现思维

画思维图的纸要足够大，当然也可以用画思维图的软件来进行，不过我认为还是直接画图比较好，画图的过程本身就是思维的过程，彩色钢笔的使用使画图的过程充满乐趣，并且新的思想由此源源涌出，在画的过程中，思想的闸门被打开了。

在对利益相关方进行访谈询问的时候，我们可以使用思维图来记笔记。当利益相关方告诉我们关于他们的工作、以及需要的新产品的特征的时候，使用思维图的好处就体现出来了。使用思

维图来记录访谈的内容，就更可能看到联系，并且发现客户没有提到但是应该解释的联系。思维图可以是一个多用途的记录工具，因为可以简单的一条线来代替联系的文字。

5）善于把松散“遥远”的元素联系起来

所谓创造性思维，是善于把一个问题中松散的“遥远”的各个元素联系起来，从周围的环境中广泛寻找自己经历过的东西，甚至无意中看到广告牌或者一次谈话都可以引发联想，得出问题的答案，这使得他们得以用已知的方法来有效的解决问题。思维图并不可能给出答案，但可以条理性的展示我们可能希望看到的东西，这就是思维图的作用。

2，文档考古学

1）检查组织使用的文档和文件可以确定某些潜在的需求

文档考古学指的是，通过检查组织使用的文档和文件来确定潜在的需求。当面对已有的系统或者遗留的系统，并且计划修改或者更新它的时候，这是很有用的。

文档考古学并不是一个完整的技巧，应该与其它的技巧共同使用，使用的时候应该十分小心。这是一种对工作使用的文档进行逆向工程，也就是说，从旧的工作所使用的材料中挖掘新的需求，或者寻找成为新产品一部分的需求。显然并不是所有的旧工作都要继续保留，但只要当前系统存在，总会有充足的材料，成为需求工作的原料。

2）文档中的名词可能代表一些概念

文档考古学的方法是这样的，检查收集到的文档（这里“文档”值得是收集到的所有东西），从中找出有用的“东西”，或者简单的找到一些名词。这些东西可能是列标题、命名的表格区域、或者只是文档中一项数据的名称。对每个名词，仔细询问以下问题：

- 此物的目的是什么？
- 谁用它？为什么？
- 系统都利用它来做什么？
- 哪些业务事件用到或者参考了此物？
- 此物有一个值吗？例如，它有一个数字、编码或者数量吗？
- 如果是这样的话，它属于哪些东西组成的集合？（数据建模的热心者立即会意识到，需要找到拥有这个属性的实体。）
- 此物的用途是什么？
- 文档中是否包含了一组重复的事物？
- 如果是这样的话，这些事物的集合称为什么？
- 能找到事物之间的联系吗？
- 什么过程建立了它们的联系？
- 每件事物附加的原则是什么？换言之，哪部分业务策略涉及该事物？
- 什么过程确保了这些规程会被遵守？
- 什么文档带给用户最多的问题？

3）问题本身不会揭示所有需求，但它会给出背景和研究的方向

这些问题本身不会揭示产品所有的需求，但是它们会给出充足的背景材料和进一步调查的建议方向。需要注意的是，在设计师拿到需求文档进行架构分析的时候，所使用的方法与文档考古学是类似的，他们所建立的第一个模型称之为“分析模型”，而在建立分析模型的时候，需要需求分析师与产品架构师共同工作来完成，下一章我们就会探讨这个问题，需求分析师在文档考古学上的经验，对架构师来说是非常珍贵的。

4）我们不是要复制旧的系统，而是构思新的产品

当进行文档考古学的时候，是在当前工作中寻找新产品所需要的功能，这并不表示我们需要复制旧的系统，毕竟我们收集需求是为了构建一个新的产品，然而，我们也需要注意到现存系统

与它的替代物之间必然存在某些共同功能。但是一定要注意，旧有的系统并不表示它就是正确的，也不表示它就是客户需要的，也许文档是没有用的，或者必须大量的修改才可能成功的复用。

5）文档考古学可以作为数据建模的重要组成部分

我们建议文档考古学作为数据建模的一部分，因为前面列出的大多数问题常常用于数据建模的原则中，小心使用这些文档就可以揭示出数据的分类以及它的属性，这个方法也可以作为面向对象开发的基础。

作为一条规则，我们应该把访谈中所有的共件（文档、照片、录像、手册以及录音）都保存起来，因为我们会常常回过头去参考它们，这样养成一个习惯。

3，业务用例研讨会

我们发现业务用例研讨会对大多数项目都是有用的，这些研讨会让需求分析师与数量较少的、专门化的利益相关方一起工作。

1）会议的沟通优于电话和 E-Mail

我们已经讨论过，业务事件触发的响应，称之为业务用例，这种业务用例清楚的界定了根据对外界的响应的工作划分。在完成这种划分以后，需要做的事情就是描述和重新制定完成的工作，或期望完成的工作。需求分析师的工作是记录下这些活动，然后通过它们导出能完成工作提供最大帮助的产品需求。

每个业务事件都有一个或多个利益相关方，他们是这部分工作的专家，他们对这个业务用例的输出有特别的兴趣。研讨会的目的，就是通过这些专家把业务用例相关的知识传授给需求分析师。这些人把自己锁起来，召开研讨会，构建一些场景展示队业务事件做出正确相应所需要的动作。在理想情况下，感兴趣的利益相关方能够有效的沟通，描述他们对工作的理解，提出问题，并给出对工作的期望。

2）用例场景描述可以使会议议题集中

在下一章中我们将讨论用例的场景，也就是以一种结构化方式来描述业务用例的故事，我们建议在讨论的时候，把业务用例的场景作为谈话的中心，但在现在阶段，只需要把业务用例的功能细分为一些合理的步骤就可以了。

会议中以场景作为手段，分析师与利益相关方一起工作，获取业务用例的知识：

- 业务用例的预测成果。
- 描述通过业务用例完成工作的一些场景。
- 异常场景描述了哪些事情可能出错，以及工作通过哪些活动来纠正它。
- 可选场景，展示工作允许的变化。
- 适用于该业务用例的业务规则，业务规则是管理的规定，可以在感兴趣的利益相关方的指导下，发现这些规则的文件，这些规则最终是业务用例和需求的一部分。
- 基本的产品用例，展示业务用例的多少部分由产品完成，在这个阶段，建立一个产品的轮廓是有好处的。
- 为这个业务用例构建的产品的可能用户特征。

- 低保真原型，用来帮助利益相关方把业务用例可视化，这个可以抛弃的原型并不打算在需求阶段结束后继续保存。

3.5 产品边界的最后确定

事实上在项目初始阶段，我们已经定义了边界，这也决定了我们需求获取的范围。但是通过需求获取我们已经获得了更多的信息，我们就可以更深入的考虑，未来产品边界究竟应该是什么样的？前期的这种考虑，可以很大程度上减少后期边界的蠕动，为产品开发打下坚实的基础。

一、最终确定产品的价值与范围

1）需求的一个关键性的任务是思考“产品应该是怎样的？”

当对工作有了比较深入而且条理化的理解之后，就可以思考“产品应该是怎样的？”这样一个问题了。遗憾的是许多项目开始的时候都有关于“产品应该是什么”的先入为主的概念，但不理解产品将成为其工作的一部分。这里我们看看为了发现优化的产品，我们可以做些什么？

对需求分析师来说，没有说出来但很重要的一项任务，就是确定工作将来应该是怎样的，以及产品是怎样才能对工作产生最大的帮助。产品是工作的一部分，工作是打算以某种程度改变的东西（通常是把它自动化），目标是找到优化的业务用例。因为业务用例通常是工作对外界服务请求的响应，所以，优化的响应就是以最低的时间、原材料或工作量成本（从组织的角度来说），提供最有价值的服务（从顾客的角度来说）。

2）思考“产品应该是怎样的？”将最终确定产品的边界

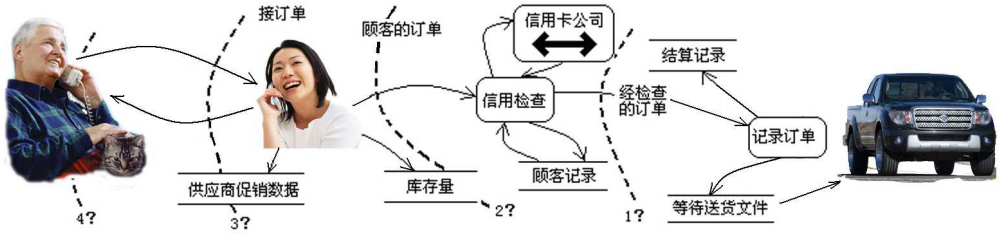
思考“产品应该是怎样的？”的直接结果，是最终确定产品的边界。在一开始就定义清楚产品边界是危险的，因为它会限制住产品的创造性思维，甚至会限制住产品发展。但是，当我们对业务用例已经进行了清晰的分析和建模，与利益相关方也进行了深入的沟通以后，我们需要最终确定当前产品的边界并把它固定下来，否则产品将无法进入真正的设计，另一方面，最后确定的边界，也是作为下一步进行产品建模的必要输入。如果范围不够正确，那么结果将使产品不能完全满足用户的需要。如果不能正确地确定工作范围，总会导致对产品的修改和需求的增强提前到来，这会带来很大的风险。

3）最佳的业务用例总是最接近工作本质的那一个

需求分析师的任务是找到最佳的业务用例，最佳的业务用例总是最接近工作本质的那一个，有时候还包括一些创新。当对业务工作非常了解之后，就要决定工作的多少部分将由产品来完成。重要的是先理解工作，然后把工作的一部分自动化，我们才能把自动化产品无缝的放到工作中去。

4）利用用创新思维来构思最终产品

让我们来看一个如何确定工作的多少部分将成为产品的例子。假定一个公司 TelCo 需要设计一个计算机系统 Acura，目的是改善电话销售的工作。原来的业务流程如下图所示，一个顾客（相邻系统）正通过电话订购某种商品。那么，产品边界的最佳位置在哪里？或者换一种思考方式，怎样才算是最有用的产品？



采用 1 号边界的产品：一个操作员输入已经检查过的订单，通过信用卡公司授权以后，产品记录订单，这并不是一个好产品，因为大部分工作留给了操作员。

那么 2 号边界又如何？操作员通过电话接下订单，把它录入产品，其余的都可以自动化完成。这个方法不坏，但还可以做的更好。

3 号边界把订单接收自动化，可以是一个语音识别系统，也可以是通过按键处理的系统，也可以通过 Web 方式处理，通过网站在线方式订购。如果顾客没有问题或者不需要太复杂的方式响应，对销售公司来说可能是个好方案，但对于客户呢？他真正想要的是什么？他喜欢与人对话还是语音发生器对话？主要销售对象群体对 Web 方式接受吗？销售公司可以提供哪些服务来保持他的忠诚度？

5）努力找到时间的真正起源

我们在讨论业务事件的时候，应该寻找事件的真正起源，可以肯定地说，起源不在操作者那里，操作者只是业务响应的一部分，它是在相邻系统作了某些事情之后发生的。在上图的例子中，起源是客户发现缺少某种商品，他发现需要订购些什么，他曾经寻找了家里有没有这个商品，或者清点了相关商品的数目，当他决定购买的时候他拿起了电话。所以，这个事件的起源在于他拿起电话前的大约 30 分钟。

6）思考客户能不能更方便些呢？

从顾客的角度，能不能使这个事情变得更方便呢？如果销售公司知道这个顾客已有商品的数目，并且知道他的消耗数目，顾客就根本不需要打电话了，公司会打电话给顾客通知他这种商品他快要消耗完了，同时安排恰当的时间送货。这个场景是把产品的边界扩展到了相邻系统思维的深处，从方便和服务的角度来说（站在顾客的角度），这是不是一个更好的想法呢？从提供服务来保持顾客的忠诚度来说（站在销售公司的角度），这是不是也是更好的想法呢？如果是，那我们的业务会做些什么样的改变呢？继续深入的思维和研究，可能会创造出更好的产品。

7）通过某个工作事件的进一步思考可以构思出真正有用的产品

所以，我们应该通过某个工作事件的进一步思考，来设想产品应该是怎样的。检查业务事件，特别是那些由人发起的事件，当事件发生的时候，相邻系统都在做什么？可以扩展产品的范围以包括这些活动吗？

我们的结论是，努力找到业务用例的真正起源，考虑自己的业务用例，他真的是在工作的边界上发生的吗？还是在它们到达组织之前就开始了？这些思考的结果，会对业务以至于产品的设计产生重大影响。

二、客户需求说明书参考模板

这里提供的客户需求说明书模板仅供参考。要注意：好的文档在于思想深邃、善于表达、重点突出、逻辑清楚，而不仅仅是格式正确。

{ 项目名称 }

客户需求说明书

文件状态： ［√］草稿 ［ ］正式发布 ［ ］正在修改	文件标识：	Company-Project-RD-UR
	当前版本：	X.Y
	作　　者：	
	完成日期：	Year-Month-Day

版 本 历 史

版本/状态	作者	参与者	起止日期	备注

0. 文档介绍

0.1 文档目的

0.2 文档范围

0.3 读者对象

0.4 参考文档

提示：*列出本文档的所有参考文献（可以是非正式出版物），格式如下：*

［标识符］作者，文献名称，出版单位（或归属单位），日期

0.5 术语与缩写解释

缩写、术语	解 释
…	

1. 产品介绍

提示：

（1）说明产品是什么，什么用途。

（2）介绍产品的开发背景。

2. 产品面向的用户群体

提示：

（1）描述本产品面向的用户（客户、最终用户）的特征，

（2）说明本产品将给他们带来什么好处？他们选择本产品的可能性有多大？

3. 产品应当遵循的标准或规范

提示：*阐述本产品应当遵循什么标准、规范或业务规则（Business Rules），违反标准、规范或业务规则的产品通常不太可能被接受。*

4. 产品的功能性需求

4.0 功能性需求分类

提示：*将功能性需求先粗分再细分，下表中的 Feature A, Function A.1 等符号应当被替换成*

有含义的名称。

功能类别	子功能
Feature A	Function A.1
	Function A.2
	…
Feature B	Function B.1
	Function B.2
	…
…	

4.m Feature M

提示：此处写一些承上启下的文字。

4.m.n Function M.N

功能描述：

…….

5. 产品的非功能性需求

5.1 用户界面需求

需求名称	详细要求
…	

5.2 软硬件环境需求

需求名称	详细要求
…	

5.3 产品质量需求

主要质量属性	详细要求
正确性	
健壮性	
可靠性	
性能，效率	
易用性	
清晰性	
安全性	
可扩展性	
兼容性	
可移植性	
…	

5.n 其他需求

附录 A：用户需求调查报告

常见需求调查方式有：

- ✧ *与用户交谈，向用户提问题。*
- ✧ *参观用户的工作流程，观察用户的操作。*
- ✧ *向用户群体发调查问卷。*
- ✧ *与同行、专家交谈，听取他们的意见。*
- ✧ *分析已经存在的同类软件产品，提取需求。*
- ✧ *从行业标准、规则中提取需求。*
- ✧ 从 Internet 上搜查相关资料。

A.1 需求标题 1

需求标题 1	
调查方式	
调查人	
调查对象	
时间、地点	
需求信息记录	

A.n 需求标题 N

需求标题 N	
调查方式	
调查人	
调查对象	
时间、地点	
需求信息记录	

3.6 需求获取问题的进一步讨论

需求获取是需求分析中具有重大影响力的过程，必须花极大的力气把它做好，下面归纳总结出一些要点，供实际需求分析工作参考。

一、需求获取的指导方针

需求获取的指导方针如下。

1）合作才能成功：需求获取可能是软件开发中最困难、最关键、最易出错及最需要沟通的方面。需求获取只有通过有效的客户—开发者的合作才能成功。分析师必须建立一个对问题进行彻底探讨的环境，而这些问题与产品有关，而不是漫无目的的盲目的收集客户所有的想法。

2）对客户群进行分类：为了方便清晰地进行沟通，就要列出重要的小组，不要假想所有的参与者都持有相同的看法，而是需要对不同的客户群进行分类，正确地对客户群进行分类是成功的需求分析的第一步。

3）考虑功能和非功能需求：对需求问题的全面考察需要一种技术，利用这种技术不但考虑了问题的功能需求方面，还可讨论项目的非功能需求。确定客户已经理解：对于某些功能的讨论

并不意味着即将在产品中实现它。对于想到的需求必须集中处理并设定优先级，以避免一个不能带来任何益处的无限大的项目。

4）在建模中学习和发现需求：我们可以把建立业务模型、在建模中学习工作、在学习工作的过程中发现需求、分析与进一步明确需求这样几个方面有机的结合起来。

5）发现真正的需求：需求获取是一个需要高度合作的活动，而并不是客户想法的简单誊本。作为一个分析师，你必须透过客户所提出的表面需求理解他们的真正需求是什么。我们可以把客户的想法扩充出去，询问一些扩充的问题，将有助于你更好地理解客户目前的业务过程并且知道新系统如何帮助或改进他们的工作。

6）发现变更和其它可能方式：我们需要调查用户任务可能遇到的变更，或者用户需要使用系统其它可能的方式。这有助于使将来的设计更有指向性。可以想像你自己在学习用户的工作，你需要完成什么任务？你有什么问题？从这一角度来指导需求的开发和利用。

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

8）把客户的假设解释清楚：我们应该尽量把客户所持的假设解释清楚，特别是那些发生冲突的部分。从字里行间去理解，以明确客户没有表达清楚但又想加入的特性或特征。我们可以使用一种“上下文无关问题”，这是一个高层次的问题，诸如“产品要求怎样的精确度”或“你能帮我解释一下你为什么不同意某人的回答吗？”。这有助于获取业务问题和可能的解决方案的全部信息。客户对这些问题的回答诸可以使我们更直接地认识问题，而这是封闭问题所不能做到的。

9）利用所有可用的信息来源：需求获取需要利用所有可用的信息来源，这些信息描述了问题域或在软件解决方案中合理的特性。一个研究表明：比起不成功的项目，一个成功的项目在开发者和客户之间采用了更多的沟通方式。与单个客户或潜在的用户组一起座谈，对于业务软件包或信息管理系统的应用来说是一种传统的需求来源。

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

11）理解用户的思维过程：我们需要尽量理解用户用于表述他们需求的思维过程。需要摆脱我们自己原有的思维定势，站在用户的角度充分研究用户执行任务时作出决策的过程，并提取出潜在的逻辑关系。流程图和决策树是描述这些逻辑决策途径的好方法。

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

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

14）我们建议需求开发过程如下：

- 定义项目的上下文视图和范围
- 确定客户类型
- 在每个客户类型中确定适当的代表
- 确定需求决策者和他们的决策过程

- 选择你所用的需求获取技术
- 运用需求获取技术对作为系统一部分的用例进行开发并设置优先级
- 从客户那里收集质量属性的信息和其它非功能需求
- 详细拟订用例使其融合到必要的功能需求中
- 评审用例的描述和功能需求
- 如果有必要，就要开发分析模型用以澄清需求获取的参与者对需求的理解
- 开发并评估用户界面原型以助想像还未理解的需求
- 从用例中开发出概念测试用例
- 用测试用例来论证用例、功能需求、分析模型和原型
- 在继续进行构思系统每一部分之前，重复用例开发以后的步骤

二、需求获取中的挑战

在需求获取的过程中，需要面临许多挑战，这里仅仅列出了一部分：

- 适应有效的需求变化。
- 对非功能性，或者说质量需求的完整甚至定量的描述。
- 分析师缺乏应用领域的知识，客户和分析师讲不同的语言。
- 功能性需求描述完整，减少遗漏“明显”的信息。
- 开发不太昂贵的原型，要理解原型只是引出需求活动的一部分，而不是可执行的组件。
- 避免强调系统的计算能力，而不是系统将会改变业务运行方式。
- 在表达需求的时候，原则上允许开发人员确定需求是否可行和可测试。
- 确定可供借鉴和学习的相似系统。
- 没有充分理解计算机功能和相关人员的限制。
- 要根据增长的知识重新作需求。
- 需要强调方法学，而不是面向技术。“面向技术”是危险的，这样会把问题扭曲为适应技术，我们更加需要的面向问题以及面向客户。
- 需求的目的是确定“做什么”，而把“怎么做”的问题放在后面。
- 分割问题，以减少复杂性。
- 项目相关人员的背景多种多样，很难获得需求的共同语言。
- 在非“瀑布”模型中，设计将与需求收集同时进行，需求很少“冻结”。

需求获取的技巧可以帮助我们做一个很困难的事情，也就是从别人那里收集并精确提取出信息，这个工作必须由人（而且是聪明的人）来担任，还没有什么工具能够代替这项工作。完成这项工作的特点是：“我们有两个耳朵、一个嘴巴，中间是大脑。”我建议用这个比例来使用它们。

首先是倾听利益相关方所说的话，并理解他们的意思，如果不会倾听，那发现客户真正想要的东西的可能性就很小了。嘴巴的作用是发问，是反馈，这对好的沟通来说极其重要，技巧的使用这个反馈通道，就可以让需求工作活起来。最后，所有的信息都需要经过思考，只有深邃的思考，才可能构建真正好的产品。

好的分析师不仅仅是倾听，更重要的是思考，不仅仅是了解用户的工作，更重要的是在分析中提出新的设想，不仅仅是交付适用的产品，更重要的是实现客户的梦想。因此，好的分析是必须要有创造性思维，从了解中发现问题，从问题中发现机会。

完成需求收集以后就需要把这些材料进行分类整理，以此为基础形成产品模型，再从这些模型中归纳整理出产品规格说明，这就是分析的过程。但是这一切的基础就是能够把需求收集的完整而深入，所以需求获取是一个极其重要的过程。

第四章　面向产品：如何开发产品需求

在开发客户需求的阶段，我们已经通过各种方法收集了利益相关方（例如，客户、最终用户、供方、构造者和测试者）的需要、期望、限制和接口，并且转换为文档化的客户需求。现在我们需要把眼光转向开发团队，理解开发团队到底有什么问题？如何才能无歧义的开发？这就需要为接着而来的产品开发进行明确定义，约定未来产品应该是什么样的，这就是产品需求。

产品需求需要站在构思和定义未来产品的角度，依据客户需求，明确开发产品和产品部件需求集及其应用场景，这也是需求分析真正有价值的结果。

4.1 复杂系统的需求分解

仅仅梳理清楚了客户想要什么东西，并不能确保项目的成功。针对一个客户需求可能会有多种解决方案，从技术的角度来看也可能某种解决方法更好。这就需要沟通和交流，把未来产品应该是什么样的沟通清楚。

1，为什么要开发产品需求

有些人认为并不需要开发产品需求，只要把已经开发完成的客户需求交给开发团队去开发就可以了，但是事实证明这种情况存在很多弊端：

- 开发团队是一个比较复杂的群体，人们的经验与认识水平差异很大。如果没有准确定义产品需求，人们只是按照自己的理解去开发软件，最后交付的产品与客户的希望差别很大，也就在所难免了。
- 不论从哪个方面来看，客户对于产品质量还是有要求的。但是如果没有定义产品质量的需求，那么软件设计的依据是什么？什么是好什么是不好？
- 软件测试的依据是需求，仅仅依靠比较抽象的客户需求，能不能开发合理的测试用例？测试的依据又是什么？
- 如果对产品的需求没有定义也没有得到客户的批准，那么验收的标准又是什么？开发团队能确信自己开发的软件能被客户接受吗？

因此，开发出完善准确的产品需求，是成功产品开发的关键。

产品需求是产品开发的约束，但是与设计不同，它仍然是一种站在产品外部描述产品的手段，关注的是做什么，而不是怎么做。在进行产品需求分析的过程中，我们确实考虑了一定的技术与管理的信息，这可能对我们深入分析问题更有利，考虑问题也会更全面，更能符合项目背景。但在定义需求的时候，就需要排除这些信息。

2，分层的需求创建过程

在产品需求阶段由于需要定义产品的方方面面，相互之间的关系比仅仅是客户需求要复杂的多。为了使大规模复杂背景下的需求可以理解、可以开发，就必须限制每一部分的范围，使大问题变成小问题。这样的需求无论对于设计还是管理，都具有很强大的支持。

我经常看到人们书写出一个伟大的似乎面面俱到的需求规格说明，这种规格说明不仅仅读者难于阅读，连编写者想读第二遍都很难下决心。它所想表达的，只不过是说我们已经尽了足够努力，至于读不读就是另一回事了。我们应该反复强调的是：需求规格说明是给读者读的。所有利益相关人，包括：客户需要审查需求；设计者需要据此完成产品设计；管理者需要据此编写项目计划；测试者需要据此编写测试案例。因此，需求的可读性是一个关乎项目是否成功的关键因素。

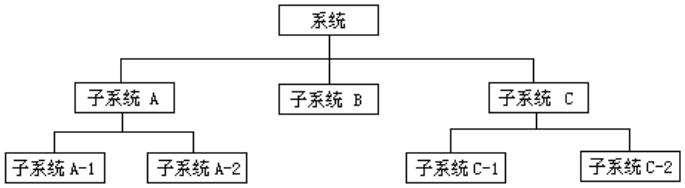
我们必须了解到，很少有需求能够在单个文档中定义，其原因是：

- 系统可能非常复杂，建档数量较大，需要有组织的和交互访问的技术。
- 该系统可能是相关产品系列的一个成员，没有什么文档可以包括所有的规格说明。
- 所构建的系统可能只是一个大型系统的子系统，仅能满足所确定需求的一个子集。
- 必须把市场和商业目标从产品的详细需求分离出来，这需要多个文档。
- 也可能在系统中建立制度或法规这样的需求，这些需求可以在其它地方进行建挡。

在大多数情况下，我们可能需要维护组成多个需求集的需求，每个需求集反映了一个特殊系统加上若干子系统的集合的需求。

1）按子系统组织需求

对于非常复杂的系统，唯一合理的方法是把它创建成由多个子系统组成的系统，这些子系统有时是其它子系统构成的系统。在特殊情况下，比如飞机控制系统，将包括上百个子系统，而每个子系统都有自己的硬件组件和软件。



2）创建系统级的需求规格说明

在这样的情况下，首先创建一个系统级的需求规格说明，在不了解或者不引用任何子系统的情况下来描述系统的功能行为。例如飞机油料的装载能力、爬升速度、飞行最大高度等等。这种规格说明并不需要描述很多细节，力求在抽象层面从期整体上把需求说明清楚。若干“用户故事”经过归纳整理，可以拼合成一些“主题”，这些主题潜在的就是一个子系统的需求描述。

3）进行子系统划分

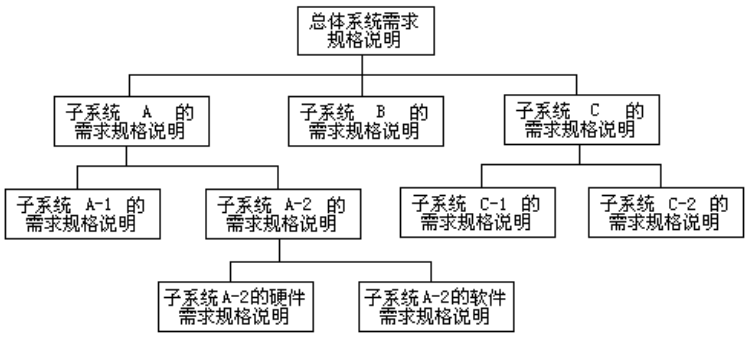
如同我们在上一节讨论过的，一旦对系统级的需求达成共识，就开始系统工程活动。我们可以把系统分解成多个子系统，描述子系统之间的接口，把系统级的需求分配到各个子系统，由此产生了系统架构描述，定义了系统划分以及系统之间的接口。

4）开发子系统需求规格说明

下一步，为每个子系统开发一个需求规格说明，这些规格说明应该在不引用它自己的子系统的情况下，完整地描述它的外部行为。在这个过程中会产生一类新的需求：派生需求。这类需求不是描述整个系统的外部行为，而是描述子系统的外部行为，因此，系统设计过程为组成系统的子系统创建了新的需求。特别是系统之间的接口成为关键需求。本质上这是子系统与另一个子系统之间的契约，或者是对子系统同意完成的事情的承诺。

5）对子系统进行划分

一旦需求达成一致，再次进行系统地分析和设计，如果需要，可以进一步把子系统分解成它自己的子系统，并且分别开发各个子系统的需求规格说明，如下图所示。



每一层，都把来自上一层的需求规格说明分配给适当的下一级需求规格说明。例如，燃料容量需求分配给燃料控制系统和燃料储备系统，同时恰当的发现和定义新的需求。直到最底层的需求规格说明指，也就是那些不能再分割的需求规格说明。

6）需求分析的分解的线路

这种按层划分的需求分析要求工作方法有两个线路：

- **自上而下**：在系统的层面，仔细分析从总体的角度有哪些要求，这些要求如何聚合成若干比较大的子系统，这些子系统又需要达到哪些目标？他们需要哪些功能集？交互上需要什么样的接口？如何进行测试？以此形成系统级的规格说明。然后每个子系统再逐渐按层细分，逐渐求精达到所需要的精细度。
- **自下而上**：仔细分析下层系统的需求，分析这些需求是如何保证达成上层需求的，已有的需求集对于上层要求是不是充分而必要的？有没有多余的部分？各个子部分交互的接口应该是怎样的？它们与上层的交互是不是符合上层接口规范？通过自下而上的分析以发现原来没有发现的问题。

这种自上而下、自下而上的过程可能要反复多次，以确保整体上需求是合理的。在需求的书写方法上不要求所有需求都书写在一起，可以根据规划把系统级、子系统级（或者更下层需求集）分开书写、分别评审并且赋给不同的开发团队。事实上从更广大的眼光来看，任何一个软件系统都不可能独立存在，它们都是更大系统的一个子部分，因此这样的处理方式是更符合实际情况、也是更有效的。

3，关于派生的需求

上述的工程方法可以看成一种问题分析技术，也是不断把复杂系统分解为简单系统的过程。在这样的情况下，我们会生成一些派生的需求，典型情况下，派生需求可以分成两种：

- **子系统需求**：这是子系统必须满足，但随最终用户未必有直接利益的那种需求（例如：子系统 A 必须计算飞行器的空速）。
- **接口需求**：这是子系统为了完成整体任务，必须与另一个子系统进行通信产生的需求，子系统创建的同时也促进了子系统之间接口的创建。

但是这些子系统需求是真的需求吗？对最终用户来说，它可能并不是重要的。但对于需求人员来说，开发人员也是需求的提供者，这些需求对于开发人员是重要的，所以也是一种至关重要的需求。还需要注意的是，这些子系统需求是由系统的分解得来的，不同的分解方式会产生不同的派生需求。所以，在前期与设计人员进行讨论是非常有意义的。

4，定义接口规范

在把需求组织成一种横向分解与层级结构的时候，就需要定义各个层次之间、各个子系统之间以及各个业务单元之间的接口。尽管设计也是从接口定义开始，但在一些关键的位置上，在初期把子系统需求分配给项目组，或者把业务单元分配给个人开发的时候，事先明确互相交互的接口规范，将会减少很多不必要的麻烦。在各种情况下，需求的接口定义提供了一种抽象，它将业务单元的接口与其实现分离，这样在开发后期可以带来多种重要效益，包括：复用、系统扩展、变化、可替换性、多态和分布式对象计算。

接口描述应该采用与技术无关的方式，例如使用下面的一张简单的接口定义表来定义接口。

接口定义表						
编号	**名称**	**输入参数**	**类型**	**输出参数**	**类型**	**说明**

很多人困惑：在需求阶段的接口定义到底要详细到什么程度？事实上在需求阶段并不需要定

义面面俱到的接口，而是要关注，哪些接口对于后期集成、测试等是重要的。另外在思考层面，在初期定义系统级需求的时候，主要考虑子系统对外的接口。在定义子系统需求的时候，则考虑各个功能单元对外的接口。换句话说，只在当前思考层面的下一层考虑接口，这样就大大减少了处理问题的难度，减少了不必要的猜测和不确定性。在更细节层面的接口，可以留给设计阶段再进行。

4.2 用例分析：描述产品部件的场景

在定义产品需求的阶段，除了宏观上定义系统级的产品相互关系以外，还需要定义清楚产品未来工作的细节，如何定义细节？这就要注意视野。一般来说，视野越宽，考虑问题越抽象。而需要考虑的问题越细致，就必须把视野缩小，把视野限制在一个个的产品部件内部去思考，就可以很好地把握细节。在这样的背景下，利用用例去描述产品部件的工作要求，并且利用用例去框定需求收集的范围，是一个好想法。

一、用例的完整概念

到底如何来定义产品需求？直观的想法是站在产品的角度把产品的功能表列出来就行了，但是实践让我们发现一些问题需要认真来解决：

- 如果不考虑使用状况，仅仅主观的列出需要什么功能，有没有可能这个功能从不会被人使用？有没有可能漏掉一些重要的功能？
- 如果不从客户的视角思考问题，仅仅从技术的视角主观的列出需求，能不能帮助分析人员发现客户真正的需要？
- 任何人对一段文字的理解都与自己的背景与体验有关，每一个以简略方式列出的功能列表，在没有应用场景支持的情况下，不同背景的人读出来的结果会不会一样？会不会出现多义性？开发人员解读这样的需求以后，会不会开发出完全不一样的东西？
- 需求的组织方式可能因每个人的喜好不同而有多种组织方式，一个技术背景深厚的人组织出的需求，从客户的眼光来看是不是容易理解？如果不容易理解，那么谈何沟通？谈何请用户帮助我们发现需求中的问题？

人们希望改变这个状态，把关注点由以产品为中心转向了是以用户为中心，事先沟通并且定义用户利用产品的工作场景，再通过这些场景发现和定义功能。与仅仅列出功能表相比，这种视角的变化很重要也很微妙，用例就是在这样的背景下提出来的。但是在实际中人们对用例方法有许多误解，因此，我们有必要对用例的完整概念作一个集中介绍。

1，什么是用例

用例是这样的定义的：用例是站在用户的角度，描述了系统完成客户目标作形成的动作序列，这一序列动作对特定参与者产生一个有价值的可见结果。

这是一个相当精确的定义。换句话说，用例是代表系统中各个项目相关人员就系统的行为达成的契约。每个用例描述一组事件，参与者通过发起与系统的一次次交互来实现某个目标，为了达成这个目标需要执行一系列的步骤（活动序列），从而得到了对这个参与者有价值的结果。

当然可以用流程图、顺序图等表示这种交互，但通常情况下用例是使用文本描述，这更容易人员之间相互沟通，而且对交互过程的表达也更加精确。

用例可以被用来记录需求，也可以描述业务过程，或者用来记录一个软件的行为。一个编写很好的用例应该具有很好的可读性，它由多个句子组成，所有句子都采用同一种语法形式描述一个简单的执行步骤。这样一来阅读用例将变得很容易。

要编写好一个用例，必须掌握三个概念：

- **范围**：真正被讨论的系统是什么？
- **主要参与者**：谁要求实现他的目标？
- **层次**：目标的层次是高还是低？

对于任何一个系统架构级别的用例，都会编写**黑盒**用例，这种用例将会专注于功能划分级别的行为，而不考虑内部细节。而对于业务过程的设计者，将会编写**白盒**用例，他会描述组织内部过程如何运作，技术开发者也需要利用白盒用例描述将来的系统具体工作情况。在不同的层次，用例的描述方法将会很不相同。

2，产品的用例模型分析

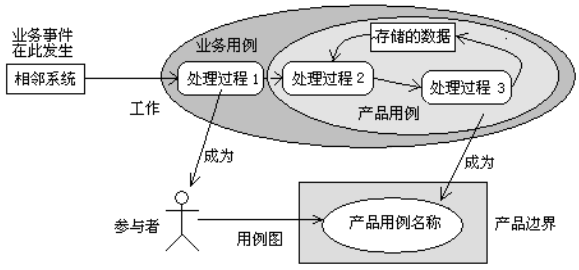
在开发客户需求阶段，我们已经强调了理解工作，而不是理解产品的重要性。通过查看更大范围的工作，可以对客户需求提出更多的问题并且构建出更好的产品。通过研究业务用例，我们主要考虑的是工作要做哪些事情，相邻系统的期望和预期的结果。

而在开发产品需求阶段，我们可以利用产品用例来描述未来的软件产品部件所需要做出的响应。但是要注意，产品用例只是业务用例在未来产品环境下的表达，我们可以通过检查工作是什么，以及工作如何响应外部请求来发现产品用例。

为什么在定义产品需求的时候把焦点集中在用例？这是一个好问题，用例的好处在于：

- 相对于传统的需求方法，用例书写简单，理解容易。
- 用例迫使开发人员从系统设计的时候就从用户的角度考虑问题。
- 用例使用户参与需求过程，帮助他理解所建议的系统，为用户提供一种沟通和记录他们需求的方式。
- 用例为需求提供一种排序的机制，人们可以说出下一个事件发生之前必须发生什么。
- 大多数情况下开发人员会参与书写用例，这意味着需求被真正的理解，而开发人员也知道要对这些需求负责。
- 用例在分析过程中是一个关键工具，帮助我们理解系统需要做什么，以及系统可能如何去做。
- 用例在设计和实现过程中也是一个关键工具，降低了把需求的表达向一种不一致的实现转化的风险。
- 用例可以直接延续到测试过程，有助于确保系统真正做它该做的事情。
- 用例也是用户文档的输入，可以很方便的把文档逐步组织起来。

实际上传统的需求规格说明方式仍然有效，有时候可以由用例引导出需求规格说明。但书写需求规格说明是一项很繁重的工作，在迭代开发的情况下，每一次迭代往往直接从用例直接进入开发。产品用例是从业务用例中推导出来的，下图表达了业务事件对产品用例的推动力。



精良的产品用例分析可以给后期工作带来巨大的好处，除了上面列出这一些，还可以在很多地方享受产品用例带来的好处：

- 它提供了一种手段，用于发现一些相关的需求并进行分组。
- 可以通过产品用例来计划实现版本。
- 测试人员可以把产品用例作为编写测试用例的输入信息。

- 产品用例为构建模拟原型提供了业务上的基础。
- 能够更早的响应变更，因为产品用例可以追踪到业务用例，然后追踪到业务事件。

3，用例作为项目连接结构

用例可以把许多其它需求细节连接在一起，为连接在一起的不同部分的信息提供了一个框架，有助于用户把概要信息、业务规则、数据格式等需求内容进行交叉连接。

从管理上说，用例也有助于组织在项目前期估算规模、制定项目计划、确定计划信息如发布时间、优先级、小组的并行开发等，它还有助于设计小组跟踪项目进程，以及规划项目测试。虽然这些信息本身并不在用例之中，但都与用例有关。

二、用例是规范行为的契约

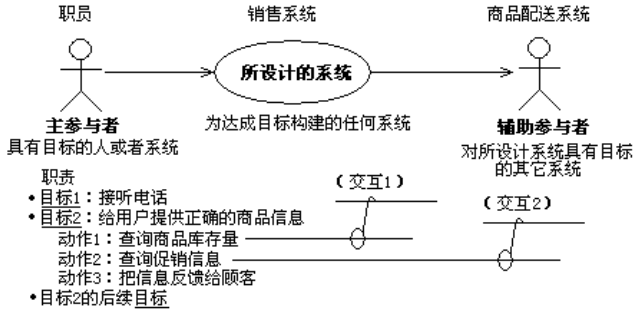
为了正确的开发系统，项目相关人员例如用户、需求分析人员、架构设计人员、编码人员、测试人员需要一个统一的契约，用例构成了契约中的行为部分。用例中的每个句子都有其价值，即它们各自描述了一项行为，该行为维护了或者增进了某些项目相关人员的利益。

因此，我们应该仅仅从具有某种目标的执行者之间交互行为的角度来考察一个用例。从概念上讲，首先是关注“执行者和目标的概念”，其次是关注“项目相关人员和利益”的概念。

1，具有目标的执行者之间交互

1）执行者具有目标

假设一个电话销售系统，职员负责处理电话购物请求（这个职员也就是主要执行者）。当客户打进电话来的时候，该职员就产生了一个目标：让计算机注册并启动这个请求。



在本例中，某些子目标要借助“辅助参与者”来实现，例如商品配送系统是原来就已经存在的业务系统，它所要完成的是一个子目标：根据订单，把正确的商品送到顾客手上。这个子目标也就是辅助参与者必须履行的承诺。

一般来说，最高的目标可以通过一系列的子目标来实现，但是不能这样无限细分下去，所以编写好的用例最大的困难，也就是目标粒度如何划分才是合理的？这需要我们对用例的分层有透彻的理解。

2）目标可能失败

如果职员记下顾客请求的时候，计算机崩溃了怎么办？这就需要为职员建立一个备选目标，例如用纸和笔记录顾客要求，并通过另外的工作流程使承诺能够实现，本质上这又出现了一个新的需求。

另一方面，系统在执行某些子目标的时候会遭遇失败，例如传送了错误的数据，或者商品配送系统可能由于某些状况不能运转（遭遇员工罢工？）。在这样的情况下，正常的工作流程无法进行，是采取备选方式呢？还是向顾客道歉？用什么方式道歉？

强调目标失败和思考目标失败后系统的反应，是用例被认为是出色的系统行为描述工具的原因之一，很多分析师和设计师都从中获得了重要的好处。

3）交互是复合的

最简单的交互是发送一条消息，但是更多的情况是一组消息序列或者场景，这称之为复合交互。例如上面的例子：

a、查询商品库存量。

b、查询促销信息。

c、把信息反馈给顾客。

d，获取顾客要求。

在更高的抽象层次上，也可以把这个活动序列进行压缩，把它作为一个单独的步骤：

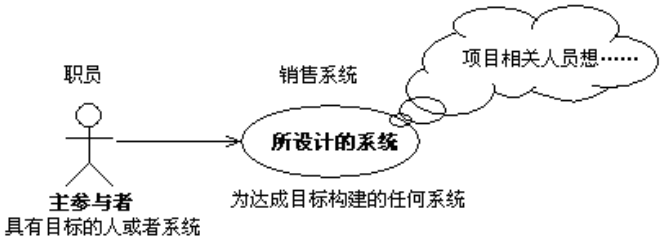
a，查询商品状态，获取顾客要求。

因此，交互可以根据需要折叠和分解，就像目标能大能小一样。其中每一步都能展开成一个单独的用例，也可以把多个交互合并成一个用例，就像上面讨论的子目标问题是一个样的。重要的是，通过对目标交互进行折叠，可以在一个很高的层次上表示系统行为，而通过把每个高层交互一点点地展开，也可以精细的刻划系统。用例常常被看成可以不断展开下去的故事，用例编写者的工作就是把这个故事书写的情节连冠、条理清楚，以便读者可以很舒适的在故事情节中切换、穿梭。

2，具有利益的项目相关人员之间的契约

执行者和目标模型解释了如何编写用例中的句子，但没有涉及如何描述行为方面的内容。出于这个原因，需要对用例基于“基于项目相关人员之间的契约”这个观点进一步扩展。

例如：ATM 必须保留一个日志来记录所有的交互过程，以备发生争议的时候项目相关人员的利益能得到保护。它也记录其它信息，以便能够记录失败以前交易到底进行到什么程度？ATM 在交付现金以前，首先要核实账号持有人是否有足够的存款，以确保 ATM 不会向客户提供比他在银行实际存款数还多的现金。



用例作为规范行为的契约，捕获了为满足项目相关人员的所有利益，并且权限于此。因此，要认真完成一个用例，就必须要列出所有的项目相关人员，根据用例的操作命名他们的利益，以及如果用例成功完成靠什么保证他们的利益。把这些搞清楚了，就可以编写用例的步骤，也就知道了用例步骤应该包括什么，不应该包括什么。

很多人并没有如此认真的编写用例，也幸运的完成了任务，很多东西是在软件开发的时候重新捕获得到。这种做法有时使项目进行得很顺利，但有时却会招致巨大的损失，这依赖于开发人员的水品。我们的任务是使项目开发不应该因为开发人员的水平波动对项目质量造成巨大影响。

3，用例的益处

用例被广泛采用的主要原因：详细描述系统被使用的时候行为细节，使得用户能够明白新系统应该是什么样的。这有利于用户尽早对系统运行的细节进行判断：是欣然接受还是拒绝？但是应用用例还有更重要的原因：

1）以用户为中心

用例方法给需求获取带来的好处，来自于用例方法的关注点由以产品为中心转向了是以用户为中心。比起使用以产品功能为中心的方法，用例方法可以更好的站在用户的角度与用户沟通，

他们到底在利用新系统做什么？这种视角的变化对于开发良好的需求非常重要。在许多 Web 开发工程中，用户代表发现，用例的方法可以真正有助于他们弄清 Web 站点的访问者到底可以做什么。用例有助于分析师与设计师理解用户的业务和应用领域，认真思考参与者与系统对话的顺序，这可以帮助我们在开发过程早期发现模糊性，也有助于从用例中生成测试用例。

2）便于明确系统目标

一般用例的命名被冠以系统的目标，并被收集整理成一个列表。这个列表声明了系统可以做什么，提示了系统的范围以及创建系统的目的。他成为项目不同的相关人员沟通的一个工具。

用户代表、主管、资深开发人员和项目经理会对这个目标仔细研究，并由此对项目的费用和复杂性作出初步的估算，然后他们再一起协商首先开发哪些功能，如何组建开发小组。这个列表也可以成为进行复杂性、经费、时间和状态度量的框架，它收集了项目生命周期中各种各样的信息。

3）强调了对异常情况处理的描述

一般在描述功能和行为的时候，大部分情况都关注系统应该做什么，而往往忽视了异常处理描述，这种分析漏洞造成的问题，在许多产品的实际运行中屡见不鲜。

用例的场景强调了对于异常情况的描述，从而便于发现许多以前没有考虑过的意外情况。当考虑例外情况的时候，通常总要和领域专家聚在一起，或者通过电话商量这种情况将如何动作，这样，就有可能发现新的项目相关人员、系统、目标和业务规则。

如果没有这些离散的用例步骤，以及对失败情况所作的集中讨论，那么许多错误情况就不可能在实际开发之前被发现。如果错误情况是在程序员编写代码的时候才被发现，这对发现新功能和新的业务规则来说就太晚了，那时业务专家已经走了，时间也非常紧迫，这时程序员只能想当然的按自己的想法编写相应代码，而不去寻求更理想的解决方案。

人们只要早期花不长的时间考虑异常步骤，就可以在后期得到更多的好处，且不说这已经挖掘出了潜在的需求，从而节省了大量的时间。

4）明确了职责

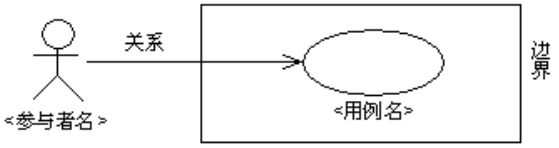
赋予职责是面向对象分析与设计（OOAD）的重要特征。职责不清晰的功能，有可能造成开发者所编写的代码从未被使用，这将是令人沮丧的。通过用例分析所得到的功能需求明确规定了用户执行的特定任务，从而防止了并没有与用户任务真正地联系在一起的“孤立”功能，这种“功能”甚至在运行中从没有人使用它们。盲目的功能列表带来的这种后果并不鲜见。

在技术方面，用例的方法也带来了好处。用例观点揭示了领域对象以及它们之间的职责。开发者运用面向对象的设计方法可以把用例转化为对象模型。进而，当业务过程随时间而变时，内嵌在特定的用例中的任务也会相应改变。如果你跟踪功能需求、设计、编码和测试以至到它们父类的用例以至于用户意见，那么这就很容易看出整个系统中业务过程是怎样级联变化的，使系统具有很强的回朔性。

三、用例模型及其创建

1，用例图

在 UML 中，利用了几个符号来表达用例中的元素，如下图所示。



参与者（actor）：具有行为能力的事务，可以是个人（由其扮演的角色来识别），计算机系统，或者组织，在 UML 使用一个小人来表达。

参与者的发现：发现参与者对提供用例是非常有用的。因为面对一个大系统, 要列出用例清单

常常是十分困难。这时可先列出参与者清单, 再对每个参与者列出它的用例, 问题就会变得容易很多。

用例：用例是关于单个参与者在与系统对话中所执行的处理行为的陈述序列。它表达了系统的**功能**和所提供的**服务**，在 UML 中，用例使用一个椭圆来表达。

2，用例的场景与事件流

1）用例图还是文档

不管怎么说，现在的情况是，很多人误以为椭圆就是用例，即使它根本没有传递什么信息。本课程多次在不同场合表达了这个观点，图比较适合表达关系，事无巨细用 200 个椭圆描绘一个开发需求，和什么都没做其实没有区别。图应该突出重点，描述一个文本的索引。而文本重在描述细节，在用例中这种细节主要表达状态和行为，而且既要考虑正常情况，也要考虑异常情况。图文并茂才是好的文档。

很多人发现，顶层用例图（外部参与者与用户目标用例）是很有用的，它提供了一个语境图，很多大的关系一目了然，用例图的价值除此以外似乎就没有什么了。

2）场景是研究一部分工作的分步骤情节

场景（scenario）：是参与者和被讨论系统之间一系列特定的活动和交互，通常被称之为“用例的实例”。通俗地讲，场景实际上是在说故事。一般来说，一个用例就是描述参与者使用系统达成目标的时候一组相关的成功场景和失败场景的集合。

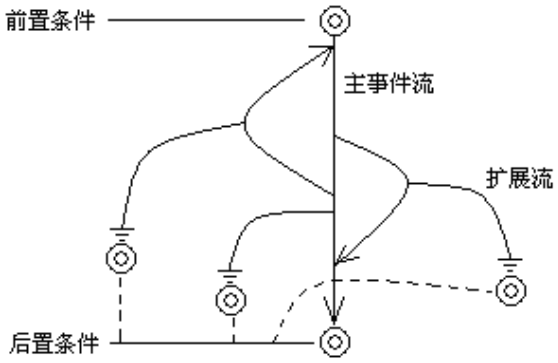
用例分析的关键是专注于“怎样才能使系统为用户提供可观测的数据，或帮助用户实现它们的目标”，而不是仅仅把系统需求用特性和功能的细目罗列出来。在需求分析中，我们必须专注于考虑系统怎么才能增加价值和实现目标。用例的主要思想是：为功能需求写出用例（而不是老式的问“系统会怎么做”的功能列表）。用例是发现和定义需求的重要方法，它反映的是功能性行为。

用例的一个场景（说故事），用来研究一部分工作的分步骤情节，而这个情节或者情景必须用规定的格式来描述。由于场景是一个中性的媒体，所有人都能够理解，业务分析师用它来对工作所要做的事情取得一致意见，在取得一致意见以后，利益相关方将决定多少工作由产品来完成，然后产生一个参与者与产品交互的场景。

3）用例事件流及应该包含的内容

作为产品用例主要的作用是构思和定义将要开发的软件产品，所以单单靠简单的用例图是不够的，还必须细化的书写和描述用例文档，这就是用例场景的描述，它也可以理解为对完成用例行为所需的事件的描述。

事件流描述了系统应该做什么，而不是怎么做。可以通过一个清晰的，易被用户理解的事件流来说明一个用例的行为。在事件流中包括用例何时开始和结束，用例何时和参与者交互，什么对象被交互以及该行为的基本流和扩展流，这可以用下图表示。



由此可以得出基本的场景描述文档的内容：

- **用例名**：这是唯一的名称，应该反映用例的主要工作。

- **主要参与者：**请求系统提供一项服务的主要项目相关人员。
- **描述：**简要描述该使用案例的作用（可以不写出）。
- **前置条件：**开始使用该用例之前必须满足的系统和环境的状态和条件（必要条件而不是充分条件）。
- **主事件流：**用例的正常流程（事件流是关注系统干什么，而不是怎么干），也称为用例的路径。可能包含有基本路径、备选路径、异常路径、成功路径和失败路径等几个方面的内容。
- **扩展流：**用例的非正常流程，如错误流程。
- **后置条件：**用例成功结束后系统应该具备的状态和条件（但不是每个用例都有后置条件）。

关于前置条件和后置条件，可以参考下面一些指导方针。

- 前置后置条件所描述的状态应该是用户能观察到的。可观察的状态例如“用户已在系统注册”或“用户已打开文档”等。
- 前置条件是用例启动时候的约束，但不是用例启动时候的事件。
- 虽然你可以在某一个用例层次上定义前置和后置条件，但它们不是只限于一个流程。
- 无论执行哪个扩展流，后置条件都应该为假，它仅对主事件流成为真。比如在后置条件中这样概括：“动作完成，或者出现错误，动作未能执行”，而不是“动作已经完成”。
- 后置条件对描述用例来说是重要的工具，你首先可以定义用例要获得什么后置条件，然后再考虑如何达成这个条件（所需要的事件流程）。

4）场景描述文档的基本要求

- **首先写出基本的路径：**这是最主要的事情，因为它是用户最关心或者最想看到的内容。
- **用例交互的四步曲：**

  参与者产生某个行为动作；
  然后系统对此动作进行响应；
  响应成功后再根据动作的要求进行状态的改变；
  最后将改变后的结果再回馈给参与者。

3，用例场景的模板格式

用例的模版可以有不同的形式，关键是要表达清楚。设想一个“酒店管理系统”的简单例子，我们描述一下场景（事件流）的文档。

用例名:	预定房间	**用例类型**
用例 ID:		
主要参与者:	客户	
描述:	该用例描述客户如何预定一个房间	
前置条件:	客户已经登录到系统	
后置条件:	成功预定之后，创建一个新的一项记录，特定时间的可用房间数将减少，如果预定失败，数据库不会发生任何改变。	
主事件流:	1，客户选择预定一个房间。 2，系统显示酒店拥有的房间类型，以及它们的收费标准。 3，客户选择需要的房间类型，并说明将停留的时间。 4，系统根据给出的周期，计算出总的费用并告知用户。 5，客户对选择的房间提出预定。 6，系统在数据库减少这个类型可预定房间的数目。 7，系统基于提供的详细资料创建一个新的预定。 8，系统显示预定确认号以及入住登记说明。 9，用例结束。	

扩展流：	6a， **重复预定** 如果存在相同的预定（同样的名字、e-mail、入住时间、离开时间）则系统显示原有的预定，并询问客户是否进行新的预定。 6a1，如果客户想继续，则系统开始新的预定，用例重新开始。 6a2，如果用户说明预定是重复的，用例结束。 6b，……
特别需求：	系统必须能处理 5 个并发预定，每个预定所花时间不超过 20 秒。

4，书写用例需要关注的几个问题

1）用例关注于未来的产品将会怎样完成工作

多年来，分析师总是利用情节或经历来描述用户和软件系统的交互方式，从而获取需求。虽然用例来源于面向对象的开发环境，但是它也能应用在具有许多开发方法的项目中，因为用户并不关心你是怎样开发你的软件，而关注于未来的产品将会怎样完成工作，从这个角度讲，用例就具有很大的优势。

2）重点是文本描述而不是图

我们需要注意的是：在这个阶段思考的重点是构思产品，因此使用用例的观点进行思维的过程，比是否画正式的用例图显得更为重要。而且为了更加严谨的定义产品，这时候的用例描述专注于文本描述而不是图。参与者可以映像到一个或多个可以操作的用户类的角色。基于“用例”方法进行需求获取的目的在于：描述用户需要使用系统完成的所有任务。

3）用例的结果集将包括所有合理的系统功能

尽管在业务分析的阶段我们已经使用了用例的概念，但是在那个阶段我们更多的是描述已经存在的业务，但是在产品用例的阶段，我们更侧重于构思某种满足客户需求的产品。这并不是简单的从业务用例到产品用例的映射，而是包括了若干创新的思考，以及对新系统开发的某种设计约束。在理论上，用例的结果集将包括所有合理的系统功能。在现实中，你不可能获得完全包容，但是比起我所知道的其它获取方法，基于用例的方法可以为你带来更好的效果。

4）用例为表达用户需求提供了一种方法

用例为表达用户需求提供了一种方法，而这一方法必须与最终产品的客户需求相一致。分析师和客户必须检查每一个用例，仔细考虑在把它们纳入需求之前决定其是否在项目所定义的范围内。为了识别系统事件，必须象前面用例的讨论一样，应该清楚的定义系统的边界，由于产品用例是用于软件开发的目的，系统边界经常被定义为软件本身，在这样的语境下，系统事件是直接激活软件的外部系统事件。

4.3 用例结构化：应对复杂性的手段

对于大多数用户和系统描述来说，上面的细化步骤多数情况可以为系统提供了充分而广泛地描述。但是随着应用的复杂性不断的增长和演变，就可能会遇到用例的分解，以及它们之间的结构化关系，下面我们来讨论有关问题。

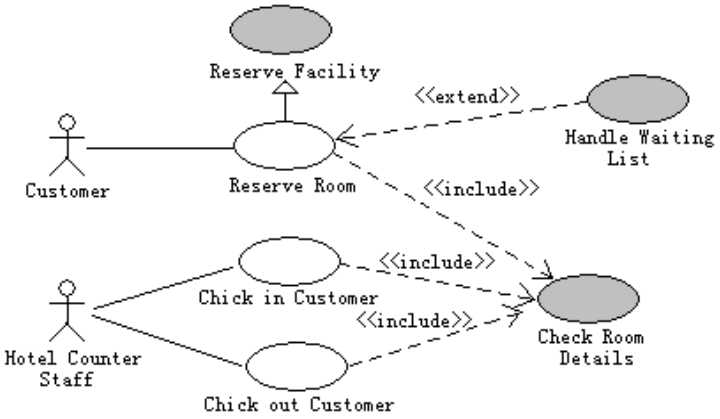
一、包含、扩展与泛化

用例通过扩展、包含、泛化等关系作为对关注点之间进行建模的手段，称之为用例的结构化。定义用例的结构化的关键之处，在于对于业务的共性与变化性的深刻理解，不同用例之间的关系如下图所示：

- **包含（include）**：一个用例的实现使用另一个用例的实现。其图形表示方法为在用例图上用一条从基本用例指向包含用例的虚箭线表示，并在线上标注构造型<<include>>。
- **扩展（extend）**：在特定的条件下，扩展用例可以在扩展点上增加新的行为，并且基本用

例不需要了解扩展用例的如何细节。其图形表示同样用虚箭线表示，并在线上标注构造型<<extend>>。

- **泛化（generalization）**：一个用例的实现从另一个抽象的用例继承。



二、包含的场景描述

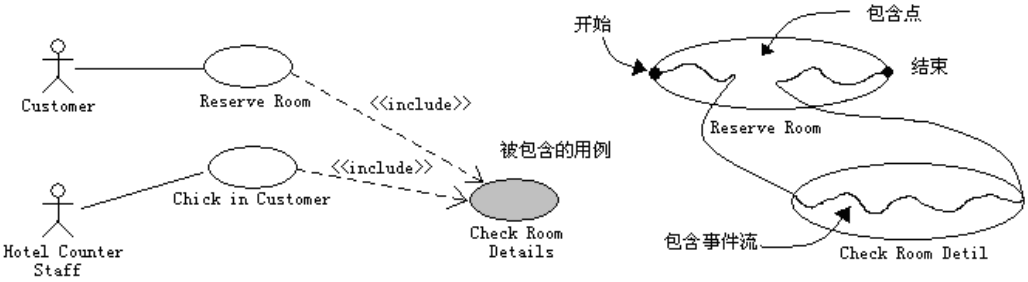
1，用例的包含（include）关系

为了确保产品设计的品质，产品开发的可追踪性与回朔性要好，因此在用例设计的时候，我们希望关注点相互独立，其重要性也不能相互比较，这就是所谓对等关注点。例如在酒店管理系统中，预订房间（Reserve Room）、登记入住（Check In Customer）、结账离开（Check Out Customer）都是对等关注点。

随着用例的细化和精化，团队会发现某些用户行为会在多个地方重复出现，事实上，很大一部分系统功能在多个地方重复出现时有可能的，比如输入口令进行用户确认。显然对相同的用户行为出现冗余的文档是不合适的，这就可以使用包含关系。包含的目的是在其它用例中引入用例。

作为分析师，我们应该注意到问题越是前期发现和解决，总的风险就越小。所以设计师希望从分析的时候就做到使关注点相互分离。在分析的时候，对于不同用例的相似步骤，可以把这些公共事件抽取出来，放在被包含的用例中，其它用例可以引用这些被包含用例中的事件流。

例如，“预定房间”和“登记入住”都需要参与者核对房间的可用性、以及查询有没有可用的房间等，这可以增加一个“核对房间清单（Check Room Details）”的包含用例。



（A）用例“预定房间”包含了用例“核对房间详情” （B）包含的业务过程

注意，大部分对子用例的调用是以包含形式表现，这使得开发团队很容易掌握，因为这类似于软件中的子程序，如果用的合理，包含关系能简化开发和维护活动，也是很重要的一种结构。

用例名:	预定房间	**用例类型**
用例 ID:	C-2	
主要参与者:	客户	
描述：	该用例描述客户如何预定一个房间	
前置条件：	客户已经登录到系统	

后置条件：	成功预定之后，创建一个新的一项记录，特定时间的可用房间数将减少，如果预定失败，数据库不会发生任何改变。
主事件流:	1，客户选择预定一个房间。 2，客户核对房间清单（C-3）。 3，客户对选择的房间提出预定。 4，系统在数据库减少这个类型可预定房间的数目。 5，系统基于提供的详细资料创建一个新的预定。 6，系统显示预定确认号以及入住登记说明。 7，用例结束。
扩展流：	4a， **重复预定** 如果存在相同的预定（同样的名字、e-mail、入住时间、离开时间）则系统显示原有的预定，并询问客户是否进行新的预定。 4a1，如果客户想继续，则系统开始新的预定，用例重新开始。 4a2，如果用户说明预定是重复的，用例结束。 4b，……
特别需求：	系统必须能处理 5 个并发预定，每个预定所花时间不超过 20 秒。

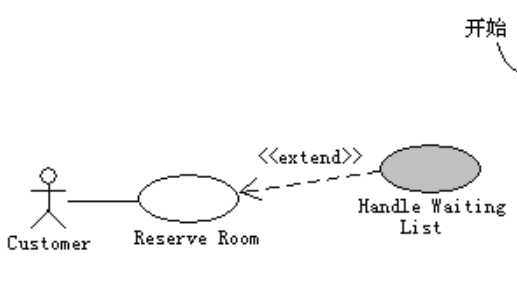
用例名:	核对房间详情	**用例类型**
用例 ID:	C-3	
主要参与者:	客户	
描述：	该用例描述客户如何预定一个房间	
前置条件：	客户已经登录到系统	
后置条件：	向客户返回总费用信息	
主事件流:	1，系统显示酒店拥有的房间细节，以及它们的收费标准。 2，客户选择需要的房间类型，并说明将停留的时间。 3，系统根据给出的周期，计算出总的费用并告知用户。	
扩展流：	………	

三、扩展的场景描述

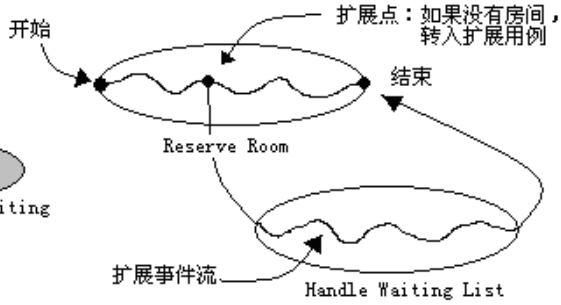
1，用例的扩展（extend）关系

随着系统随时间的不断演变，需要附加一些特性和功能来满足新的和当前的用户需要。假定有这样的情况，一个项目范围是团队一次迭代所能完成规模的两倍，你就需要砍掉一部分功能，把某些用例推到下一个周期完成。在第一个迭代周期完成一个主要的功能，下一个迭代周期完成一些扩展的功能。

例如，酒店管理系统中有一个功能“待分配房间的预订人等候名单”，如果没有房间，系统就会把客户放进这个“等候名单（Waiting list）”，因此，这个“Waiting list”就是“预订房间（Reserve Room）”的扩展。把扩展分离出来，可以使问题容易理解，这就就不至于被过多的问题所纠缠。下图表示了这个扩展用例的存在发生了什么。



（A）“等待列表”扩展了“预订房间”



（B）扩展业务过程

使用扩展用例的理由有三条：

- 作为一个实体，扩展用例能够简化用例的维护，同时允许团队关注并细化扩展的功能，

而不需要重读基本用例本身。

- 把扩展看成是用例开发，扩展点可以在基本用例中给出作为“通向未来特性”的途径。
- 扩展用例可以表示随意性的行为，而不是一个新的基本或扩展流程。

最后一条往往是最有用的。还需要注意需求的变更，标识出哪些需求将来可能变更，尽早提出一些解决办法，比如把可能变更的功能独立出来（至少建议这样做），这样就可以避免将来的需求变更带来不利的影响，所以，对于产品用例的考虑，我们可以在早期注意以下几个方面：

- 用例对应着功能包，所以用例的大小要合适。
- 注意到缠绕状态，标识出并且提出建议方案。
- 研究并且标识出易变性，把易变的功能独立起来。

从这些角度说，产品用例是在业务用例的基础上更进一步的思考。

2，用例扩展关系的场景描述

扩展用例包括一个或者多个扩展事件流，扩展事件流与备选事件流很相似，只不过它是在另一个用例中添加行为，例如在“酒店管理系统”中：

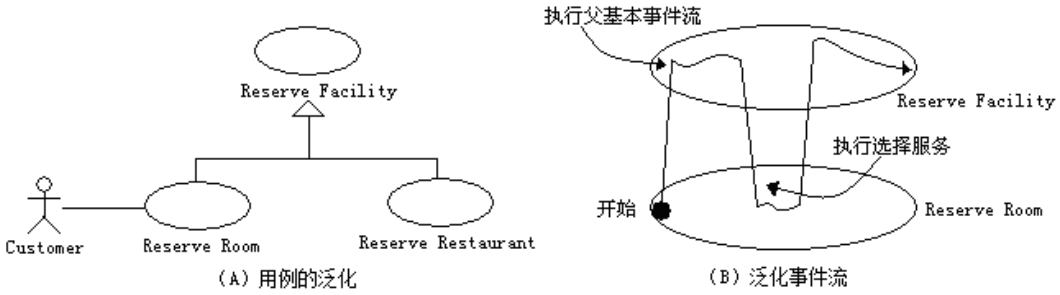
用例名:	预定房间	**用例层次**
用例 ID:	C-2	用户目标
主要参与者:	客户	
描述:	该用例描述客户如何预定一个房间	
前置条件:	客户已经登录到系统	
后置条件:	成功预定之后，创建一个新的一项记录，特定时间的可用房间数将减少，如果预定失败，数据库不会发生任何改变。	
主事件流:	1，客户选择预定一个房间。 2，系统显示酒店拥有的房间类型，以及它们的收费标准。 3，客户核对房间的收费（S1）。 4，客户对选择的房间提出预定。 5，系统在数据库减少这个类型可预定房间的数目（Ea）。 6，系统基于提供的详细资料创建一个新的预定。 7，系统显示预定确认号以及入住登记说明。 8，用例结束。	
扩展流:	……	
扩展点	Ea：更新房间可用性 Ea1，当没有客户所选择的房间的时候，该扩展事件流发生。 扩展事件流为房间预订队列（CS-5-EF1）	

用例名:	处理等候列表	**用例层次**
用例 ID:	CS-5	子功能
基本事件流:	……	
扩展事件流:	**EF1：房间预定队列。** 1，创建一个等候预定，并根据所选择的房间类型生成一个唯一标识符。 2，把等候预定放入一个等候列表中。 3，显示这个等候预定的唯一标识符。 4，基用例结束。	

四、用例的泛化关系及场景描述

泛化表达用例之间存在“is-a-kind-of”关系。当一组用例拥有相同的事件流序列，或者相似的一组约束的时候，可以使用例的泛化。泛化建模在特定领域架构中特别有效，在面向特征的领域模型中，泛化被用来描述领域中的共性与应用中的个性相互间的关系，使两者清晰分离，但通过一定的约束使两者结合起来。

一般父用例是抽象的，而子用例将继承父用例的特性，但是实现了变化的部分。如下图所示，“预订服务”是一个抽象用例，其中“选择服务”是一个抽象事件流。而“选择服务”的事件实现在“预定房间”用例中。事件流执行的规则如下：实例化首先出现在子用例中，沿着基本事件流运行，如果子用例没有定义基本事件流，则沿着父基本事件流执行，上面的例子由于没有定义子基本事件流，所以沿着父基本事件流运行。在运行到“选择一项服务”的时候，执行子用例定义的“选择服务”子用例。



（A）用例的泛化　　（B）泛化事件流

事件流的描述如下：

用例名:	预定服务	**用例层次**
用例 ID:	C-3	用户目标
主事件流:	该用例开始于某个客户预定一个服务 1，系统显示可提供的服务。 2，客户选择一项服务（S1）。 3，系统显示所选服务的总费用。 4，系统在数据库中减去相应服务的总数量。 5，系统为所选服务创建一个新的预定。 6，系统显示预定确定号。 7，用例结束。	
扩展流：	……	
子事件流：	S1：{abstract}选择服务	

用例名:	预定房间	**用例层次**
用例 ID:	C-4	用户目标
主事件流:		
子事件流：	C-3-S1，选择服务： 1，客户选择预定房间。 2，客户选择房间类型并说明停留时间。 3，系统根据给出的周期计算出总的费用。	

要注意到的问题是，首先不要混淆泛化和扩展，用例中的扩展与泛化是完全不同的，扩展事件流只不过是挂接到原有用例事件流的某个位置上，以添加新的行为，而且是在同一层级解决问题。另一个方面不能混淆泛化和包含，泛化和包含都是用于抽取用例的公共行为，因此容易混淆，其实它们是实现复用的两种不同手段，泛化关系要求父用例和子用例之间拥有“is-a -kind-of”关系，这样继承才有意义，但包含无需拥有这种关系。

五、利用用例描述需求要注意的问题

1，正确编写用例的提示

用例文档的编写最困难的地方在于，这是一种单调的写作方式，又需要富有完美的表达能力，使读者愿意阅读。当用例书写完毕以后，需要分析和回顾已写完的用例，使思路不断地被完善和清晰起来。用例编写的注意力应该放在文字上而不是画图上，对于正确的写作风格，我们将给出一些有益的提示。

1）使用例易于阅读

要是需求文档短小简明，而且易于阅读。从文法上说：要在现在时态中使用主动词。不是使用被动语态，要使用主动语态。要明确句子的主语在哪里，只写真正是需求的东西，不要提及与需求无关的东西。下面的一些习惯可以使你达到这个目的。

- 让问题短小、切题。长用例当然可以满足大的需求，但很少有人喜欢阅读。
- 从头开始，用一条主线贯穿始终。顶部是一些对全局有重要意义的用例，再从这里分离出用户目标和最终的子功能用例。
- 用动词短语来给用例命名，这些动词表达了用例所要达到的目的。
- 从触发事件开始一直连续，直到目标实现或者被取消，并且系统完成所有与这次事务有关的记录。
- 用完整的主动语态句子来描述所要完成的子目标。
- 确保每一步中参与者及其意图是可见的。
- 突出失败条件，并使恢复动作是可读的，最好是不必为每个步骤编号，人们就能清楚地知道下一步该做什么。
- 把可选行为放在扩展部分，而不是放在用例主事件流的条件语句中。
- 只在非常必要的情况下才生成扩展用例。

2）仅使用一种句型

在编写用例的每个执行步骤的时候，只采用一种句型。

- 现在时态的句子。
- 在主动语句中使用主动动词。
- 描述参与者成功到达的目标，这些目标推动了整个过程的前进。

在扩展的条件部分可以使用不同的语法形式，这样就不会使它和执行步骤产生混淆。

3）“包含”子用例

很重要的一条经验就是，大部分子用例的加入使用包含关系。混合使用饱含、扩展与泛化往往使读者产生混淆。也就是只在必要的时候使用扩展，当然在框架（Framework）设计的时候使用泛化有一定的好处，但只是在需要的时候使用。

4）两个结局

每个用例都有两个可能的结局：成功和失败。

记住：当一个执行步骤调用一个子用例的时候，被调用的用例可能成功也可能失败。如果是在主事件流中调用，则把失败处理放在扩展中。如果调用来自扩展，则成功和失败的处理均放在同一个扩展中。

对于目标的成功与失败，编写者实际上有两个职责：一个是确定对每个被调用用例的失败都进行了处理；另一个是确定用力满足了每一个项目相关人员的利益，特别是目标失败的情况下。

5）项目相关人员需要保证

用例不仅仅记录了主参与者和系统之间公共的可见的交互操作。如果用例仅仅完成了这些操作，那么它不是一个可接受的行为需求，而仅仅是一个文档化了的用户界面。

系统是为了达到项目相关人员利益上的目的。其中一个相关人员便是主参与者，在一般的用例格式中，其它相关人员并没有被列出来。但是用例需要满足这些项目相关人员的利益，并提供满足这些利益的保证。

一般主事件流和它的扩展瞄准了项目相关人员的利益，所以从主事件流开始阅读，看看是不是考虑了所有相关人员的利益，你会发现遗漏是经常的事情。例如，没有想到失败，因而没有记录下恢复信息，检查一下全部的失败处理是否保护了所有相关人员的利益。

在编写主事件流之前先编写保证条件是一个好的策略，因为在第一遍编写的时候就会考虑必要的保证，而不是后来发现它们再返回来对文本进行修改。

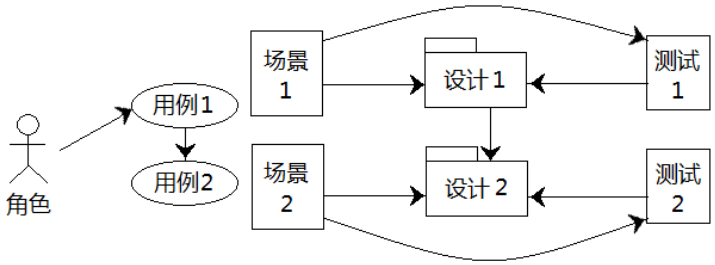
6）对用例进行通过/失败测试

简单的通过/失败测试，可以让我们知道什么时候已经部分的将用例编写完毕。下面是一个检查表，正确的情况是每一项都回答“是”。

对用例进行通过/失败测试	
域	**问题**
用例标题	1，是用主动动词短语命名主参与者的目标吗？ 2，系统能完成这些目标吗？
范围与层次	3，这些域填写了吗？
范围	4，用例是把范围内的系统作为一个“黑盒”对待吗？（如果是高层系统需求文档，答案“是”；如果是白盒业务用例，答案“否”）。 5，如果对范围内的系统进行设计，设计者是只设计范围内的事情，而不管范围之外的事吗？
层次	6，用例的内容与所描述的目标层次匹配吗？ 7，目标是在所描述的目标层次上吗？
主参与者	8，他有行为吗？ 9，他有目标与被讨论的系统的服务承诺矛盾吗？
前置条件	10，它是强制的吗？它们能在某些地方被所讨论的系统设置吗？ 11，在用例中，它们真的从来不需要检查吗？
项目相关人员及其利益	12，他们被指明了吗？系统一定能满足他们所描述的利益吗？
最小保证	13，所有项目相关人员的利益都被保护了吗？
成功保证	14，所有项目相关人员的利益都被满足了吗？
主事件流	15，它有 3～9 个步骤吗？ 16，它能从触发事件到交付成功保证运行吗？ 17，在执行过程中，它允许适当的改变吗？
在任何场景中的每 个步骤	18，它描述了一个成功目标吗？ 19，在它完成后，整个过程明显地向前移动了吗？ 20，是否清楚地指出哪个参与者正在操作目标？（谁在“踢球”？） 21，参与者的意图是否清楚？ 22，该步骤的目标层是否低于整个用例的目标层？它是恰好比用例目标层低一点吗？ 23，你确定该步骤没有描述系统用户接口的设计吗？ （例如 GUI 界面操作与交互） 24，在该步骤内传递的信息是否清楚？ 25，它的“确认”与“检测”条件相匹配吗？
扩展条件	26，系统能够并且必须发现和处理它吗？ 27，它是系统确实需要的吗？
技术与数据变动列表	28，你能否确定这不是主事件流的一个普通行为扩展？
整个用例的内容	29，对投资者和用户：“这是你们想要的吗？” 30，对投资者和用户：“在交付的时候，你们能识别出这是你们所想要的吗？” 31，对开发者：“你们能实现它吗？”

2，用例能不能单独成为需求

在软件开发中，用例及其场景是非常有用的。我们可以针对每一个用例引伸出多个功能需求，从而组合成一种功能结构。这种功能结构与参与者如何执行相关的任务结合起来，就使原本比较散乱、抽象、死板的功能描述获得了生命。而且基于用例的软件设计，可以保证需求、设计、测试前后的一致性，便于进行需求跟踪。



那么，用例场景能不能单独成为需求文件，用以控制开发与交付呢？这要根据具体情况。一般来说，用例场景只是描述了产品的活动，而且描述方法上有一定的抽象性。尽管功能涵盖在活动之内，但并没有很清晰的表达出来。另一方面，用例也没有严格的定义质量属性和约束，其作为技术文件的严格性是存在缺陷的。

但是用例及其场景确实是指导开发非常好的依据，各方依靠用例及其场景进行沟通和交流也很方便，所以用例及其场景越来越受到各方的喜爱（其进一步的普及化，称为用户故事）。从目前已有的经验来看，可以根据具体情况用多种方法来编写与一个用例相联系的功能需求文档，而且避免在不同地方产生信息冗余，因为冗余信息将使需求管理变得更加困难。基本上可以分成以下三种情况。

1）仅利用用例的方法

如果产品是组织内部开发的，客户的代表是产品策划部门派出的产品经理，双方可以深度交流。而且产品开发采用迭代模型，很多需求可以在开发过程中逐步呈现。此时，仅仅使用用例及其场景作为需求是合理的。

如果功能的描述是需要的，可以把功能需求包括在每一个用例的说明之中。其中的问题是可能会发生交叉引用在多个用例中重复的那些功能需求。解决这个问题的一种方法是，是可以通过先前讨论的“包含”的关系来阐述，在这个关系中，一些公共功能（如用户身份验证）被分割到一个独立的，可重用的用例中。

2）利用用例和 SRS 相结合的方法

如果产品开发与客户存在严格的合同关系，就需要对功能、质量和约束进行清晰的描述，这样仅仅采用用例描述需求就不能满足要求了。此时可以把用例说明限制在抽象的用户需求级上，并且把从用例中获得的功能需求编入软件需求规格说明（SRS）中，让用例与规格说明各司其责。

在这种方法中，首先要明确需求到底是以用例为主，而规格说明仅仅是一个对模糊问题的补充（此时往往命名为“补充规格说明”）呢？还是以用例作为线索，以规格说明作为开发的依据。这两种方式对文档的要求是不一样的。

在任何情况下，我们都需要在用例和与之相关的功能需求之间建立可跟踪性。最好的方法是把所有用例和功能需求存入数据库或者客户需求管理工具中，这将允许我们定义这些跟踪性联系。在相对比较简单的情况，功能需求编写的时候可以依据用例进行组织，并且使用标注来进行追踪。

3）仅利用软件需求规格说明的方法

很多组织规范要求以严格而正规的需求规格说明作为开发的唯一依据，这时候用例的场景描述就成为一个辅助的手段，仅仅用来帮助我们发现需求。在采用这种方法的时候，就不需要独立编写详细的用例文档，更多的精力是放在编写规格说明上。在这种情况下，把用例作为一种思考线路仍然是有益的，它可以使规格说明的组织更加合理，特别是在需要确定冗余的功能需求时，由于对每个功能需求仅需要陈述一次，而用例可以帮助我们梳理出这个功能将会在哪些地方出现。

3，注意几个用例陷阱

在用例的方法中应注意如下的陷阱：

- **太多的用例**：如果你发现自己陷入用例的爆炸之中，你就可能不能在一个合适的抽象级上为之编写文档。不要为每一个可能的说明编写单独的用例。而是把普通过程、可选过程以及例外集成起来写入一个简单的用例。也不要把交互顺序中的每个步骤看成一个单独的用例。每一个用例都必须描述一个单独的任务。你将获得比客户需求多得多的用例，但你的功能需求将比用例还多。每一个用例都描述了一个方法，用户可以利用这个方法与系统进行交互，从而达到特定的目标。
- **用例的冗余**：如果相同的函数出现在多个用例中，那么就有可能多次重写函数的实现部分。为了避免重复，可以使用“包含”关系，在这一关系中，将公共函数分离出来并写

到一个单独的用例中，当其它用例需要该函数时，可以请求调用它。

- **用例中的用户界面的设计用例**：应该把重点放在用户使用系统做什么，而不是关心屏幕上是怎么显示的。强调在参与者和系统之间概念性的对话的理解，而把用户界面的细节推迟到设计阶段。任何屏幕上的画面和用户界面机制的影响只是使参与者—系统交互的可视化，而并不作为主要的设计说明。
- **用例中包括数据定义**：我见过一些包括数据项和数据结构定义的用例，可以在该用例中操纵这些数据项和数据结构，包括数据类型、长度、格式和合法值。这个方法使项目的参与者难于找到他们所需要的定义，因为用例中说明它所包含的每一数据定义是不明显的。这也可导致冗余的定义，当一个用例改变时，其它的没有改变，但破坏了同步性。应该把数据定义集中在适用于整个项目范围的数据字典中，以便在使用这些数据时减少不一致性。
- **试图把每一个需求与一个用例相联系**：用例可有效地捕捉大多数所期望的系统行为，但是你可能有一些需求，这些需求与用户任务或其他参与者之间的交互没有特定的关系。这时你就需要一个独立的需求规格说明，在这个规格说明中可以编写非功能需求文档、外部接口需求文档以及一些不能由用例得到的功能需求支持。

小结：

分析师发现对于一些复杂的用例，画出图形分析模型是有益的，为了避免模糊性，更应该仔细书写模型的文档。反复对照模型和文档，就可以经常发现在文档中不易被发现的遗漏、模糊和不一致性的地方。

在每一个专题研讨会之后，分析师给出用例的说明和与研讨会参与者的有关的功能需求，研讨会的频繁程度应该有一个限制。每天开一个研讨会，将会使参与者难于在文档中发现错误。从紧张的思维行为中脱离出来，让头脑休息一两天，可以使你从一个新的角度来看待早先的工作。一周最好只主持二或三个研讨会。在完成每个用例的描述以后，我们就可以集中力归纳功能和非功能需求了。

第五章　深入分析：如何分析与确认需求

分析与确认需求的目的是确定预定运行环境对满足利益相关方需要、期望、限制和接口的能力有什么影响。必须考虑可行性、任务需要、费用限制、潜在市场规模、以及获取策略等所有因素。在考虑这些问题的时候还依赖于对产品目标的理解，必需的功能性定义也必须加以确定，并产生对实现关键功能序列的优先级分析。

5.1 功能性需求：产品应该如何工作？

发现功能需求的的目的，是要形成构建的产品的一份合约，因此，功能性需求必须十分详细的描述预期的产品将执行哪些活动。为了满足这个指标，功能性需求必须包含足够的细节，让开发能够构造出正确的产品，使需求分析师与利益相关方的误解减少到最低程度。

一、关注细节：事务与功能需求

在产品需求阶段已经对用例描述了工作场景，场景表达了为完成用例目标，用利益相关方能够识别并且熟悉的语言描述的一系列行为步骤。但是同时由于它抽象程度比较高，没有描述产品功能的细节。因此,通过场景发现和精确定义需求，成为需求分析必须考虑的问题。

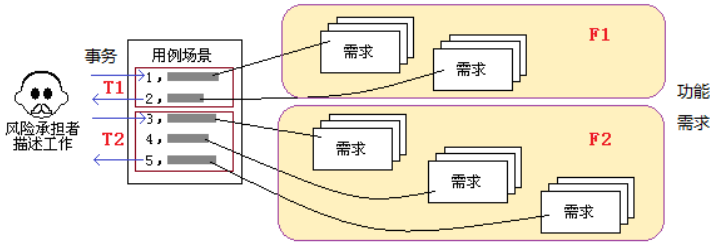
1，如何更好地组织需求

用例方法对于我们如何发现和组织需求带来了一个新的视角，为了使需求更规范、清晰、有条理，便于在用例和与之相关的功能需求之间建立可跟踪性，并且对于用户来说更加符合实际状况，就需要考虑发现和组织需求的方法。

我们可以这样来考虑：

- **首先：**把每个用例看成一个大的需求集，描述清楚这些需求集之间的交互关系，这就自然的使需求与用例之间有了映射关系。
- **其次：**把视野集中在每个用例。一个用例表现了为实现一个大的目标所必需的行为，而每一个大目标需要若干小目标来实现，这个小目标就是事务。在用例点规模估计中，我们已经定义了事物就是从用户到系统，再回到用户的一个“环形路线”，我们可以把每个用例所包含的若干事务分离出来并进行命名。这就有了一个清晰的需求框架。
- **最后：**再把视野集中到每个事务，仔细分析为了完成这个小目标（事务），系统需要哪些功能？这些功能有哪些模糊不确定的地方？能不能据此定义开发？

这种从功能集到事务再到功能的层层分解方式，不但减少了需求定义的难度，而且是最后形成需求文档和用例文档之间有很好的追踪关系，并且避免了需求描述的信息冗余，下图展示了这种渐进式得到功能性需求的方法。



2，通过事务发现功能

我们用已经讨论过的“预订房间”作为案例，讨论如何在场景中发现事务并细化功能的，其用例的场景如下。

用例名：	预定房间	**用例类型**
用例 ID:		
主要参与者:	客户	
描述：	该用例描述客户如何预定一个房间	
前置条件：	客户已经登录到系统	
后置条件：	成功预定之后，创建一个新的一顶记录，特定时间的可用房间数将减少，如果预定失败，数据库不会发生任何改变。	
主事件流:	1，客户选择预定一个房间。 2，系统显示酒店拥有的房间类型，以及它们的收费标准。 T1 提供类型选择 3，客户选择需要的房间类型，并说明将停留的时间。 4，系统根据给出的周期，计算出总的费用并告知用户。 T2 计算费用 5，客户对选择的房间提出预定。 6，系统在数据库减少这个类型可预定房间的数目。 7，系统基于提供的详细资料创建一个新的预定。 T3 创建预定 8，系统显示预定确认号以及入住登记说明。 9，用例结束。	
扩展流：	6a， **重复预定** 如果存在相同的预定（同样的名字、e-mail、入住时间、离开时间）则系统显示原有的预定，并询问客户是否进行新的预定。 T4 处理重复预定 6a1，如果客户想继续，则系统开始新的预定，用例重新开始。 6a2，如果用户说明预定是重复的，用例结束。 6b，……	
特别需求：	系统必须能处理 5 个并发预定，每个预定所花时间不超过 20 秒。	

据此可以发现功能需求如下：

F1：提供类型选择
客户请求预订，显示房间类型，显示收费标准。
F2：计算费用
客户选择类型，客户声明周期，系统计算费用，显示费用。
F3：创建预定
客户确认预定，更新预定记录，建立预定，显示预定信息。
F4：处理重复预定
发现重复预定，提示重复预定，处理重复预定。

3，细节的清晰化

在发现功能的时候，有一些细节的含糊之处要清晰化。例如：在道路除冰系统中，有一个名为“产生道理除冰调度表”的用例，该用例参与者是卡车库负责人，用例图如下。

用例名：	产生道路除冰调度表	**用例类型**
用例 ID:		
主要业务参与者:	卡车车库负责人	
基本事件流:	1，工程师提供调度日期和区域标识符。 T1 确定调度日期与区域 2，产品选择相关的热象图。 3，产品运用热象图，区域温度读数和气象预报来预测该区域每个道路的温度。 T2 确定道路结冰状态 4，产品预测哪条道路会结冰，何时会结冰。 5，产品从相关的车库中调度可用的卡车。 T3 车辆调度 6，产品向工程师提供建议的时间调度表。	

这个用例一共有 3 个事务个事务：确定调度日期与区域，确定道路结冰状态，车辆调度；需

要与利益相关方一起对每个步骤问一个问题：为了完成这些步骤，产品必须做些什么？

1）第一个事务：确定调度日期与区域

步骤 1：工程师提供调度日期和区域标识符

第一项功能需求就非常明显了：接收调度日期。

当你问利益相关方关于调度日期有没有特殊的要求的时候，他们会告诉你调度日期不会比当前日期超出两天，这项信息提供了另一项功能需求：判断日期在 2 天之内。

进一步的另一项需求是：接收有效的地区标识符。

问一下“有效”的含义，就会发现另一项需求。如果标识符标识了工程师负责的一个地区，那它就是有效的。如果它与工程师所希望的地区的标识符相符，它也是有效的。这导致了另外两项功能性需求：

验证地区在除冰职责之内；

显示验证结果。

从一个事务中导出的需求的数量并不重要，但是经验告诉我们，它通常少于 6 个。如果每个事务只发现了一个需求，这表明这个场景的粒度太细，要么就是用例很复杂。

发现功能需求的目标是要发现足够的功能需求，让开发者能够构建出客户希望的产品，也是参与者用它来完成工作的产品。

2）第二个事务：确定道路结冰状态

让我们考虑一下这个事务中的一个用例步骤：

步骤 4：产品预测哪条道路会结冰，何时会结冰。

这个步骤导致了 3 项功能需求：

确定地区中何处预计会结冰。

确定地区预计将会结冰路段。

确定路段何时会结冰。

现在以同样的方式，处理场景中的每个步骤，当穷尽了所有步骤以后，就为这个用例写好了功能性需求。通过与一组同事一起对需求复查了一遍，来测试是否完成了这个用例。在这个过程中，你应该能够清楚地展示为参与者提供了正确的结果的用例。

最后明确的功能需求如下所示：

F1：确定调度日期与区域
接受调度日期，判断日期在 2 天之内，接受有效地区标识符，验证地区在除冰职责内，显示验证结果
F2：确定道路结冰状态
接受区域热象图，接受区域温度读数，接受气象预报读数，预测道路温度；
确定预计结冰区域，确定结冰路段，确定路段中结冰时间，显示预测图像；
F3：车辆调度
显示相关车库状态，调度可用卡车，提供建议的时间调度表；

4，异常和可选方式

一个用例场景至少有一个基本流程，它也至少应该有一个事务。没有事务的流程是没有意义的，因为系统在没有刺激源时什么都不会做，用户在没有弄清系统的反应之前也不会提供任何新的刺激源。

异常是不希望但不可避免的对正常情况的偏离，异常场景反映了产品怎样从不期望发生的事情中恢复过来。用例几乎都会有描述处理例外或异常的流程（扩展、备选）。每一个例外流程都至少含有一个事务，这点也适用于一个可选择的流程。

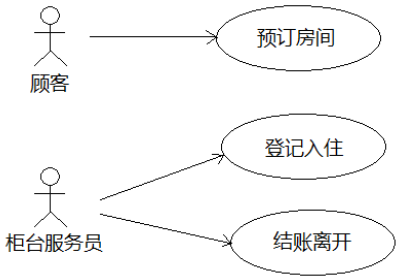
在发现功能需求的时候，需要针对每个异常步骤，确定产品为了完成这个步骤所必须做的事情。要关注处理异常和可选场景，事实上有时候这构成了大多数需求。因为人们在使用软件产品的时候能够作出最奇怪的动作，所以需要确定大量的恢复功能。

5，提取出重复的可以共享的功能

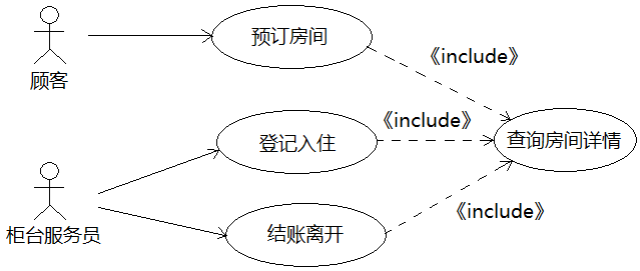
在完成了每个场景的功能提取之后，需要整体审视一下已经形成的功能表，看看哪些功能是重复的可以共享的，并把它们提取出来。虽然在编写用例场景的时候，已经利用用例的包含关系做了一些这方面的工作，但是基于下面一些原因可能是不充分或者不正确的：

- 最初掌握的信息不足，不足以发现更多的共享功能；
- 场景可能是由不同的人编写的，限于范围，不足以发现更大范围内的共享。

在掌握更多信息的情况下，我们会发现原来用例场景的不足，从而修正用例的场景，使他们趋于完善。我们用上一章讨论过的宾馆管理作为例子，最初编写场景的时候针对顾客和柜台服务员的应用场景，是由不同的人编写的，所做出的用例可能是下面的样子。



在针对每个场景提取功能需求之后，在统一审视的情况下，人们发现在每个用例中都存在“查询房间详情”的功能集。于是把它们提取出来，反过来修正原来的用例，如下图所示。



我们不可能一下子掌握所有情况，对情况的掌握是在过程中逐步完成的，这种后一步骤对前一步骤的修正称之为反馈。在软件工程中，有效的利用反馈过程的原理，可以大幅度提升每个过程的质量。

例如，前面我们在明确了客户需求之后，根据新掌握的信息，可能会修正原来确定的范围（总比到了最后范围蠕变强）；在功能提取之后，可能会修正原来确定的场景；在进入设计阶段，根据掌握的更多设计信息，可能会修正最初确定的需求；在每个阶段集成测试并评审之后，我们可能会根据已经获得的新情况，修正已有设计，甚至提出需求变更要求。

遇到这些情况不要奇怪，这正是符合人们认识事物的螺旋上升法则，与其避开这种螺旋性，还不如承认它，并利用它帮助我们提升质量。不要怕修改，精雕细刻与粗制滥造根本性的区别，就是一个是反复修改的，一个是做出来就算了。那么这会不会影响效率呢？从一个节点看是会的。但是从整体上看，前期付出的努力会在后期获得丰厚的回报，并且会大大提升整体的效率。这种精益求精的文化，无论对于企业还是个人，都是良好的正向推动力。

二、避免误解：如何减少二义性

不论需求是书面的还是访问的口头描述，都应该避免二义性和因此带来的误解。

相同的词在不同的人群中，或者在不同的专业圈子里使用，慢慢就就会导致意义不同。在编写需求的时候，不但要关注同名异义的词，如果产品的上下文不清楚，也会引起二义性，例如：“传递 24 小时天气预报。”这个含义取决于需求类型，是产品要传递未来 24 小时的天气预报吗？还是仅仅传递某种天气情况 24 小时？

建议根据产品用例把需求分组，这种系统的组织方式将会某种程度上减少二义性。例如，考虑如下需求：“产品将显示**所有**预计会结冰的道路。”

其中的“所有”指的是产品知道的所有道路吗？还是仅仅指的用户正在检查的那些？如果用例场景告诉我们前面参与者曾经指定了一个地区，或者地区的一部分，那么就可以说，“所有”指的是在所选择的地理区域内。实际上，几乎所有需求的含义都取决于它的上下文。其实这是一件好事，因为不需要费力气限定每项需求的每个词，那会浪费利益相关方的时间，当场景为需求设定了上下文的时候，就减少了二义性的危险。

类似的，一个工程师说：“我们希望结冰之前让卡车处理路面。”显然他指的不是卡车结冰前，而是路面结冰前，这可以通过上下文理解清楚。

在需求规格说明的“命名惯例和定义”部分，可以记录下项目中使用的每个词的含义，这种做法是消除二义性的起点。

在编写一项需求的时候，可以把它读出来，让利益相关方确认大家的理解是相同的意思，有的时候这个方法很有效。另外，真正的需求将在编写验收标准的时候展现出来，在那以前，有一份好的需求就可以了。

5.2 非功能性需求：产品的质量特征

在某些情况下，产品的非功能性方面是做产品的主要原因，某些产品的功能可能是一样的，但是，在易用性、性能和可靠性方面下的功夫，也可能是新产品最主要的需求。非功能性需求并不改变产品的功能，也就是说不管增加多少属性，功能需求都会保持不变。但是更复杂的是，非功能性需求也可能会为产品增加功能，比如产品可能更易于使用、更安全或者更直观。

任何非功能性需求对一个系统设计都有重要影响。这些影响包括可靠性、时间表、技能和成本的约束。比如，在时间紧迫、技能有限同时资金充足的情况下，更好的办法是购买和外包，而不是内部开发所有的组件。然而，对架构设计最具影响的因素，包含功能、可靠性、性能、支持性、实现和接口。通常是非功能性属性（如可靠性和性能）决定了某个架构的独到之处，而不是功能性需求。正是这些特点，在需求分析的时候必须关注非功能性需求的问题。非功能性需求很大程度上定义了产品的质量属性

一、关注质量：产品的特征与独到之处

非功能性需求有很多类型，大部分情况下与产品的质量属性有关，比较常用的是：

- 观感需求：产品的外观精神实质。
- 易用性和人性化需求：产品的易用性程度，以及更好的用户体验所需要的特殊可用性考虑。
- 执行需求：功能是实现必须多块、多可靠、能有多少处理量、多精确。
- 操作和环境需求：产品的操作环境，以及对该操作环境必须考虑的问题。
- 安全性需求：产品的安全性、保密性与可恢复性。
- 文化和政策需求：对于产品的操作所设计的人文化和习惯带来的特殊需求。
- 法律需求：哪些法律和表顺适用于该产品。

在描述非功能需求的时候，应该表达这个需求的描述和理由，先记录下利益相关方对这项需求的意图，将来会为每项需求加上一个测量指标（称之为验收标准）。

1）观感需求

例如，对于观感需求，当仔细研究了产品的目标受众以后，利益相关方希望产品对受众有吸引力，可能会要求如下的一些：

- 显而易见的使用简单性；

- 平易近人，这样人们就会不犹豫的使用它；
- 权威性，这样用户会依赖它和信任它；
- 与客户的其它产品兼容；
- 对儿童或者其它特定群体有吸引力；
- 不显眼，这样人们就不会注意到它；
- 创新并表现出艺术的方式；
- 看上去很专业；
- 令人兴奋；
- 酷，

2）易使用性

如果人们认为易用性是实符合用户能力及其对使用体验的期望，我们可以看一下需求规格说明中对用户的分类，他们需要哪种类型的人来完成他们的工作？他们需要哪种类型的产品来完成他们的工作？产品的易用性对用户使用产品的工作效率、错误率以及用户对新产品的接受程度都会产生影响。所以在编写易用性需求之前，需要考察客户需要产品达到什么目标。

比如：该产品应该易于使用。

这个需求就太模糊了，可以采取下面的写法：

产品应该对公众易于使用，他们也许不能看懂英语。

产品应该对具有认证资格的机械工程师使用。

下面是一种描述方式：

描述：产品应该为个人顾问提供一种喜欢的工作方式。

理由：为了树立起顾问对产品的信心。

验收标准：在经过 8 周的熟悉期以后，75%的顾问将愿意使用该系统。

还要注意“易于使用”和“易于学习”有轻微的不同，易于使用的产品着眼点是出促进持续的工作效率，他们在使用前可能经过了培训。易于学习的产品设计着眼点是针对不太经常使用的功能，他们可能忘了它的用法，但很快能够学会并使用，比如报表处理等。

下面是一种描述方式：

描述：当未经培训的公众第一次尝试使用该产品的时候，它应该易于学习。

理由：潜在用户可能从未使用过这类产品。

验收标准：由公众组成的测试小组中，90%的人在第一次使用该产品的时候，能在 45 秒内成功地通过产品购买一张票。

易用性还可以包括：

- 用户的接受率或者采用率；
- 通过引入该产品而导致的的生产效率的提高；
- 错误率（或者错误率的降级）；
- 被不说英语的人采用；
- 对残障人士的可用性；
- 被没有计算机使用经验的人使用（这是一项重要的，但往往被遗忘的考虑）。

对于操作和环境的需求，需要研究当产品与伙伴产品合作的时候，或者与其它信息系统接口的时候，或者与其它提供信息的系统接口的时候，也许要写出操作需求，这些系统在用例图中表现为参与者，在上下文模型中表现为相邻系统。

3）安全性

安全性可能是最难指明的一种需求，并且如果不正确的话可能会给产品带来极大的风险，一般来说，安全性包括 3 个方面：

- **保密性：**在编写这类需求的时候，实际上是在指明允许的访问（授权），在什么情况下授

权是有效的，对每个得到授权的用户来说，允许访问哪些数据和功能。

- **可得性**：得到授权的用户可以不受组织的访问数据，有时候也需要考虑防止数据丢失或者恢复丢失数据的需求。
- **完整性**：产品所保存的数据与相邻系统发送给产品的数据应该保持一致。

除此以外可能还需要一些完整性检查：

- 防止得到授权的用户以不希望的方式使用。
- 进行审计以检测不正确的用法。
- 在一些非正式的事件之后（断电、异常操作、火灾、水灾、爆炸），验证产品的文整性。

4）审计：

必要的时候，可以把审计列在需求规格说明的安全部分中，大多数会计产品都有审计方面的需求，审计需求要求产品包括一份审计跟踪记录，审计需求的准确实现必须与审计师沟通，例如可以这样写：

描述：产品应该保留法定的时间内所交易的日志。

理由：这是审计师所要求的。

在描述软件质量问题的时候，我们可以利用一些业内公认的软件质量标准。软件的质量由一系列质量要素组成，每一个质量要素又由一些衡量标准组成，每个衡量标准又由一些量度标准加以定量刻画。质量度量贯穿于软件工程的全过程以及软件交付之后，在软件交付之前的度量（内部属性）主要包括程序复杂性、模块的有效性和总的程序规模。在软件交付之后的度量（外部属性）则主要包括残存的缺陷数和系统的可维护性方面。

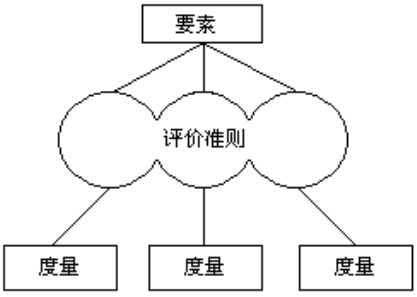
5）ISO 9126 软件产品评价：质量特性及使用指南

ISO 9126 称为“软件产品评价：质量特性及使用指南”。在这个标准中，把软件质量定义为“与软件产品满足声明的或隐含的需求能力有关的特性和特性的总和”，可以分为 6 个特性，即：功能性、可靠性、效率、可使用性、可维护性以及可移植性。这 6 大特性，每个特性包括一系列副特性，其中各质量特性和质量子特性的含义如下：

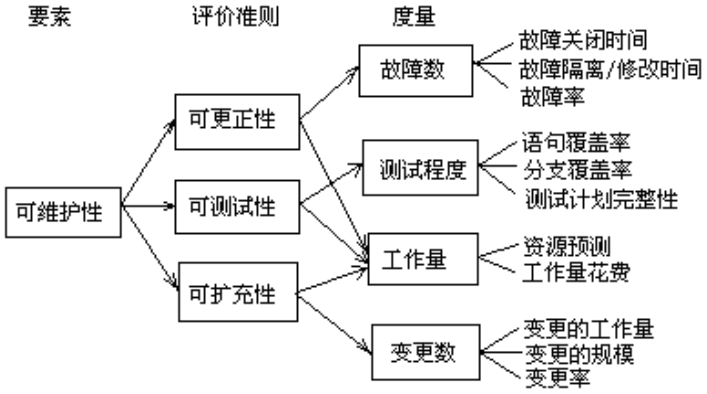
	特性	副特性
1	**功能性(functionality)**：与一组功能及其指定的性质的存在有关的一组属性。功能是指满足规定或隐含需求的那些功能。	**适合性(suitability)**：与对规定任务能否提供一组功能以及这组功能是否适合有相关的软件属性。 **准确性(accurateness)**：与能够得到正确或相符的结果或效果有关的软件属性。 **互用性(interoperability)**：与同其他指定系统进行交互操作的能力相关的软件属性。 **依从性(compli ance)**：使软件服从有关的标准、约定、法规及类似规定的软件属性。 **安全性(security)**：与避免对程序及数据的非授权故意或意外访问的能力有关的软件属性。
2	**可靠性(reliability)**：与在规定的一段时间内和和规定的条件下，软件维持其性能的有关能力。	**成熟性(maturity)**：与由软件故障引起实效的频度有关的软件属性。 **容错性(faulT tolerance)**：与在软件错误或违反指定接口的情况下，维持指定的性能水平的能力有关的软件属性。 **易恢复性(recoverability)**：与在故障发生后，重新建立其性能水平并恢复直接受影响数据的能力，以及为达到此目的所需的时间和努力有关的软件属性。
3	**易使用性(usability)**：与为使用所需的努力和由一组规定或隐含的用户对这样所作的个别评价有关的一组属性。	**易理解性(understandability)**：与用户为理解逻辑概念及其应用所付出的劳动有关的软件属性。 **易学性(learnability)**：与用户为学习其应用(例如操作控制、输入、输出)所付出的努力相关的软件属性。 **易操作性(Operability)**：与用户为进行操作和操作控制所付出的努力有关的软件属性。

4	**效率(efficiency)**：在规定条件下，软件的性能水平与所用资源量之间的关系有软件属性	**资源特性(resource behavior)**：与软件执行其功能时，所使用的资源量以及使用资源的持续时间有关的软件属性。
5	**可维护性(maintaina bility)**：与进行规定的修改所需要的努力有关的一组属性。	**易分析性(analyzability)**：与为诊断缺陷、失效原因或判定待修改的部分所需努力有关的软件属性。 **易改变性(changeability)**：与进行修改、排错或适应环境变换所需努力有关的软件属性。 **稳定性(sta、bility)**：与修改造成未预料效果的风险有关的软件属性。 **易测试性(testability)**：为确认经修改软件所需努力有关的软件属性。
6	**可移植性(portability)**：与软件可从某一环境转移到另一环境的能力有关的一属性。	**适应性(adaptability)**：与一软件无需采用有别于为关软件准备的处理或手段就能适应不同的规定环境有关的软件属性。 **易安装性(installability)**：与在指定环境下安装软件所需努力有关的软件属性。 **一致性(conformance)**：使软件服从与可移植性有关的标准或约定的软件属性。 **易替换性(replaceability)**：与一软件在该软件环境中用来替代指定的其他软件的

其软件质量模型包括 3 层，即高层：软件质量需求评价准则(SQRC)；中层：软件质量设计评价准则(SQDC)；低层：软件质量度量评价准则(SQMC)，也就是：质量要素 (factor)、评价准则(criteria)、度量(metric)。



ISO 认为这 6 个特性是全面的，也就是说软件质量的任何一部分都可以用这 6 个特性中的一个，或者多个特性的某些方面来描述。这 6 个特性中的每一个都定义为“一组与软件相关方面有关的属性”，并且能细化为多级子特性。例如：



可维护的分解图

二、抓住重点：明确关键质量属性

1，对质量属性分级

我们必须根据用户对系统的期望来确定重要质量属性。定量描述质量属性提供了对用户期望的清晰理解，这将有助于设计者提出最合理的解决方案。然而，大多数用户并不知道如何回答诸

如“互操作性对你的重要性如何？”或者“软件应该具有怎样的可靠性？”等问题。这就需要分析员想出了对于不同的用户类型可能很重要的属性，并根据每一个属性设计出许多问题，利用这些问题询问每一个用户类型的代表，可以把每个属性分成一级（不必多加考虑的属性）到五级（极其重要的属性）。这些问题的回答有助于分析员决定哪些质量特性用作设计标准是最重要的。

2）确定可测量的、可验证的需求

然后，分析员与用户一起为每一属性确定特定的、可测量的和可验证的需求。如果质量目标不可验证，那么就说不清你是否达到这些目标。在合适的地方为每一个属性或目标指定级别或测量单位，以及最大和最小值。如果你不能定量地确定某些对你的项目很重要的属性。

3）由错误发现质量需求

另一个定义属性的方法，是确定任何与期望相冲突的系统行为。通过定义不能发生什么事情（一种反向需求），我们就可以定义出强制系统必须表现出哪些行为。这种方法最适用于要求安全性能很高的应用程序，在这些应用程序中，系统的差错可能会导致生命危险。

4）目前需求中常用的质量属性

下表所列出的质量属性也是常用的。

有效性（availability）	健壮性(robustness)
高效性(efficiency)	可用性(usability)
灵活性(flexibility)	可维护性(maintainability)
完整性(integrity)	可移植性(portability)
互操作性(interoperability)	可重用性(reusability)
可靠性(reliability)	可测试性(testability)

下面我们将简要地介绍列在表中的每一个质量属性，并提出一些有助于用户陈述他们对质量属性期望的问题。虽然这些例子有些简单，但提供了由实际项目项目总结出来的样本目标的陈述。我们可以选择一种明确、无歧义方法来表达每一个质量属性需求，这样就可以指导开发者进行设计选择。

2，对用户重要的属性

1）有效性

有效性指的是在预定的运行时间中，系统真正可用并且完全运行时间所占的百分比。更正式地说，可以用下面的公式表达：

$$\text{有效性} = \frac{\text{平均故障时间（MTTF）}}{\text{平均故障时间（MTTF）} + \text{故障修复时间}}$$

有些系统有严格的有效性要求，例如，一个战场指挥系统当用户要执行一个任务，但如果系统在那一时刻不可用时，就会引起严重的后果。我们可以询问用户需要多高的有效性，并且是否在任何时间对满足业务或安全目标有效性都是必须的？一个有效性需求可能这样说明:“在一个工作日期间，在当地时间早上 6 点到午夜，系统的有效性至少达到 99.5 %，在下午 4 点到 6 点，系统的有效性至少可达到 99.95 %。

2) 效率

效率是用来衡量系统如何优化处理器、磁盘空间或通信带宽。如果系统用完了所有可用的资源，那么用户遇到的将是性能的下降，这是效率降低的一个表现。

拙劣的系统性能可能激怒等待数据库查询结果的用户，或者可能对系统安全性造成威胁，为了在不可预料的条件下允许安全缓冲，我们可以这样来定义效率:“在预计的高峰负载条件下，10 %处理器能力和 15 %系统可用内存必须留出备用。”这样的定义是可测试的。还要注意，在定义性能、能力和效率目标时，考虑硬件的最小配置是很重要的。

3) 灵活性

可扩充性、可增加性、可延伸性和可扩展性表现出了系统的灵活性，它表明了在产品中增加

新功能时所需工作量的大小。如果开发者能够预料到系统的扩展性，那么他们就可以选择合适的方法来最大限度地增大系统的灵活性。

灵活性对于通过一系列连续的发行版本，并采用增量型和重复型方式开发的产品是很重要的。例如在一个图形工程中，灵活性目标是如下设定的：“一个至少具有 6 个月产品支持经验的软件维护程序员可以在一个小时之内为系统添加一个新的可支持硬拷贝的输出设备。”

4) 安全性

安全性主要涉及：防止非法访问系统功能、防止数据丢失、防止病毒入侵并防止私人数据进入系统等。

安全性软件已成为互联网软件的重要议题。电子商务系统的用户关心的是保护信用卡信息，Web 的浏览者不愿意私人信息或他们所访问过的站点记录被非法使用。安全性需求不能犯任何错误，数据和访问必须通过特定的方法完全保护起来。我们应该用明确的术语陈述完整性的需求，如身份验证、用户特权级别、访问约束或者需要保护的精确数据。一个完整性的需求样本可以这样描述：“只有拥有查账员访问特权的用户才可以查看客户交易历史。”

5) 互操作性

互操作性表明了产品与其它系统交换数据和服务的难易程度。

为了评估互操作性是否达到要求的程度，我们必须知道用户使用哪一种应用程序与你的产品相连接，还要知道他们要交换的是什么数据。例如，“地理信息系统”的用户习惯于使用一些商业工具处理图形，所以他们提出如下的互操作性需求：“地理信息系统应该能够从××工具中导入任何有效地理信息结构图。”

6) 可靠性

可靠性是软件无故障执行一段时间的概率。健壮性和有效性有时可以看成是可靠性的一部分。衡量软件可靠性的方法包括正确执行操作所占的比例，在发现新缺陷之前系统运行的时间长度和缺陷出现的密度。我们应该根据如果发生故障对系统有多大影响，以及对于最大的可靠性的费用是否合理来定量确定可靠性需求。如果软件满足了可靠性需求，那么即使软件还存在缺陷，也可以认为达到其可靠性目标。要求高可靠性的系统也是为高可测试性系统设计的。

例如，一个武器系统测试设备的软件，这些设备在技术准备时对武器系统进行自动化测试。用户要求真正与测试相关的那部分软件要高可靠性，而其它系统功能，例如周期性地记录测试场地环境数据等，则对可靠性要求不高。对于该系统的一个可靠性需求说明如下：“由于软件失效引起武器系统测试失败的概率应不超过 5‰”。

7) 健壮性

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

例如我们要进行一个图形引擎的可重用软件组件的开发，该引擎将把通过卫星扫描的数据文件转换成三维图形显示。其它许多应用程序要请求调用这个图形引擎。由于在图形引擎中，我们将无法控制这些应用程序的数据，所以此时健壮性就成为必不可少的质量属性。一个健壮性需求是这样说明的：“所有的绘图参数都要指定一个缺省值，当输入数据丢失或无效时，就使用缺省值数据。”这个例子反映了对一个“用户”是另一个软件应用程序的产品，其健壮性设计的方法。

8) 可用性

可用性也称为“易用性”和“人类工程”，它所描述的是许多组成“用户友好”的因素。可用性衡量准备输入、操作和理解产品输出所花费的努力。我们必须权衡易用性和学习如何操纵产品的简易性。

例如“武器系统测试项目”的分析师会这样询问用户：“能快速简单地使用这个软件，这对你

有多重要？”和“一个普通技术人员使用这个产品完成测试大概需花多少时间？”对于定义使软件易于使用的许多特性而言，这只是一个简单的起点。对于可用性的讨论可以得出可测量的目标，例如“一个培训过的用户应该可以在平均 3 分钟或最多 5 分钟时间以内，完成从测试项目表中请求一种测试操作。”

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

3，对开发者重要的属性

1) 可维护性

可维护性表明了在软件中纠正一个缺陷或做一次更改的简易程度。

可维护性取决于理解软件、更改软件和测试软件的简易程度，可维护性与灵活性密切相关。高可维护性对于那些经历周期性更改的产品或快速开发的产品很重要。我们可以根据修复一个问题所花的平均时间和修复正确的百分比来衡量可维护性。

例如“武器测试系统”包括如下的可维护性需求：“在接到来自上级重新修订的武器测试报告的规定后，对于现有测试过程的更改操作必须在一周内完成。”又比如在图形引擎工程中，我们知道必须不断更新软件以满足用户日益发展的需要，因此，确定了设计标准以增强系统总的可维护性：“函数调用不能超过两层深度“，并且“每一个软件模块中，注释与源代码语句的比例至少为 1∶2。”我们必须认真而且精确地描述设计目标，以防止开发者过度设计，从而做出与预定目标不相符的愚蠢行为。

2) 可移植性

可移植性是度量把一个软件从一种运行环境转移到另一种运行环境中所花费的工作量。软件可移植的设计方法与软件可重用的设计方法相似。可移植性对于一个具体工程的成功是不重要的，对工程的结果也无关紧要。如果可移植是必要的，那么就应该陈述产品中可以移植到其它环境的哪一部分，并确定相应的目标环境。于是，开发者就能选择设计和编码方法以适当提高产品的可移植性。

3) 可重用性

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

4) 可测试性

可测试性指的是测试软件组件或集成产品时查找缺陷的简易程度。

如果产品中包含复杂的算法和逻辑，或者具有复杂的功能性的相互关系，那么对于可测试性的设计就很重要。如果经常更改产品，那么可测试性也是很重要的，因为我们会经常对产品进行回归测试来判断更改是否破坏了现有的功能性。

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

三、避免冲突：质量属性的取舍

1）如果不能同时实现两项需求，那么这两项需求就是冲突的

质量属性很多情况下会发生相互冲突，我们不可避免地要对一些特定的属性对进行取舍。必须和用户以及开发者一起确定哪些属性比其它属性更为重要，并定出优先级。在他们作决策时，要始终遵照那些优先级。

如果不能同时实现两项需求（一项需求的解决方案妨碍了另一项需求的实现），那么这两项需求就是冲突的。例如，一项需求是要求产品“被所有人使用”，而另一项需求要求产品“绝对安全”，那么这两项需求是不能按要求实现的。

2）发现冲突的第一个途径就是把需求按它们的类型分类

发现冲突的需求的第一个途径就是把需求按它们的类型分类，然后检查每个类型的所有需求项，寻找那些验收标准相互冲突的需求。在讨论非功能性需求的讨论中，通过质量属性冲突需求的检查矩阵，可以用来研究有冲突的需求。

3）质量属性之间一些典型的相互关系

下表描述了主要质量属性之间一些典型的相互关系，当然也可能会遇到一些例外。一个单元格中的加号表明单元格所在行的属性增加了对其所在列的属性的积极影响。例如，增强软件可重用性的设计方法也可以使软件变得灵活、更易于与其它软件组件相连接、更易于维护、更易于移植并且更易于测试。

	有效性	高效性	灵活性	完整性	互操作性	可靠性	健壮性	可用性	可维护性	可移植性	可重用性	可测试性
有效性								+		+		
高效性			—		—	—	—	—		—	—	—
灵活性		—		—		+	+	+			+	
完整性		—			—				—		—	—
互操作性		—	+	—			+					
可靠性	+	—	+					+			+	
健壮性		—	+		+	—			+		+	—
可用性	+	—	+			+				+	+	+
可维护性		—	+	—	+	+	+	—			+	
可移植性	+	—						+				+
可重用性	+	—	+			+		+				+
可测试性		—								+	—	

仔细研究这张表，可以发现一些质量属性之间内在的关系。一个单元格中的减号表明单元格所在行的属性增加了对其所在列的属性的不利影响。

例如，高效性对其它许多属性具有消极影响。如果你编写最紧凑，最快的代码，并使用一种特殊的预编译器和操作系统，那么这将不易移植到其它环境，而且还难于维护和改进软件。类似地，一些优化操作者易用性的系统或企图具有灵活性、可用性并且可以与其它软硬件相互操作的系统将付出性能方面的代价。

又比如，比起使用具有完整的制定图形代码的旧应用系统而言，使用外部的通用图形引擎工具生成图形规划将大大降低性能。所以我们必须在性能代价和提出的解决方案的预期利益之间作出权衡，以确保作出合理的取舍。

4）通过优先级分析达到质量属性的最佳平衡

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

- 如果软件要在多平台下运行（可移植性），那么就不要对可用性抱有乐观态度。

- 可重用软件能普遍适用于多种环境中，因此，不能达到特定的容错（可靠性）或完整性目标。
- 对于高安全的系统，很难完全测试其完整性需求；可重用的类组件或与其它应用程序的互操作可能会破坏其安全机制。

5）发现至关重要的质量属性

在软件中，其自身是不能实现质量特性合理平衡的。在需求获取的过程中，加入对质量属性期望的讨论，并把我们所了解的要求写入软件需求规格说明中。这样，才有可能提供使所有项目利益相关方满意的产品。

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

6）列出一些经常发现需求冲突的情况

在需求规格说明中，一项需求与其它需求发生冲突是完全有可能的，为了发现冲突的需求，下面列出一些我们经常发现需求冲突的情况：

- 使用相同数据的需求（按照使用的术语进行匹配查找）；
- 相同类型的需求（按照使用术语进行匹配查找）；
- 使用相同的度量尺度的需求（查找那些验收标准使用相同度量尺度的需求）。

在进行这种检查的时候，可以使用电子表格等工具。

7）需要考虑相同类型需求的不同场景

对于功能性需求，可以从它的结果中寻找需求。例如，假定有一项需求要求除冰车走最短的距离，而另一项需求指定要给最快的路线。这两项需求所指的并不一定是相同的路线，或者说可能得到的是不同的结果。

对于非功能性需求，要考虑相同类型需求的不同场景，例如，用户是科学研究者，要处理大量的数据，那么易用性就不应该包括“用户将不需要任何培训就能够执行所有用例”这样的验收标准。它可以是容易使用的，但这样的产品可能需要数月的培训才可能真正发挥作用。这就是说，验收标准与用户的指定是冲突的。

8）注意不同的用户可能会提出不同的要求

冲突的产生可能是因为不同的用户提出了不同的要求，或者用户提出的需求与客户的想法不一致，这对于大多数的需求收集工作来说是正常的，这表明需要有一种冲突解决机制。

作为需求分析师，冲突越早、越快解决就越有效，所以需求分析师应该扮演领导者的角色，当把冲突的需求隔离出来以后，单独与每个用户接触（这就是记录每项需求来源的原因），与用户一起查看需求并确保用户对需求的理解是相同的。

9）对冲突双方的用户单独进行需求收集和明确

对冲突双方的用户单独进行这项工作，注意让他们重新对客户满意和不满意度评分，如果某个用户对不满意度打了低分，就有可能说服他改变某些要求。在单独与用户交谈的过程中，找出他们的理由，他们真正想得到的结果是什么？如果其它需求优先，该需求是否可以妥协？大多是情况下我们可以用这种方式解决问题。注意我们使用的词是“冲突”，不需要争辩，不需要立场，用户也不需要知道冲突的对方是谁。当需求分析师研究清楚需求的成本、相关的风险等问题以后，把当事人召集在一起看看能否达成某种折中，一般情况下，这种折衷都是可以达成的。

5.3 验收标准：可测量的的需求

“验收”意味着解决方案完全满足了需求，但是为了测试解决方案是否满足需求，需求本身必须是可测量的。

一、验收需要标准的原因

如果产品有一项需求，要执行某个功能或者具备某种属性，那么测试活动就必须展示产品确实执行了该功能，或者具备了该项期望的属性。为了进行这样的测试，需求必须有一个测试基准，这样测试者才可能把提交的产品与最初的需求进行比较。测试基准就是验收标准，也就是产品的某种定量描述，它说明了产品必须达到的标准。

作为提交给架构师的需求，我们有理由认为，当他知道了产品的验收标准以后，他就会按标准构建，例如验收条件是：在水下 15 米处连续工作 15 小时。那么他们不太可能使用不防水的材料构建这个产品。

根据一个公认的测量指标对一项需求进行测试，其中最难的是定义一个测量指标，如果利益相关方要求产品“优秀”，那么必须找到一种方式来测量它“优秀”的程度。当然，这项指标必须得到利益相关方的同意，你可能不知道他说的“优秀”是什么意思，但与利益相关方就测量标准达成一致意见以后，构建者就可以创造出正确的产品，测试者也就可以证明它是正确的产品。

于是，我们需要发现一些“优秀”的测量指标，你可以提问“优秀”指的是什么？如果回答是:“职员喜欢使用它”,进一步的探究就会是:“产品应该职员本能的喜欢它，毫不犹豫的使用它。”

这样的回答就给出了一些可以测量的东西：再见到产品品质前的犹豫，到完全采用它需要多少时间？或者运用一段时间以后满意度（通过员工调查表），当与利益相关方达成一致意见以后，就有了一个“优秀产品”的测试标准。

要记住，所有的需求都可以测量，你要做的只是找到合适的尺度。

二、测量的尺度

测量的尺度是用于测试产品符合程度的单位，如果需求是针对一定的操作速度，那么尺度就是完成给定动作所需要的时间。对于一个易用性要求，可以测量学习产品所需要的时间，或者达到公认的能力水平的时间。对于可升级性，尺度可能使升级一项功能所需要的成本（钱币），或者是工作量（人/月）。几乎所有的东西都有测量尺度，关键是我们使用正确的尺度。

三、明确理由

理由就是需求的原因，或者存在的道理，我们发现，为需求加上理由，会使理解真实的需求变的更加容易。利益相关方常常会告诉你一个解决方案，而不是他们真实的需求，或者他们会告诉你一个模糊的需求，以至于没什么用处。

当用户说需要一个“优秀”的产品的时候，一个理由就是用户乐于使用，这时候的测试条件前面已经讨论过了。但如果理由是“容易使用”，那对应的需求就有了完全不同的意义，导致相应的验收标准也不相同。

理由不仅仅是帮助你发现验收标准的指导，还可能帮助你发现混合在一起的模糊的需求，比如前面说的“优秀”的产品，几个利益相关方可能有不同的理由，比如：易于使用、令人兴奋、还有用户愿意再次使用它。这样就会有三项需求，每项都有不同的不同的属性，以及相应的测量标准。

在向利益相关方询问理由的时候，最常说的词可能就是“为什么？”这是对的。

需求分析师的职责就是问为什么，不断地问为什么，直到理解了需求的含义。但事情还没有完，我们还需要做很多工作来测量需求的真正含义，这就是为什么需要导出验收标准。

为了找到合适的验收标准，要从分析得到的需求描述和理由开始，例如：

描述：产品应该让购买者容易找到他选择的音乐。

这比较模糊，让我们再看看理由：

理由：音乐购买者习惯了方便，不能忍受很不方便的找到音轨。

那么，速度成了合理的测量单位，而且与音乐购买者有关。市场人员调查相关群体，得到的结论是说：购买者应该在 3 个动作之内找到音轨，其中 10 秒钟是忍受极限。为了比对手做得更好，于是导致如下验收标准：

验收标准：平均音乐购买者应该能够在 6 秒钟之内，通过不超过 3 个动作，定位任意一段音乐。

由于真实世界、技术甚至成本的限制，某些验收标准可能不能实现，这是对验收标准就需要进行调整，以适应产品的操作环境、使用意图和客户的预算。请把这些调整堪称业务误差，因为我们可以肯定某些用户的操作水平会在平均水平之下，所以可以将验收标准调整如下：

验收标准：90%的音乐购买者应该能够在 6 秒钟之内，通过不超过 3 个动作，定位任意一段音乐。

四、非功能需求的验收标准

非功能需求是产品必须具备的品质，某些非功能需求看上去很难量化，但是总可以把它加上数字标准，如果不能对一项需求进行量化和测量，它就不一定是一项真的需求。它可能是多项需求合并在一起了，或者没有仔细考虑好、没有理由、偶然的，也许根本就不是需求。对这样的需求，或者是想办法量化它，或者是删除它。

例如，有这样的需求：

描述：产品应该对用户友好。

初看起来很模糊，但是可以找到测量它的方法，可以问一下用户，在他们心目中，“用户友好”的准确含义是什么？是易于学习，还是易于使用，或者是吸引人，还是其他含义？

假定用户呈请了他们的意图，比如：“我希望我的员工能够快速学会使用这个产品。”这就说明了一个尺度（快速），比如培训的时间等，我们可以这样写：

验收标准：新用户在第一次使用该产品的时候，应该能在 30 分钟内完成数据的添加、更改、删除操作。

请注意，这个测量标准定义的是具体用户关于“产品友好”的含义，但不是所有“产品友好”的含义。像前面关于“优秀”的含义，下面的表达式可测量的：

验收标准：在引入该产品的 3 个月内，60%的用户应该用它完成工作，在这些用户之中，应该有大于 75%的用户对产品表示赞许。

要注意验收标准对需求是起到澄清作用的，通过讨论测试尺度，需求从一个模糊的、二义性的意图，变成了一个完整的、可测量的需求。利益相关方在一开始是不太可能用这样的方式来表达自己的意图的，建议你顺着这样的流程走下去，这样并不会因为需要可测量就放慢自己的需求收集过程，而是可以更深入的了解利益相关方的意图以及理由。通过分析，可以编写对验收标准的最佳诠释。

1，产品是否失败

验收标准可以以一个问题来确定：“什么会被认为产品失败？”假定有如下需求：

描述：产品必须在可接受的时间内产品道路除冰调度计划。

显然测量的尺度是时间，那么如果客户告诉我们超过 15 分钟才产生计划是不能接受的，就可以得到如下验收标准（加上了业务误差）：

验收标准：在 90%的情况下，工程师应在在 15 分钟之内得到产品成生的道路除冰调度计划，任何情况下都不得超过该时间的 20 秒。

但是有的时候对测量标准不能达成一致意见，这种情况就需要考虑需求是不是有几项需求组成？能不能分解成不同测量方式的几个独立的需求。极端的情况是需求根本不切实际，比如：“该产品应该足够好，如果我的祖母还在世，她将会喜欢它。”显然这个需求是没有办法测试的。

2，主观测试

某些需求需要通过主观测试来进行测量。例如，有一项文化需求：“不冒犯任何团体。”那么验收标准可能是这样的：

验收标准：产品应该不会让测试组 85%的人感觉被冒犯。测试组有可能与产品发生联系的人员代表组成，测试组所代表的业务团体感觉被冒犯的不超过 10%。

不能指望 100%的人通过测试，在这个情况下，业务误差保护了产品不至于受到少数极端观点的攻击，同时又允许对“感觉被冒犯”进行测量。你可以使用原型而不是最终产品来进行这类测试，这样可能更有成本效益。另外，百分比数字不是任意制定的，它必须来自于经验数据，并且得到用户的认可，这种认可包括用户对这一目标的原因有很好的理解。

3，观感需求

观感需求是关于产品的外观和行为的精神、情绪和风格。

很多用户要求产品具备本公司的代表颜色，这种要求要么是坚持品牌标准，要么是希望强化顾客的认知，但这两种情况稍有不同。

坚持品牌标准：

验收标准：产品应该有市场部领导认证符合今年的公司品牌标准。

强化顾客认知：

验收标准：60%的目标用户在第一次见到该产品的 5 秒钟内，就意识到该产品属于该公司。

观感需求虽然是对意图的感觉式说明，但经过反复询问理由，还是可以找到可测量的方面。

4，易用性和人性化需求

易用性和人性化需求规定了产品对目标用户的方便性与合适性。下面来看一个例子：

描述：产品应该是直观的。

为了测量“直观的”，必须考虑直观是针对什么样的用户而言的，如果用户是道路除冰工程师，他们拥有工程学位，并拥有气象方面的经验。

理由：工程师必须认为产品直观，否则他就不愿意使用它。

有了这个理由，“直观”就具备了不同的意义。

验收标准：在首次使用该产品时，在不参考产品以外的帮助的情况下，道路工程师应该能够在 10 分钟内得到一封正确的除冰预报。

有时候“直观”值得是易于学习，这是必须询问可以花多少时间用于培训，从而得到这样的验收标准。

验收标准：在经过一天的培训以后，10 个道路工程师中应该有 9 个能够成功的完成[选择的任务清单]。

易用性需求的验收标准也可以使用量化来完成给定任务完成时间、允许的差错率、用户的满意率、易用性实验室的评分等，重要的事情事发现需求的真正含义，验收标准能正确测量这个含义，从而确认需求的含义。

5，执行需求

执行需求是关于产品的速度、精度、容量等方面的需求，大多数情况下，执行需求的本质将指出测量的尺度是什么，例如：

验收标准：95%的情况下，响应时间不超过 1.5 秒，其它情况下不超过 4 秒。

可访问性：

验收标准：在系统投运的前 3 个月中，在早上 8:00 至晚上 8:00 之间，产品的可用时间应该

达到 98%。

验收标准也可以使一个范围，例如：

验收标准：产品应该允许每小时 3000 次下载，但是每小时 5000 次更好。

使用范围的原因是防止开发者构建过于昂贵的产品，在预算与设计限制之间取得合理的平衡。

6，安全性需求

安全性需求包括了产品的许多方面，例如：

描述：只有使用 A 类登录的工程师能够进行增加、修改或者删除气象站的数据。

验收标准：在 1000 次气象数据的增加、修改或者删除中，全部由 A 类登陆的工程师完成，没有例外。

至于数据的一致性等等问题，都属于安全性考虑的范围，这里就不再讨论了。

五、功能性需求的验收标准

1，功能需求验收标准的描述

一般来说，对于功能需求来说，不存在测量的尺度，动作要么完成，要么没有完成。功能需求可能是不同类型的动作，例如：

描述：产品应该记录气象站的数据。

理由：准备除冰调度表需要这些读数。

验收标准：记录下来的气象站数据应该与负责传输这些数据的气象站数据相同。

在这里，验收标准并没有说如何被测试，这种陈述保证了测试者用它来确保符合需求。

如果功能需求是进行某种计算，则验收标准就应该指出：“计算结果应该与某机构的看法一致……”。功能需求的一般原则是：验收标准确保功能被正确执行，从中导出测试用例。

5.4 设定优先级：哪些需求是最重要的？

当客户的期望很高、开发时间比较短并且资源有限时，我们必须尽早确定所交付的产品应该具备的最重要的功能。合理建立每个功能的相对重要性，有助于我们规划软件的开发重点，使我们可能以最少的费用提供产品的最大功能。

一、为什么要设定需求的优先级

每个项目负责人都必须权衡合理的项目范围、进度、预算、人力资源以及质量目标的约束。比较常用的权衡的方法是：当接受一个新的高优先级的需求或者其它项目环境变化时，删除低优先级的需求，或者把它们推迟到下一版本中去实现。如果客户没有以重要性和紧迫性来区分它们的需求，那么项目负责人就必须自己作出这样的决策。但这样一来就会有风险，因为客户可能并不赞成项目经理所设定的优先级。当项目有很多选择可以完成一个成功的产品时，应该尽早在项目中设定其优先级。

二、从多个角度考虑设定优先级

事实上，在优先级上达成一致是有困难的，让每一个客户都来决定他们的需求中哪一些是最重要的，大部分情况下他们会把 85 %的需求设定为高优先级，10 %的需求设定为中等优先级，5 %的需求设定为低优先级。这种设定方式和没有设定优先级并没有太大的区别。要在众多具有不同期望的客户之间达成一致意见就更难了，因为人们心中都存在个人的利益，并且他们并不总能与其它群体的利益相妥协。这就需要一些方法对需求的优先级进行分析，这样更便于就优先级问题

达成共识。正是这样一些困难，我们需要对这个问题仔细进行考虑。

1，不同角色的人处理优先级

优先级设定的困难来自于两个方面：

- **客户方面**：客户的心理往往是：“我需要所有的特性，只要以某种方式使它发生即可。”如果用户知道低优先级需求可能不会实现，那么就很难说服用户设定需求的优先级。
- **开发者方面**：开发者的概念是，既然需求被写入软件需求规格说明，那么就应该不遗余力地去实现这些需求。开发者更喜欢避开优先级设定的原因，在于他们觉得建立优先级与他们要向经理表示的“我们可以全部完成产品”的态度相冲突。

但是在现实中，一些特性确实比其它特性更重要，把握需求的重点对项目的成功极其有意义。例如：在项目接近尾声时，开发者可能会需要抛弃掉一些不必要的功能以保证按时完工，在最后阶段匆忙决定可能会使项目陷入灾难，而在项目的早期阶段设定优先级将有助于我们在项目过程中逐步作出相互协调的决策。如果在项目的后半期我们感觉需要考虑哪些是低优先级需求，但这时我们已经实现了将近一半的特性，那这将是一种浪费。正是上述的背景告诉我们，需要以一种专业的眼光来审视优先级设定的方方面面。

2，双因素优先级设定方法

当产品负责人提出“要根据业务价值确定优先级”的伟大建议的时候，业务价值到底是什么？因此需要有更确切的指导原则，我们可以考虑最重要的 2 个因素：

- 获得这些功能所带来的经济价值。
- 开发这些功能所减少的风险。

由于多数项目都考虑经济价值并且尽可能必考风险，所以这两个因素往往比较受到重视。

1）价值

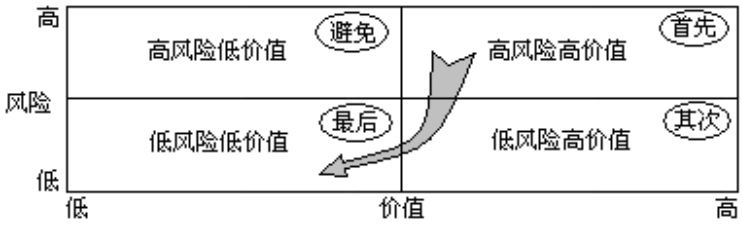
确定优先级的第一个因素是主题的经济价值，获得了某个主题的功能，可以让客户获得或者节省多少钱？这本身就具有“要根据业务价值确定优先级”的含义。一般来说，价值越高优先级也越高。确定某个主题价值的理想方法，是估计它在一段时间内（几个月或者几年）所带来的经济影响。问题在于确定经济回报是很困难的事情，这种估计难度颇高。因此常常用一些替代方式对价值进行估计，由于用户满意度常常与价值有关，所以常用满意度等非经济度量来表示价值。

2）风险

几乎所有的项目都蕴藏着大量风险。风险指的是目前尚未发生但是可能发生，而且会妨碍或者限制项目成功的事情。在项目中不同的风险包括：进度风险、成本风险以及功能风险。此外，风险还可以分为技术风险和商业风险。早期缓解高风险是个良好的做法，也就是说风险越高，优先级也越高。

3）综合排序

在按某个单独因素排序之后，我们还需要综合考虑。例如：下面关于风险与价值关系的 4 象限图。正确的方法是要“避免”高风险低价值的功能，但这个避免不是绝对的，今天被认为是要避免的，6 个月后随着开发的进展，或者风险降低，或者价值提高，这个功能可能会处于首先要处理的位置。开发的优先级是从高风险高价值开始，沿着下图所示的曲线安排比较合理。



上面的优先级评估方法还是比较定性的，更精确的方法需要某些定量研究。我们发现，确定优先级牵涉到不同的因素，而这些因素常常互相冲突，利益相关方也可能会有不同的目的，达成一致往往比较困难。因此，我们可以通过一种矩阵方法来综合考虑各种因素，这种矩阵方法也称之为结构化方法。通过这种结构化方法可以消除一些情感、政策以及处理过程中的推测，使考虑问题比较全面。

3，多因素矩阵排序方法

如果需要在优先级分析中考虑更多的因素名就需要使用矩阵方法。比较常用的是来自于质量功能开发（QFD）关于客户价值的概念。QFD 认为客户价值取决于两个方面：

- 一方面，如果实现了特定的产品特性，那么将为客户提供利益；
- 另一方面，如果不能实现产品特性，就要受到损失。

产品特性的诱人之处在于它所提供的价值成正比，而与实现该特性时的费用和技术风险成反比。一切都是平等的，只有那些具有最高的价值/费用比率的特性才应当具有最高的优先级。这个方法还有一个特点，也就是首先必须把产品的核心功能分离出来，下面几种功能必须首先实现：

- 产品核心业务功能和特性；
- 被视为产品的独特之处的功能；
- 为符合政府规定所要求的特性。

一旦分清对于产品交付必不可少的特性，就可以对其它的需求采用模型来确定其相对优先级。换句话说，我们只能把这个设定优先级的矩阵应用于非最高优先级的可变动的特性上，必须在连续的区间上分配估计的优先级，而不只是把它们分成几个不同的优先级层次。下面的例子描述了“仓库货品跟踪系统”的优先级评估。

“仓库货品跟踪系统”优先级设计的矩阵范例									
相对权值：	**2**	**1**		**1**		**0.5**			
用例	**相对利润**	**相对损失**	**总价值**	**价值%**	**相对费用**	**费用%**	**相对风险**	**风险%**	**优先级**
1．查询供应商订单的状态	5	3	13	8.4	2	4.8	1	3.0	1.345
2．建立仓库存货清单报表	9	7	25	16.2	5	11.9	3	9.1	0.987
3．查看一个特定货品的历史记录	5	5	15	9.7	3	7.1	2	6.1	0.957
4．打印货品进出数据表	2	1	5	3.2	1	2.4	1	3.0	0.833
5．维护常用货品列表	4	9	17	11.0	4	9.5	4	12.1	0.708
6．更改预定货品请求	4	3	11	7.1	3	7.1	2	6.1	0.702
7．建立柜台存货清单报表	6	2	14	9.1	4	9.5	3	9.1	0.646
8．在供应商目录中查询一个特定的货品	9	8	26	16.9	7	16.7	8	24.2	0.586
9．在数据库中查询一种常用货品记录	3	4	10	6.5	4	9.5	2	6.1	0.517
10．从结构绘图工具导入货品的结构	7	4	18	11.7	9	21.4	7	21.2	0.365
总计	54	46	154	100	42	100	33	100	- - - -

在设定优先级的过程中典型的参与者有：

- 项目经理、他指导全过程，解决冲突，并且在必要的时候调整其它参与者的方案。
- 重要的客户代表，例如产品的代表者，他可以提供受益和损失程度。
- 开发者代表，例如开发组的技术指导者，他提供了费用和风险程度。

这个矩阵的使用方法如下：

1）列出所有需求

在矩阵中列出需要设定优先级的所有需求、特性或用例；所有项都必须在同一抽象级别上；不要把个人需求与产品特性混合在一起。

如果某些特性具有逻辑上的联系，例如只有包括特性 A 的情况下才能实现特性 B，那么在分析中只要列出驱动特性就可以了。这种模型在其有效范围内可以容纳几十种特性。如果有更多的项，那么就把相关的特性归成一类，并建立一个可管理的初始化列表。如果需要的话，也可以在更详细的级别上进行第二轮分析。

2）估计客户的相关利益

为了估计每个特性提供给客户或业务的相关利益，可以用 1～9 划分等级，1 代表可忽略的利益，9 代表最大的价值。这些利益等级表明了与客户需求的一致性，客户代表是判断这些利益的最佳人选。

3）估计损失

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

4）计算相对利润和相对损失

总价值栏是相对利润和相对损失的总和。在缺省情况下，利润和损失的权值是相等的。

作为一种精化，我们可以更改这两个因素的相对权值。在上表的例子中，所有利润估价的权值都是损失估价权值的两倍。在矩阵算出了特性价值的总和并计算出每个特性价值占总价值的百分比（价值百分比栏）。

5）估计每个需求的相对费用

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

6）估计风险

下一步，我们必须要估计出与每个特性相关的技术或风险相对程度，并利用 1～9 划分等级。1 级表示可以轻而易举地实现编程，而 9 级表示需要极大地关注其可行性、缺乏具有专门知识的人员，或者使用不成熟或不熟悉的工具和技术。在矩阵中计算出每个特性所产生的风险百分比。在缺省情况下，利润损失，费用和风险的权值是相等的，但是也可以在矩阵中调整其权值。在表中，风险权值是费用权值的一半，而费用权值与损失权值相等。

如果无需在模型中考虑风险，就可以把风险的权值设为 0。

7）计算优先级

一旦把所有的估算写入矩阵，就可以利用如下公式计算出每一特性的优先级：

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

8）排序优先级

最后，我们可以按照计算出的优先级用降序排列表中的特性。处于列表最顶端的特性是价值、费用和风险之间的最佳平衡，因此必须具有最高优先级。

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

这个模型有助于我们在评估需求的时候作出合理的决策。把需求优先级的设定由妄加评论转向以客观和分析为基础的评估过程，在项目中我们所采取的措施将可以提高以最合理的序列实现

最重要功能的能力。

小结：

本章我们详细的讨论了从已经构建的产品模型中，抽取出功能需求与非功能需求的方法，更深一步，我们还对用户需要的重点能力进行了研究。但是，所有这些内容需要整理出一份完整的软件规格说明书。

软件规格说明书是未来现实世界与机器世界的接口，也是作为要构建的产品的一份合约，也是需求分析阶段最重要的产品。为了让开发者能够构造出正确的产品，使需求分析师与利益相关方的误解减少到最低程度，下功夫编写软件规格说明书是完全必要的。

第六章 总结归纳：编写需求规格说明

现在到了出成果的时候。我们必须以文档的形式编写完整的软件需求规格说明（Software Requirements Specification，SRS）。这个规格说明将作为开发、测试以及客户验收的依据。要注意到，需求的编写需要思想，仅仅把感兴趣的内容拼一拼并不能成为好的需求，关键是要考虑：我到底是想表达什么？

6.1 需求规格说明书模板

需求分析师需要一份编写需求的指南，这样就会更容易，更方便，这就需要一个模版。需求规格说明书模板就像一个容纳需求的容器，通过检查需求文档，把需求分成类型，这有利于我们识别和提取需求。每个类型在模板中占据一部分，它一共包含了 5 个大的部分内容。

在研究前面章节的时候，如果仔细浏览这个模版，就会发现关于编写需求的许多内容，以及收集各种需求的一些思考方向，典型的需求规格说明模版如下：

项目驱动：描述项目的理由与动机。

1，项目的目标：投资创建产品的理由以及希望取得的业务上的好处。

2，客户、顾客和其它利益相关方：产品涉及他们的利益以及对他们的影响。

3，产品的用户：预期的最终用户，以及他们对产品可用性的影响。

项目限制条件：加在项目和产品上的约束条件。

4，需求限制条件：项目的局限性和产品设计的约束条件。

5，命名标准和定义：项目的词汇表。

6，相关事实和假定：对产品产生一定影响的外部因素。

功能性需求：产品的功能。

7，工作的范围：针对的业务领域。

8，产品的范围：定义预期产生的边界。

9，功能与数据需求：产品必需作的事情以及所操作的数据。

非功能性需求：产品的品质。

10，观感需求：预期的外观。

11，易用性和人性化需求：产品让预期用户使用应该是什么样子的。

12，执行需求：速度、大小、精度、人身安全、可靠性、健壮性、可伸缩性、持久性和容量等需求。

13，操作和环境需求：产品预期的操作环境。

14，可维护性和支持需求：产品的可改动性必须达到什么水平，以及需要什么样的支持。

15，安全性需求：产品的信息安全、保密性和完整性。

16，文化与政策需求：人和社会因素。

17，法律需求：满足适用的法律。

项目问题：这些是适用于构建产品的项目。

18，开放式问题：那些尚未解决的问题，可能对项目的成功有影响。

19，立即可用的解决方案：利用已有的组件，而不是重新开发。

20，新问题：引入新产品而带来的问题。

21，任务：把产品投入使用必须要做的事情。

22，迁移：从现有系统转换的任务。

23，风险：项目最有可能面对的风险。

24，费用：早期对构建产品的成本或者工作量的估计。

25，用户文档：创建用户指南或者文档的计划。

26，后续版本需求：可能在产品将来发行版本中包括的需求。

27，解决方案的想法：我们不想错失的设计想法。

下面我们对书写规格说明书的各个方面，比如需要注意的问题、思考问题的重点、一些容易混淆的问题等加以说明。

6.2 项目驱动与问题描述

项目的目标和问题描述是需求文档的第一部分，这是对产品整体方向和价值的描述，必须认真对待。

一、项目目标

这部分描述的是客户希望构造一个新产品的原因，也就是说描述客户所面对的业务问题，并解释产品将如何解决问题。

1，用户的业务或项目工作的背景

这里需要描述用户所作的工作，以及为什么他没有像期望的那样顺利进行，一定存在某些重要的改进机会，否则不会要求开发一个新产品。这段描述可能非常简要，比如：

道路管理部门对由于结冰引发的道路事故数量不满。

也可能完整描述操作上的问题，常常对严重问题作了最多的陈述，以及目标产品要解决的主要问题，比如：

道路冬天结冰引发交通事故，我们需要预测和市道路可能结冰，然后我们的车库可以及时调度除冰卡车来防止道路结冰。除了气象预报以外，我们还希望使用地区的热象图以及设置在公路上的气象站发出的道路温度数据。我们希望新的系统能够发出更精确的冰清预报，使我们比现在更及时地进行除冰，从而减少交通事故。我们也希望能不加选择的化学除冰，因为那样会浪费除冰化合物，同时也污染环境。

也可能并没有什么问题，但是有一个重要的商业机会，客户希望抓住它，在这种情况下，可以对该商业机会进行描述，比如：

道路除冰情况对许多国家都是一个问题，目前没有准确的预报办法，道路除冰的处理方式极不科学，极为随意。如果我们能开发一个准确冰情预报和处理调度的产品，将会有很大的市场。我们的市场部门已经找到了 25 个国家愿意立即购买这样的产品。

或者，产品的目标是做一些探索或者可行性调查，在这样的情况下，项目的提交产物不是新产品，而是一份文档，指出一项产品的需求可能或者不可能满足，比如：

冰情预报和卡车调度过去一直是有不同的实体完成，如果把冰情预报和卡车调度结合在一起，我们预计调度工作会有效的多。通过共享策略和优化策略，这两项活动都会受益。我们希望调查是否存在为我们组织开发一个新产品的机会。

2，项目目标

这部分描述我们想做的事情，或者产品将对工作的总体目标做什么贡献。不要让这部分篇幅过长（对产品目标作简要解释，比作长而且离题的论述要有价值）。一个短的、明显的目标，会使利益相关方更清楚，因此也就更可能在目标上取得一致意见。比如：

通过精确预报结冰道路的除冰工作来减少交通事故。

如果目标是有价值的，就需要表达它所提供的利益，一般说有 3 种利益：

- 在市场上的价值。
- 减少运作成本。
- 增加客户价值或服务。

好处必须是可度量的，否则就不能认为是一个有价值的目标，下面的写法只能认为是一个胡言乱语：

需要一个分布式的、以质量为中心的、委托式的群体协调机制和分阶段的、动态的联盟监视解决方案。

一般来说，建议是用我们已经讨论过的 PAM（目标、好处、度量标准）来描述，例如：

目标：精确预报道路结冰时间并分排除冰卡车。

好处：通过预报道路结冰情况来减少道路事故。

度量标准：因结冰而发生的事故数将低于冬季发生的事故总数的 15%。

目标：在冬季道路养护支出上节省费用。

好处：减少除冰道路养护的费用。

度量标准：除冰费用将在目前道路养护费用的基础上降低 25%，冰对道路的损伤将降低 50%。

这些度量是可测试的，它提供了一种指导，帮助项目团队把注意力集中在对目标有最大贡献的需求上。项目目标是一个整体目标，而不是每个需求的罗列。目标是在项目启动阶段确定的，必须保证目标是合理的，可以达到的，表述简单的，有价值的，并且有度量目标，就可以对提交的产品进行测试，以确保它满足了目标。

如果目标没有这些属性，那么历史已经告诉了我们，该项目不大可能提交任何有用的东西。

二、客户、顾客和其它利益相关方

这部分描述利益相关方，利益相关方包括客户、用户、业务或需求分析员（负责收集客户需求并编写文档，以及负责客户与开发机构之间联系沟通的人）、开发人员、测试人员、用户文档编写者、项目管理者和客户管理者。

也就是说，利益相关方是在产品中有利益关系的人，画一些时间来准确确定并且描述这些人是值得的，因为不知道他们是谁，所带来的惩罚将会很严厉。

每个利益相关方都对产品有要求，他们都是需求的来源，如果没有找到所有的利益相关方，就有可能不能找到所有的需求。只要有可能，就给出利益相关方的名字和联系方式。

三、产品的用户

产品用户是直接与产品交互的人，他们尽管是一种特殊类型的利益相关方，但由于他们在需求收集工作中的特殊重要性，我们在模板中引入单独的一节来描述。

在网罗活动中确定了产品边界以后，用户是谁就清楚了，在需求规格说明的这一部分，需要说明他们的特征，他们的习惯和思维方式，对产品最后的表现有很大的影响。

功能需求源自于研究用户的工作，所以重要的是正确的确定用户当前实际在做的工作。用户故事得越好，就越容易确定易用性需求（易用性对不同背景的人是不一样的）。记得要包括用户的任何不寻常的特征，他们是在发怒？还是在赶时间？在行走中？还是只用一只手（比如侍者）。

规格说明书的这一部分详细记录对用户的了解，在需求过程中利用这部分的工作来确定与这些特定用户相关的需求。

6.3 产品限制条件的确定

产品限制条件放在第二部分的原因，是因为这约束了所有问题的展开，也是至关重要的部分。

1，强制的限制条件

限制条件是全局的需求，它是适用于整个产品的一些因素。产品必须在列出的限制条件下构建，有些限制条件是管理层确定的，这就需要认真考虑，它们限制了需求分析师可以做的事情，为产品定了型。

限制条件包括以下几个方面：

1）解决方案限制条件

这是一些设计决定，例如产品必须使用某种语言，必须运行在某种计算机上，或者必须与原来的产品兼容。另外，如果使用某些商业或者开源的软件产品，它们必须正确的交互。

尽管我们不希望分析阶段包含任何设计，但忽略这些组织上的指令是不现实的，从组织的观点看也是不可接受的。

我们必须提醒检查所有的限制条件的动机，研究强调某种技术是带来了真实的好处吗？这类解决方案有真实的业务需要吗？是否仅仅为了描述业务，却给出了一些常见的解决方案呢？

2）当前系统的实现环境

有时候，限制条件反映出管理层不愿意购买一些新的硬件。比如，产品必须运行在现有的服务器上，因此，需求就不能给出一个需要不一样的计算能力才能实现的产品。

3）伙伴应用或者协作应用

几乎所有的软件项目都会不同程度的于其它产品交互，这些产品可能在组织内部，也可能在组织外部，本节告诉开发者，这些需要交互的产品是什么？并解释为什么他们被看作当前产品的伙伴应用。

4）立即可用的软件

指定采购的软件，有些产品提供的功能几乎可以满足某个领域的一切要求，通常利用这样的产品是有意义的。这里必须描述打算采用哪些采购的软件，可能还会指定相应的接口和特殊功能，不过，我们发现把这些功能和其它功能需求放在一起可能会更清楚一些。

5）预期工作场地环境

描述工作场地，不过只有工作场地对产品需求有影响的时候才包含这部分内容。

6）时间进度限制条件

指定构建产品允许的时间，这个时间会约束项目团队需求收集的时间，让需求的内容允许产品在规定的时间内构建出来。

7）预算限制条件

本节描述项目可用的资源（资金、人员、工作量）。目的是约束人们万丈的雄心，防止项目团队用只够一个小的项目资金预算，去收集一个庞大项目的需求。

2，命名惯例和定义

建立命名惯例的字典，说明一些关键名词的含义。在需求规格说明中使用的名称应该是通常的业务名称，避免名称上的二义性甚至发生误导的名称。

3，相关事实和假定

相关事实是一些外部因素，它们对产品产生影响，但不属于规格说明书的其它部分。他们不一定会转化为需求，但提醒开发者的一些需求要考虑的情况和约束。

但是，软件开发有时候需要一些假定，这些假定很可能是项目团队做出来的，目的是提醒管理层注意那些对产品有影响的因素。这些假定往往牵涉到产品的目标操作环境，也可能是其它的事情，但是如果假定没有成为事实，将对产品的成功产生有害的影响。

通过指明这些假定，分析师需要向所有的利益相关方说明这些假定，判断这些假定是不是有事实依据，例如假定：处理过的道路至少在 2 小时内不需要重新处理。如果这个假定与事实不符，需要及早纠正。

6.4 功能性和非功能性需求的描述

产品功能性与非功能性需求是软件规格说明的主体，也是篇幅最大的部分。在很多情况下这一部分是多人共同书写，很容易造成前后文和上下文的不一致。这一部分的书写决定了规格说明的质量，是最需要下功夫的部分。

一、工作的范围

首先要建立的范围是工作范围，它确定了要研究的业务领域的边界，并概要性的描述它与环境的关系。当理解了工作以及它的限制条件的时候，就可以建立起产品的范围。

我们可以使用一个上下文模型来描述工作范围以及围绕工作的相邻系统。建立正确的工作范围对于产品的成功很关键，值得投入精力和时间来确保这个上下文范围对于理解客户的业务是足够的，客户的业务将从产品中受益，如果不这么做，得到满意产品的机会就降低了。

工作的范围往往比较大，需要对工作进行划分，这种划分可以通过事件列表完成，每个业务事件应该包括：

- 事件编号；
- 事件名称；
- 输入或者触发数据流；
- 输出数据流；
- 对事件响应/业务用例的一句话总结。

如果已经构建了对业务用例的响应模型，可以把模型放在规格说明的这一部分，可以用附录的形式，附上适当的事件编号。

二、产品的范围

1）产品边界

可以用多种形式描述产品范围，但是一般模型比文字更容易理解。在上下文图中，中间部分是产品，相邻系统是参与者。

2）产品用例清单

如果存在大量用例，那么产品用例清单比直接在文档上写出用例要好管理些。

三、功能性需求和数据需求

1）需求收集卡片

我们前面提到过的需求收集卡片式是非常有意义的，一次性的找出一项需求的所有部分以后再找下一项的做法是不切合实际的，所以建议做一些小卡片，把需求框架打印在卡片上，在研究问题的时候，发现一项就快速的记录下来，随着对需求理解的加深，逐步地完成这些卡片。这种做法给我们带来很大的灵活性，是我们在利益相关方那里收集需求的时候不至于带来很大的障碍。我们可以根据它所属的用例进行分组，“事件/用例”编号便于在桌面上查找某张卡片。

需求编号：	**需求类型：**	**事件/用例编号：**
描述：		
理由：		
来源：		

验收标准：		
顾客满意度	**顾客不满意度**	**冲突**
优先级：		
支持材料：		
历史：		

2）原子需求

一个单项、完整、形式化的需求应该有如下的构成，由于有关问题前面谈得很多，这里就不再详细叙述了：

1）需求编号；

2）需求类型；

3）事件/用例编号；

4）描述；

5）理由；

6）来源：首次提出这个需求的人，或者需求可以归因于他，这样需求就有了参考点。

7）验收标准；

8）顾客满意度和不满意度；

9）优先级；

10）冲突；

11）支持材料：有些材料对需求来说是重要的，可以简单的在此处引用。

12）历史：这里记录需求首次提出的时间、更改日期、删除日期、通过质量关的日期。如果有帮助，也可以加上负责人的名字。

四、非功能性需求

对于非功能性需求我们已经作了很详细的讨论，这里就不再重复阐述，主要的是非功能需求需要仔细研讨可度量的标准，因为非功能性需求大多属与质量需求有关，而且还牵涉到验收标准，必要的时候，需要聘请相关技术人员参与研究。

6.5 阐述项目问题

有一些问题可能会对项目产生影响，需要提醒利益相关方，就要在这一部分阐述。

1，开放式问题

这一部分主要描述已经发现，但还没有解决的问题。例如，当需求获取的时候，一些问题浮出水面，但不能立刻有答案。又比如利益相关方可能不能确定有些事情将来会如何发展。例如：

气象中心的数据库可行性研究还没有完成，这影响到气象数据的处理。

要知道变化时刻都有可能发生，这些变化中的一部分肯定会影响到产品，开发式问题包括对预期变化点的预测，尽管结果不能确定，但这些预期对设计的影响很大。

2，立即可用的解决方案

有时候会有一些现成的解决方案，但需要提醒客户构建产品与购买组件或使用开源代码的优劣分析，需要讨论 3 个方面的问题：

- 是否有现成的产品，是否可以购买商业软件？
- 现有的组件是否可以用于该产品？
- 是否有一些我们可以复制的东西？

针对每个问题，找出您觉得合适的替代方案，或者给出外购组件的原因。在这一部分，指出可能的费用、可得性、实现的时间以及其它可能影响决定的因素是有意义的。

这里主要是提出将来的可选方式，而不是限制条件。

3，新问题

改变原有的秩序可能会带来相反的结果，因此我们需要检查并指出安装新系统会带来的任何问题。例如，可能完成工作的方式发生改变，与现有系统的接口发生了改变，人们目前的工作习惯发生了改变等。

我们并不需要描述一切方面，但也需要提醒客户他们可能面对的问题和机会。

4，任务

为了交付产品，必须采用哪些步骤？本部分的意图是突出构建产品所需要的工作量，或者购买一个解决方案需要的步骤。不论计划采取何种途径把产品安装到用户的业务领域中去，都需要在需求阶段知道途径并且描述它。尽管组织有构建产品或者购买产品的标准方法，还是存在由产品需求导致的一些变化。

管理层会利用这部分提供的信息来评估产品的可行性，考虑它所需的工作量和费用。

6.6 需求文档编写的若干建议

需求文档是在前面所有工作的基础上编写的，如果没有这个基础和分析的结论，所编写的文档尽管格式良好但也一定的没有用处的。再一次提醒，文字是表达思想的，如果思想是匮乏的，那么再漂亮的格式都是没有用的。

一、产品需求规格说明书参考模板

这里提供的产品规格说明书模板仅供参考。再重复一遍：好的文档在于思想深邃、善于表达、重点突出、逻辑清楚，而不仅仅是格式正确。在描述的时候要图文并茂，用图（例如 UML）表达关系，用文字（例如场景）表达细节，问题之间的线索要清晰，层次要分明。这样书写出来的文档才可能真正发挥作用。

{ 项目名称 }
产品需求规格说明书

文件状态： ［√］草稿 ［ ］正式发布 ［ ］正在修改	文件标识：	Company-Project-RD-PRS
	当前版本：	X.Y
	作　　者：	
	完成日期：	Year-Month-Day

版 本 历 史

版本/状态	作者	参与者	起止日期	备注

0. 文档介绍

0.1 文档目的

0.2 文档范围

0.3 读者对象

0.4 参考文档

提示：*列出本文档的所有参考文献（可以是非正式出版物），格式如下：*
[标识符] 作者，文献名称，出版单位（或归属单位），日期

0.5 术语与缩写解释

缩写、术语	解 释
…	

1. 产品介绍

提示：
（1）说明产品是什么，什么用途。
（2）介绍产品的开发背景。

2. 产品面向的用户群体

提示：

（1）描述本产品面向的用户（客户、最终用户）的特征，

（2）说明本产品将给他们带来什么好处？他们选择本产品的可能性有多大？

3. 产品应当遵循的标准或规范

提示：*阐述本产品应当遵循什么标准、规范或业务规则（Business Rules），违反标准、规范或业务规则的产品通常不太可能被接受。*

4. 产品范围

提示：*阐述本产品“适用的领域”和“不适用的领域”，本产品“应当包含的内容”和“不包含的内容”。说清楚产品范围的好处是：（1）有助于判断什么是需求，什么不是需求；（2）可以将开发精力集中在产品范围之内，少干吃力不讨好的事情；（3）有助于控制需求的变更。*

5. 产品中的角色

提示：*阐述本产品的各种角色及其职责。各种角色的具体行为将在功能性需求中描述。*

角色名称	职责描述

、

6. 产品的功能性需求

6.0 功能性需求分类

提示：*将功能性需求先粗分再细分，下表中的 Feature A, Function A.1 等符号应当被替换成有含义的名称。*

功能类别	子功能
Feature A	Function A.1
	Function A.2
	…
Feature B	Function B.1
	Function B.2
	…
…	

6.m Feature M

提示：*此处写一些承上启下的文字。*

6.m.n Function M.N

名称、标识符	

功能描述	
优先级	
输入	
操作序列	
输出	
补充说明	

提示：*此处也可以描绘每一个用例的场景，以用例图表达参与者各个用例以及用例之间的关系，对应每个用例，可以书写相应的场景，包括：名称、主要参与者、简述、前置条件、后置条件、主事件流、扩展流、补充说明等。在必要的时候，也可以描述各个概念对象之间的时序关系。不论是书写传统的功能操作序列，还是书写用例场景，所表达的信息是基本一致的，可以根据情况选用。*

……

7. 产品的非功能性需求

7.1 用户界面需求

需求名称	详细要求
…	

7.2 软硬件环境需求

需求名称	详细要求
…	

7.3 产品质量需求

主要质量属性	详细要求
正确性	
健壮性	
可靠性	
性能，效率	
易用性	
清晰性	
安全性	
可扩展性	
兼容性	
可移植性	
…	

7.n 其他需求

附录 A：需求建模与分析报告

提示：*建议用 UML 对产品需求进行建模与分析，也可以使用 DFD 进行数据流的过程分析，这要根据具体情况选用。*

A.1 需求模型 1

A.n 需求模型 N

附录 B：需求确认

提示：*需求确认规程主要分两步：（1）需求评审，（2）需求承诺。对需求的评审应当采用“正式技术评审方式”，将产生一份“需求评审报告”，规程请参见 SPP-PROC-TR。在获取责任人（Stakeholders）对需求的承诺之前，该《产品需求规格说明书》必须先通过需求评审。*

需求评审报告摘要	
需求文档	*输入名称，标识符，版本，作者，完成日期，...*
需求评审报告	*输入名称，标识符，评审日期，...*
评审结论	[ ] 工作成果合格，“无需修改”或者“需要轻微修改但不必再审核”。 [√] 工作成果基本合格，需要作少量的修改，之后通过审核即可。 [ ] 工作成果不合格，需要作比较大的修改，之后必须重新对其评审。
评审意见	
评审小组成员	*输入评审小组成员*

需求承诺	
需求文档	*输入名称，标识符，版本，作者，完成日期*
客户承诺	*承诺...* 签字，日期
项目经理承诺	*承诺...* 签字，日期

二、善于书写良好的文档

1，书写良好的文档

很常见的情况是，开发人员对于书写文档有种种非议，但是一个正规的软件项目，即使带来了表面上的工作效率降低，也必须花费精力书写文档，为什么呢？

任何时候，只要是两个人以上参与项目，都离不开口头沟通，但是每个人在传递这个口头沟

通的内容的时候，都会把原来的意思歪曲一点点，人类的记忆能力就是这样的。一个软件项目永远离不开口头沟通，我们不必要去停止这种沟通方式，但是当开发人员比较多的时候，就可能带来很多麻烦，某些情况需要大家都知道，需要其他人员与之交互等等，解决的办法就是把所有的情况记录下来，或者是要强调书写文档。利用文档记录有五个好处：

1）拓展人脑所掌握的记忆范围

在任何一个大到必须编写需求文档的项目中，信息的含量都会超过一个人可以掌握的规模，书面文字可以供将来参考，而不会像记忆一样慢慢的褪去。

2）为项目团队所有成员提供相同的信息

书面文档阅读的时候内容是相同的，所有的人员都阅读相同的材料，这是任何人对人的口头沟通无法达到的效果。

3）减少项目人员流动的困难

项目中发生人事变动是正常的，当老成员离开，新成员加入的时候，任何口头沟通都很难让人很快地掌握情况，一份很好的文档可以让他花更少的时间融入到团队中去。

4）保护智力资产

在大多数情况下，项目中只有少数的人理解问题域，他们知道是否需要变更，软件是否存在某些漏洞，内心中对软件是不是有什么新观点。如果这些人把他们掌握的信息以正规的方式记录下来，项目对他们的依赖就会减小，如果他们获得了更好的机会，他们的智力资产也不会被带走。

5）帮助书写人员更好的理解问题域

以书面形式描述需求和规格说明，将促使相关人员花精力以比口头沟通更标准和更准确的方式表达。人们在写作过程中可以发现自己对需求理解存在漏洞，甚至概念上也是不完整的。难怪人们说：“文档本身并不重要，重要的是写文档的过程。”当然，这样写出来的文档对别人也是重要的。

从长远的角度看，需求的整体文档都是重要的，而从短期来看，规格说明书是最重要的。规格说明书很明确的告诉程序人员和测试人员我们需要开发什么测试什么。如果没有这些背景知识，他们就没有办法权衡技术实现方法，也没有办法提出建设性的意见和建议。

很多公司实际上是很严肃地对待文档的，但他们并没有从文档中得到好处，相反在实践中口头沟通仍然是主要手段，其关键原因是所书写的文档并不是为了开发而是为了应付检查。从格式上看很漂亮，内容面面俱到，但在文档组织上把信息分散到各个地方，不容易查找，使文档提供的好处尽失。如果我们希望项目开发小组很好的使用文档中的信息，就需要考虑什么内容是开发小组最关注的，怎么组织文档才是对开发来说最容易查找，如何编写才是最容易阅读和有效的。

2，走捷径及其风险

确实在很多情况下可能需要走捷径，例如时间紧迫，由于需求分析占用了大量的时间而使编码发生延误，最终很可能影响整体质量。当然需求不足也会影响整体质量（bug 增多），很多情况下这两者是需要权衡的，作为项目负责人就需要对走捷径的利弊进行分析和权衡，下面我们对常见的捷径与风险做一些讨论。

1）省略问题域

捷径：在需求文档中，省略掉对问题域的描述，也就是说只写需求陈述，这将使需求文档从篇幅上大大缩短，恐怕三页纸敲定需求文档也不是没可能。

风险：当需求人员离开自己目前的岗位，恐怕就没有人理解客户为什么需要这些能力，也没有人能够理解客户的业务，对应用程序的维护将变得很困难，对项目的修改和优化也将会变得极其被动。

不考虑问题域的描述，还会导致先入为主的解决方案。通过理解问题域，评审人员、设计人员可能会提出一些很好的建议，有时还可能提出新的需求，这些建议也可能使需求人员从没有想

到的。如果没有对问题域的理解，那设计人员唯一能做的就是被动执行。

另外一个风险在于，如果不是把需求与具体的业务背景关联起来，需求陈述本身很可能被错误的理解。

2）省略需求分析文档

捷径：把整个需求分析文档都省略掉，只是提供一个需求规格说明，描述软件与外部世界接口的行为。

风险：没有明确的需求描述，只是依靠与外部世界接口的需求规格说明。设计人员就可以天马行空的设计他们认为够酷的功能了，只要符合规格说明描述的接口规定就行。这些功能对用户来说可能是毫无意义的，也可能省略掉对用户来说非常看重的功能，原因很简单：因为这种功能开发起来没意思，不够酷，不能反映水平。

3）省略用户接口文档

捷径：用户接口文档描述的是用户界面（包括屏幕和其它的人机交互方式），很多分析人员往往省略这种文档，而让程序员编写程序边设计用户接口。

风险：用户接口是直接与用户打交道的，人类的天性就是这样的：程序员倾向于忽略单调乏味的、实现起来费时费力又不能表现水平的用户接口思想。但是细心的用户接口人员会花很多时间细心的体会用户的感受，会在几乎不被人注意的地方用某种技巧性的方式使产品易于使用。在用户接口文档中，会包括很多程序员不愿意考虑的、充满想象力的、不同寻常的特性，这些特性会大大提高产品的价值。

4）省略规格说明与需求分析

捷径：这是最极端的情况，但我确实见过这种捷径。也就是说在软件开发过程中，不去书写任何需求文档，让程序员一边做需求分析、一边做程序设计。只是通过传统的口头方式与用户方沟通需求，问清楚一部分实现一部分，因此，既使产品已经制造出来了，对需求还只是一个比较模糊的印象。

风险：除了上面已经说到的风险外，这种捷径必然带来大量的 bug。这是因为，缺乏规格说明，测试人员无法建立测试计划，如果你总是在走这条捷径，恐怕你就不需要任何测试人员了。我见过的类似项目，项目组就没有测试人员，这样的项目居然也投运了。

用这样的方式开发项目，是不可能实现优化的，因为所有的事情都是以一种模模糊糊的方式进行的，这样的项目是不可能获得任何短期利益的（除非用户对产品质量没有任何要求），混乱、bug、不必要的重新编写，花费更多的人力去交付第一个版本。

需要注意的是：写一大堆任何人都没有办法理解或者阅读的文档，与不写任何文档从效果上是一样的，只是花费了更多的成本而已。

3，文档编写的建议

1）防止编写成智力拼图

许多需求文档写成了类似智力拼图的玩具，很多不同的相关定义与内容散落在文档的不同地方。读者不会预料到在文档的其它地方会不会还有相关说明。为了阅读这样的文档，人们需要把一切都记在脑子里，智力拼图是把散落在各处的小块拼在一起，但人的大脑很难做到这一点。

要记住，任何大的文档都不同程度的具有智力拼图性质，但我们的任务是减少智力拼图块的数量。不要让用户在一个地方看到的信息块做出了推论，而在另一个地方才发现这个理解是错误的，这就很危险。

组织文档的原则，大多数情况下是防止出现智力拼图的情况，如果需要，可以把文档中对某个名称的定义和描述都找出来，统一放在某个固定的、唯一的位置上，必要的时候还可以建立术语表。如果无法将相关信息都集中在一个章节，就需要增加引用页面，这样读者就不会迷惑了。

我们这个课程讲义的编写本身就是一个需求过程，除了定义清楚问题域以外，整个课程的思

考方式是基于问题、研究对策、寻找解决方案，而不是依据于某个人说过什么，某个标准是什么（何况标准也是在变化的）。在书写方式上，以读者为中心，力求通俗易懂、内容连贯、条理清楚。在内容组织上，力图减少不必要的拼图游戏，减少不必要的前后查阅。相似的概念和名词，只要在各个章节中，都力图自成逻辑体系，使得读起来省力清晰。这些特征，在需求文档的编写中都是需要注意的。

2）防止编写成鸭叫需求

有些需求编写出来为什么开发人员不愿意看呢？为什么似乎已经描述的很清楚的需求，在读者感觉模糊不清难以阅读呢？我们来看一下下面的需求描述：

R-461 机票预定数据验证函数应该验证机票预定数据。

这样的句子确实使人莫名其妙，这到底是什么意思？如何测试它？这很容易使人联想到一种鸭叫演讲，单调、温顺、平庸、符合官方标准，但是叫人提不起精神，看起来描述了很多，但实际上什么也没有描述。

把自己写的句子大声念出来，如果发现自己写了很多毫无疑义的句子，只是一味的迎合标准要求，就要考虑重新框定问题，使文档写出来具有精神，读者读起来也有精神。

3）不要为了文档而文档

软件质量保证是每个开发组织都很关注的，为了保持一致的软件质量，我们需要一套独立定义的质量准则，所以，过程的定义与文档规范就成为很多组织很关注的内容。但也会发生文档的使用与功能正确实现无关的情况，由于文档记录要与文档标准保持一致，结果使用起来将非常繁琐而散乱。

很多需求文档都没有被开发人员阅读，其原因是人们无法理解他，不适合开发人员的习惯，也没有办法把文档与自己的开发工作联系起来。这种书写文档的理念只是为了应付检查，而不是促进效率与更好的开发，是为了文档的本身，是为了文档而文档的书写方式。书写这种文档的目的只有一个，那就是证明我们正在做和官方保持一致的事情，如果有什么差错，那不是我的责任。至于项目能不能完成，那不是我考虑的问题。这样的文档怎么能够发挥作用呢？

第七章 质量控制：需求的管理、验证与确认

需求管理是需求工程的重要组成部分，而验证和确认是软件工程的必要环节。那么，在实施需求管理、验证和确认的过程中我们到底有哪些困惑？到底如何才能把需求做得更好？这些都是需要加以深入研究的。

7.1 需求管理的目的与任务

为了确保需求的质量，除了进行正式评审之外，在整个开发过程中都应该对需求进行管理。特别是当人们普遍面对着范围蔓延、偏离方向的期望、放弃、迷茫的顾客等情况的时候，我们应该认识到需求管理是一项最基本、最基础的管理职责。

软件最大的特点就是软件的“软”字。换句话说，软件最大的特征就是需求极易发生变化，而需求的变更又是造成产品质量问题最重要的原因之一。需求不断变更是我们常常遇到的情况，由于人们认识事物的规律，是不可能早期把一切都描述清楚的，需求变更是不可避免的。因此，需求工程中强调了需求管理，加强需求管理是提升产品质量的有效手段。

需求管理的根本目的是在项目整个生命周期期间，通过需求管理活动，确保开发过程有序、改善产品质量、降低维护成本并且提高开发效率，确保项目的整体目标得以实现，其侧重于如下五个任务：确保各方对于需求的理解一致、获取利益相关方对于需求的承诺并进行监督、进行需求变更控制、通过维护需求的双向可追溯性以确保工作产品和需求一致、以及标识工作产品与需求的不一致并进行处理。

- **确保各方对于需求理解一致：**可以通过用文档记录需求、识别利益相关人、分配需求以对其评审以及鼓励反馈意见等多种形式达成目的。
- **获取利益相关方对于需求的承诺：**需要在各方反馈意见达成一致的基础上建立需求基线，并且向管理层和客户证明项目已经达到可以继续开发的状态，从而获得批准。
- **需求变更控制：**首先必须知道需求的来源，记录更改的理由。以组织（技术状态控制组）的形式执行变更请求管理流程，分析利弊并且实施变更管理策略，以受控的方式管理变更。项目经理还需要跟踪反映需求更改程度的测量项，寻找需求变更的规律，并且通知相关方维护由需求变更所带来的计划、测量、质量保证以及配置管理方面的变更。
- **维护需求的双向可追溯性：**必须建立相应的需求跟踪矩阵，通过审核跟踪信息帮助项目经理确认所有需求都被应用，并且使得在需求发生变更以及维护的时候能够正确、完整地确保需求得以实现。当由于技术实现方面的原因造成需求变更的时候，也能够快速回溯到相应的需求源，以确保工作产品和需求一致，达到提高效率的目的。
- **标识工作产品与需求的不一致性：**必须对所有被发现的不一致性进行记录，说明它们的来源与原因，并且采取相应的纠正措施。

7.2 获得对需求一致的理解

项目组与顾客一起对需求进行分析，确保对需求的含义达成共识，最终达成一致的需求集，并且获得利益相关方的承诺，这对于需求能否落到实处尤其重要。

一、建立利益相关方理解需求的渠道

需要通过各种正式和非正式方式建立各方面的沟通渠道，以期利益相关方对于需求有一致无歧义的理解。对模糊的有争议的问题需要进一步澄清，挖掘隐含的需求，并达成一致。

1，为什么必须寻求一致的理解

软件是一种思维，软件开发是一种典型的知识工作。如果没有对问题一致的理解，各有各的想法，就不可能有共同的目标，也不可能有合理的计划与策略。

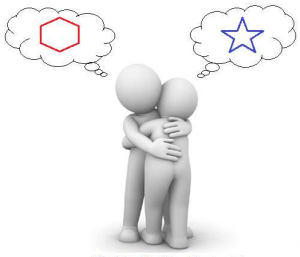
研究发现，知识工作本身就是一个互动的过程，每一方对于产品以及他们参与的工作会有不同的理解。随着项目参与人员越来越多，由于参与人员的背景越来越复杂，更多的问题出现了，这又产生一个更大的互动过程。随着越来越多的人对需求的共同理解，整个项目的利益相关方将会形成一个有凝聚力的整体，此时，人们都会看到项目成功的概率很高了。

在一开始，分析师对于产品应该是什么样的？应该如何定义产品事实上也不是太有把握。此时他们虽不能对所有的结论达成一致，但可以就当前哪些问题尚不清楚？如何把这些问题弄清楚？等等形成共识。之后就进入了交互的步骤：找出迷惑和分歧之处，就如何解决分歧达成一致，并且付诸行动。接着再进入一个更细节的层次，找出迷惑和不同意见，再解决它们。随着他们统一到共同的理解上，实际上也统一了产品的预期和需求。

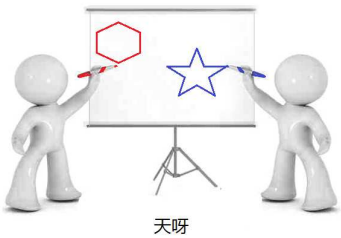
许多优秀的需求分析人员都会惊讶的发现，这种矛盾、迷惑、分歧、统一的不断循环，是需求开发的自然组成部分。依靠它们，利益相关方找出了需要解决的问题，从了推动了需求与设计走向了更高层次的创造性过程。

2，寻求一致理解的方法论

虽然我们使用相同的语言和辞藻来描述需求，但只有开始交付时才会意识到：我们想的完全不是一回事，这对项目来说是致命的。问题不在于我们有没有协调一致的愿望，这是很自然的，而是在于人们的内心还有歧异的时候，我们武断地认为人们已经达成一致了，多数项目就是这样被扼杀的。

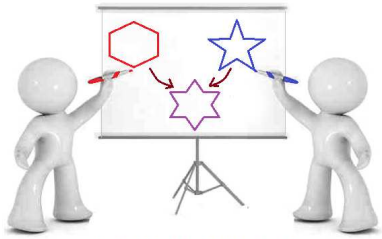


我们已经达成了一致

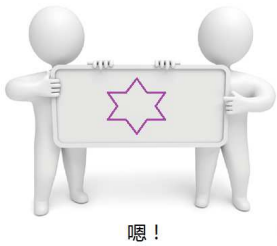


天呀

因此，对需求一致理解是需求管理最基本的目标。从管理的角度，需要建立一个过程以一致的、可重复的方法达到这个目标。其中最重要的方法，就是信息公开，让所有人的看法都能公开表达出来，并且真正的达成一致。



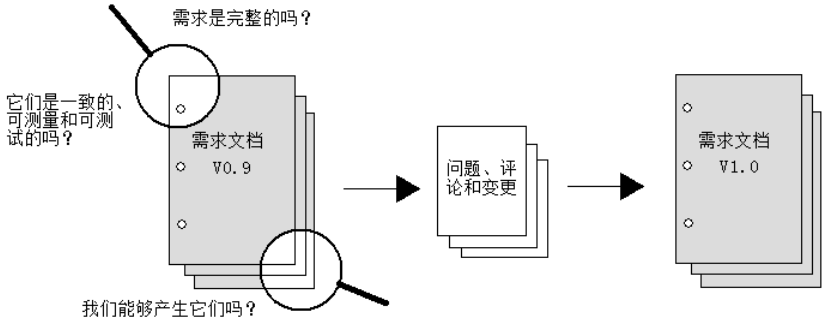
如果我们这个做会怎样



嗯！

例如：在初步的需求完成之后，要给利益相关人评审和理解需求的机会，然后对需求进一步

澄清。一旦达成协议，需求的基线版本就可以发布了，如下图所示。



在这个过程中，一般有五个事情要做：

1）用文档记录需求

用文档记录需求似乎是个不言而喻的事情，但问题是我们的需求文档书写的怎么样？是不是容易阅读？是不是图文并茂？是不是把握了重点？功能需求、非功能需求、约束条件是不是描述的准确全面？更重要的，需求是不是以可测试的方式来描述，特别是质量属性？文档的格式很重要，它反映了书写者是不是重视，如果 UML 图形符号用的不对，可能立刻就暴露出书写者的水准不够，或者不认真。需求书写的认真和正规，立即就可以引起评审者的重视。

2）识别利益相关人

关键是要识别到底谁对需求感兴趣，利益相关人是与需求有直接关系的人：批准需求的人、审查需求的人、使用需求工作的人。我们与这些人一起工作，以确信获得了正确的需求集合，并能够最终满足顾客的需要。

3）分配需求以对其评审

从管理领域的角度来看，并不是强迫人们去理解需求。项目管理者的工作是保证所有的人都有机会去理解需求。为了培育这样的机会，就可能制定一种分发机制（电子邮件、打印的副本等），可能还需要不时分发更新的版本，建立一种分发流程也可能是好想法。

4）给评审一定的时间

利益相关人需要获得对需求充分的理解，那就需要有充足的时间来理解需求，因此需要留有时间，而不是急冲冲的开个评审会。

5）鼓励反馈意见

如果利益相关人觉得他们的评论得到重视，他们就可能会出色的完成需求评审工作。鼓励反馈并给利益相关人提供反馈的方式，例如电子邮件、问题日志和任何有利于记录问题的形式。

二、获取对需求的承诺

承诺本质上属于一种道德规范，人们达成了某种协议并希望大家都遵守它，这就是承诺。没有承诺，随之而来的就是混乱，当主意不断的改变时，失败的命运就已经注定了。

正是由于承诺，人们在前期就需要下功夫对需求获取一致的理解，并且估算任务得出可以作出负责任承诺的结论。没有承诺的要求，也就不可能去理解需求。承诺之所以重要还有以下几个原因：

- 首先，它创建了一个认可的气氛，人们已经同意工作可以继续了。
- 其次，它意味着外界的反馈意见：要求批准需求的人有机会不批准需求，因为他认为需求还没有达到可以继续工作的地步。
- 最后，承诺建立了多数人共识的一种可见的途径，它向管理层和顾客证明，项目组确实可以在一起工作。

但为什么有些时候人们害怕做出承诺呢？当面前一片黑暗的时候，谁会有勇气做出承诺？所以，只有基于良好的需求才可能做出良好的承诺。所以如果发现人们害怕做出承诺，就需要检查

目前需要做出承诺的信息是否还有含糊不清的地方？这也成了良好需求的一个驱动力。

很多人认为承诺就是顾客在需求上确认签字。但为什么很少有人能够寻求到这个签字呢？这牵涉到人们对于需求的理解。最初的需求并不是结果，它只是一个基线，也就是在现在我们的认识水平下，我们已经达成了共识，项目可以开始了。签字并不等于冻结需求，因为我们的都认可随着项目进展，我们对问题会有新的看法，如果需求没有变化，那也不需要需求管理了。这个观点，应该在需要签字的表格中表达清楚。

这种沟通非常重要，当批准需求的人面对着一个陌生的复杂的解决方案，我们常常会发现他们在批准和发布需求的时候犹豫不决。因为他们觉得需求一旦发布，自己就没有了任何发言机会了。项目负责人应该充分表达这样的观点，通过确保需求管理过程是一个持续的、迭代的过程，并且有充足的机会调整和调节需求（在项目计划中已经预留了这种调整的时间），这种顾虑完全可以消除。

需求承诺目标是建立一个需求基线。需求基线将起到公共出发点的作用，并且作为未来所有迭代的基准集。如果利益相关人觉得需求可以容纳未来的变更，那他们将很愿意提交这个基线版本，并且帮助我们“获得批准签名”。

承诺有多种方式，最传统的就是签名，白纸黑字是好东西，它表明了承诺的最明确的途径。但也可能有多种形式，电子邮件、口头认可也未尝不可，但都不如签名好。关键是要表达清楚，签字并不等于冻结需求，它只是表明目前我们已经达成了共识，项目可以进行下去了。

由于在需求演化的过程中，需求的变更可能会在项目的全生命周期间存在，因此需求的承诺并不仅仅处于早期需求开发阶段，在每一次需求变更发生以后，都可能出现要求需求承诺的活动。基本的需求承诺活动可以有如下几个步骤：

1）了解需求承诺的人和形式

项目负责人首先需要明确需求承诺的责任人和形式。

2）明确需求承诺的意义

项目负责人必须与利益相关人沟通，并且说明需求的承诺并不等于冻结需求，而是表达在目前的认识水平下相关各方已经达成了一致意见，经过批准以后可以起到公共出发点的作用，并作为未来所有需求变更的基准集。

3）定义多种承诺方式

项目负责人对各种不同的利益相关人，定以多种承诺方式，例如签名、电子邮件、在相关表格上表达意见等。

4）纳入配置管理系统

需求基线一旦被建立就需要被纳入配置管理系统。这是因为在需求的演化过程中可能会出现多个需求版本，因此必须进行版本控制。

7.3 需求跟踪

通过需求跟踪来提供一种产品与合同一致的方法，把系统设计、编程、测试等阶段成果与需求文档进行比较，建立与维护“需求文档、设计文档、代码以及测试用例”之间的一致性，确保产品严格依据需求文档进行开发。

一、需求跟踪的动机与方法

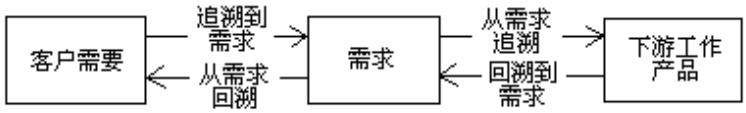
1，需求跟踪动机

需求跟踪提供了一个表明与合同或说明一致的方法。它可以改善产品质量，降低维护成本，而且很容易实现重用。项目中的需求跟踪能力有如下好处：

- 帮助审核确保所有需求被应用。
- 在需求发生增、删、改时可以确保不忽略每个受到影响的系统元素。
- 使得维护时能正确、完整地实施变更。
- 在重新设计时，可以列出传统系统中将要替换的功能，记录它们在新系统的需求和软件组件中的位置。
- 在设计新系统时发现旧系统相关资源。例如：功能设计、相关需求、代码、测试等。
- 减小风险，使部件互连关系文档化，减少关键成员离开项目带来的风险。

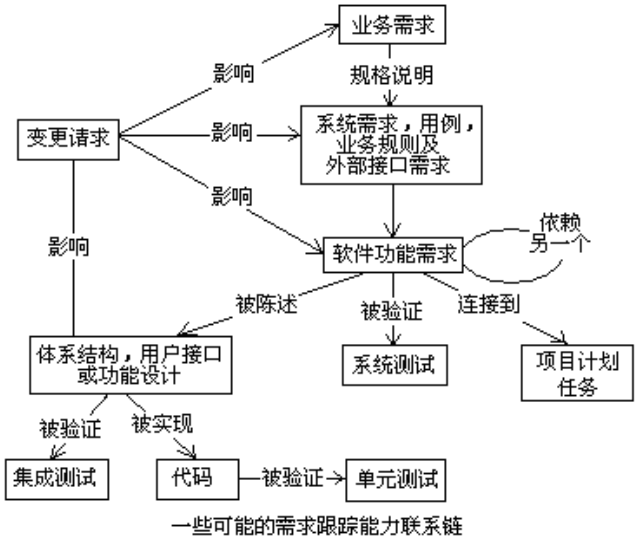
2，需求跟踪能力链

下图说明了四类跟踪能力。客户需求可向前追溯到需求，这样就能区分出开发过程中或开发结束后由于需求变更受到影响的需求。这也确保了需求规格说明书包括所有客户需求。同样，可以从需求回溯相应的客户需求，确认每个软件需求的源头。如果用用例的形式来描述客户需求，图的左半部分就是用例和功能性需求之间的跟踪情况。



四类需求可能跟踪能力

下图说明了许多能在项目中定义的直接跟踪能力联系链。



一些可能的需求跟踪能力联系链

3，需求跟踪能力矩阵

表示需求和别的系统元素之间的联系链的最普遍方式是使用需求跟踪能力矩阵。下表展示了这种矩阵。

一种需求跟踪能力矩阵				
用例	**功能需求**	**设计元素**	**代码**	**测试实例**
UC-28	catalog.query.sort	Class	catalog.sort()	search.7
		catalog		search.8
UC-29	catalog.query..import	Class	catalog..import()	search.8
		catalog	catalog.validate()	search.13
				search.14

每个功能需求向后连接一个特定的用例，向前连接一个或多个设计、代码和测试元素。设计元素可以是模型中的对象。加上更多的列项就可以拓展到与其它工作产品的关联，例如在线帮助文档。包括越多的细节就越花时间，但同时很容易得到相关联的软件元素。这样的跟踪能力联系

链可以定义各种系统元素类型间的一对一，一对多，多对多关系。

4，需求跟踪的基本策略

应用需求跟踪能力来管理工程时，需要通过以下策略来达到要求：

- 首先要决定定义哪几种联系链。
- 选择使用的跟踪矩阵的种类。
- 确定对产品哪部分维护跟踪能力信息。并不是处处需要跟踪，而是由关键的核心功能、高风险部分或将来维护量大的部分开始做起。
- 通过修订过程和核对表来提醒开发者在需求完成或变更时更新联系链。
- 制定标记性的规范，用以统一标识所有的系统元素，达到可以相互联系的目的。
- 确定提供每类联系链信息的个人。
- 培训项目组成员，使其接受需求跟踪能力的概念和了解重要性、这次活动的目的、跟踪能力数据存储位置、定义联系链的技术。
- 一旦有人完成某项任务就要马上更新跟踪能力数据，即立刻通知相关人员更新需求链上的联系链。
- 在开发过程中周期性地更新数据，以使跟踪信息与实际相符。

二、需求跟踪中的管理活动

需求跟踪矩阵可以用多种方式建立，在一般情况下可以用手工建立需求跟踪矩阵，在需求和产品关系比较复杂的情况下，提倡使用适当的需求跟踪软件，以提高工作效率。需求跟踪的基本管理活动如下：

- 项目负责人指定一个项目成员担负需求跟踪员，负责需求跟踪工作。
- 需求跟踪员使用需求跟踪矩阵记录需求跟踪状况。
- 项目负责人制定标记规范，以统一的标识所有系统元素，达到可以互相联系的目的。
- 项目负责人确定提供每一类联系链信息的个人。
- 培训负责人组织对项目组成员的培训，使其接受需求跟踪能力的概念和了解其重要性。这个活动还同时包括说明联系链数据存储的位置、需求管理工具的使用（如果使用工具）等内容。以确保项目组成员明白自己所担负的责任。
- 一旦有人完成某项任务即立即通知需求跟踪员进行需求跟踪矩阵的更新。
- 需求跟踪员在开发过程中需要周期性的跟踪数据，以使跟踪信息与实际相符。如果发生需求变更，经 CCB（配置控制委员会）小组研究同意后，立即通知相关人员更新需求，并通知需求跟踪员更新需求跟踪矩阵。项目负责人如果发现跟踪矩阵数据不正确，要立即通知需求跟踪员修正，以期达到正确的效果。

三、查找和消除不一致

项目负责人在项目开发期间，可以利用需求跟踪矩阵发现后续工作成果之间的不一致性，以期早期发现问题解决问题，例如

- 后续工作成果没有实现需求文档中的某些需求；
- 后续工作成果实现了需求文档中不存在的需求；
- 后续工作成果没有正确实现需求文档中的需求；

项目负责人把所发现的不一致性记录在“需求跟踪报告”之中，并通告给相关责任人。为了消除不一致必须：

- 相关责任人给出消除不一致的计划，项目负责人将该措施和计划记录到《需求跟踪报告》之中；

- 相关责任人消除不一致性之后，项目负责人要通知需求跟踪员更新需求跟踪矩阵。

7.4 需求变更控制

在项目期间需求可能会由于多种理由被更改。当出现需要更改的需求或者随着项目的进展产生附加需求的时候，可能不得不对现有需求进行变更。需求变更是一个高风险大影响力的活动，需要通过需求管理过程确保需求变更受控。

一、确定需求变更类型

需求变更通常有两种不同的类型，采取的方法和策略也不相同，包括：

1）功能变更

功能变更是为了增加与删除某些功能，这种变更必须通过正式的变更评价过程以权衡利弊。如果变更的代价比较小，同时对其它方面影响也比较小，通常批准这种变更。如果变更的代价比较大，在权衡利弊之后，就需要进一步确认由谁来支付变更所需要的费用，而且进行效益分析，以确认是否值得做这个变更。

2）缺陷修补

为修复漏洞所作的变更，在项目前期它是必须进行的，不需要从管理的角度进行审查和批准。在项目的后期由于变更影响较大，通常需要进入正式的变更流程。

二、审批变更申请

需求变更对项目成果影响极大，因此必须采取正规的管理活动来处理，审批变更申请的活动如下：

- 在明确需求变更类型的基础之上，由需求变更申请人撰写“需求变更申请书”，递交给项目负责人或客户方负责人；
- 需求变更申请书必须阐述变更原因、变更内容以及此变更对项目造成的影响；
- 项目负责人和客户共同审批“需求变更申请书”，如果任何一方不同意变更，则退回变更请求，项目按原需求文档执行；如果双方都同意变更则进入变更请求管理流程阶段。

三、管理变更请求

软件的变化性是导致软件开发困难的一个重要原因，诸如市场的变化、技术的进步、客户对项目认识的深入等都可能导致需求变更请求的提出。如果缺乏软件变更请求的有效管理能力，不断发生的变更会使开发管理失控。缺乏有效的变更请求管理会导致以下问题：

- 软件产品质量低，对一些缺陷的修正被遗漏。
- 项目经理不了解开发人员的工作进展，缺乏对项目现状进行客观评估的能力。
- 开发人员不了解现在手头工作的优先级别，可能会把紧急的事情放下，去处理一般优先级的任务。

有效的变更请求管理优点如下：

- 提高产品管理的透明度。
- 提高软件产品的质量。
- 提高开发团队沟通效率。

帮助项目管理人员对产品的现状进行客观的评估。

变更请求管理是由需求变更控制小组集体完成的管理过程，该过程包括 7 个阶段，它们是：变更请求提交、变更请求接收、变更请求评估、变更请求决策、变更请求实现、变更请求验证以

及变更请求完成。

1）变更请求提交

在提交变更请求的时候，需要对变更软件系统进行记录。根据变更的分类，变更请求通常也被分成两大类，具有不同的请求策略。

- **增强请求：**是指系统新增加的特征，或者是对系统已有设计行为的变更。增强请求来源较广，在很多情况下，它来自于客户并且直接通过市场部门或者客户支持部门到达工程部门。增强请求的关键数据是：这个变更对客户来说有多重要，另外诸如变更的详细细节以及请求提出人的标识也很关键。有的情况下，增强请求来自于内部的测试和内部试运行的结果。这类变更需要分析变更的代价，并且作出正确的决定。
- **缺陷：**存在于一个已经交付的产品中的异常现象或者缺陷。缺陷的来源大多数是内部测试发现的，在提交期间的关键数据是：如何发现缺陷、如何重现缺陷、缺陷的严重程度以及谁发现了缺陷。缺陷也可以由用户发现，此时关键数据包括：遇到问题的客户标识、该客户所感觉到的严重程度以及他使用的是哪个版本的软件等。

一般来说，缺陷需要立即处理。但是如果在项目开发的后期，就需要分析缺陷修复的代价，对变更的批准需要谨慎。

2）变更请求接收

指定接收和处理变更请求的责任人，确认变更请求。变更接收需要检查变更请求的内容是否清晰、完整、正确，具体包括：

- 是否已存在重复请求或误解。
- 确定请求是缺陷还是增强请求。
- 对变更请求赋予唯一的标识符。
- 建立变更跟踪记录。

在请求接收后，必须对这些请求进行评估。

3）变更请求评估

在评估期间，必须浏览所有的新提交的变更请求，并且对每个请求的特征作出决定，确定变更影响的范围和修改的程度，为确定是否有必要进行变更提供参考依据。

如果是缺陷，则必须是可重现的和可确认的，一般缺陷的优先级，可以根据缺陷的严重程度和修复缺陷的重要性加以确定，通常缺陷由工程部门加以评估。

增强请求不需要加以确认，但需要和其它增强请求以及产品需求一起比较以确定优先级。在增强请求评估期间，要记录有多少用户提出了相同的请求，提出请求用户的重要性，对市场的可能影响，对销售人员和客户支持人员的影响等。通常增强请求由产品管理部门进行评估。

4）变更请求决策

决策阶段需要对变更请求作出决定，比如推迟实施或者永远不实施等。缺陷和增强变更请求一般使用不同的方法处理。

对于增强请求，有多种原因影响着决定，包括：产品销售的容易程度、怎样经得起竞争、客户到底需要的什么、需要进行什么样的变更才可以进入新市场等。一般把所有的增强型变更放在一起权衡，要对一个新的发布版本进行实现、推迟实现以及不实现的决策。

缺陷的决策主要根据开发周期中所处的位置和工作量的大小来决策，在项目前期，一般缺陷可以分配给某个开发人员，由他自己决定做什么。如果缺陷是可重现的，开发人员会尝试在当前发布版本中修复该缺陷。

在开发周期的后期，就需要为缺陷建立正式的复审过程，比如开发人员可以作出评估，但他不能决策，而是由需求变更控制小组做出最后的决定。

在生命周期的不同点上关注点会不一样，比如，在产品开发的初期，会关注增强请求，而在产品开发的最后阶段，一般关心的是产品质量，尽管这时可能有增强请求，但关注点是缺陷。

5）变更请求实现

缺陷修复和增强都会带来更多的设计工作，而且缺陷修复需要构造一个环境，在这个环境中需要对这个缺陷进行重现测试，并且测试相应的解决方案。

正规的需求是以文档提交的，在实施变更期间就需要对文档进行变更，比如新功能必须文档化。对于缺陷，如果修复不影响可见的功能，则可以不修改文档，但如果影响到用户的可见行为，则也需要修改文档。当决定不修复的时候，有时候也会改变文档，比如使用变通的解决办法等，需要表现在发布版本中。

6）变更请求验证

验证发生在最终测试文档的制作阶段，增强请求的测试通常涉及验证所作的变更是否满足需要。缺陷测试则首先用一个正式项目的构建来复现这个缺陷，然后测试本次变更是否解决了问题。

7）变更请求完成

变更请求的完成可能是实现或者不实现变更，但都会导致变更请求者关闭请求循环。变更请求管理可能分成多个层次，但更多是集中在项目级。

7.5 验证与确认的基本概念

验证（Ver）和确认（Val）是软件工程重要的组成部分，也软件产品质量控制的重要手段。它来自于这样一个理念：“质量从来都不是偶然达到的，它总是智慧工作的结果”。

1，验证与确认的定义

在实践中，验证与确认关系比较紧密，容易给人以误解，因此首先必须给出明确的定义。

- **验证**：是确保所选的工作产品满足指定的要求。为了实现验证，需要定义一套验证测试方法，用于按指定的要求检验工作产品。验证一般还会要求同行评审，这是一种在早期排除缺陷的方法，可以对正在开发和维护的工作产品进行有价值的深入考察，而且其有效性已经得到了证明。
- **确认**：是按顾客需求增量式地确认产品已经达到了要求。确认可以在运行环境或仿真环境中进行，也可以通过正式评审。确认活动使用与验证类似的方法，例如：测试、分析、审查、演示或模拟。最终用户与其他利益相关方需要共同参与确认活动。

确认和验证活动常常并发进行，或者是不断交替进行的，其工作方式也有很多类似之处，并可以使用相同环境的一些部分，这很容易造成某些误会，因此必须明确这两者的区别。要注意到确认和验证的根本区别在于目标。

验证的目标在于发现问题和解决问题，所强调的是过程。其关注的是工作产品是否合适地反映了指定的需求，并尽早发现隐含的缺陷。从方法上来说：验证大部分使用非正式的评审方法（同行评审、走查、质量关），验证的结果可能是一张问题表，用于提醒相关人员修正缺陷。换句话说，验证确保“你正确地做了”。

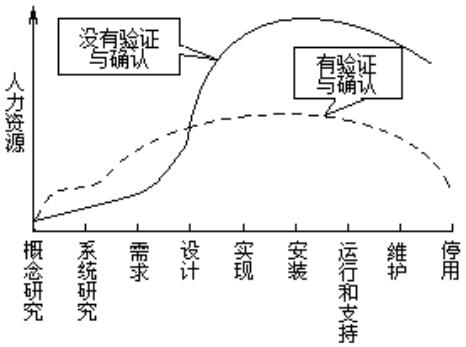
确认的目标在于确定工作产品是否正确，强调的是结果。确认证实所提供的产品符合预定的要求。从方法上来说：确认大多数使用正式评审，其结果是被接受的阶段或最终产品，这个结果往往作为重要的基线存在。换句话说，确认则确保“你做了正确的东西”。

2，利于早期发现问题降低成本

验证和确认强调早期、持续，而不是后期一步到位，但为什么要这么做呢？

如果提前针对验证和修改过程中花费一些金钱，将在后期的测试、维护和客户支持中节省更多的时间。如果组织建立了有效的验证程序，并对质量进行跟踪控制的话，这种付出就是值得的，总的回报远远超过这一个过程的支出。

人们早已认识到，验证与确认有利于早期发现缺陷，可以减少项目的总体成本和工作量，而且加快了交付进度。缺陷如果进入到了后期测试阶段，就可能需要执行多项测试来确定错误，并要进行额外的工作来获得测试信息，一旦发现遗漏的缺陷，正在生产过程中的程序员必须放下手上的工作，集中精力来修正错误，并确定其正确性，结果使总体工作效率大大下降。所有的研究都表明后期修复错误成本的增加非常巨大，越早发现错误，修正它的成本就越低。下面的图表达了这个关系。



但令人费解的是，验证经常是强行推行的，没有人对它的非经济价值提出异议，但在一个混乱的进度表和拙劣的管理模式下，验证在经济上的好处往往并不能被人感觉出来。令人欣慰的是，过去 20 年中，人们收集了很多数据证明了验证的好处，例如，Watts Humphrey 的研究发现，需求期间的 1 个小时能修正的错误，到设计阶段变成 3-6 小时，到编码阶段是 10 小时，到测试活动就变成 15-40 小时，到验收阶段是 30-70 小时，到操作阶段就是 40-1000 小时，成本相应大幅的增加。

3，同行复审

验证的重要手段就是同行复审（也称之谓走查），它除了经济效益以外还有非经济收益，包括可以抵消任务中与人有关的问题：

- 同行复审可以发现无关的部分，而测试不能。复审可以遵循所有的分支路径，而测试不能测试所有路径，在不频繁进入的路径中，经常会留下错误。
- 不同的思维方式、个性和生活经历，有助于发现各种问题，仅凭一个人的视角，往往很难发现问题。
- 当开发人员知道具有同级人员复审这样一张安全网的时候，会激发更多的创造性，并且在灵感闪现的时候，能够更加现实。
- 工作与人员分离，复审者可以把精力放在发现问题上。
- 由于复审需要合适的“块”，强化了开发人员模块化的思考方式，并利用高内聚低耦合的知识，提升整体上的质量。
- 在开发人员复审环境中，人们可以通过别人的工作，吸取他人的有益设想，并进行传播，有人认为这是复审的最重要的结果。
- 增加了团队的一致性，确立了事实上的团队行为标准。
- 项目经理能够知道产品的真实状态，有时开发人员认为很重要的事情，尽管没有恶意，但这个行为可能会破坏项目。

经验表明，在开发过程中进行同行复审，有助于减少人力资源的投入，大大减少了测试工作量，以及设计和编码的重复工作量，结果是从整体上节约了成本，还缩短了进度。

4，需求的质量关

在我们对验证有比较深入理解的基础上，让我们花一点时间来考虑需求质量控制的方法，毕竟这需要投一些工作量来确保需求的正确性。

在前面大量工作的基础上形成的“需求规格说明”，并不是最后评审过的包括版本号、发布日期以及批准签名的正式版本，而仅仅是正在形成正式基线版本之前的一项或者多项需求的集合，不论它以什么形式存在，甚至包括头脑中的需求。

以每项需求为单位的验证措施也称之为质量关，其意义在于能够帮助我们早期发现错误。需求质量关实际上是一个关卡，防止不受欢迎的需求进入规格说明。每项需求都必须尽过验证测试，以判断它的正确性和合适性，不受欢迎的需求包括：不完整的、不可追踪的、不一致的、无关的、不正确的、二义性的、不可行的、属于解决方案的、镀金的或者蔓延的需求。被拒绝的需求将被返回给提供者。

7.6 需求验证测试的步骤

大型项目都会需要一份完整的需求规格说明，而且即使需求中的一个小错误，到后来也可能成为大问题，这就需要验证来确保需求的质量。需求验证测试需要有一套检查方法来确保“你正确地做了”。事实上也需要一个检查单来逐项检查需求编写的质量。

很多人认为只要需求能表达清楚问题就可以了，殊不知很多后期发现的需求歧义，都是由于前期编写上的缺陷造成的。正确的需求编写非常重要，应该保证每个人读到同一本需求应该得到同一个结论，而不是各自不同的结论。

但如果最终规格说明书太厚，对整份文档的有效验证测试就会受影响（人们不会认真阅读超过 15 页的文档，往往会采用略读）。在这样的情况下，一般在产生需求的时候测试单页的需求或者一组内聚的需求比较容易成功。这个想法对于大型项目是不错的，以每项需求为单位进行验证测试，如果这项需求通过了这些测试，它就成为规格说明的一部分，最后总的规格说明中就只包括测试过的、正确的需求。

1，测试需求的完整性

我们可以把需求项框架看作是针对单项需求的一个分格的容器，每一个是这项需求的一个组成部分，前面建议大家使用的需求收集卡片，可以用来帮助测试需求的完整性，下面展示的卡片是一个完整需求的例子，需求分析师已经注明这项需求与其它需求没有冲突，这项需求应该通过完整性检查。

需求编号：75	**需求类型**：9	**事件/用例编号**：6
描述：如果一个气象站传送数据失败，产品将发出警告。		
理由：传送数据失败，可能表明该气象站失效并需要维护，并且用于预测结冰的数据可能不完整。		
来源：George ，Shaw， 道路工程。		
验收标准：对每个气象站，当每小时记录下来的各类读数个数，不在制造商规定的范围之内时，产品将通知用户。		
顾客满意度	**顾客不满意度**	**冲突**
3	5	无
优先级：版本 1		
支持材料：Rosa 气象站规格说明书。		
历史：GBS 在 2005-07-28 提出。		

1）测试是否存在遗漏部分

完整性的第一项测试就是把需求与框架中出现的组成部分进行比较，尽管不是每项需求都有所有的组成部分，但如果缺少某些部分，应该有很好的理由才行。

通过多年的实践，我们发现这些组成部分是有价值的，有助于说清楚一项需求和它的理由，它们或者是度量方法，或者是解释，或者是指向其他信息，或者是将在项目的一些阶段用到的需求，所以强烈建议只要适合该项信息的需求，都加入进来。

但有时候并不是所有部分都是需要的，比如有的时候描述本身就说明了为什么这项需求很重要，再加上理由就没什么意义了。有时候如果有清晰可读的验收标准，描述也可以去掉。有时候也可能没有支持材料。

很自然，如果缺少了一个组成部分，原因应该是它不必要，而不是它太难写或者应为没有注意到。如果缺少某个组成部分是因为还在调查它，可能性是等待某个人的回答，那么就需要在需求上注明，例如：

支持材料：2008-10-05 等待用户方工程师提供支持材料的细节。

完整性测试指出每项需求都应该具备所有相关的组成部分，否则应该记录下他们没有完成或者不能完成的原因。

2）测试是否对所有利益相关方都有意义

对需求的组成部分满意以后，应该确保每个组成部分都增加了需求的含义，促进共同的理解，爱因斯坦说过：“每件事都应该尽可能的简单，而不是简单一点”。这意味着需求的编写要尽可能清晰、言简意赅，但必须确保写下的需求包含了所需要的信息。

例如有这样的部分：

支持材料：Rosa 气象站规格说明书。

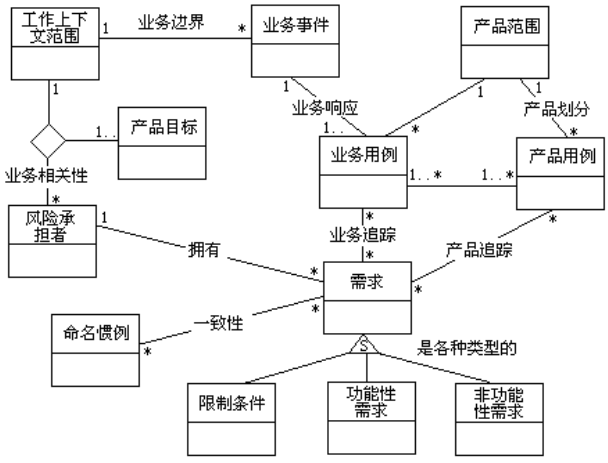
可以问一下有没有可能导致误解的部分，比如可以这样问：“这会让我们产生误解吗？”“Rosa 气象站规格说明书是不是只有一个版本？”这样的问题有助于我们更精确的表达我们的意思，修改后的结果如下：

支持材料：2007-1-22 日发布的 Rosa 气象站规格说明书 1.1 版。

2，测试需求的可追踪性

我们经常会发现，队伍在向后面传递口令的时候，到最后传递的结果是面目全非。软件开发与这个情况类似，每个阶段都需要作某些转换，在转换的过程中，开发所涉及的众多人员中的某个人可能误解需求、错误的使用需求、或者搅乱需求。

因此很重要的一件事情就是，能够把最初的需求与最后提交的产品中实现该需求的部分联系起来，或者换句话说，就是要把开发过程的开始到它结束之间保持联系，确保指定的需求就是最后实现的需求，下图展示了需求的各个部分与它们实现之间的联系。



这是一个典型的需求知识模型，展示了需求规格说明元素之间的联系，这种联系对于开发过程中追踪个个组件是必需的。例如：“产品范围”与一些“产品用例”连在一起，每个“产品用例”有一系列的需求。

一项原子需求可以是一个“限制条件”、一项“功能性需求”或者一项“非功能性需求”。

“工作上下文范围”由一些业务事件组成，每个业务事件对应一个“业务用例”。

为了维护可追踪性，我们需要系统的了解哪些需求属于那些产品用例，哪些业务事件由哪些业务和产品用例来完成。在质量关中，我们需要注意可追踪性的需求一般应该具备以下特性：

- 唯一的标识符号；
- 需求或者限制条件的类型说明（使用已经定义的小节号等）；
- 该需求所属的所有业务用例或产品用例的参考引用；
- 对提出该需求的风险承担着的参考引用；
- 使用一致的术语。

如果需求是可跟踪的，那么当变更发生的时候，找出产品那些部分将受到变更的影响就更加容易，评估更改对产品其它部分的影响也更加容易。要记住需求可能会在产品生命周期的任何阶段发生变更。确保需求可追踪性意味着可以设计出最有效的方式进行更改。

3，测试术语的统一性

需求分析与文学作品是不同的，文学作品尽力使不同的读者触发不同的、富于变化的景象，而需求分析必须不同的人阅读只有一种解释，否则就存在危险。

为了指定只能有一种方式解释的需求，应该在使用说明书中对使用的术语进行定义。

在需求模板中，包括了命名惯例和定义，这个定义应该经过风险承担着的认可。

保持一致性的第二件事就是：检查每项需求使用术语的方式都与它的定义相符，这就是说只有定义是不够的，还必须确保每个属于的使用都是正确的。一词多义会使设计者选择一个对她来说最有意义的含义来实现它，如果利益相关方不认可这个含义，就会认为系统没有符合要求。

应该把寻找不一致性作为确定需求过程的一部分，当发现不一致的时候，就意味着需要进一步的调查和协商。

4，确定是否与目标相关

在开始需求获取的时候，肯定会遇到一些很吸引人的、说明清晰的、完整的需求，但它们与产品的目标没有关系。大多数项目都会遇到这样的事情，热情的用户开始加入他们能想到的任何东西，开发者也想让产品无所不包，结果这导致了产品的需求蔓延，结果是寓所超支或者项目滞后。

在项目启动的时候，应该很好的讨论并确定产品的目标，并且以量化的方式把它写出来，这些量化的目标成为项目的仲裁者。

要测试一项需求的相关性，只要把它的需求与产品的目标进行比较。测试方法是：这项需求是否为项目的目标做出贡献？这项需求是否增加了一些东西，直接或者间接的使项目更接近于目标？

作为一个例子，我们来考察下面的需求：

描述：产品应该维护一个查找表，表中记录一年中日出和日落的时间。

项目目标：精确预报道路结冰时间并有效的调度，进行合适的除冰处理。

初看起来好像是相关的，并通常在晚上形成，这项需求似乎对产品目标做出了贡献。但是进一步研究就会发现，道路结冰是由道路表面温度决定的，而温度是由气象站监视并传输的，冰在白天也可能形成，所以没有必要知道白天和黑夜相关的知识。

有些需求与目标看起来没有直接联系，但如果没有这些需求，产品将不能达到目标，例如如下需求：

描述：产品应该记录卡车活动的时间长短。

初看起来和产品目标没什么关系，但是看看这项需求的理由：

理由：除冰卡车 24 小时内不能被调度超过 20 小时。

就发现这个需求是必要的，质量关的负责人可以放行这项需求，因为她对满足项目的目标做

出了贡献。

许多非功能需求也可以看成对项目目标的间接贡献。

无关的需求被退回来的时候，应该附上一个简短的解释，而且不要害怕为此花一点时间。因为出现“无关的需求”可能意味着某个利益相关方对项目的目标有误解，或者意味着某个新的业务领域正在被打开。换句话说，你必须判断需求是不是真的是无关的，或者是否因为业务的改变，原来的范围已经不正确了。

产品的范围也决定了需求的相关性，通过对上下文图与相邻系统的信息流的研究，可以测试相关性。例如：

描述：产品应该记录卡车驾驶员的加班情况。

要查看以下产品的边界，通过产品的上下文模型图，以及进入合离开产品的信息流，研究是否有某个信息流与记录卡车驾驶员的加班情况有关呢？是否有信息流表明产品将处理工作时间问题？在上下文模型中，是不是有什么东西表明产品应该存储加班情况呢？

在道路除冰的例子中，没有什么可以证明记录加班情况是相关的需求，这项需求也不与其它任何部分发生联系，所以这是一项冗余的功能。

5，测试验收标准

需求可能是二义性的，实际上任何语言的陈述都可能有二义性，会产生不同的主管理解。所以，编写需求的同时就需要编写“验收标准”，只要可能，验收标准就要用数字来量化，因为数字是没有二义性的。

每项需求都有一个验收标准是很重要的，第一个要问的问题就是：“需求是否有一个正确定义的验收标准？”任何没有验收标准的需求必须被认为是不完整的，因为这样的需求还存在很大的不确定性（如果无法度量它，就不能理解它）。

对验收标准的下一个问题是：“它是否被用作设计接受测试的输入信息？”这可能要测试人员来帮助回答这个问题。

验收标准不是测试设计，但说明了需要测试什么以确保提交的解决方案与需求相符，在这个阶段，也可以考虑测试的成本是否可以接受。

虽然验收标准使用数字来表达需求，但数字本身并不是主管确定的，要基于事实依据，例如：

描述：产品应该易于学习。

验收标准：用户在第一次使用该产品的时候，应该能在开始 30 分钟内学会处理一项申请。

首先要问的是：“30 分钟这个数字从何而来？”它是否来自于某个利益相关方的突发奇想？或者它是来自于一些证据，这些证据表明超过 30 分钟用户将感到受挫并且放弃？或者需求的编写者在需求的支持材料中包括了对这类证据的引用？

验收标准度量方法可能包括一些业务误差，比如上面的例子要指定一组人进行测试，就要允许其中的一小部分人失败，这称之为业务误差。但业务误差的确定，应该以适当的事实为依据。

6，确定在限制条件下是否可行

可行的需求需要符合项目的限制条件，限制条件包括诸如构建产品可用的时间、预期的工作环境、产品的用户、对产品设计的限制等。

必须检查每项需求，已确定它们在限制条件下是否可行，这意味着必须对每项需求问一下，是否与限制条件存在某种冲突。

例如有如下的需求：

卡车驾驶员应该接收气象预报，并安排调度他们自己的除冰行动。

这个需求是不行的，卡车司机并不具备预报道路结冰时间所需的信息，特们也不知道哪些道路已经处理过了，协调司机需要集中处理。

可能还需要考虑组织机构是否成熟，是否具备实现这项需求的技术能力？写出一项需求是容易的，但为它构建一个能工作的解决方案是另一回事情。

还有一个更困难的事情，就是对现有开发能力进行评估，需求是否超出了现有开发团队的开发能力，可以从团队经验、项目允许时间以及预算的角度考虑这些问题。

需要考虑是否有时间和财力来实现这项需求，考虑这项需求在总预算中所占的份额。如果超过了预算，就要考虑这项需求所代表的顾客价值：高价值的需求需要沟通和谈判、低价值的需求作为放弃的候选项。

还要注意某项需求是否为所有的利益相关方所接受？这是一种自我保护的措施。如果某些利益相关方不认可某项需求，历史告诉我们这项需求在最后验收的时候会出现问题，甚至在他们会设法破坏产品的开发，由于功能与他们的想象不一样，他们也可能放弃使用该产品。

我们还需要考虑：是否存在限制条件使该项需求不可行？是否存在一些伙伴应用或者期望的工作环境与该项需求冲突？是否存在一些解决方案限制条件（也就是设计限制条件）？是否存在难于实现或者不可能实现的需求？

7，区分需求还是解决方案

对需求的描述往往不自觉地用解决方案的形式说出，这是由于个人经验所致，这种方案可能可行，但不一定是最合适的，从而隐蔽了真正的需求。

应该检查需求是不是包含了技术方面的因素？一般越是抽象的描述，越不可能是解决方案，例如如果写下：

产品应该易于使用。

这是一项需求，但是如果写下：

产品应该使用 JavaScript 来实现界面。

那就是解决方案，因为它牵涉到了具体的技术。

在检查需求的时候问一个为什么，往往能发现真正的需求，例如遇到了如下需求：

用户应该用口令来访问系统。

为什么这是一项需求？“因为我们不希望未经授权的人能访问保密的信息。”这就引发了真正的需求：

产品应该只允许得到授权的用户访问保密的信息。

显然，口令只是达到这个目的的一种形式，还有多种方式可以达到这个目的。

当然，很多当初提出来的解决方案可能是很好的方案，我们应该记录下这些方案，实际上只要想到好的方案就记录下来，但不要经不起诱惑，歪曲产品的需求来适应某种解决方案，解决方案可能是了不起，但它解决的可能是其它的问题。

8，顾客价值

需求所附的顾客满意/不满意度评分，说明了顾客对于该项需求价值的认知。

顾客满意度	顾客不满意度
如果这项需求成功实现，利益相关方的高兴程度（评分 1-5，1 为不感兴趣，5 为非常高兴）	如果这项需求没有成为座钟产品的一部分，利益相关方的不高兴程度（评分 1-5，1 为无所谓，5 为非常不高兴）

我们发现，分成两步式的评分有助于人们可观的考虑需求，而不是机械的给出 1~10 的评价。顾客、或者由重要的利益相关方组成小组，在需求分析师的帮助下，对需求的设定价值评分，以后，这些评分将与实现需求的成本加权，在需要的情况下，这个评分将帮助我们在需求之间取舍。这看起来很费时费力，但精确了解一项需求的价值是非常有用的。

如果大量的需求得出的价值都是 5/5，这表明价值没有被仔细评定。不过，经验表明这种评定是客户最不愿意做的事情，应该让客户理解：为什么理解什么需求对他们的业务是最重要的这件

事情非常重要，为什么我们的关注点要放在对客户最重要的需求上，尽管我们会尽可能避免，但在发生必须要折衷的时候，我们就会选择最重要的需求来实现，同时这样做我们就会防止镀金的需求等等。所以，对于没有被仔细评定的评分，应该动员客户再做一次。

9，镀金需求

“镀金”的词来自于浴室的龙头，镀金的龙头出水并不比其它龙头好，它唯一的特点是更贵。在顾客价值评定中，不满意度评分很低的需求，往往就是镀金的需求。

要指出的是，我们并不一定需要把镀金的需求完全排除在外，有些额外的东西有时候也是不错的想法，有时候一点小小的镀金，会使客户非常乐意接受这个产品（想想 windows 中的屏保程序）。但是实现镀金的需求必须是有意识的选择，如果发现项目限制条件下不能实现所有的需求，那么排除在外的首先就是镀金的需求。

7.7 需求确认与正式评审方法

验证的目的，是把不好的需求拒之门外，而确认是通过正式评审保证需求是正确的。在进行正式评审之前，需要再一次复查需求规格说明的质量。评审前的复查需要检查需求是否存在遗漏或者矛盾的需求，保证所有的需求相互一致。简而言之，复查工作是确保规格说明书是完整的和恰当的，这样就可以转向正式评审阶段。

需求确认是一个正式的评审过程，以确保“你做了正确的东西”。经过正式评审并且发布的的需求，将作为一个基线，约束了开发、测试、集成和交付等各项活动。需求确认可能是一个增量迭代过程，确认会与验证交替重复进行。

1）需求文档的评审是一项精益求精的技术

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”。

2）需求评审为利益相关方提供了在特定问题上达成共识的方法

需求评审也为利益相关方们提供了在特定问题上达成共识的方法。有一个项目经理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，需求分析师和项目经理很恼火，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。

3）验证和确认提高的投资回报率

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。

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。本章将沿着如下图所示思路逐渐展开。

4）我们需要认真审查需求文档的每一行

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？哪些不重要部分只要用非正式评审就

能满足质量要求？

一、正式评审过程

正式评审需要有一个过程来控制它的进程。

1，参与者

评审参与者必须代表三个方面的观点：

1）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。

2）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架构师，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，评审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。

3）要根据正在评审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。这些评审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。

评审组中的评审人员应限制在 7 个人左右或者更少。如果评审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，从而在评审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，评审中每个成员扮演的角色

一些评审组中的成员在评审期间扮演着特定的角色；这些角色随着不同的评审过程而不同，但其所起的作用是相似的。

作者：作者创建或维护正在被评审的产品。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”的非正式评审中，作者经常主持讨论。然而，作者在评审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。

由于作者在评审中不起积极作用，因此，他只能听取其它评审员的评论，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评审员没有觉察到的错误。

调解者：调解者（moderator）或者评审主持者所做的是，与作者一起为评审制订计划，协调各种活动，并且推进评审会的进行。调解者在评审会开始前几天就把待评审的材料分发到各个评审员，按时召开会议，从评审员那获得评审结果，并且使会议集中在发现错误上，而不是解决提出的问题。

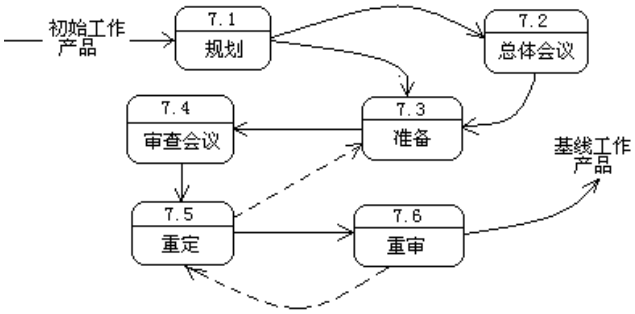
向经理或者那些收集评审数据的工作人员汇报评审结果也是调解者的责任。调解者最后的角色是督促作者对规格说明做出建议性的更改，以保证向执行者明确说明在评审过程中提出的问题和缺陷。

读者：读者的角色由评审员扮演。在评审会进行期间，读者一次评审规格说明中的一块内容，并做出解释，而且允许其它评审员在评审时提出问题。对于一份需求规格说明，评审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评审员不同的解释，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评审会中提出的问题和缺陷。记录员必须仔细评审所写的材料以确保记录的正确性。其它的评审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。

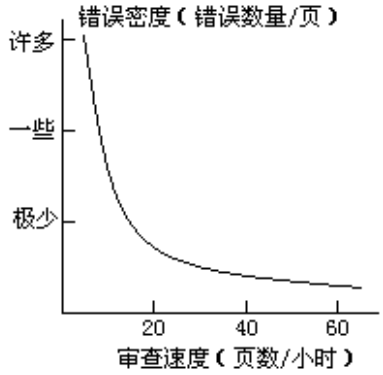
3，评审阶段

典型的评审过程如下图所示。



评审是一个多步骤事件。每一个评审过程阶段的目的简要地总结如下：

1）规划（planning）：作者和调解者协同对评审进行规划，以决定谁该参加评审，评审员在召开评审会之前应收到什么材料并且需要召开几次评审会。评审速度对能发现多少错误影响甚大（Gilb and Graham 1993），如下图所示。



评审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，就会引起评审速度的下降。）由于不能把太多的时间用于需求评审，所以应根据忽略重大缺陷的风险来选择一种合理的速度。

每小时评审 4～6 页是合理的，但应根据如下因素调整评审速度：

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

2）总体会议（overview meeting）：总体会议可以为评审员提供了解会议的信息，包括他们要评审的材料的背景，作者所作的假设和作者的特定评审目标。如果所有的评审员对要评审的项目都很熟悉，那么你就可以省略本次会议。

3）准备（preparation）： 在正式评审的准备阶段，每个评审员以典型缺陷（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。评审员所发现的错误中有高达 7 5 %的错误是在准备阶段发现的，所以这一步骤不能省略。

如果评审员准备不充分，将会使评审会变得低效，并可能作出错误的结论，此时，评审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。

4）评审会议（inspection meeting）：在评审会进行过程中，读者通过软件需求规格说明指导评审小组，一次解释一个需求。

当评审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。

评审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。

评审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，评审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。

一些研究者认为评审会议是劳动密集型的，以至于很难说清它们的价值。然而，我发现评审能发现评审员在进行个人准备时没有发现的错误。即使有这些评审质量的活动，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。

5）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，作者必须在评审会之后，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，那么进行需求评审将是无意义的。

6）重审（follow-up）：这是评审工作的最后一步，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。重审结束了评审的全过程并且可以使调解者做出判断：是否已满足评审的退出标准。

4，进入和退出评审的标准

当软件需求文档满足特定的前提条件时，你就可以进行需求评审了。在评审的准备阶段，进入评审的标准为作者设定了前进的方向。这些标准还可以使评审小组避免把时间浪费在评审之前就应该解决的问题上。调解者在决定进行评审之前，可以把进入评审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评审的标准：

- 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。
- 已经获得了评审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评审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，在调解者宣布评审结束之前，你应该定义所满足的退出评审的标准。这里有一些关于需求文档的退出标准：

- 已经明确阐述了评审员提出的所有问题。
- 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。
- 检查是否已将评审过的资料送到有关收集处。

5，需求评审清单

为了使评审员警惕他们所评审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。这些清单可以提醒评审员以前经常发生的需求问题。下面的表描述了评审软件需求规格说明书的清单。

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、评审得以验证或分析？
- 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（如系统需求或用例）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？

下面表包含了一个用例的评审清单。

- 用例是否是独立的分散任务？
- 用例的目标或价值度量是否明确？
- 用例给操作者带来的益处是否明确？
- 用例是否处于抽象级别上，而不具有详细的情节？
- 用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用例？
- 是否简明书写、无二义性和完整地记录了每个过程的对话？
- 用例中的每个操作和步骤是否都与所执行的任务相关？
- 用例中定义的每个过程是否都可行？
- 用例中定义的每个过程是否都可验证？

没有人可以记住冗长清单中的所有项目。可以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。可以让不同的评审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：赋予评审员特定的查错责任，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，这比仅仅向评审员发放一份清单所产生的效果要好得多。

二、评审前复查规格说明

复查也为重新评估产品的费用和风险提供了机会，既然拥有了一份完整的需求，对产品的了解就会比启动会议更多。在需求规格说明完成之后，对产品的范围和功能有了一个准确的认识，

所以此时就是重新更准确的估计产品规模的时候了，另外诸如可选择的解决方案架构、预估费用等都可以更清晰的完成。

这个阶段有些需求会引入巨大的风险，开发方应该评价自己的能力，并且预估风险。

软件需求说明的评审主要需要审查下面几个问题：

1．审查需求的一致性

2．审查需求的现实性

3．审查需求的完整性和有效性

软件需求说明评审中的问题：

1，所规定的软件目标和任务与系统的目标和任务相符合吗？

2，与所有系统成分的重要接口都已被描述了吗？

3，研制项目的数据流图、数据字典、数据结构充分确定了吗？

4，图表都清楚吗？每个图表在不加补充说明的情况下能被理解吗？

5，主要功能在规定的范围之内吗？每一种功能被充分说明了吗？

6，设计的限制条件是现实的吗？

7，开发的技术风险是什么？

8，考虑过软件需求的其它方案吗？

9，检验标准是否详细？他们能否确认系统是成功的？

10，有无遗漏、重复或不一致的地方？

11，用户是否审查了初步的用户手册？

12，软件计划中的估算是否需要修改？

三、需求评审的问题分离技术

经常遇到的情况是，需求评审以一种比较随意的方式进行，虽然也有专家会议、讨论、结论这几个环节，但由于过程比较宽泛，视野比较散，难以发现需求的缺陷。因此，在评审中需要把问题分离开来，一个问题一个问题的解决。

1，评审是否存在遗漏的需求

评审时的第一项检查就是确定适合产品的所有需求类型是否出现在产品规格说明书中。项目的目标和产品的战略计划常常能够说明适合的需求类型。例如，如果开发一个财务产品，但却没有安全性需求，那肯定有什么东西漏掉了。

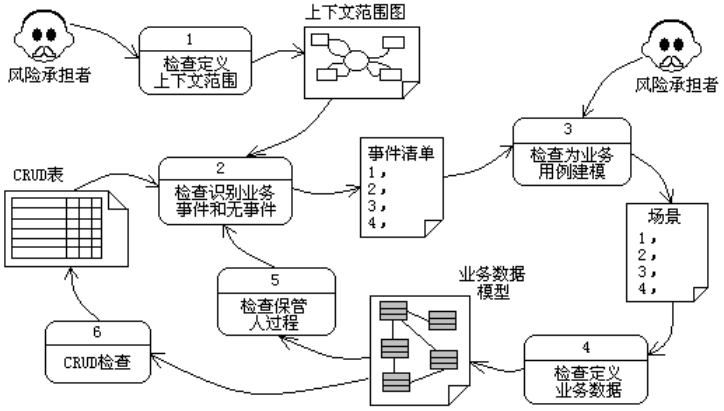
功能性需求应该足够完成每个用例的工作，为此可以和拥护一起把每个产品用例推演一遍，如果做了需求所要求的事情，是否能得到用例的成果？用户是否满意地认为，产品会完成他们的工作需要？

要寻找产品必须处理的异常情况，是否产生了足够的可选场景，以处理这些不测事件？

针对非功能需求类型检查产品的每个用例，它是否适合该类用例所有非功能性需求？依次查看每个非功能性需求，阅读它们的描述，并确保所有可能适用的非功能需求已经被包含在规格说明之中。

2，评审是否发现所有的业务用例

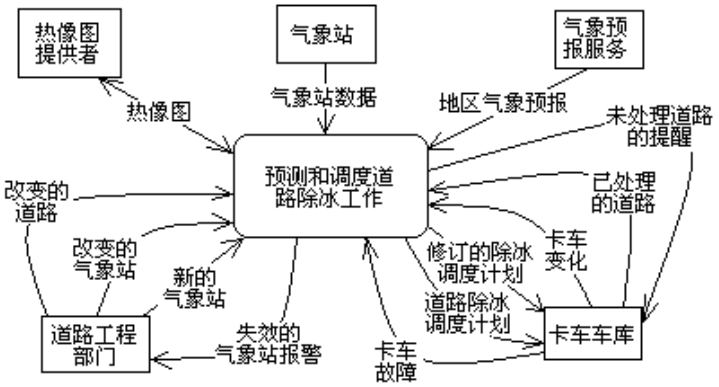
作为良好的需求分析，我们的基本策略是：针对每个业务事件，确定工作的响应（业务用例），进而研究响应的哪部分需要由产品完成（产品用例），然后每次收集一个产品用例的需求。但是，怎么知道是否已经发现了全部业务事件了呢？我们可以利用需求过程和系统建模的输出信息，有条理的、形式化的进行审查。下图展示了确认业务事件的完整过程。



这个过程是一个迭代式过程，执行每项活动直到“识别业务事件和无事件”的活动不再发现新的业务事件为止，在需求活动的这个阶段，应该已经完成了这个工作中的绝大部分。

1）检查定义上下文范围

我们这里所说的上下文范围是要研究的工作的范围，上下文范围是在“项目启动”阶段确定的，假定我们已经确认的上下文模型如下。



在需求分析过程中，我们会对这个上下文范围精化，也可能由于创新的需要而变动这个上下文范围。检查的过程是对上下文范围的完整性提出疑问，如有必要，就需要对它进行更新。

2）检查识别业务事件和“无事件”

这是检查的重点和难点，如果遗漏了事件，就必然遗漏了功能，假定我们已经确认的事件清单如下。

道路除冰系统事件清单			
编号	**事件名称**	**输入数据流**	**输出数据流**
1	气象站传送数据	气象站读数	
2	气象局预报天气	区域气象报告	
3	道路工程师通知改变的道路	改变的道路	
4	道路工程师安装了新的气象站	新的气象站	
5	道路工程师改变了气象站	改变的气象站	
6	到了测试气象站的时间		失效的气象站告警
7	卡车车库改变了卡车	卡车改变	修订的除冰调度计划
8	到了检查结冰道路的时间		道路除冰调度计划
9	卡车处理了一条道路	已处理的道路	
10	卡车车库报告卡车出问题	卡车故障	修订的除冰调度计划
11	到了监控道路除冰的时间		对没有处理的道路进行提醒

必须仔细核对上下文图和数据流，检查有没有遗漏的地方。发现“无事件”是防止遗漏的一个好办法，始终问一下“如果事件没有发生将会怎样？”则可以帮助我们发现一些遗漏的事件。

3）检查为业务用例建模

这不是需求复查工作的一部分，而是已经完成的工作。但是我们可以阅读产品模型中需求场

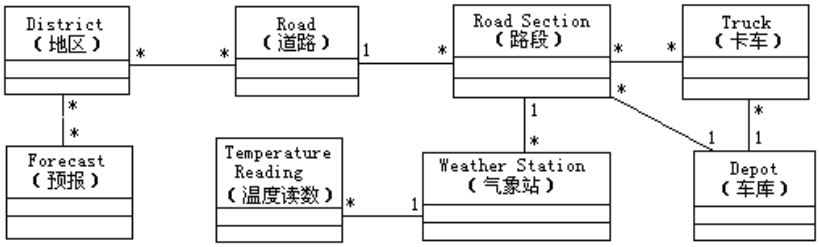
景的模型，判断这是不是可以很好有助于我们理解和沟通业务用例的功能和数据。

4）检查定义业务数据

复查过程的下一步，这一步应该已经完成了的，就是构建正研究的工作所需要存储的数据模型。可以用各种方法构建这个模型，比如类图、实体关系图等。根据需求调研成果，这个数据模型是不断确认和更新的。

一般来说，构建这类模型需要获得数据库人员的帮助，尽管这个阶段并不需要构建具体的数据库模型，但是，坚持让数据库人员帮助构建“业务数据”的模型是有好处的，但要注意的是这决不意味着开始数据库设计，这是两码事。

下图是已经构建了的道路除冰系统数据模型，实体类代表了表，而属性代表了字段。



在定义的时候需要标识主键，如果存在关联就需要标识外键。

不需要在确定数据类上花费太多的时间，只要做的足够好，不用花费整月的时间，由一些启发对于确定属性和类是有帮助的：

- 业务中使用到的具体或者抽象的东西；
- 可以标识的东西：账户、销售机会、顾客；
- 数据的主题，而不是数据的本身；
- 有确定商业目的的名词；
- 产品或者服务：抵押证书、服务协议；
- 组织机构的分支、位置或者结构；
- 角色：专案负责人、雇员、经理；
- 要记住的事件：协议、合同、订购；
- 来自上下文范围模型的相邻系统。

也可以通过上下文范围模型来发现类，上下文范围模型展示出进出工作的数据流，一般来说，工作存储的数据通过数据流流入、也通过数据流离开。可以这样认为：如果工作内存在数据，必定是某些数据流带来的。这意味着必须仔细研究上下文模型中的数据流，并且寻找它们的属性。针对它们问一下：“这个属性描述了什么？”或者“这项数据的主题是什么？”，这些主题就是类。

分析上下文范围模型中的所有数据流，包括进入和离开的，在寻找满足前面列出的类的属性的“东西”，分析所有的数据流以后，就几乎已经确定了所有业务数据类了。

5）CRUD 检查

需要在类模型中检查每个类的数据，他们可能被创建（Create）、被引用（Referenced）、被修改（Update）、被删除（Delete），这称之为 CRUD 检查。我们可以创建一个 CRUD 表，以确定是否每个实体上都执行了相应的动作。在表中，每个单元个显示了业务事件的编号，这些事件对实体执行了相应的操作。

CRUD 表				
类	**Create**	**Referenced**	**Update**	**Delete**
车库		7		
区		2，8，10		
气象预报	2	8，10		
道路	3	4，8，9，10	3	
路段	3	4，8，10，11	3，9	
温度读数	1	6，8，10		

卡车	7	8，9，10	7，10	
气象站	4	1，6	5	

如果一个类被引用但却没有被创建的时候，这表明创建事件被遗漏了。如果实体被创建却没有被引用，表明要么遗漏了事件，要么它是多余的数据。有些类（并非所有的类）会被更新或者删除，当然，为此它们必须被创建。

CRUD 表中任何一个空位都提出了工作范围完整性的问题。例如，车库、区实体没有创建事件，这意味着上下文范围是不完整的，因为它没有展示创建这些存储数据实体的输入数据流。

这种分析的结果，可能需要重新访问用户，以发现这些遗漏事件的更多知识。当它们被发现以后，应该记录在事件列表中，更新 CRUD 表，然后继续该过程。

CRUD 标的“删除”列只展示引为业务策略的原因而删除的类，这与数据归档或者清理是不一样的，例如，如“车库”将停业，那么它将被删除，但是,“气象预报”永远不会删除，这种分析需要与业务策略结合起来考虑。

6）重复直到完成

上述业务事件的发现是迭代式的，也就是说，必须持续不断的重复这个过程（确定业务事件、对业务用例建模、假如类模型、检查类被创建、引用、更新和删除），直到在“发现事件和无事件”不再能发现新的事件为止，这样就可以确信再也没有其它与工作相关的业务事件了。

3，评审顾客价值

每项需求都应该有满意度和不满意度评分，这个评分反映了价值。通常可以把它附在每项需求后面，更多的情况，可以把它附在每个用例的后面，以反映成功的交付这部分工作对客户的价值。

应该鼓励客户告诉您在每项需求上的价值，如果客户在需求评分上有困难，可以试试自己来完成再看他们的反应，很显然您的结果可能与客户有很大差别，但从这里开始就可以入手寻求一个正确的评分。

在评分之后，需要考虑需求的价值。例如，一项需求的满意度评分是 5，不满意度评分也是 5，那么客户是真得想要这项需求。满意度评分是 2，不满意度评分也是 2，那么客户并不在意最终产品中是不是包括这项需求，那么要么可以从需求规格说明中去掉，要么延迟到下一个版本来实现。

4，评审是否存在冲突的需求

在评审的时候，需要再一次检查有没有冲突的需求。前面已经讨论过，如果一项需求的解决方案妨碍了另一项需求的实现，那么这两项需求就是冲突的。当发生冲突的时候，需要研究成本、风险等因素，把当事人召集在一起看看能否达成某种折中。我们发现，除非极端的情况，利益相关方一般都愿意达成某些折中，如果他们希望体面地解决问题而不是丢面子的话。

5，评审规格说明的二义性

规格说明如果要做到实用，应该避免二义性。不应该是用代词，注意到形容词和副词都可能引起二义性。如果使用“应该（should）”这个词，不要使它暗示该项需求是可选的，但是仅仅注意这些还是不够的。

验收标准是对每项需求量化的机制，从而保证需求的无二义性。我们应该尽可能使每项需求都是可度量可测试的，如果正确的应用了验收标准，那么规格说明书将没有二义性。

对于需求描述当然二义性越少越好，但如果验收标准没有二义性，那么不好的需求描述也不会带来多大的问题。

如果对需求描述的质量比较在意的话，我们建议您随机选取 50 项需求，让一些利益相关方给出他们对需求的解释。如果所有的利益相关方的意见是一致的，就可以认为需求描述的质量是可以接受的。如果对某项需求有争议，那再多选 5 项（随机选取）。重复这个动作，直到规格说明可

以接受，或者争议的列表变得很长，表明说明书的问题显然存在。

如果发现问题很糟糕，可以考虑让一个有资历的技术作者来重写规格说明。

要检查说明书用到的所有术语和命名规则是不是得到明确的定义，如果每个词都有达成共识的定义，而且一致的使用术语，那么整个规格说明的意思就是一致的和无二义的。

6，评审优先级的设定

排列需求的优先级使大家可以选择哪些需求在产品的哪些版本中实现。确定优先级是比较复杂的，因为它牵涉到不同的因素，而这些因素常常互相冲突，利益相关方也可能会有不同的目的，达成一致往往比较困难。优先级问题上，还要考虑功能的关注点、重要性和影响性，综合起来考虑问题。

四、需求评审的困难

我们发现，不少企业在审查需求文档的时候面临着一些困难。下面是一些普遍的问题，并附有解决的方案。

1，大型的需求文档问题

1）没有人可以坚持到审查完大型文档的最后一部分

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。

2）在开发软件需求规格说明时，采用非正式的、渐增式的审查

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，庞大的审查小组问题

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。

设想有一个有 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。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的问题

1）地域上的分散性使需求审查更加困难

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）。

2）可以采用共享网络文件夹形式

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也有助于进行远程讨论。

3）任何方法都不如面对面讨论

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 %。

小结：

需求确认的目的是确保“你做了正确的东西”。只有经过正式评审并且发布的的需求，才可以作为一个基线纳入配置管理系统，从而约束了开发、测试、集成和交付等各项活动。在软件开发生命周期中，需求确认并不仅仅只有一次，而且会与验证交替重复进行。

需求评审需要利益相关方全身心地投入，按照一定的阶段有步骤有目的的确认需求。这种投入可以带来更好的产品质量和客户更大的满意度，也可以降低产品生存期中的维护和客户支持的费用，因此是值得的。

需求管理是软件开发中应该引起足够重视的事情，要采取适当步骤来确保需求集受到管理。需求管理并不是需求开发的后续活动，而是在项目全生命周期中都发挥作用。需求管理的活动包括：获得对需求一致的理解、获得对需求的承诺以及需求变更控制。对需求管理的重视首先来自于需求的不稳定性。要注意到需求并不会冻结，整个项目开发期间应该建立沟通机制，以促进利益相关人之间的沟通，确保需求变更不至于对项目造成破坏性的影响。

7.8 结语：执著的追求卓越

在软件组织中，分析师的作用举足轻重。通过上面的讨论，我们发现需求分析相当专业，而且对软件开发的方方面面都有至关重要的影响。当企业把一个方向的生命线托付给你的时候，责任也是重大的，因此我们必须十分谨慎和细致，最后我给你提如下一些建议：

1，分析师重要的工作是构思创新的产品

在人类历史上，著名的创新总是与爱迪生、贝尔这些振聋发聩的名字联系在一起，似乎只有他们以及那些颠覆性的创新，才构成了人类不断演进的历史。但实际上，企业是为了生存和进化而进行创新，并不能仅寄希望于天才和奇迹。

不思进取的企业将在竞争者的不断追逐之下迅速丧失市场份额和领先地位。随着经济全球化的进程、信息通信技术革命的深入，在日益激烈的商业竞争中，这一情势已经显得尤为突出。今天，一个重要的事实就是，当企业面临瞬息万变的市场环境时，创新已不再是可有可无之物，它已经成为企业生存与发展的必要条件。

2，把创新纳入有效的管理规划之中

企业唯有将创新纳入有效的管理规划之中，遵循明确的指导原则和方法论，进行持续不断的系统化创新，才能长久地保持竞争优势。因此分析师经常应该考虑的问题，是我们如何能够在持续成功的创新之中做到“适者生存”，避免遭遇被市场无情淘汰的商业“达尔文法则”。

今天，网络作为一种改变和决定未来的力量，正在深入社会经济的各个层面，成为提供全新

通信、应用和生活体验的包罗万象的平台，也让每个国家、地区、企业和个人都有能力在这个平台之上通过创新改变自己的竞争力和命运。改革开放近 30 年以来，中国正成为推动世界经济增长的重要引擎，并从“中国制造”向“中国创造”迈进。中国企业不仅需要在技术研发、产品服务、商业流程和模式上开展多渠道的创新，更需从管理思想上认知创新的价值和规律，从而真正掌控创新之舵，驶向充满机遇的蓝海。在这个领域中，分析师是可以大有作为的。

3，分析师需要努力保持简单

分析师的立身之本是分析的头脑，要善于抓住问题的本质，也要善于抓住问题的重点，要学会把纷乱复杂的大问题分解成小问题，更要善于理清问题之间的关系。这种分析方法的锻炼，不仅在需求分析上，即使在其它工作中，也会因此受益无穷。

建议“最简单的解决方案就是最好的”，你不应该过度制作软件。在分析上你不应该描述用户并不真正需要的附加特性一个辅助的原则就是：“模型来自于目的”。

这个原则引发了两个核心的实践。

第一个就是描述模型的文档力求简单明了，切中要害。

第二个就是在分析中要考虑避免不必要的复杂性，以减少不必要的开发、测试和维护工作

4，要善于书写良好的文档

分析师要善于书写文档，要站在读者的角度书写文档，文档应该图文并茂，语言通俗。但是，你的文档只要足够好，并不需要完美无缺，事实上也做不到，因为建模的原则就是“拥抱变化”。你的文档应该重点突出，这样可以增加受众理解它的机会，同样这个文档会不断更新，因此如何管理好文档显得十分重要。

5，迭代和递增的工作

这种迭代和递增的工作，对项目管理和软件产品开发，事实上提出了更高的要求，你必须时时检验你的项目进展，不要使它偏离了方向。

6，执著的追求卓越

良好的需求是产品成功的关键，但是展望未来，我们还会面对很多疑问：我们所处的领域将会如何变化？我们怎样才能应对艰巨的挑战？没有人知道未来是什么，但有一点是明确的，那就是我们的发展受限于增强自身的能力和才干。

自问要从项目中获得什么，实际上也是自问想从人生中获得什么。人生极少会按照自己的职业规划那样发展，而更大的机会也可能是在意料之外不期而至。问题在于，我们做好准备了吗？我们必须执著的追求卓越，只有在执著的追求卓越的旅程中，我们才有可能趋于卓越。最终，我们会为自己的人生而自豪，而这种自豪，源自于我们所做的每一件事情都是卓越的。

附录　用例点项目估算方法

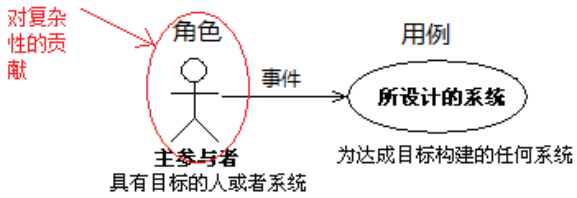
在需求开发中建立良好的用例场景，不但为设计打下了基础，也为项目管理提供了支持。项目管理成功的关键是制定计划，而计划的可靠性来自于对产品规模正确的估算。在众多估算方法中，基于用例点（Use Case Point，UCP）的估算方法是目前具有代表性的一种，它提供了一个规则，避免了同一项目因估算人员不同而导致结果存在较大差异的现象。

一、用例点估算的基本思路

在需求分析阶段，通过完成一个用例模型，就可以包含了所有角色的列表（用户或者外部系统），以及用例业务场景。这些信息能够在项目的早期阶段帮助我们对系统的规模达成一致意见。用例点估算的基本思路如下。

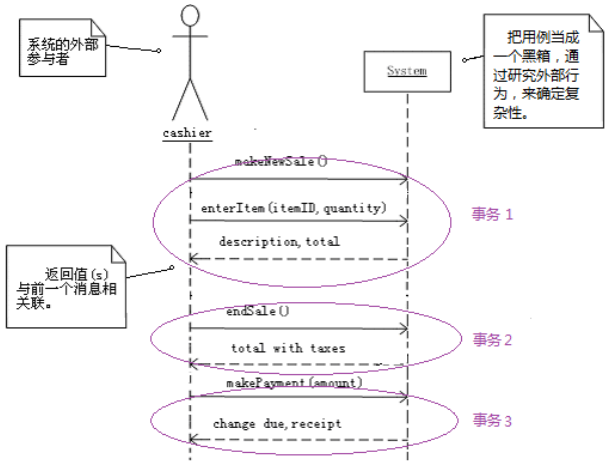
1，角色复杂性

角色（参与者，actor）：这是一个具有行为能力的事物，可以是人（由其扮演的角色来识别），计算机系统，或者组织。分析清楚它用什么方式与系统交互，由此带来不同的复杂度，这就形成了第一组数据：**角色复杂性（UAW）**。

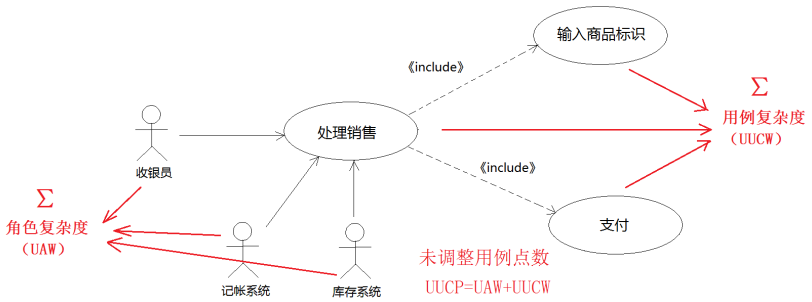


2，用例复杂性

估算是从外部来看系统，需要避开过细的场景步骤，以事务为单位来计算复杂性。什么是事务呢？事务指的是：角色输入一个消息，系统发生反应，然后将处理的结果返回给用户，这就形成一个“环形路线”， 每一个环代表一个完整业务，称之为事务，如下图所示。



统计表明，在一个用例中，事务的多少可以严重影响用例的复杂性，这就得到了**用例复杂性（UUCW）**。把 UAW 与 UUCW 相加，就得到了**未调整用例点数（UUCP）**。



问题在于项目早期能获得这些信息吗？事实上，需求分析的早期就是要和客户一起进行这种精细的交互分析，通过不断地发现事件，弄清楚系统响应，从而把产品如何工作定义清楚。

由于在项目初期利益相关方都在，他们都非常熟悉这些业务事件，也可以去咨询拥有类似系统和背景工作经验的同事，因此，用事务来统计用例复杂性是可行的。

3，复杂性调整因子

在上述基本数据的基础上，还要考虑一些其它因素，对复杂性进行调整，它包括两类：

- **技术复杂性调整因子（TCF）：**由于采取不同的技术路线，技术性能要求的不同，可能带来不同的复杂性，这称为技术复杂性调整因子。
- **环境复杂性调整因子（ECF）：**由于团队特征、需求稳定性等因素，也会带来额外的复杂性，这称为环境复杂性调整因子。

考虑到了这些因素，就可以得出以**用例点（UCP）**表达的系统规模了

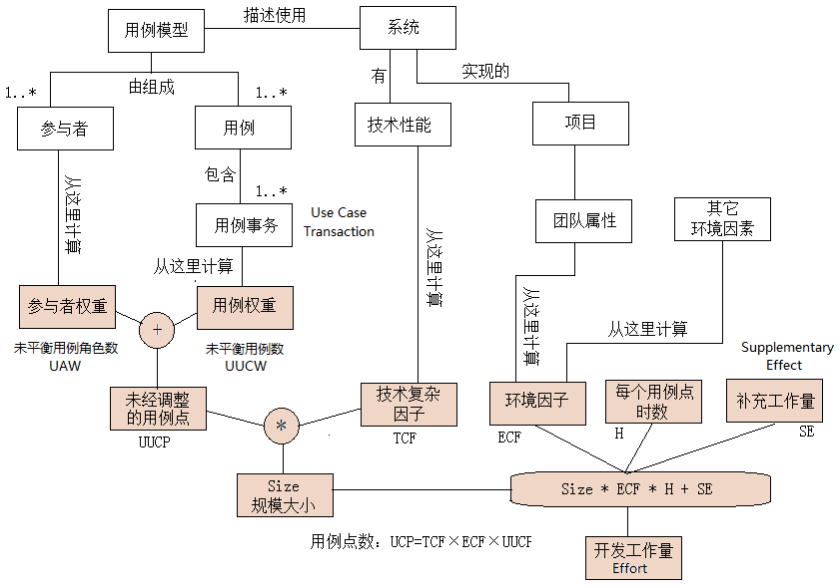
4，工作量的估计

为了估计完成系统的工作量，可以利用历史数据统计出每个用例点所需工作量，这就是生产率（人时/UCP），乘以相应的用例点数就可以得到估算出的工作量。

另外，有些工作量是无法用用例点计算的，比如项目管理、质量保证、配置管理等。这可以加上一个**补充工作量（SE）**，最后得到项目的总工作量。

显然，UCP 估算准确度取决于用例分解层次及对 UML 技术的熟练程度。因此，为了提高估算准确度，要对项目成员及项目经理进行 UML 技术培训，在需求分析的时候，要强调需求用例编写规范。

用例点估算方法的基本概念模型如下图所示。



二、确定未调整用例点数

用例点计算关键是要确定未调整用例点数（UUCP），而这个数据有来自于角色复杂度（UAW）和用例复杂度（UUCW）两个数值之和。

1，角色复杂度（UAW）的计数

1）复杂度计算

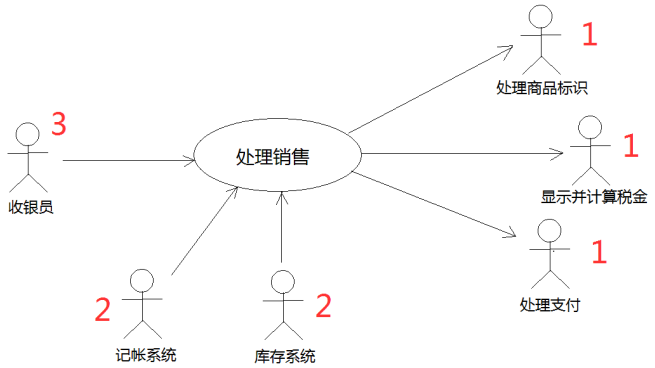
在所计算的部件中，角色（参与者）被划分为简单, 中等，复杂 3 个复杂度等级。其等级划分原则及权重说明如下：

角色复杂度			
序号	复杂性	说明	权重
1	简单	用例角色通过已定义的 API 或接口与系统进行交互	1
2	中等	用例角色通过某种协议（如 TCP/IP）与系统进行交互	2
3	复杂	系统的最终用户（即人）通过 GUI 或 Web 界面与系统交互	3

计算未平衡用例角色数 （Unadjusted Actor Weight，UAW），就是将每一个等级的用例角色数汇总，并乘以对应等级加权求和。

2）计算案例：

有一个超市销售系统中的一个部件（POS 处理销售），其用例如下图所示。针对每个角色根据特点赋予一个复杂度（其中：处理商品标识，显示并计算税金、处理支付是处理相应业务的组件，处理销售在需要的时候通过 API 调用这些组件）：



得出：3Simple×1＝3

2Average×2＝4

1Complex×3＝3

UAW＝3+4+3＝10

2，用例复杂度（UUCW）的计数

1）复杂性和权重

用例的发明者（Jacobson）已经把用例事务定义成从用户到系统，再回到用户的一个“环形路线”。基于每个用例事务数目对用例复杂度的影响，划分为简单，中等，复杂 3 个等级。其等级划分原则及权重说明如下：

用例复杂度			
序号	复杂性	说明	权重
1	简单	用例事务数小于或等于 3	5
2	中等	用例事务数在 4 和 7 之间	10
3	复杂	用例事务数大于 7	15

计算未平衡用例数 （Unadjusted Use Case Weight，UUCW）就是将每一个等级的用例汇总，并乘以对应等级权重，最终求和。

例如：

5 个用例完成业务请求响应环数小于 3 个，4 个用例完成业务请求响应环数在 4～7 个，没有大于 7 个的复杂用例，则：

5Simple×5＝25

4Average×10＝40

0Complex×15＝0

UUCW＝25+40+0＝65

经验：

在使用 UCP 估算方法的过程中，软件功能结构分解对估算的有效性有很大影响。因此，功能分解层次在 3 ~5 层之间为宜。例如在“处理销售”的例子中，分解层次为：产品-产品部件-处理框架三个层次。特别是如果发现一个用例的事务数>>7，就要考虑分解是不是足够，否则会带来过大的误差，而且开发难度大大增加。

3，用例事务不是什么?

在用例点度量中，正确的识别和划分用例事务是成功的关键。我们已经明确了用例事务是什么，下面进一步说明用例事务不是什么，从而进一步明确概念。

1）用例事务不总是一个用例步骤

注意，用例事务并不是定义为“用例流程中的步骤”。用例事务只是一个系统步骤，这是不准确的，因为这会得出步骤越多系统越复杂的结论。

2）用例事务并不是一个数据库活动

尽管很多况下事务要和数据库发生联系，但是在一个环形路线中，系统根本不用查询数据库也是可能的。此时，数据可能来自系统以外或其它位置。因此得出用例事务一定会与数据库中的事务联系起来的结论是不合适的。

3）用例事务不是一个“刺激源”

尽管一次事务总是从一个刺激源（消息）开始（就是用户进行了一项触发系统反应的操作），但刺激源本身并不是完整的事务。例如：在一个场景中，假设已经拥有一个用例描述为：

（1）用户选择一个 X。

...

（n）用户提交。

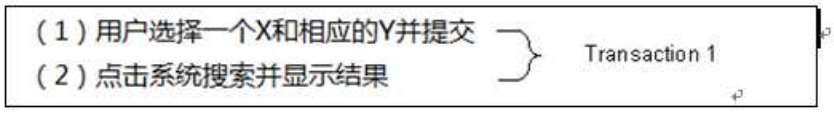
...

这是两个刺激源，但是还不清楚系统是对步骤（1）和（n）中刺激源共同作出了反应，这样两个刺激源可以组成一个事务呢？还是对（1）和（n）分别作出反应，形成两个事务？因此，事务并不取决于刺激源，而是取决于刺激源和回应的组合。

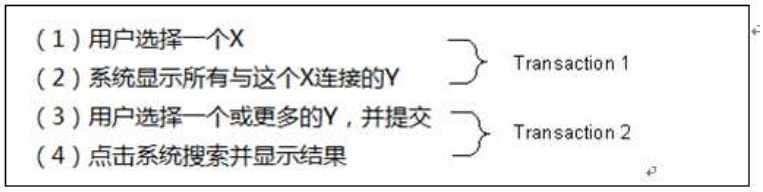
4，度量对于需求的反馈作用

一般的顺序来说，是先有需求后有度量。但是，正是在对规模的度量中，也可以帮助我们发现需求的含糊之处，这种度量对于需求的反作用，显示了度量的价值。

例如：一个用例描述的第一个草稿如下：



我们可能会发现这太简单含糊了，不足以描述事务，于是促使我们进一步的修改文字：



这就对提高需求开发的质量形成了一个很好的反馈。

5，怎样计算用例事务

既然我们已经有了什么是以及什么不是用例事务的清晰解释，让我们迎接在用例中计算事务的一些挑战。

1）不需要关注具体内容

计算事务是从发现工作入口的消息开始，而且只关注往返而不需要关注具体的内容。

2）用例事务的最少数量

一个用例场景至少有一个基本流程，它也至少应该有一个事务。没有事务的流程是没有意义的，因为系统在没有刺激源时什么都不会做，用户在没有弄清系统的反应之前也不会提供任何新的刺激源。

用例几乎都会有描述处理例外的流程（扩展、备选）。每一个例外流程都至少含有一个事务，这点也适用于一个可选择的流程。这就给了一个任何用例场景中用例事务最小数量的指示。

3）显示事务和计算事务

识别出了用例事务，我们是否需要对它们平等的重视？特别是存在某种框架的时候，处理相应的事务并不会形成更多的工作量。例如，一个用例具有十个事务，但是它们中只有七个值得处理，另外三个可以直接利用框架处理，这就不需要计算复杂性。下表显示了这样一个例子。

不同假设性用例计算的事务				
用例	**发现事务**	**计算事务**	**原因**	**UC 权重**
1 申请工作	4	3	1 个可利用框架	简单
2 找工作	3	3		简单
3 评估申请	10	7	3 个可利用框架	平均

4）早期发现用例复杂性

事务分析还可以帮助我们早期发现用例复杂性，如果事务的权重已经到了“复杂”，就要考虑能不能合理的分离，这会对提高开发效率有正面的影响。一般的经验中，过于复杂的用例能够满足不止一个目标，所以，考虑创建一个新的用例就是值得的。

5）难以计算的事务

如果用例使用在缺乏用户交流的情况之下，那么怎样才能把业务的概念转化成一个环形路线？坦白来说，数圈在这里并不适用。我们需要其他的方式来估算这种用例的权重。特别是一些批任务，建议由专家估算来完成。下表显示了它们是怎样在计算表中显示的。

业务与添加的批任务同时计算				
用例	**发现事务**	**计算事务**	**原因**	**UC 权重**
1 申请工作	4	3	1 个可利用框架	简单
2 找工作	3	3		简单
3 评估申请	10	7	3 个可利用框架	平均
4 读取文件到数据库中	批处理	--	专家估算	复杂性

如果批任务要比一个复杂的用例还要大，同样它也应该有不止一个目标，因此这个工作可以分解成更多的用例。如果不能找到一个好的办法去分解一个批任务，还可以转化成用例点概念模型中提到的“补充影响（Supplementary Effect，SE）”。

6，计算用例复杂度（UUCW）的案例

针对前面提到的“处理销售”的案例，用例的场景与事务识别如下：

用例示例	
用例名	处理销售
用例 ID	TSPOS-6
主要参与者：	收银员
相关人员：	顾客、销售公司、政府税务机关、信用卡支付机构
前置条件：	收银员已经得到识别和授权
后置条件（成功后的保证）：	存储销售信息；准确计算税金；更新帐目和库存信息；记录提成；生成收据；记录支付授权服务的许可。
主流程：	1，顾客携带购买的商品到达 POS 机收费口。 2，收银员开始一次新的销售。 3，收银员输入商品标识（TSPOS-6-1）。 T1 输入商品标识 4，系统显示总值计算税金（TSPOS-6-2） T2 显示总值计算税金 5，收银员请顾客付款。 6，顾客支付（TSPOS-6-3）。 T3 处理支付 7，系统记录并显示完整的销售信息，把销售和付款信息发送到外部的记账系统（进行记账和提成）和库存系统（更新库存）。 T4 记录销售信息 8，系统打印发票。 T5 打印发票 9，用例结束 使用框架，所以不予统计。
扩展流程：	*a（*a 代表任何时候的第一种情况），如果在场景中的任何一步检测到异常、若机等现象，系统将失败： 1，收银员重启系统，登录，请求恢复上次的状态。 2，系统恢复重建之前的状态。 T6 异常恢复 2a，系统恢复过程中检测到异常。 1，系统向收银员指示错误，记录此错误，并进入一个清空状态。 2，收银员开始 次新的销售。 3-5a（3-5 代表第 3-5 步之间的任何位置）如果顾客要求收银员从已输入的商品中去掉一个商品： 1，收银员输入商品标识，并输入删除信息。 2，系统显示更新后的累加值。 T7 取消商品 3，从主事件流第 3 步继续执行。 3-5b，如果顾客要求收银员取消交易： T8 取消交易 1，收银员在系统中取消交易。 3-5c，如果收银员希望暂停销售： T9 暂停销售 1，系统记录销售信息，使收银员能够在任何一台 POS 终端上恢复操作。 5a，如果顾客随身携带现金不足，且无法用其他方法付款： 1，顾客告诉收银员，他要取消此次销售，收银员在系统上取消此次销售。 5b，顾客出示优惠卷： 1，收银员记录每张优惠卷，系统记录每张优惠卷以备记账之用，系统扣除相应价值。 1a，输入的优惠卷并不适用所购买的商品。 T10 优惠卷处理 1，系统向收银员指示错误。 2，从主事件流第 4 步继续执行。 T11 处理回扣 8a，如果有的商品有回扣： 1，系统给出回扣表格，并为每个回扣商品提供回扣收据。 8b，如果顾客要求礼物收据（不显示价格）： 1，收银员请求礼物收据，系统给出礼物收据。 T12 打印礼品收据
特殊需求	1，要求收银员使用触摸屏方式操作。 2，支持中文和英文两种语言显示。 3，在步骤 3～7 中，可以插入新的业务规则。 事务数=9 复杂

可以统计出，事务数=9，可见这个用例太复杂了，可以考虑进一步分解，例如可以考虑把第 8 步包括 8a、8b 两个扩展单独做成扩展用例（打印发票），这样就可以减少 3 个事务。在本例中暂不作这个考虑。

计算：1Complex×15＝15

UUCW＝15

这个例子为了简化问题，只考虑了一个用例的情况。

7，计算未调整用例点数（UUCP）

将 UAW 与 UUCW 相加得出未调整用例点（Unadjusted Use Case Point, UUCP）。续上例：

UUCP＝UAW+UUCW＝10+15＝25

用例事务的概念是这样一种特征：它最好与一个环形路线结合，从用户启动的刺激源到系统的反应形成闭合环。与这个概念结合，我们需要对怎样以及什么时候计算事务作出一些判断。它更像是一种艺术，而不是一门科学，与常识和经验一起应用这些方法，就可以帮助我们作出更有效的努力，以早期评价项目的成本。

三、计算复杂度因子及开发工作量

1，计算技术复杂度因子 TCF

需要计算计算技术复杂度因子（Technical Complex Factor, TCF），根据项目复杂度不同，可将 TCF 中每项因子赋予 0~5 间的任意值（评分），因子赋予的分值越高，该因子对项目的影响越大或关联性越强。

技术复杂度因子		
TCF	**说明**	**权重**
TF1	系统分布式程度	2.0
TF2	系统性能要求	1.0
TF3	最终用户使用效率要求	1.0
TF4	内部处理复杂度	1.0
TF5	复用程度	1.0
TF6	易于安装要求度	0.5
TF7	系统易于使用程度	0.5
TF8	可移植性	2.0
TF9	系统易于修改程度	1.0
TF10	并发性要求	1.0
TF11	特殊安全功能特性要求	1.0
TF12	为第三方系统提供直接系统访问	1.0
TF13	是否需要特殊的用户培训设施	1.0

计算 TCF：为表中 TF1~TF13 各项因子 0-5 打分，再将每项因子得分与其对应用权重相乘，然后求和得到 TFactor。由此计算得出，TCF=0.6+(0.01×TFactor)。这个调整量应该在 0.6～1.3 之间。

2．计算环境复杂度因子(ECF)

环境复杂度因子（Environment Complexity Factor, ECF），是根据项目所处的环境（人员、团队等）的不同，将 ECF 中每项因子赋予 0~5 间的任意值。任一因子赋予的分值越高，该因子对项目的影响越大或关联性越强。ECF 因子描述及权重见下表。可以看出，一个非常重要的环境因子是需求的稳定性。需求越不稳定，复杂度（规模）就越大。

环境复杂度因子		
ECF	**说明**	**权重**
EF1	UML 精通程度	1.5
EF2	系统应用经验	0.5
EF3	面向对象经验	1.0
EF4	系统分析员能力	0.5
EF5	团队士气	1.0
EF6	需求稳定度	2.0
EF7	兼职人员比例高低	-1.0
EF8	编程语言难易程度	-1.0

为表中 EF1~EF8 各项因子 0-5 打分，再将每项因子得分与其对应用权重相乘，然后求和得到 EFactor。由此计算得出，ECF=1.4+（-0.03×EFactor）。这个调整量应该在 0.725～1.4 之间。

3，计算软件规模 UCP

用例点数为：

UCP=TCF×ECF×UUCP。

续上例：

假定调整量为：TCF=0.9, ECF=0.905，则

UCP=0.9×0.905×25=20.36(20)。

经验：

由于 UCP 方法没有给出 TCF 和 ECF 各项因子打分的准则，因此需要累积每一次估算时项目经理打分的理由，并最终制定出明确的评分准则。可以按照项目类型收集 UCP 与工作量数据，并引入统计分析方法，建立工作量预测模型。以此避免不同项目类型及因项目成员开发经验不同而对估算结果带来的影响。

4，估算项目开发工作量

利用历史数据给出基于每 UCP 完成的工作量，即生产率（人时/UCP）就可以计算得出项目开发工作量。

UCP 发明人建议：每 UCP 为 16 人时~30 人时，均值为 20 人时，对一个规模为 17 个 UCP 的项目，所需要的开发工作量为：

Effort=UCP×Productivity=20×20=400 人时，约为 10 人周。

最后，在用例点模型中添加有经验得出的补充工作量（SE）（例如项目管理时间，质量保证、集成测试等），然后估算就完成了。

经验：

每 UCP 完成的工作量在不同的企业或者团队是不一样的，一开始采用的数据可能是推荐的数据，当然这是不准确的。当项目完成之后就需要总结一下，对于同样的 UCP 实际用的工作量到底是多少？发生这种差异的原因在哪里？从而修正每 UCP 工作量的计算值，经过几个循环之后，估算的准确度就会大幅度的上升。

度量的目的是为了比较，能够比较的先决条件是统一度量衡。如果一个企业各个项目团队都统一使用 UCP 进行规模估算，并且在项目结束的时候都进行了类似的总结，就有可能建立企业的历史数据库。基于这个数据库，将有利于分析不同的团队、不同类型的人在工作效率上的不同，不但为正确估算打下基础，也为更准确的发现问题和解决问题，为企业管理提供了支持。

统一使用 UCP 进行规模估算还可以迫使需求团队认真编写用例场景，可以帮助我们尽早发现需求开发中的问题（如果难以估算是不是我们的需求做的太粗糙了呢），这对于提升需求开发的质量，进而改善设计、测试以及其它方面的质量，都有着正向的推动力。