# 第2章　需求基础

## 2.1. 需求的定义

需求一直是软件工程中较为模糊的词汇之一[McConnell2000]，提起需求（Requirement），不同背景的人（用户、开发者）会有不同的看法，因此需求是需求工程当中一个非常难以准确定义和解释的概念。

在各种不同的需求词汇解释中，本书更倾向于使用IEEE的需求定义[IEEE1990]：

（1）用户为了解决问题或达到某些目标所需要的条件或能力；

（2）系统或系统部件为了满足合同、标准、规范或其它正式文档所规定的要求而需要具备的条件或能力；

（3）对（1）或（2）中的一个条件或一种能力的一种文档化表述。

IEEE的定义当中同时包括了用户的观点（第一种条件和能力）和开发者的观点（第二种条件和能力），它强调了“需求”的两个不可分割的方面：一、需求是用户为中心的，是与问题相联系的；二、需求要被清晰、明确地写在文档上。

## 2.2. 满足需求就是解决问题

### 2.2.1. 问题与需求

需求源于问题，要准确理解需求，就必须明确它与问题的关系。[Jackson1995a, Jackson1997]认为当现实的状况与人们期望的状况产生差距时，就产生了问题（如图2-1所示）。问题中的差距要么是某些事件、事物的状态不理想，要么是某些事情的发生过程不理想。要解决问题，就需要改变这些事件、事物的状态，或者改变它们状态变化的演进顺序，使其达到期望的状态和理想的演进顺序。

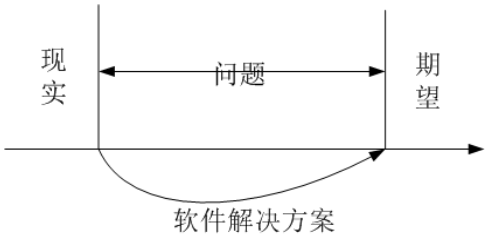


图2-1 问题与软件解决方案关系示意图

人们开发软件系统的目的就是希望用它作为解决方案，解决问题，使得现实改善到期望的状况。解决问题、改善现实、满足用户期望的条件与能力就是需求。

例如，一个利润率仅为2%的企业会认为利润不够高，希望通过开发和应用一个软件系统，能够将利润率提高到5%。那么2%的利润率就是现实，“利润率低”（低了3%）就是企业面临的问题，利润率为5%是期望的状况，将利润率提高3%或者将利润率提高到5%就是需求。

### 2.2.2. 问题解决的两个方面——问题域与解系统

1．问题域

问题在现实世界与软件系统的互动中得到解决[Jackson1995b]。如图2-2所示，软件系统在应用于现实之后，就成为现实世界的一个部分。当然，软件系统不会也不需要与整个现实世界互动，它只需要与现实世界中的一部分互动即可。这个部分就是问题的发生地，也是问题解决的基本范围——解决问题必需涉及的事件和事物，[Jackson1995b]将它们称为问题域（Problem Domain）。

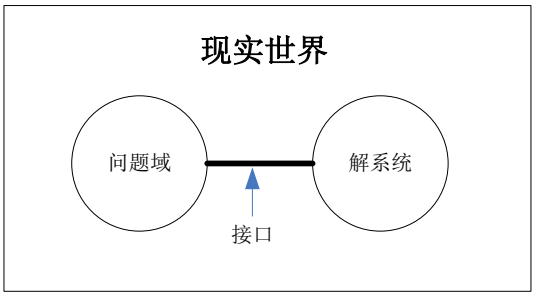


图2-2 问题域与解系统示意图

问题域是需求的背景，要理解需求就必须先理解问题域[Bray2002]。例如，要准确理解需求“将利润率提高到5%”，就需要弄明白利润由哪些部分组成，各自的比例多少，工作是如何完成的……再例如，要准确理解需求“用户可以查询商品详细信息”，就需要了解哪些用户在哪些任务中需要查看哪些商品详细信息……总之，虽然表达期望的需求看起来比较简单，但是只有明白问题域的复杂背景信息，才能真正理解需求的含义。

问题域的背景信息又被称为问题域特性（Problem Domain Feature）。与需求相区别的是，问题域是自治的，它有自己的运行规律，而且这些规律不会因解系统的引入而发生改变。需求是一种对未来的期望，是可以打折、部分满足、甚至不予满足的；而问题域特性是既定现实、可以改善但不能忽视更不能违背的。例如对于需求“将利润率提高到5%”，可以部分满足，只提升3%。但是如果用户的销售市场遍布全国（问题域特性），就不能仅考虑一个地点的销售工作状况。

2．解系统

软件系统通过影响问题域，帮助人们解决问题，所以[Jackson1995b]称之为解系统（Solution System）。在解系统当中，软件起着主要的作用，它是软件解决方案在通用计算机上的实现。

解系统是问题的解决手段，并不是问题的产生地，所以，解系统并不是问题域的一个部分。解系统与问题域之间存在可以互相影响的接口，以实现交互活动。

需求工程师要注意区分用户与软件开发人员在关注点上的不同：用户关注于问题域，软件开发人员更关注解系统。需求工程师扮演着桥梁的作用，一方面使得用户不需要了解和关注解系统，因为用户并不懂得软件开发的专业知识；另一方面使得软件开发者不需要关注问题域，让软件开发者将精力集中到软件构造工作中。尤其需要注意的是：虽然需求工程师通常是技术人员，来自于解系统领域，但是他们也必须要懂得如何站在用户的立场，与用户进行基于问题域的交流。

3．问题域与需求

虽然解决问题和满足需求的手段是引入解系统，但问题和需求都来自于用户。用户关注的是问题域，所以需求是用户对问题域当中的实体状态或事件的期望描述，如R1、R2所示。

R1：一旦书籍被借出，则在归还之前，系统应该不允许它被再次借阅。

R2：如果超过30天的归还期限，系统在书籍归还时应该进行超期处罚。

需求并不针对解系统，它的描述应该尽可能使用问题域的语言，尽量不涉及解系统的专业名词。例如，需求R3、R4中就含有“数据仓库技术”、“客户关系管理系统”、“按钮”、“界面”等多个计算机词汇，是不恰当的。相比之下，R5、R6的描述更能被用户所接受。

R3：系统应该使用数据仓库技术建立客户关系管理系统CRM以扩大5%的销售额。

R4：如果用户在销售列表信息界面选中一个商品，并点击“查看”按钮，系统将显示商品的详细信息界面。

R5：应用系统12个月后，销售额应该扩大5%。

R6：在用户请求查看具体商品时，系统应该显示该商品的详细信息，包括条形码、名称、价格、厂家。

需求开发的最原始出发点就是用户需求，或者需求的源头——问题。

4．解系统与需求规格说明

解系统的核心是软件解决方案和解决方案在通用计算机上的实现。虽然解决方案及其实现都关注于软件系统本身，但相互间也有所不同。解决方案描述的是软件系统与问题域交互的过程，侧重于软件系统中与外界交互的部分。实现部分则主要是软件内部的组成元素、结构关系、物理实现等软件系统的构造要素。需求工程所关心的仅仅是解决方案，不涉及软件的实现细节。

在需求开发过程中，问题域中的用户提出问题与需求。需求工程师接收用户问题与需求，分析问题域背景，建立软件解决方案，并将解决方案传递给后续软件开发者。软件开发者负责将软件解决方案变为软件实现。在整个工作衔接中，需求是用户与需求工程师的协作基础，解决方案是需求工程师与软件开发者的协作基础。

因为解决方案从对外交互的方式定义了软件系统的功能，所以解决方案被称为软件系统的需求规格说明（Specification）。需求开发最终的目的就是提供一个高质量的需求规格说明，它定义了一个能够解决用户问题、满足用户需求的软件对外交互方案，是后续软件开发活动的工作基础。需求规格说明的典型描述方式是：“系统能够...”或者“如果用户提出...请求，那么系统应该...”。

[IEEE1990]将规格说明定义为：以一种完全的、精确的、可验证的方法规定系统或部件的需求、设计、行为或者其他特性的文件，并经常指明判定这个规定是否满足的过程。

[IEEE1990]将需求规格说明定义为：规定系统或部件的需求的文档。典型地包括功能需求、性能需求、接口需求、设计需求和开发标准。

问题域、需求、解系统、需求规格说明之间的关系示意如图2-3所示。

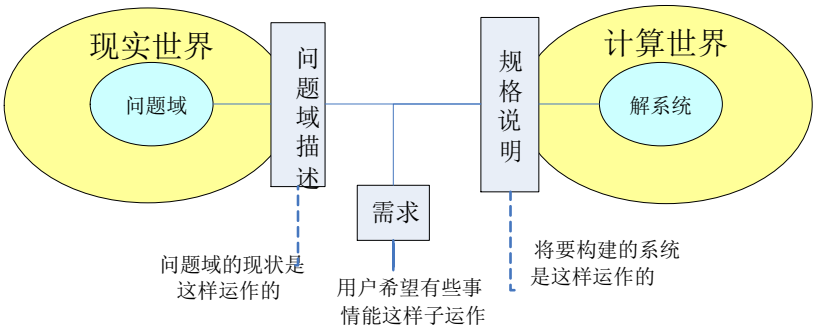


图2-3 问题域、需求、解系统、需求规格说明关系示意图

### 2.2.3. 问题解决的基础——模拟与共享现象

处于问题域之外的解系统之所以能解决问题域中的问题，是因为问题域与解系统之间存在有效的互动，并在互动中互相影响。而问题域与解系统能够形成互动的基础是解系统部分模拟了问题域，[Jackson1995b]将这种模拟性称为共享现象（Share Phenomenon）。

初看上去，问题域与解系统原本是两个相互独立的系统，相互独立性使得它们之间难以互相影响。但是一旦认识到解系统对问题域的模拟性，它们就会变得紧密联系而不是相互独立，互相影响也自然会形成。

简单地将，模拟是指其中一方仿制另一方的信息。解系统对问题域的模拟则更加复杂一些，它们之间的模拟性带有交互性：一方面，解系统会在自身中保持一份与问题域现象一致的信息，并随着问题域现象的变化而变化；另一方面，问题域会期待着在解系统中看到一致的信息，并据此展开自己的行为。

例如，一个图书馆中有图书、借书人、借书规则等现实信息，图书馆管理系统中就会建立相应的数据表（Table-Book；Table-Borrower；Table-Rule），这个是简单的仿制。如果图书馆中有一本书《软件需求工程》因为磨损而报废了，那么Book表中就需要删除“Name=《软件需求工程》”的数据行。如果在表Borrower中有一个数据行是“Name=张三，BorrowedNum=5”，那么管理员就会认为张三已经借走了5本书。如果张三实际上只借走了3本书，那么管理员就会认为管理系统出了错误，工作不再正常。这种带有交互性的模拟就是解系统与问题域能够互相影响的原因和途径。

解系统与问题域模拟的交互性其实是由人在意识中强制建立的。如果用户并不将现实发生的情况实时地输入到软件系统中，或者用户在工作时完全忽视软件系统提供的输出，那么软件系统就会失去影响和改变现实的能力，就不可能解决现实问题或者满足现实需求。这充分说明软件系统必须得到用户的认可，否则就会失去价值。

因为问题域与解系统并不相互独立，所以相比于图2-2，[Jackson1995b]认为图2-4更准确地描述了问题域与解系统的关系。

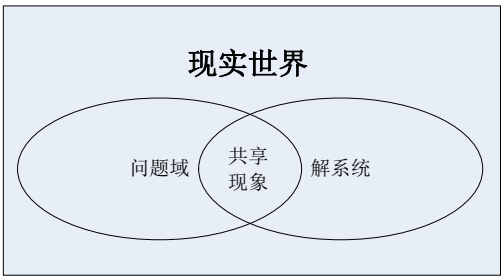


图2-4 问题域和解系统的关系

共享现象就是解系统所模拟的问题域部分，该部分知识在两个系统中同时存在。除了共享现象之外，问题域还有一些没有被解系统模拟的知识，因为现实世界非常复杂，不可能也没必要在解系统中完全重现。例如，一本图书的质地、每页纸是否损坏、是否被涂抹等信息不需要在软件系统中建模。解系统会从特定的角度对问题域知识进行抽象和简化，并模拟简化后的知识。

除了包含共享现象的知识模型之外，解系统也有一些并非来自于现实模拟的特征，例如数据库管理系统的选择、模型的范式化、索引的建立等等，这些因素并不对应于任何的问题域知识，却是解系统必不可少的部分。

### 2.2.4. 问题解决的方法——直接与间接

因为模拟后的知识——共享现象，是解系统的一部分，所以解系统可以对其施加操作，适当改变这些知识，知识的改变会通过交互性传递给问题域，问题域在会接受改变的基础上继续规律性的运作，使得问题得以解决。例如，要用软件跟踪记录用户在银行的存储款情况，可以将用户在银行的账户建模为表Account（ID, Name, Balance），如果用户张三在现实世界中存储了1000元钱，那么软件系统就给“Name= ‘张三’”的Account记录的Balance增加1000。当其他银行职员看到软件系统中这条记录时，也会接受张三存储了1000元这个现实。再例如，仓储用户想降低库存成本，软件系统可以将出入库信息建模为表Import和Export，然后软件系统通过计算表Import和Export过去一段时间的数值，给出将来一段时间会有多少仓储差额的预测数值，仓储人员就会接受该预测数值以保持最佳库存，自然能降低库存成本。

模拟并操纵共享现象是软件系统满足需求最直接的方法，但有些情况下软件系统也会使用间接的方法解决问题：软件系统操纵共享现象影响问题域的一部分，然后利用问题域内在的规律性自动影响另一部分。例如，图书管理员希望能够督促那些超期的借书者尽快归还图书，直接的解决方式是软件系统将借书者的联系方式建模为表Contact，并自动使用Contact的数据完成督促告知（比如发送邮件）。但是如果软件系统中没有存储借书者的联系方式Contact，也就是说软件系统的共享知识中没有解决问题需要的信息，就只能通过间接的方式来满足需求了，这时软件系统可以将超期者的名单告知图书管理员，然后由图书管理员逐一电话进行督促归还。

考虑问题解决和需求满足的方法时，成本是重要的因素，如果成本能够接受，就尽量使用直接的方式进行解决，如果成本太高，就可以折中使用间接方式进行解决。

间接解决方式也提醒需求工程师，考虑到问题域内的规律性，在设计解决方案时要防止解系统的引入在问题域当中引发未预见的连锁反应，这种反应可能会使得解决方案达不到预期目的，甚至造成不良的负面效

应。

防止未预见的连锁反应尤其要关注间接特性。间接特性不会与解系统直接交互，不会受到解系统的直接影响，但是却可能因为连锁反应而受到影响。例如在一个车辆调度系统当中，由调度员根据用车的请求统一安排车辆的使用，安排过程中车辆的驾驶员并不和调度系统进行直接的交互，但在车辆和驾驶员固定配对的情况下，对车辆的调度就决定了驾驶员的工作安排，因此，车辆调度的方法自然会影响驾驶员的工作情况，如果他们的相关因素没有被认真对待，就可能导致不良后果，致使在驾驶员请假的时候进行了车辆分配或者驾驶员的工作量分配不均等。

### 2.2.5. 问题解决方案——需求规格说明

因为解系统解决问题的方法是改变共享知识，影响问题域的运行，进而满足用户的需求。所以需求规格说明主要包括两个部分（如图2-5所示）：（1）对共享现象（模型）的描述；（2）系统对共享现象所施加的操作的描述。这也就是软件系统中最为核心的两个部分：数据与功能。

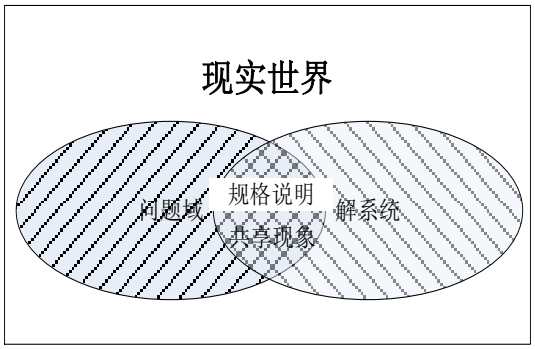


图2-5 需求规格说明

### 2.2.6. 问题解决的困难性

如果拥有描述明确的问题域特性E和定义良好的系统行为S，就可以很容易的发现将系统应用到问题域后会产生的效果。这种效果如果符合预期的需求R，那么系统就是满足人们需要的系统。所以需求工程的目的就是根据E，构建S，使得E和S的联合作用效果符合需求R：$E,S \mapsto R$。

从这里可以发现需求工程的困难之处：（1）不存在描述明确的E；（2）不存在确定的针对S的评估标准R；（3）根据问题域特性和系统行为推测系统应用效果是简单的推理过程，即 $E,S \mapsto R$ 是简单的，但根据问题域特性和期望的系统应用效果构建系统行为的过程是困难的，$E,R \Rightarrow S$ 是一个创造性的过程。

这些困难也恰好说明了需求工程的主要工作：（1）进行需求开发，确定用户的期望效果R；（2）研究问题背景，描述问题域特性E；（3）构建解系统，描述解系统行为S，使得 $E,S \mapsto R$。

## 2.3. 需求和问题都有层次性

需求是问题解决的期望，问题是可大可小的，期望自然也是可大可小的。例如，一个超市收银员的问题可以是“工作效率太低（大）”，也可以是“商品销售过程太繁琐（中）”，还可以是“销售时计算总价不方便（小）”。

问题和期望粒度不同的现象被称为需求的不同抽象层次[Bühne12004]。如图2-6所示，需求最为常见的抽象层次有三个[Wiegers2003, Kauppinen2005]：业务需求（Business Requirement），针对整个业务的期望，例如R7；用户需求（User Requirement），针对具体任务的期望，例如R8；系统级需求（System Requirement），针对用户与系统一次交互的期望，例如R9。

R7：在系统使用3个月后，销售人员进行销售处理的工作效率应该提高20%。

R8：收银员可以使用系统完成销售处理。

R9：在收银员请求计算已输入商品的总价时，系统应该根据规则Rule1计算总价并显示。

Rule1：总价=∑（价格×数量×折扣）；

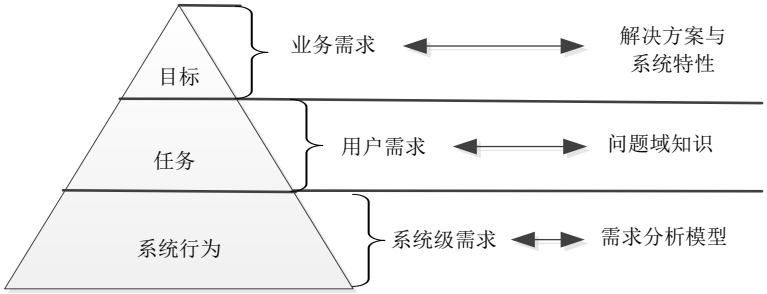


图2-6 需求的层次性

### 2.3.1. 战略问题与业务需求

业务需求是抽象层次最高的需求，是系统建立的战略出发点，表现为高层次的目标（Objective），它描述了组织为什么要开发系统。例如超市管理系统可以有业务需求BR1～BR4。业务需求通常来自项目的投资人、购买产品的顾客、实际用户的管理者、市场营销部门或产品策划部门。

BR1：在系统使用6个月后，商品积压、缺货和报废的现象要减少50%

BR2：在系统使用3个月后，销售人员工作效率提高50%

BR3：在系统使用6个月后，店铺运营成本要降低15%

范围：人力成本和库存成本

度量：检查平均每个店铺的员工数量和平均每10,000元销售额的库存成本

BR4：在系统使用6个月后，销售额度要提高20%

最好情况：40%

最可能情况：20%

最坏情况：10%

业务需求必须是可验证的，其验证标准可以是一个数值指标，例如BR1~BR4，也可以是一个直接的有无、是否等判定，例如BR5、BR6验收时就是有与没有的判定。

BR5：跟踪记录储户的存取款情况

BR6：跟踪记录VIP顾客信息

业务需求可验证的数值指标是通过研究问题域的背景资料得出的，例如如果大多数商品从入库到出库的

销售周期是6个月，那么就可以将BR1的第一个条件设定为“系统使用6个月”，如果详细列举原来导致商品积压、缺货和报废的原因及比率，将软件系统能解决的原因及比率累加后达到50%，就可以将BR1的后一个验证条件写为“商品积压、缺货和报废的现象要减少50%”。再例如，如果收银员从新手到熟练使用系统的培训周期为3个月，就可以将BR2的第一个条件写为“系统使用3个月后”，如果实例研究中收银员使用与不使用系统的工作时间为1：2，就可以将BR2的第二个条件写为“销售人员工作效率提高50%”。

为了满足用户的业务需求，需求工程师需要描述系统高层次的解决方案，定义系统应该具备的特性（Feature）。高层次的解决方案及系统特性指出了系统建立的方向，参与各方必须就它们达成一致，以建立一个共同的前景（Vision），保证涉众朝着同一个方向努力。以支持业务需求的满足为衡量标准，系统特性说明了系统为用户提供的各项功能，它限定了系统的范围（Scope），定义良好的系统特性可以帮助用户和开发者确定系统的边界。

为满足超市管理系统的业务需求BR1～BR4，可以设计特性为SF1~SF10的高层次解决方案。其中SF1、SF3、SF7针对BR1与BR3，SF2针对BR1与BR4，SF4、SF5、SF8、SF9针对BR4，SF10针对BR2与BR3

SF1：分析店铺商品库存，发现可能的商品积压、缺货和报废现象

SF2：根据市场变化调整销售的商品

SF3：制定促销手段，处理积压商品

SF4：与生产厂家联合进行商品促销

SF5：制定促销手段进行销售竞争

SF6：掌握员工变动和授权情况

SF7：处理商品入库与出库

SF8：发展会员，提高顾客回头率

SF9：允许积分兑换商品和赠送吸引会员的礼品，提高会员满意度

SF10：帮助收银员处理销售与退货任务

### 2.3.2. 任务问题与用户需求

高层次的目标是由组织的决策者提出的，但普通用户才是组织当中任务的实际执行者，只有通过一套具体并且合理的业务流程才能真正的实现目标。用户需求就是执行实际工作的用户对系统所能完成的具体任务的期望，描述了系统能够帮助用户做些什么。用户需求主要来自系统的使用者——用户。在有些情况下，系统的直接用户是不可知时，例如通用的软件系统或者社会服务领域的软件系统等，所以用户需求也可能来自间接的渠道，例如销售人员、售后支持人员等。

例如，在超市管理系统之中，收银员用户的需求如UR1～UR5所示。

UR1：收银员可以使用系统逐一记录销售的商品

UR2：收银员可以使用系统计算商品账单并处理付款情况，账单计算需要使用促销策略

UR3：收银员可以使用系统为顾客打印收据

UR4：收银员可以使用系统退回顾客已经购买的商品

用户需求是对任务的期望，所以其基本表达方式为“××用户可以使用系统完成××任务”。用户任务应

该是目标性、有价值的活动。例如在ATM机系统中，取款、存款、转账都是合理的用户需求，因为它们各自代表了用户的一个目标，但“向ATM中插入银行卡”就不是一个合理的用户需求，因为用户不会无目的的“向ATM中插入银行卡”。再例如，UR1是一个合理的用户需求，因为它表达了收银员的一个任务，但是“收银员使用扫描仪扫描商品条形码”就不是合理的用户需求，因为收银员的目的是记录商品，而扫描条形码只是手段。

用户的任务可以有粒度不同的抽象表述，大的任务可以包含（分解为）小的任务，例如“销售”是任务，可以分解为UR1~UR3。所以UR1~UR3是合理的，但UR4也是合理的。在实践当中，用户需求到底使用哪个粒度与抽象层次，要依据软件系统的复杂度而定。本书建议使用UR1~UR3的粒度，它们的特点是无法再进行任务分解。但是在比较复杂的系统中，抽象度稍高的用户需求也是可以接受的。还可以为用户需求建立嵌套层次结构，例如将UR1~UR3命名为UR4.1~UR4.3以表明它们是对UR4的细化。

UR4：收银员可以使用系统完成销售处理过程

需要注意的是：在三个层次中，只有用户需求在表述时在不可验证性上要求较为宽松。因为用户需求具有下面几个特点：①模糊、不清晰。用户需求允许适度使用形容词和副词，使得描述常常带有模糊和不清晰的特性；②多特性混杂。在用户进行需求描述时，常常将功能需求和非功能需求混杂在一起；③多逻辑混杂。用户需求是对用户任务的描述，而任务本身往往含有前后相继的多个逻辑处理过程，即一个任务需要进行多次的系统交互才能够完成。

用户需求表达了用户对系统的期望，但是要透彻和全面的了解用户的真正意途，仅仅拥有期望是不够的，还需要知道期望所来源的背景知识。因此，对所有的用户需求，都应该有充分的问题域知识作为背景支持。而在实际工作当中，用户表达自己的期望时，通常不会提及需求所涉及问题域知识，所以需求工程师需要根据用户的需求整理完整的问题域知识。例如对UR1，需要补充问题域知识如下：

Data：需要记录的商品信息包括ID、名称、描述、价格、特价、数量、总价；

Format：ID为×××格式的条形码；

Rule：总价= 特价×数量。

### 2.3.3. 系统行为问题与系统级需求

业务需求描述了系统的目标与效益，适合决策者，用户需求描述了具体任务，适合用户，但它们都不适合于软件开发者。适合软件开发者的需求层次是系统级需求，它关注的是软件系统的行为，尤其是系统与外界的交互行为：在接收到一个外界请求时，软件系统应该给外界提供的响应。所以系统级需求的典型形式是“系统可以×××”或者“在××用户提出××请求时，系统应该×××”。

例如，对用户需求UR1，可以将之转化为如表2-1所示的系统级需求，其中SR1及其嵌套、SR2属于细节的功能需求，DR1、Rule是对功能需求的补充需求，分别属于数据需求和规则约束需求（具体参见2.4节）。

表2-1 系统级需求示例

| 需求ID | 需求描述 |
| --- | --- |
| SR1 | 在收银员输入商品目录中已存在的商品标识时，系统显示输入商品的信息，包括ID、名称、描述、价格、特价、数量、总价。ID的规则参见DR1 |

表2-1 系统级需求示例

| 需求ID | 需求描述 |
| --- | --- |
| SR1.1 | 在收银员要求输入数量时，系统应该允许收银员输入商品的数量 |
| SR1.1.1 | 在收银员输入大于等于1的整数时，系统修改商品的数量为输入值，并更新显示 |
| SR1.1.2 | 在收银员输入其他内容时，系统提示输入数量无效 |
| SR1.2 | 系统应该计算并显示输入商品的总价 |
| SR1.2.1 | 如果存在适用（商品标识、今天）的商品特价策略（参见Rule3），系统将该商品的特价设为特价策略的特价，并计算分项总价为（特价×数量），并将其计入特价商品总价 |
| SR1.2.2 | 在商品是普通商品时，系统计算该商品分项总价为（商品的价格×商品的数量），并将其计入普通商品总价 |
| SR1.3 | 在显示商品信息0.5秒之后，系统显示已输入商品列表，并将新输入商品添加到列表中 |
| SR2 | 在收银员输入商品目录中不存在的商品标识时，系统不予处理 |
| DR1 | ID是规则为...的商品条形码 |
| Rule3 | 适用（商品标识，参照日期）的商品特价促销策略：<br>（促销商品标识=商品标识）而且（（开始日期早于晚于参照日期）并且（结束日期晚于等于参照日期）） |

系统级需求比用户需求更加详细和准确，包含更多的技术细节[Maiden2008]。例如，表2-1的描述中SR1.1.2、SR2都是技术实现问题，不是任务的业务部分。对于开发人员来说，系统级需求对功能的定义更准确，每一条系统级需求恰好就是开发人员需要完成的一个设计决策。例如，表2-1中的每一条需求都是开发人员需要考虑的一个决策点，依赖于表2-1进行的开发工作更不可能出现决策遗漏与偏差。

因为系统级需求比用户需求更细节，数量更多，所以为了节省开发时间，在实际开发中有些开发者更愿意使用用户需求而不是系统级需求作为后续开发的基础。这种做法在一定程度上是可行的，但也有自己的风险：用户需求不够准确，给开发人员提供了过大的发挥空间，可能导致开发人员在需求理解上出现偏差[Maiden2008]，因为开发人员未能从用户需求描述中得到足够信息以准确地完成设计与实现工作，就只能以自身的经验进行假设[Albayrak2009]，这些假设未必是合理的。如果可能的话，本书还是建议开发者尽可能使用系统级需求作为后续开发的基础。

一个软件系统的系统级需求集合定义了相应业务需求及用户需求的解决方案，构成了需求规格说明的主体部分。解系统及其需求规格说明都是不属于现实的，是人们为了解决问题而构建的，所以系统级需求无法直接从现实中得到的。相比之下，业务需求直接或间接地来源于决策者，用户需求直接或间接地来源于用户，而系统级需求就只能通过技术加工获得。技术加工过程被称为需求分析，其源对象是用户需求及相关的问题域知识，处理方式是利用分析方法、技术建立需求分析模型并基于需求分析模型将用户需求及问题域知识转化为系统级需求。将用户需求转化为系统级需求是一个复杂的活动，详细情况请参见本书需求分析部分。

### 2.3.4. 需求开发要遵从层次性

功能需求的三个不同抽象层次之间有紧密的联系，如图2-7所示。

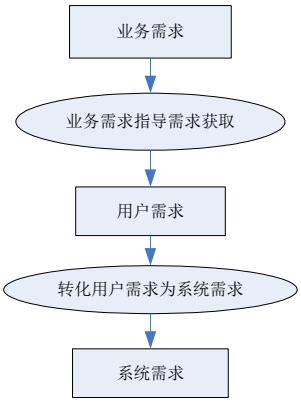


图2-7 不同抽象层次需求之间的联系

在三个不同层次的功能需求当中，业务需求具有明显的目的性和较高的抽象性，比较容易获取和确认。所以需求开发往往从获取业务需求开始。有了业务需求之后，就可以确定系统的最终目标和努力方向，进而指导具体的需求获取活动，发现用户需求。用户需求经过明确和细化的处理，可以转化为系统级需求。

从另一个角度讲，系统开发者理想中的需求是系统级需求，因为开发者可以直接将系统级需求映射为系统行为 ,进行设计和开发。

但是因为系统级需求是无法直接或间接从现实中获取的，所以开发者只能退而求其次——获取用户需求，并通过分析活动将其转化为系统级需求。

随之而来的另一个问题是，用户需求的获取过程非常复杂，涉及众多参与者和诸多问题，要成功的获取用户需求，就首先要协调参与者的立场和问题的范围，而这只能通过对业务需求的处理进行解决——根据业务需求，协调涉众的立场，限定问题的范围，指导用户需求的获取过程。

## 2.4. 需求的分类与表述

需求需要被文档化表述，这要求需求工程师搞清楚需求有哪些类型以及每种类型如何进行表述。

### 2.4.1. 需求的分类

分类的目的是为了区别对待，否则分类就失去了意义。需求分类的目的是为了将需求划分为需要区别对待的不同类型，每种类型会被文档化到不同的部分，服务于不同的读者、不同的目的。

1.广泛意义上的需求谱系

人们在软件开发中谈论“需求”时，通常是指软件需求，本书中使用“需求”一词时也主要用来指称软件需求。但有时“需求”一词也会别用来指称其他类型的需求，为了能够更清晰地理解后面的需求分类，这里还是要区分一下不同的“需求”指称，如图2-8所示。

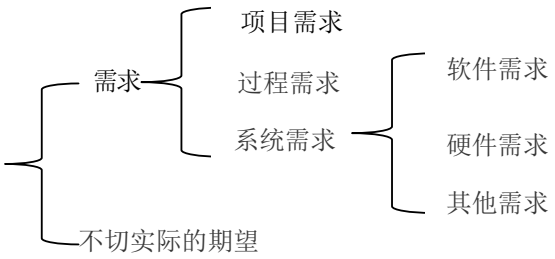


图2-8 “需求”一词的常见含义及其关系

“需求”一词可能被用来指称针对项目的期望，例如R10、R11，它们被称为项目需求。项目需求针对的对象是作为项目核心的计划，包括项目的成本、资源、时间、进度等。

“需求”一词可能被用来指称针对开发过程的期望，例如R12、R13，它们被称为过程需求。过程需求针对的对象是软件开发过程，包括开发人员、工具、方法等。

R10：项目的成本要控制在60万元人民币以下。

R11：项目要在6个月内完成。

R12：在开发中，开发者要提交软件需求规格说明文档、设计描述文档和测试报告。

R13：项目要使用持续集成方法进行开发。

要解决一个问题，人们需要将软件、硬件和人力资源联合起来，这种联合的形式被称为系统工程，包括软件工程、硬件工程和人力资源管理。虽然在系统工程中，软件可能处于最为重要的地位，但是硬件与人力也不可忽视。因此，人们在表述需求时，除了会表达对软件的期望之外，也可能会表达对硬件、人力等因素的期望。这样，所有针对系统工程的需求都被称为系统需求，其中与硬件相关的需求被称为硬件需求（如R14），与软件相关的需求被称为软件需求，与人力资源相关的需求以及软件、硬件、人力之间协同的需求被称为其他需求（如R15）。

R14：系统要购买专用服务器，其规格不低于....。

R15：系统投入使用时，需要对用户进行1个星期的集中培训。

在软件开发项目中还有一个不得不强调的“需求”形式是不切实际的期望。严格来说，不切实际的期望不属于需求，因为它虽然表达了一种期望，但却是根本无法实现的期望。常见的不切实际期望有三种类型：技术上不可行，例如R16；在有限的资源条件下不可行，例如R17（财务分析系统非常复杂，比整个销售系统都要复杂）；超出了软件所能影响的问题域范围，例如R18（因为软件系统根本无法限制收银员的行为，正确的形式应该如R19所示）。

R16：系统要分析会员的购买记录，预测该会员将来一周和一个月内会购买的商品；

R17：系统要能够对每月的出入库以及销售行为进行标准的财务分析；

R18：在使用系统时，收银员必须要在2个小时内完成一个销售处理的所有操作。

R19：如果一个销售处理任务在2个小时内没有完成，系统要撤销该任务的所有已执行操作。

从严格的意义上来说，项目需求与过程需求都不能算是需求，因为它们并不是用户对问题解决的期望，而是客户对软件开发活动本身的要求。硬件需求与其他需求也不属于用户对问题解决的期望，而是为了让软件能够成功运营而需要适应的环境与活动。但是在文档化需求的材料中，经常会出现项目需求、过程需求、硬件需求和其他需求，因为它们对需求工程师及开发者正确理解软件需求甚至整个产品具有极其重要和不可缺少的作用[Gotel2006]，所以它们经常出现在需求文档中（一般位于非主体部分，例如需求文档的开头或末尾部分），供项目管理者和系统工程师阅读。

2.严格意义上的软件需求分类

从严格意义上将，软件需求是直接或间接关系到软件系统功能的期望。根据不同的分类标准，可以将需求分成不同的种类。在各种需求的分类当中，最常见的是[IEEE1998]的分类，[IEEE1998]将需求分成下列类别：

- 功能需求（Functional Requirement）：和系统主要工作相关的需求，即在不考虑物理约束的情况下，用户希望系统所能够执行的活动，这些活动可以帮助用户完成任务。功能需求主要表现为系统和环

境之间的行为交互。

- 性能需求（Performance Requirement）：系统整体或系统组成部分应该拥有的性能特征，例如CPU使用率、内存使用率等。
- 质量属性（Quality Attribute）：系统完成工作的质量，即系统需要在一个“好的程度”上实现功能需求，例如可靠性程度、可维护性程度等。
- 对外接口（External Interface）：系统和环境中其他系统之间需要建立的接口，包括硬件接口、软件接口、数据库接口等等。
- 约束（Constraint）：进行系统构造时需要遵守的约束，例如编程语言、硬件设施等。

除了上述5种明确的软件需求类别之外，[IEEE1998]还指出项目中也可能会出现数据需求（Logical database requirements）等其他特殊类型的需求。

除功能需求之外的其他四种类别需求又被统称为非功能需求（Non-Functional Requirement）。在非功能需求当中，质量属性对系统成败的影响极大，因此在某些情况下，非功能需求又被用来特指质量属性。

### 2.4.2. 功能需求

功能需求是软件系统需求当中最常见、最主要和最重要的需求，同时它也是最为复杂的需求。

通常，一个软件系统的绝大部分需求都是功能需求。虽然在类别划分上，功能需求只是5种类别之一，但在比例上，功能需求占所有需求的90%以上也并不例外。进行这样不均衡比例的划分，是因为功能需求的处理方式是一致的。功能需求是一个软件产品得以存在的原因，是软件系统能够解决用户问题和产生价值的基础，也是整个软件开发工作的基础。所有的开发者都需要了解功能需求。在复杂的系统中，功能需求数量太多，所以需要将它组织为多个独立部分，然后按照分工原则由不同的开发者来处理不同的部分。

因为在大规模软件系统中，其功能需求比较复杂，所以它是最需要按照三个抽象层次进行展开的需求类别，也就是说功能需求的开发要围绕“目标→任务→交互”（例如BR2→ UR1～UR4 ，UR1→SR1～SR2、DR1、Rule3、 ）的路线进行，对“目标”、“任务”和“交互”三个概念的关注是功能需求开发的重中之重。

### 2.4.3. 性能需求

[IEEE1990]对性能的定义是：一个系统或者其组成部分在限定的约束下，完成其指定功能的程度，例如速度、精确性、内存使用程度等。性能需求定义了系统必须多好和多快的完成专门的功能。

常见的性能需求包括：

（1） 速度（Speed），系统完成任务的时间，例如PR1。

PR1：所有的用户查询都必须在10秒内完成。

（2） 容量（Capacity），系统所能存储的数据量，例如PR2。

PR2：系统应该能够存储至少10万条销售记录。

（3） 吞吐量（Throughput），系统在连续的时间内完成的事务数量，例如PR3。

PR3：解释器每分钟应该至少解析5000条没有错误的语句。

（4）　负载（Load），系统可以承载的并发工作量，例如PR4。

PR4：系统应该允许200个用户同时进行正常的工作。

（5）　实时性（Time-Critical），严格的实时要求，例如PR5。

PR5：监测到病人异常后，监控器必须在0.5秒内发出警报。

性能需求的定义要适合于运行环境，过于宽松的性能要求会带来用户的不满，过于苛刻的性能要求会给系统的设计造成不必要的负担，所以给出一个合适的量化目标是非常关键的，但同时也是非常困难的。更加常见的方法是在限定性能目标的同时给出一定的灵活性（例如PR6）或者给出多个不同层次的目标要求（例如PR7）。

PR6：98%的查询不能超过10秒。

PR7：（最低标准）在200个用户并发时，系统不能崩溃；

（一般标准）在200个用户并发时，系统应该在80%的时间内能正常工作；

（理想标准）在200个用户并发时，系统应该能保持正常的工作状态。

将性能需求划分为独立的类别，文档化为独立的部分，是因为它有其他类别需求都不具备的动态性——理论上说，只有开发完成并实际运行系统，才能确定软件系统的性能。实际工作中，软件体系结构师、系统工程师等人员需要特别关注性能需求，必要时需要专门进行模拟。

### 2.4.4. 质量属性

1.质量属性的概念

在软件系统的开发和使用过程当中，人们很自然的关注系统的功能，它是系统能够为用户提供帮助的第一要素，但成功的软件系统除了满足功能性需求之外，还需要满足更多的要求，例如易于使用、少出错等等。

功能性需求是用户对软件系统的显式要求，用户在软件系统创建之前就可以清晰的向开发者表达这种要求。而非功能需求属于隐式要求，用户在软件系统创建之前无法清晰的告诉开发者他们希望该系统具备什么样的非功能性特征。但是在软件系统投入使用之后，他们却可以快速的判断出软件系统的哪一部分非功能需求不满足他们的条件。例如，在市场买一双鞋子时，对于鞋子功能（休闲、跑步还是踢足球）的要求是显式的，但是对鞋底是否会脱胶、鞋面坚韧度等特性的要求就是隐式的，虽然不会有明文规定鞋底不能脱胶，但是一旦脱胶就会被认为是鞋子质量不合格。一般认为，具备职业素质的人员能够在用户不提及的情况下认识到用户的隐式要求，否则该人员就是不合格的。

要建立成功的软件系统，系统必须满足显式的及隐含的各种要求。系统为满足显式的及隐含的要求而需要具备的要素称为质量。

为了度量一个系统的质量，人们通常会选用系统的某些质量要素进行量化处理，建立质量特征，这些特征称为质量属性。

需要注意的是质量属性需求包含性能需求，只是性能需求比较特殊，所以被独立为单独的类型。

实际工作中，软件体系结构师会比较关心质量需求，因为妥善解决质量问题是软件体系结构的主要工作[Clements2007]。

2.质量模型

为了更好地根据质量属性描述和评价系统的整体质量，人们从很多质量属性的定义当中选择了一些能够相互配合、相互联系的特征集，它们被称为质量模型。最为常见的质量模型有[ISO/IEC 9126-1]和[IEEE1016-1992,1998]两个，分别如表2-2、2-3所示。

表2-2 ISO/IEC 9126-1的质量模型

| 特征 | 子特征 | 简要描述 |
| --- | --- | --- |
| 功能性 | 精确性 | 软件准确依照规定条款的程度，规定确定了权利、协议的结果或者协议的效果 |
| | 依从性 | 软件符合法定的相关标准、协定、规则或其他类似规定的程度 |
| | 互操作性 | 软件和指定系统进行交互的能力 |
| | 安全性 | 软件阻止对其程序和数据进行未授权访问的能力，未授权的访问可能是有意，也可能是无意的 |
| | 适合性 | 指定任务的相应功能是否存在以及功能的适合程度 |
| 可靠性 | 成熟性 | 因软件缺陷而导致的故障频率程度 |
| | 容错性 | 软件在故障或者外界违反其指定接口的情况下维持其指定性能水平的能力 |
| | 可恢复性 | 软件在故障后重建其性能水平，恢复其受影响数据的能力、时间和精力 |
| | 依从性 | 软件符合法定的相关标准、协定、规则或其他类似规定的程度 |
| 易用性 | 可理解性 | 用户认可软件的逻辑概念和其适用性需要花费的精力 |
| | 可学习性 | 用户为了学会使用软件需要花费的精力 |
| | 可操作性 | 用户执行软件操作和控制软件操作需要花费的精力 |
| | 吸引性 | 软件吸引用户的能力 |
| | 依从性 | 软件符合法定的相关标准、协定、规则或其他类似规定的程度 |
| 效率 | 时间行为 | 执行功能时的响应时间、处理时间和吞吐速度 |
| | 资源行为 | 执行功能时使用资源的数量和时间 |
| | 依从性 | 软件符合法定的相关标准、协定、规则或其他类似规定的程度 |
| 可维护性 | 可分析性 | 诊断软件中的缺陷、故障的原因或者识别待修改部分需要花费的精力 |
| | 可改变性 | 进行功能修改、缺陷剔除或者应付环境改变需要花费的精力 |
| | 稳定性 | 因修改导致未预料结果的风险程度 |
| | 可测试性 | 确认已修改软件需要花费的精力 |
| | 依从性 | 软件符合法定的相关标准、协定、规则或其他类似规定的程度 |
| 可移植性 | 适应性 | 不需采用额外的活动或手段就能适应不同指定环境的能力 |
| | 可安装性 | 在指定的环境中安装软件需要花费的精力 |

表2-2 ISO/IEC 9126-1的质量模型

| 特征 | 子特征 | 简要描述 |
| --- | --- | --- |
| | 共存性 | 在公共环境中同分享公共资源的其他独立软件共存的能力 |
| | 可替换性 | 在另一个指定软件的环境下，替换该指定软件的能力和需要花费的精力 |
| | 依从性 | 软件符合法定的相关标准、协定、规则或其他类似规定的程度 |

表2-3　[IEEE1061-1992,1998]的质量模型

| 因素 | 子因素 | 简要描述 |
| --- | --- | --- |
| 功能性 | 完备性 | 软件具有必要和充分功能的程度，这些功能将满足用户需要 |
| | 正确性 | 所有的软件功能被精确确定的程度 |
| | 安全性 | 软件能够检测和阻止信息泄漏、信息丢失、非法使用、系统资源破坏的程度 |
| | 兼容性 | 在不需要改变环境和条件的情况下，新软件就可以被安装的程度。这些环境和条件是为之前被替代软件所准备得。 |
| | 互操作性 | 软件可以很容易地与其他系统连接与操作的程度 |
| 可靠性 | 无缺陷性 | 软件不包含未发现错误的程度 |
| | 容错性 | 软件持续工作，不会发生有损用户的系统故障的程度。<br>也包括软件含有降级操作（degraded operation）和恢复功能的程度 |
| | 可用性 | 软件在出现系统故障后保持运行的能力 |
| 易用性 | 可理解性 | 用户理解软件需要花费的精力 |
| | 易学习性 | 用户理解软件时所花费精力的最小化程度 |
| | 可操作性 | 软件操作与目的、环境、用户生理特征相匹配的程度 |
| | 通信性 | 软件被设计得与用户生理特征相一致的程度 |
| 效率 | 时间经济性 | 在指明或隐含的条件下，软件于适当的时间限度内，执行指定功能的能力 |
| | 资源经济性 | 在指明或隐含的条件下，软件使用适当数量的资源，执行指定功能的能力 |
| 可维护性 | 可修正性 | 修正软件错误和处理用户意见需要花费的精力 |
| | 扩展性 | 改进或修改软件效率与功能需要花费的精力 |
| | 可测试性 | 测试软件需要花费的精力 |
| 可移植性 | 硬件独立性 | 软件独立于特定硬件环境的程度 |
| | 软件独立性 | 软件独立于特定软件环境的程度 |
| | 可安装性 | 使软件适用于新环境需要花费的精力 |
| | 可复用性 | 软件可以在原始应用之外的应用中被复用的程度 |

3.质量属性的重要性

质量属性应该和功能需求一样得到足够的重视。真实的现实系统中，在决定系统的成功或失败的因素中，满足非功能属性往往比满足功能性需求更为重要[Taylor2009]。

质量属性非常重要是因为它对设计的影响很大。在软件设计当中，对任何指定的功能都会有多种可选的方案，不同的方案选择产生不同的设计结果。这些不同的设计结果都体现了共同的功能特性，但它们之间却有着很大的区别，差异之处即在于拥有不同的质量因素。设计方案的质量因素往往包含很多不同的质量属性，而且不同的质量属性之间互有折中（例如，提高可移植性往往会导致效率的降低），很难会出现某一个设计方案的质量属性完全优于其他方案的情况，因此，软件设计必须根据需求的质量属性在多种方案中选择一个最优的方案。而如果不存在事先定义好的质量属性需求，设计方案的选择将完全没有依据，结果就很有可能导致软件不被用户所接受。

对于一个已经完成的设计，如果需要修改它的功能，则需要对设计进行一定的调整或拓展。但如果需要修改它的质量属性要求，在复杂的情况下就可能会需要重新进行设计方案的选择，受到的影响就是整个设计而不再仅仅局限于它的某个部分[Jansen2005]。所以，在设计开始之初就确定质量属性要求非常重要，而且对越复杂的系统越为重要。

4.质量属性需求的开发

虽然用户会在和需求工程师交流的过程当中表达一些和质量属性相关的想法，但因为他们并不了解软件系统的开发过程，也就无从判断哪些质量属性会在怎样的程度上给设计带来多大的影响，也无法将他们对软件系统的质量要求细化成一组组的可量化的质量属性，所以一般来说，他们并不能明确地提出他们对产品质量的期望。

[Chung2000]认为，在用户的叙述当中，质量属性大多是和功能需求联系在一起的，因此为了发现用户对质量属性的要求，需求工程师需要对照软件的质量属性检查每一项功能需求，尽力去判断质量属性存在的可能性；对于一些不和任何功能需求相联系的全局性质量属性，需求工程师要在碰到特定的实例时意识到它们的存在。[Cysneiros2001]通过实践还发现，用户在描述中使用的形容词和副词通常意味着质量属性的存在。

在发现质量属性后，要想了解用户的真正想法，需求工程师还需要和用户以及开发人员一起从多个角度进行质量的定义。进行质量属性的定义时，应当将其与相联系的功能需求关联起来。不能和功能需求建立联系的全局性质量属性应该统一进行处理。

5. 常见的质量属性需求

常见的质量属性有：

⑴　可靠性（Reliability）：在规格时间间隔内和规定条件下，系统或部件执行所要求功能的能力，例如QR1。

QR1：在进行数据的下载和上传中，如果网络故障，系统不能出现故障。

QR1.1：分店子系统应该检测到故障，并尝试重新连接网络3次，每次15秒；

QR1.1.1：重新连接后，分店子系统应该继续之前的工作；

QR1.1.2：如果重新连接不成功，分店子系统应该等待5分钟后再次尝试重新连接

QR1.1.2.1：重新连接后，分店子系统应该继续之前的工作；

QR1.1.2.2：如果重新连接仍然不成功，分店子系统将数据恢复到同步之前的状态；

QR1.2：总店子系统应该检测到故障，并等待分店子系统的消息

QR1.2.1：在等待10分钟仍然没有接到分店子系统的消息时，分店子系统将数据恢复到同步之前的状态

⑵　可用性（Availability）：软件系统在投入使用时可操作和可访问的程度或能实现其指定系统功能的概率，例如QR2。

QR2：系统的可用性要达到98%。

⑶　安全性（Security）：软件阻止对其程序和数据进行未授权访问的能力，未授权的访问可能是有意，也可能是无意的，例如QR3。

QR3：收银员只能查看，不能修改、删除会员的信息。

⑷　可维护性（Maintainability）：为排除故障、改进质量或适应环境变化而修改软件系统或部件的容易程度，包括可修改性（Modifiability）和可扩展性（Extensibility）例如QR4。

QR4：如果系统要增加新的特价类型，要能够在2个人月内完成。

⑸　可移植性（Portability）：系统或部件能从一种硬件或软件环境转换至另外一种环境的特性，例如QR5。

QR5：服务器要能够在1人月内从Window 7操作系统更换到Solaris 10操作系统。

⑹　易用性（Usability）：与用户使用软件所花费的努力及其对使用的评价相关的特性，例如QR6。

QR6：使用系统1个月的收银员进行销售处理的效率要达到10件商品/分钟。

[Vara2011]在调查中让调查者分别列举出最为常见的5个质量属性需求以及最为重要的质量属性，结果如图2-9所示。从中可以发现，最为常见的质量属性是：易用性、可维护性、性能、可靠性和灵活性。

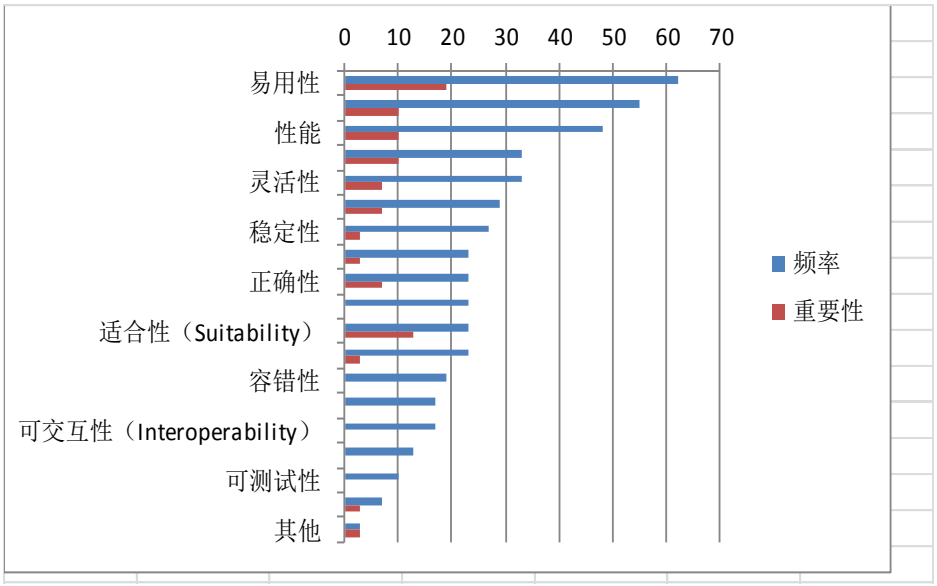


图2-9 常见的质量属性需求及其重要性，数据源自[Vara2011]

### 2.4.5. 对外接口

对于解系统而言，问题域中的其他软件系统也属于问题域的一个部分，一个比较特殊的部分。因此用户有权利对解系统和其他系统之间的软硬件接口提出要求，解系统的对外接口也是一种重要的需求。

对系统之间的软硬件接口需要说明以下内容，如IR1所示：

- 接口的用途
- 接口的输入输出
- 数据格式
- 命令格式
- 异常处理要求

IR1：客户端在每个工作日的20:00向服务器端发送数据包，更新商品数据

数据包使用XML格式，XML文件包括Message Header和Message Content

Message Header的Root Tag为“ShopProduct”，其各个字段如下：

ShopID：String，店铺号，2位区位码+3位店铺码；

Date：DateTime，日期，当天日期时间；

Message Content的父标签为“ShopProduct”，其各个字段如下：

Product：商品数据

ID：String，商品ID，36位数据代码；

SaledNum：Number，销售出的数量；

ReturnedNum：Number，退货数量；

RuinedNum：Number，报废数量；

ImportNum：Number，入库数量；

错误码为：

00：操作正确

01：数据错误

10：网络故障

11：其他类型错误

用户界面在有些情况下也会被视为系统的对外接口，被作为一种重要的需求。但[CMU/SEI1991]认为，和其他的需求相比，用户的界面更经常发生变化，进而影响需求的稳定性，所以[CMU/SEI1991]建议如果将用户界面作为需求一部分的话，一定要进行单独的处理和组织。通常，对于人机交互复杂的系统，使用单独的人机交互设计文档记录用户界面需求，否则可以将其作为需求文档的一部分进行记录，如需求IR2所示。

IR2 订单管理：系统应该使用如下图所示的Form风格，帮助销售经理完成订单管理。

IR2.1 在销售经理开始订单管理时，系统显示所有未签收的订单列表

IR2.2 在销售经理选择待修改的订单后，系统显示该订单的所有信息，并允许销售经理修改其状态。

IR2.3 在销售经理修改订单状态后，系统保存更改后的订单列表，并刷新当前订单列表。

对外接口是设计师需要的知识，软件体系结构师设计需要根据对外接口安排软件系统的外部环境，详细设计人员要将所有的对外接口与业务逻辑进行隔离，交互设计师要为人机交互需求设计人机交互方案。

### 2.4.6. 约束

约束是不受解系统影响，却会给解系统带来极大影响的问题域特性。因为不受解系统的影响，所以从解决问题的角度来看约束不会要求解系统为其进行专门的设计。但是如果解系统不满足约束，那就意味着问题域并不能够提供解系统要求的运行环境，解系统将无法在问题域内成功的部署和运行。因此，约束是在总体上限制了开发人员设计和构建系统时的选择范围。

这些约束通常会影响后续的体系结构设计、详细设计、实现、测试等软件开发活动。

常见的约束主要有：

- 系统开发及运行的环境，例如Constraint-1。包括目标机器、操作系统、网络环境、编程语言、数据库管理系统等。
- 问题域内的相关标准，例如Constraint-2。包括法律法规、行业协定、企业规章等。
- 商业规则。用户在任务执行中的一些潜在规则也会限制开发人员设计和构建系统的选择范围。
- 社会性因素。文化、信仰等社会性因素。

Constraint-1：系统要能够在Windows X和Linux两种操作系统上运行。

Constraint-2：系统的保密性能要符合×××法律第×××条的要求。

在2000s之后，随着Internet的发展，软件系统在社会生活中出现的越来越多，影响越来越大，所以约束变得越来越重要，尤其表现在法律规则[Otto2007]、社会性因素[Yu2010, Milne2011]这两个方面。

规则的往往是需求文档的一个重要部分，[Gottesdiener2002]建议按照表2-4的格式进行描述。

表2-4 规则描述样式与示例

| 类别 | 描述样式 | 示例 |
|---|---|---|
| 术语 | [限定词]<名词/业务术语>是指<文字描述> | Rule2 :退货是指顾客在一个时间点上凭之前1周内的购物发票，退回其中一项或多项商品的行为； |
| 事实 | [限定词]<名词/业务术语1>[条件限定]必须\|可能<动词或动词短语>[限定词]<名词/业务术语2> | Rule3 :同样的商品在不同的时期内可能有不同的价格； |
| | <名词/业务术语1>的特征有<名词/业务术语2> | Rule4：每件商品都有一个条形码 |
| 约束 | [限定词]<名词/业务术语>必须满足<条件> | Rule5：商品条形码符合EAN-13标准 |
| | <名词/业务术语>必须/不能<动词或动词短语><条件> | Rule6 :商品特价的折扣率不能超过50% |
| 推导 | <名词/业务术语>的计算方式为<数学计算表达式> | Rule7：普通商品项总价 = 价格×数量 |

表2-4 规则描述样式与示例

| 类别 | 描述样式 | 示例 |
| --- | --- | --- |
| 推理 | 如果<条件1>[和/或者<条件2>...]，那么<结论> | Rule8：如果销售日期在一周之前，或者销售是用积分付款的，或者销售的非积分付款余额已经不足以支付商品退款额，那么该商品就属于不可退货商品 |

项目管理者和软件体系结构师、详细设计师后续开发人员都需要关注约束，以保证最终的系统能够成功运营。

### 2.4.7. 其他需求

实践中，需求文档还常常会文档化其他类型的需求，例如安装需求（例如OR1）、培训需求（OR2）、数据需求等，其中数据需求较为常见。

OR1：在安装系统时，要初始化用户、商品库存等重要数据。

OR2：系统投入使用时，需要对用户进行1个星期的集中培训。

数据是软件系统中非常重要的知识内容，但它可以在表达功能需求时进行描述，例如在2.3.3节提及的需求SR1，描述了交互功能的同时也描述了使用的数据。如果在功能需求中没有描述数据内容（例如SR3），就需要补充描述数据信息（例如DR1~DR2）。

SR3：在收银员输入商品标识时，系统显示商品信息，商品信息参见DR1、DR2；

DR1：ID是规则为...的商品条形码；

DR2：商品信息包括：ID、名称、描述、价格、特价、数量、总价；

## 2.5. 优秀需求的特性

理想情况下，需求应该既是解决用户问题所需要的，又是表述清晰的，既是用户的需要，又是开发者的需要。为此，人们定义了一些优秀需求应该具备的特性[Kar1996,IEEE1998, Firesmith2003]，下面是其中比较重要的部分。

### 2.5.1. 完备性

优秀的需求是完备（Complete）的，它不需要做更多的扩展就可以充分的说明用户需要的系统功能。完全性的判断标准是：需求是否描述了开发人员设计和实现这项功能需要的所有信息。只有完备的需求在开发当中才可能被独立出来，单独对待。在需求开发的过程当中，对于不清晰的信息可以标记为TBD（To Be Determined，待确定），但在需求开发结束之前，所有的TBD都必须被解决。

例如R20就是不完备的，仅仅根据需求描述，开发人员并不能明确到底如何输入商品信息，以及显示哪些商品信息。相对而言SR1、SR3更加完备。

R20：在收银员输入商品时，系统显示商品信息。

[Firesmith2005]建议对不同类型的需求可以从不同的方面来保障需求描述的完备性：

- 对功能需求的描述，要确保下列方面都得到了描述：
  - 行为的触发者（Trigger），它使得系统执行功能需求的行为，常见的触发者包括：数据输入、接收的请求、要处理的异常等。
  - 行为的前置条件（Precondition），它是系统成功满足功能需求的前提，常见的前置条件包括：系统的模式或状态、其他外部系统的状态、任何系统数据的值等；
  - 行为（Action），在前置条件下接受到触发者时，系统必须执行的行为；
  - 后置条件（Postcondition），一旦系统成功执行行为后所处的状态，常见的后置条件包括：系统的模式或状态、其他外部系统的状态、任何系统数据的值等；。
  - 不满足前置条件（Failed preconditions）的情况，以及相应情况下的结果（Resulting failure postconditions）。
- 对数据需求（或者功能需求描述中的数据内容）要注意下列内容以保障完备性：
  - 类型；
  - 语义（例如在数据字典或项目术语表中描述数据的含义）；
  - 组成成分（例如属性，子部分等）
  - 初始值或缺省值；
  - 可能的取值范围；
  - 度量单位；
  - 数据量；
  - 更新频率；
  - 可以对其施加的合理操作；
  - 对外关系，例如关联、聚集、泛化等。
- 对对外接口的描述要注意下列内容以保障完备性：
  - 接口的名字；
  - 接口的定义，简短的描述；
  - 接口的方向声明（In，Out，或者双向）；
  - 服务请求，包括：
    - 语法，例如名称、参数、返回值。
    - 语义，含义，协议，例如前置条件、后置条件与不变量或者状态模型等；
    - 异常，语法与语义；
    - 接口定义的特殊数据类型；
    - 质量属性要求（例如性能、可靠性、安全性、保密性等）
- 对质量属性（含性能）的描述要注意下列内容以保障完备性：
  - 针对系统或其部件的质量标准；
  - 明确了需要满足质量标准的情况与条件；
  - 衡量质量标准所使用的单位；

- 质量度量的阀值；

- 与需求源头和相关事项的专家确认，以保障约束描述的完备性。

### 2.5.2. 正确性

每一项需求都必须正确的描述所需要的系统功能，要真实的反应用户的意图，所以需求的正确性又常被称为真实性（Real）。需求的正确性只有提出需求的人才能加以判断，所以需求在传递给开发人员之前，必须请需求的提出者予以确认。

正确性是一个看上去很简单，但实践中很难满足的特性[Gilb2005]。实践一再表明不真实的需求是最为常见的需求错误之一[Lawrence2001,Leffingwell1999, Sawyer1997, Lubars1993] ,必须得到足够的重视。

需求正确性难以达到的主要原因有：

一、用户在表达自己的需要时，可能会在潜意识下进行一定的加工。常见的情况是：用户的问题是A，但用户认为如果提供了方法B，则问题A自然可以得到解决，为此用户向需求工程师反映的便是B，而不是真实的A。所以为了发现用户的真实需求，需求工程师一定要进行问题分析，尽力发现问题背后的问题。

二、在人际交流当中，信息会发生自然的衰减，甚至扭曲，导致需求工程师理解的并非是用户所表达的，对此情况的解决方法是在需求传递给开发人员之前，请提出需求的用户进行仔细的检查和确认。

[Gilb2005]提出了一些发现真实需求的方法，总结起来包括下列几点：

①多问用户“为什么”，将涉众的需求描述从具体情景、技术环境等约束中抽离出来，发现涉众更广范围和更高层次上的目标。例如如果用户描述“需要将联系人信息放到web站点上”，需求工程师就应该问为什么，将web站点这个技术环境剥离，转变为“交流共享联系人信息”这个更深层的目标。

②从业务方面描述需求，而不是从技术方面描述需求。虽然需求开发最终要产生的是系统级需求，它关注于软件系统的对外交互，不免涉及技术观点。但涉众更易于从业务角度理解需求，所以在与涉众交流时，要用业务的方式描述需求。例如“收银员输入商品标识，系统记录商品...信息”比“收银员使用扫描仪扫描商品条形码，系统将...信息存储入数据库”更能为涉众所理解。

③关注涉众的想法。包括关注涉众对需求的效益评价，关注涉众对需求的优先级划分，真正理解需求对于涉众的意义。

④在演化中理解需求。最初始的需求并不一定就是最终的需求，要将最初始的需求展示给涉众，并利用涉众的反馈发现真正的需求。

⑤通过量化手段准确理解需求。尤其是在描述质量属性需求时，试图量化需求的过程中，对验证标准的反复推敲可以更准确地理解涉众的需求含义。

### 2.5.3. 可行性

需求必须能够在系统及其运行环境的已知条件和约束下实现。用户无法判断需求的技术可行性，所以需求的可行性是由开发人员进行检查的。在检查的过程中，开发人员可能需要进行一定的分析和研究，而不是单纯的凭借经验和直觉。对于难以判断的需求，必要的时候要通过开发原型来加以验证。

不可行的需求又被称为不切实际的期望（Unrealistic Expectations），是实践当中常见的需求定义问

题，而且它在很大程度上影响着项目的成败[Standish1995]。

因为用户并不掌握关于软件系统构建的相关技术知识，所以用户可能会提出一些已有软件技术无法实现的期望，或者在限定的项目环境下固执的要求不可能同时满足的多项要求，这就通常是不切实际的期望的来源[Gabb1999]。

面对不切实际的期望，要求软件开发者提供可行性、成本等足够的技术参考信息，帮助用户对其进行取舍和调整。

### 2.5.4. 必要性

每一项需求都应该是必要的，它是满足用户的业务需求所必需的。如果一条需求被忽略之后，系统仍然能够以同样的效果解决用户的问题，那么它就不值得在开发的过程中消耗额外的资源。

不必要的需求也是实践中常见的一个问题[Wiegers2003, Young2002, McDermid1989]，可能因为多种原因而出现：

其一是用户将之作为和开发人员谈判的筹码，然后通过自己对不必要需求的进退取舍而在和开发人员的谈判当中取得真正想要的利益，例如金钱。对此问题，唯一需要的就是开发人员代表的谈判技巧。

其二是用户在交流当中，总是害怕信息有所遗漏，并因而产生不利后果，因此用户总是倾向于表达各种各样的需要。要解决这个问题，就需要开发人员在进行用户需求的获取之前，先定义明确的业务需求，然后根据业务需求进行用户需求的过滤和选择。

其三是需求开发人员“画蛇添足”，添加“用户肯定会喜欢”的功能，该类功能既会造成项目额外的耗费，又不会给用户带来更多的帮助。这就要求需求开发人员要保持以用户为中心。该事项开发人员尤其需要注意。

### 2.5.5. 无歧义

需求能够正确传递知识的前提是传递者和受众能够形成共同的理解，因此每一项需求都应该有而且只能有一种解释，即需求无歧义。为了让需求可理解，一般倾向于以用户的语言描述需求，而用户的语言往往含有大量容易导致歧义的因素，因此在保证需求描述的无歧义时要格外注意需求描述中的词汇选择，通常在需求开发当中要定义一个可以共同理解的词汇表（Glossary），然后再在其基础上进行需求的描述。

模糊和歧义也是实践中常见的需求错误类型[Wiegers2003,Sawyer1997, Lubars1993, McDermid1989]，它多数是被无意写出的，少数情况下也可能被有意写出。

无意中写出模糊和歧义的需求定义往往是因为选词造句不当，导致不同的人对同一项需求产生了不同的理解。[Wiegers2003]建议在描述需求时避免使用表2-5内的词汇以防止需求描述出现歧义。

表2-5 应该避免的歧义词汇，源自[Wiegers2003]

| 歧义词汇 | 改进方法 |
| --- | --- |
| 可接受的、足够的 | 具体定义可接受的内容，说明系统怎样判断“可接受”或“足够” |
| 大概可行的、差不多可行的 | 不要让开发人员来判断“大概”和“差不多”到底是否成立。应将其标记为待确定问题并标明解决日期 |

表2-5 应该避免的歧义词汇，源自[Wiegers2003]

| 歧义词汇 | 改进方法 |
| --- | --- |
| 至少、最小、不多于、不超过 | 明确指定能够接受的最大值和最小值 |
| 在……之间 | 明确说明两个端点是否在范围之内 |
| 依赖 | 描述依赖的原因，数据依赖？服务依赖？还是资源依赖？等等 |
| 有效的 | 明确“有效”所意味的具体实际情况 |
| 快的、迅速的 | 明确指定系统在时间或速度上可接受的最小值 |
| 灵活的 | 描述系统为了响应条件变化或需求变化而可能发生的变更方式 |
| 改进的、更好的、更快的、优越的 | 定量说明在一个专门的功能领域内，充分改进的程度和效果 |
| 包括、包括但不限于、等等、诸如 | 应该列举所有的可能性，否则就无法进行设计和测试 |
| 最大化、最小化、最优 | 说明对某些参数所能接受的最大值和最小值 |
| 一般情况下、理想情况下 | 需要增加描述系统在异常和非理想情况下的行为 |
| 可选择地 | 具体说明是系统选择、用户选择还是开发人员选择 |
| 合理的、在必要的时候、在适当的地方 | 明确怎样判断合理、必要和适当 |
| 健壮的 | 显式定义系统如何处理异常和如何响应预料之外的操作 |
| 无缝的、透明的、优雅的 | 将词汇里面所反映的用户期望转化成能够观察到的产品特性 |
| 若干 | 声明具体是多少，或提供某一范围内的最小边界值和最大边界值 |
| 不应该 | 试着以肯定的方式陈述需求，描述系统应该做什么 |
| 最新技术水平的 | 定义其具体含义，即“最新技术水平”意味什么 |
| 充分的 | 说明“充分”具体包括哪些内容 |
| 支持、允许 | 精确地定义系统的功能，这些功能组合起来支持某些能力 |
| 用户友好的、简单的、容易的 | 描述系统特性，用这些特性说明词汇所代表的用户期望的实质 |

有意产生的模糊和歧义的需求定义往往是为了应付对需求持有不同立场的用户，这些用户关于需求的目标互相冲突，需求工程师遂采用了模糊化的处理方法。但软件的生产是无法进行模糊化处理的，所以开发者最终仍要面对一个两难局面。对于用户立场冲突的正确解决方法是在项目前景的指导下，促进用户之间的协商解决。

### 2.5.6. 可验证

需求应该是可验证的，也就是说通过分析、检查、模拟或者测试等方法能够判断需求是否被满足。如果需求不可验证，就无法判断完成的系统是否满足了该需求，开发人员也无法去选择一个能够实现该需求的方法。通常，不可验证的需求往往是因为描述模糊或者过于抽象，所以在进行需求的描述时要让需求具体化，小心形容词和副词的使用，避免程度词的使用。

# 引用文献

[Albayrak2009] Albayrak,ö., Kurtoglu,H., Bıçakçı, M., Incomplete Software Requirements and Assumptions Made by Software Engineers, 16th Asia-Pacific Software Engineering Conference, 2009.

[Bray2002] Bray, I. K., An Introduction to Requirements Engineering (1st edition), Addison Wesley, 2002.

[Bühne12004] Bühne1,S., Halmans1,G., Pohl1,K., Defining Requirements at Different Levels of Abstraction, Proceedings of the 12th IEEE International Requirements Engineering Conference (RE'04), 2004.

[Chung2000] Chung, L., Nixon, B., Yu, E. and Mylopoulos, J., Non-Functional Requirements in Software Engineering, Kluwer Academic Publishers 2000.

[Clements2007] P. Clements, R. Kazman, M. Klein, D. Devesh, S. Reddy and P. Verma, The Duties, Skills, and Knowledge of Software Architects, Proc. 6th Working IEEE/IFIP Conf. Software Architecture (WICSA 07), 2007.

[CMU/SEI1991], Software Engineering Institute, Requirements Engineering and Analysis Workshop Proceedings, 1991.

[Cysneiros2001] Cysneiros,L.M., Leite, J.C.S.P. and Neto, J.S.M., A Framework for Integrating Non-Functional Requirements into Conceptual Models, Requirements Engineering Journal —Vol 6 , Issue 2 Apr. 2001.

[Firesmith2003] Firesmith, D., Specifying Good Requirements ,Journal of Object Technology, vol. 2, no. 4, July-August 2003, pp. 77-87.

[Firesmith2005] Firesmith, D., Are Your Requirements Complete? JOURNAL OF OBJECT TECHNOLOGY VOL. 4, NO. 1, January-February 2005.

[Gabb1999] Gabb, A. P., Requirements Denial - Examining the Excuses, Conference of the Systems Engineering Society of Australia (SESA) and The International Test and Evaluation Society (ITEA), SETE99, October 1999.

[Gilb2005] Gilb, T., Real Requirements: How to Find Out What the Requirements Really Are?, Fifteenth Annual International Symposium on Systems Engineering, Seattle, Washington: International Council on Systems Engineering, 2005.

[Gotel2006] Gotel,O., In Search of the System Concept, IEEE SOFTWARE, January/February 2006, pp102-103.

[Gottesdiener2002] Gottesdiener, E., Top Ten Ways Project Teams Misuse Use Cases–and How to Correct Them, The Rational Edge, Date, 2002.

[Lawrence2001] Lawrence, B., Wiegers, K., and Ebert, C., The Top Ten Risks of Requirements Engineering, IEEE Software, 2001.

[Leffingwell1999] Leffingwell, D., Widrig, D., Managing Software Requirements: A Unified Approach, Addison-Wesley, 1999.

[Lubars1993] Lubars,M., Potts,C., Richter,C., A Review of the State of the Practice in Requirements Modeling, Proceedings of IEEE International Symposium on Requirements Engineering, RE'93,1993.

[Maiden2008] Maiden, N., User Requirements and System Requirements, IEEE Software, March/April 2008, pp90-91.

[McConnell2000] McConnell, S., What's in a Name? IEEE SOFTWARE,Septembr/October 2000, pp7-9.

[McDermid1989] McDermid, J. A., Requirements Analysis: Problems and the STARTS Approach, In IEE Colloquium on 'Requirements Capture and Specification for Critical Systems' (Digest No. 138), 4/1-4/4. Institution of Electrical Engineers, November 1989.

[Milne2011] Milne,A., Maiden, N.A.M.,Power and Politics in Requirements Engineering: A Proposed Research Agenda, Proc. 18th IEEE Int'l Requirements Eng. Conf., IEEE CS, 2011, pp. 187–196.

[IEEE1061-1992] IEEE Std 1061-1992, IEEE Standard for a Software Quality Metrics Methodology, Institute of Electrical and Electronics Engineering, Inc., 1992.

[IEEE1061-1998] IEEE Std 1061-1992, IEEE Standard for a Software Quality Metrics Methodology, Institute of Electrical and Electronics Engineering, Inc., 1998.

[IEEE1990] IEEE Std 610.12–1990, IEEE Standard Glossary of Software Engineering Terminology, Institute of Electrical and Electronics Engineering, Inc., 1990.

[IEEE1998] IEEE Std 830-1998, IEEE Recommended Practice for Software Requirements Specifications, Institute of Electrical and Electronics Engineering, Inc., 1998.

[ISO/IEC 9126-1] ISO/IEC 9126-1, Software Engineering - Product Quality - Part 1: Quality Model, ISO/IEC Ed., International Organization for Standardization and International Electrotechnical Commission, 2001.

[Jackson1995a] Jackson, M., Problems and Requirements, a Keynote Address at RE'95; Proceedings of the IEEE Second International Symposium on Requirements Engineering; ACM Press, 1995.

[Jackson1995b] Jackson, M., The World and the Machine, a Keynote Address at ICSE-17; Proceedings of ICSE-17; ACM Press, 1995.

[Jackson1997] Jackson, M., The Meaning of Requirements, Annals of Software Engineering Special Issue on Software Requirements Engineering pages 5-22, 1997

[Jansen2005] A. Jansen and J. Bosch, Software Architecture as a Set of. Architectural Design

Decisions, In Proceedings of WICSA 2005, 2005.

[Kar1996] Kar, P. and Bailey M., Characteristics of Good Requirements, the 6th INCOSE Symposium, 1996.

[Kauppinen2005] Kauppinen, M., Introducing Requirements Engineering Into Product Development: Towards Systematic User Requirements Definition, Doctoral Dissertation, 2005.

[Otto2007] Otto, P. N., Antón, A. I., Addressing Legal Requirements in Requirements Engineering, 15th IEEE International Requirements Engineering Conference, 2007

[Robert2002] Robert L. Glass, Facts and Fallacies of Software Engineering, Addison-Wesley, 2002.

[Sawyer1997] Sawyer, P., Sommerville, I., and Viller, S., Requirements Process Improvement through the Phased Introduction of Good Practice, Software Process - Improvement and Practice 3, 19-34, 1997.

[Standish1995] Standish Group, CHAOS, 1995

[Taylor2009] R.N. Taylor, N. Medvidovic, E.M. Dashofy, Software Architecture: Foundations, Theory and Practice, Wiley 2009.

[Vara2011] Vara, J. L., Wnuk,K., Svensson,R. B., Sánchez1 J., Regnell, B., An Empirical Study on the Importance of Quality Requirements in Industry, 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE 2011), 2011.

[Wiegers2003] Wiegers, K., Software Requirements, second edition. Redmond, WA: Microsoft Press, 2003.

[Young2002] Young, R. R. , Effective Requirements Practices, Boston et al.: Addison-Wesley, 2002.

[Yu2010] Yu, E.,et al. (eds.), Social Modeling for Requirements Engineering, MIT Press, 2010.