6

第 6 章

实例化需求
——如何写出高质量需求

第2～5章讲解了如何发掘业务需求、如何定义利益相关方需求、如何编制功能需求规约和非功能需求规约。但是，这些内容和软件系统的代码实现之间没有直接的关联，可能造成需求与代码实现的不匹配。本章聚焦如何把需求提升到可执行水平，使其成为代码的一部分，从理论上消除从需求到代码实现之间的信息衰减。同时总结出编写实例化需求的CARUN规则。本章的讲解重点如下：

- 为什么要引入实例化需求？
- 实例化需求是什么？
- 实例化需求怎么做？

6.1　为什么要引入实例化需求

在软件系统研发过程中，做正确的事固然是一件困难的事情，正确地做事也不是轻而易举的。传统上，哪怕是一个中型的软件系统，它的需求规约可能也是长篇巨著：几十万字的软件系统需求规约非常普遍。为了更精细地描述软件系统对外提供的功能，上百万字的需求规约并不鲜见。软件开发人员对于满篇充斥的“一定要做到……”“一定不能违反……”的规定性描述文字，一方面感到非常枯燥，另一方面非常容易出现理解偏差。在没有实际系统做演示的情况下，只有这些强制性要求的需求信息传递效果非常糟糕，因为它们虽然看起来像是很精确，却包含很多潜在的和可能产生理解不一致的地方。每个软件开发人员都有自己的假设，这些假设影响着他们对这些文字的理解。因此，急需一种有效的方法，消除需求团队和开发团队之间对于需求理解的不一致，使软件系统的原始需求所蕴含的信息能够准确、一致、高效地在各个团队之间传递。相对于抽象的理论、概念，人们对于具体的实践、故事更容易理解和接受。就像同样深奥的经文，我们自己读起来可能莫名其妙，而经过大师旁征博引、结合实际，就能轻松明白其中的深奥道理一样，用“实例”来描述需求，能更好地提升信息的传递效率，保持它的一致性（图6-1）。

图 6-1　大师讲经，舌粲莲花

回想一下需求团队如何从最终用户处挖掘他们的需要：聆听他们在实际工作中发生的“故事”，或者期望能够完成工作的“方式”；通过对这些故事的总结、归纳、提炼、精细化，最终形成抽象、格式化、命令式的需求。为了验证、确认和诠释这些抽象的需求，开发团队通常会重新构建一些“业务场景”，以方便和需求团队进行沟通（图6-2）。在这个过程中，是不是存在什么问题？

图 6-2　需求形成和验证过程

一个非常直接和自然的想法是：为什么不能够直接把最终用户的“故事”固化下来，在不同团队之间传递呢？这样不同的团队看到的不就是同样的信息，能够消除信息传递过程中的衰减了么？

Gojko Adzic总结了业界在这方面的研究和实践，并在2011年出版了《实例化需求：团队如何交付正确的软件》一书，完整描述了实例化需求（Specification by Example，SBE）和用这种方法推进软件系统研发的模式。实例化需求就是用实例来精细化需求，其核心是使用实例作为需求、开发和测试等团队之间沟通的桥梁，以实例为载体，多方协作制定需

求，从而保证团队所有成员对需求的理解一致，具有以下好处。①作为多方一起工作的成果，在技术的支持下，能够“在不修改的情况下实现自动化验证”，消除信息传递衰减的风险；②既可以作为开发的目标，也可以作为验证软件是否按照预期执行的标准，还可以作为和客户沟通的依据；③能够实现持续验证，从而保障其与系统代码的一致性，避免出现“需求文档完成即过时”的问题。

作为促进需求信息有效传递的最佳实践，实例化需求引入了一组过程模式，协助研发团队正确研发软件系统（图6-3）。

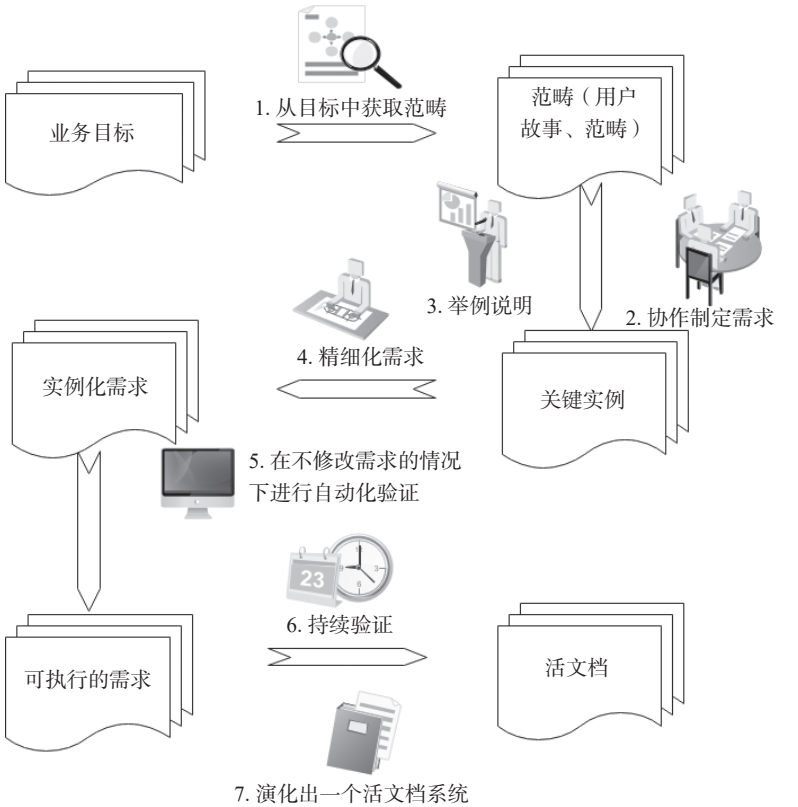

图 6-3　实例化需求的关键流程模式

1. 从目标中获取范畴

在从业务目标抽取任务范畴的过程中，研发团队不能盲目地接受软件需求作为一个未知问题的解决方案，而是要从目标中获取任务的范畴。他们从客户（或他们的代表）的业务目标出发，和客户一起共同定义实现业务目标的任务范畴。研发团队和客户一起确定解决方案。首先，用户聚焦于传递期望需要的范围和他们期望从中获得的价值；接着，实现团队基于自身的专业能力，一般能够给出一个比客户自己提出的更经济、快速、容易发布和维护的解决方案。

2. 协作制定需求

在制定软件系统需求的过程中，成功的团队不是依赖单独一个人（一种角色），而是协同业务人员在内的所有角色一起制定详细需求。不同背景的人有不同的观点，并基于他们自己的经验和技术来解决问题。业务专家深知业务场景和流程，技术专家知道如何更好

地应用支撑技术或创新技术；测试人员知道到哪里查找潜在的问题，后续团队应当如何规避这些问题。所有这些信息，在定义需求时都需要被获取。协同制定需求不但能够利用整个团队的知识和经验，而且也能产生集体需求意识，使每一个人都深入参与需求定义的过程中。

3. 举例说明

在软件系统实现过程中，优秀的团队擅于使用来自实际的、具体的、鲜活的例子准确展现需求的内涵。在团队中，需求（业务）、开发、测试等角色协同识别、描述被期望功能的关键实例。在这个过程中，需求人员是主角，引导实例的定义过程，而开发人员和测试人员经常能够提供额外的例子来说明边缘情况，并解决系统中比较突出的问题。这种方式有效消除了需求和真实需要功能的不一致，并确保涉及的每个人对什么被需要有共同的理解，避免由于误解和信息传递导致的软件系统实现谬误。

4. 精细化需求

实例化需求的核心是“实例”，通过前面的努力，识别和定义出可用的实例。这些实例应当是精确、完整、真实和容易理解的，并且包含了非功能的需求。在实例化需求中，使用生成的实例来精细化需求，去除多余的信息，并针对开发与测试的应用上下文，定义出适当数量与详细程度的实例化需求。注意，在需求中，核心内容是描述打算做什么，而不是如何做软件系统。

5. 在不修改需求的情况下进行自动化验证

需求承载了外界对软件系统能够提供什么价值的信息，是软件系统“做正确的事”的具体体现。在软件实现的传统方法中，需求验证是通过和它没有“紧密绑定”关系的测试用例完成的，在针对需求设计测试用例的过程中，不可避免地存在理解偏差和信息传递不一致的情况。为了解决这一问题，实例化需求实现了在不修改需求的情况下进行自动化验证。出现这一提升的原因，是把传统的需求精细化成了可执行的需求，这也是能够使需求信息无缝传递给团队后续角色的关键。

6. 持续验证

成功团队通过持续地执行所有可执行的需求，很快就能发现软件系统和需求之间的差异。因为可执行的实例化需求具体、鲜活、容易理解，所以也有助于研发团队和用户讨论需求变化，以及决定如何解决这些变更问题。可执行需求能够通过持续验证需求与实现之间的一致性，促使软件研发一直按正确的方向前进。

7. 演化出一个活文档系统

软件系统=算法+数据+文档。其中，算法和数据以可执行程序的形式对外提供价值。文档中包括两部分：一是外界期望从软件系统中获得的价值，即可执行程序的需求信息输入；二是对可执行程序输出的描述，即软件系统的研发成果描述。这两部分文档信息的交互点是软件系统的代码，活文档系统正是以此为切入点。首先，把需求和代码绑定，使其成为可执行程序的一部分；其次，从代码生成软件系统的描述文档，从而保障其及时性和准确性。这种以代码为依托的信息传递方式，由于能够高效、及时、准确地提供各种格式的文档，因此被称为“活文档系统”。

6.2 实例化需求实践

下面结合用户认证与授权系统来说明如何从目标中获取范畴、协作制定需求、举例说明、精细化需求、在不修改需求的情况下进行自动化验证、持续验证以及演化出一个活文档系统的方法和注意事项。

6.2.1 从目标中获取范畴

界定范畴是软件系统研发中至关重要的第一步。如果没有界定出准确的范畴，后续的所有努力就像建造于流沙之上的城堡，由于没有稳固的根基，最终都会徒劳无功。如果软件系统研发团队过度依赖客户提供的原始需要信息，不加分析、提炼就直接把它们当成需求，实则是将有针对性地设计解决方案以及解决问题的重任转嫁给了客户，从而造成只有等到系统上线后，才发现所开发的软件无法达到客户业务目标要求的问题，造成巨大浪费。专业的软件研发团队所确定的软件范畴，从来都来源于客户，而又高于客户的描述。他们会紧盯客户的业务目标，围绕业务目标的实现，和客户一起共同定义任务的范畴（图6-4）。

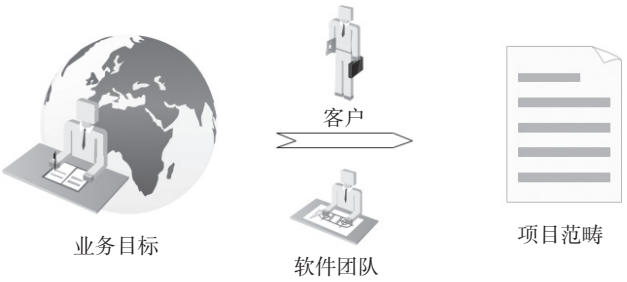

图 6-4 客户与软件研发团队共同从业务目标定义项目范畴

1. 构建正确的范畴

在软件系统需求调研过程中，往往能够收集到大量的应用场景、用户故事，幸运时还能得到在用系统详尽的产品功能清单，但是却很少能够直接获得客户明确、清晰的业务目标。

2.1节我们假定A公司发出了一份《用户认证与授权需求建议书》，在实际中这算是比较正规的需求了，其中也仅描述了A公司对于用户认证与授权系统应该达到的功能和非功能要求，和它的业务目标毫无关系，更不用说我们假定的对开发用户认证与授权系统感兴趣的B公司，它的业务目标是“董事会为公司定下三年内公司销售额翻番的战略目标”，甚至和所开发软件系统的本身完全没有关系。

但是，从B公司的定位“提供标准化的通用软件产品”，结合《用户认证与授权需求建议书》中的具体软件系统需求，应该能够界定出：B公司将要投资建设的用户认证与授权系统是以实现用户认证与授权业务领域标准功能为核心，其主要目的是形成能够在市场上作为通用的、专业解决方案销售的产品，而对于A公司的需求建议书中超出标准功能的部分，就不是B公司重点关注的内容了，严格来说，不在B公司规划的用户认证与授权系统的范畴之内。

实际上，界定一个软件系统研发的范畴，其信息的来源不仅依赖于直接用户的原始需要信息，更依赖于所在组织的发展目标、高层对于目标软件系统的定位信息，也正是由于这个原因，在界定范畴的过程中，一定要让高层领导加入，并把界定的范畴结果提交其批准。

2. 在没有整体控制权的情况下，协作确定范畴

对于大型的、集成了大量子系统的软件系统的子系统研发团队而言，范畴由软件系统整体界定，子系统研发团队只负责其中的一部分，可能无法直接和商业目标挂钩。而作为一个大型的集成系统，有它最优的实现方案，而整体的最优方案往往是多个子系统之间互相支持、彼此配合的结果，但对于某个子系统本身，却未必是最优方案。在这种情况下，子系统研发团队充分了解客户的真实目标，有助于子系统研发团队专注于真正重要的事情。

例如，在B公司，用户认证与授权系统是一个独立的软件，具有明确范畴。在A公司中却不然，它只是某个业务系统的一部分，而整个业务系统的范畴是由A公司界定的，遵循A公司的业务目标。在这种情况下，A公司最好能够把范畴向用户认证与授权系统团队说清楚，以便他们能够在B公司的范畴之外为A公司提供额外的能力。

6.2.2　协作制定需求

多角色协作完成软件系统研发是敏捷开发的核心理念（敏捷宣言的第一条就是个体和互动高于流程和工具），并形成了以需求、开发、测试（神勇三剑客）为核心的协作团队，共创软件系统（图6-5）。实例化需求方法是敏捷开发的一员，它的生命力同样源自多方的努力，只靠单一角色达不到它所期望的效果。

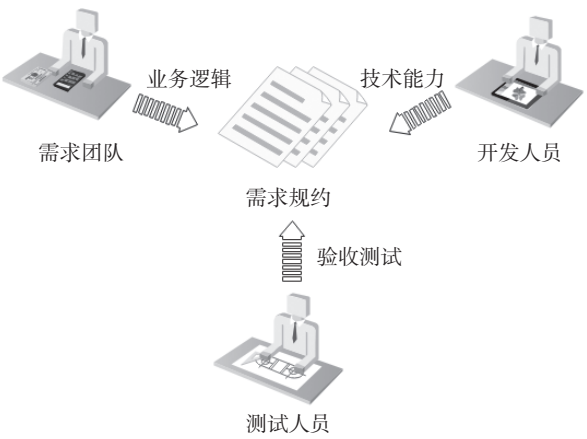

图 6-5　协作制定需求规约之“神勇三剑客”

1. 为什么需要协作制定需求

协作制定需求可以保证团队的各个角色就需要完成的内容达成共识。同时，不同角色各有专业优劣：需求团队知道业务逻辑是什么，但不知道怎么组织需求才易于进行自动化测试；开发人员掌握技术能力，知道什么样的需求更容易进行自动化测试并驱动开发，但对需求和测试的工作并不擅长；测试人员能够编写验收测试，但无法说明它们是否验证了实际的业务需要。因此，需要各种角色相互协作才能写出真正高质量的需求。

2. 常用协作方式

协作方式有很多，各有优劣，分别适合不同的场景，供需求团队在完成相应的工作时选用。

1）全体工作坊

全体人员参加的大型工作坊是建立共识并获取需求最有效的途径之一。由于团队全体都参与，以后就不必再做知识传递。在大型工作坊中，开发和测试人员可能会提出需求团队原来没有考虑到的业务场景，有助于需求团队查漏补缺，编写完整需求。

但是，大型工作坊的准备工作相对困难，协调各方同时参与的准备工作很难解决，并

且由于参与方较多，达成共识也比较困难。为了全体工作坊能够顺利开展，可以尽量通过定期会议的方式，提前预约客户或利益相关方的时间。

2）小型工作坊

小型工作坊是敏捷开发最常用的协作方式，几乎日日、时时都在应用。甚至在系统研发过程中，由于需求、开发、测试的相关人员经常协同工作，他们进行的小型工作坊也称为作战会议。与全体工作坊相比，小型工作坊更加灵活，更容易组织，并且不需要预先计划，更能以问题为导向。组织小型工作坊，一般要求与会人员在目标问题上有相似深度的理解。一般情况下，会议的方向多被领域知识较强的人所左右，只有知识水平相当的人，才能在会议中互相印证、互相补充。达不到要求水平的参与者应当提前准备。

3）结对编写

需求人员和测试人员结对：需求人员提供正确的业务逻辑，测试人员知道编写测试的最佳方法。需求人员单独撰写需求，容易专注于编写一个大而全的用例，这个用例可能会影响现有的很多测试，降低软件系统的可测试性。通过结对编写需求，可以使需求更容易实现和验证。

4）开发人员评审需求

资深的开发人员通过评审需求，往往能给需求人员提出合理的建议，使需求人员定义的需求更全面，减少需求必要信息缺少或难以实现的风险。

也许有的读者会质疑，为什么需要开发人员评审需求呢？他能起到相应的作用么？

我们还用“验证账号”用例来展示开发人员对于需求质量提升的作用，对于这个用例，基于描述它的用户故事，在5.3.1节抽取出相关的质量需要。

QN1：为了防止密码泄露，在用户名/密码认证过程中，从输入页面传输到服务端需要加密传输。

QN2：认证过程不能过长，需要在3秒内完成。

QN3：为了规避恶意撞库，认证失败5次后，锁定登录账号10分钟。

如果不是经验丰富的开发人员，像质量需要3（QN3）这样的内容是很难主动想到的。

6.2.3 举例说明

下面从一个例子——“验证账号”用例的实现开始。“验证账号”用例的用户故事描述了这个用例实现的业务流程（参见下面文本框中内容）。需求人员是怎么得到这个流程的呢？一定是访谈和观察了多个用户（例如UserA、UserB）如何应用系统，然后根据他们的输入、获得的结果及过程中的操作总结出来的。开发人员根据用户故事，设计算法逻辑和数据结构，最终编码实现这个用例，其中，数据结构在面向对象的上下文中，以实体类的方式代表系统的用户（包括UserA、UserB等）。测试人员根据用户故事设计多个测试用例，使用不同的账号（UserA、UserB等）登录系统，验证是否按照需求返回了正确的界面。

> 为了完成业务工作，作为用户认证与授权系统的最终用户，通过业务系统，使用用户认证与授权系统提供的认证功能，输入用户名、密码等认证信息后，成功登录业务系统，业务系统能够提供最终用户需要的业务功能。

为什么UserA、UserB到处出现呢？因为“验证账号”用例就是定义的用户登录系统，UserA、UserB作为用户的代表，理所当然出现在各个环节，在实例化需求理论中，把这种代表实际应用对象的例子称为“实例”（图6-6）。

由于实例代表了客观存在的实体，采用实例作为沟通手段，能有效消除理解歧义，是确保信息精确无误传递的有用工具。经实践检验，精心构造的实例有助于打破“传话游戏”的困境，确保软件系统各参与方对需求有清晰、一致的理解。

举例说明需求的想法很简单，但是实施起来却没那么容易。因为如何能够找到一组具有代表性的实例来说明需求是很大的挑战。能够有效诠释需求的实例，应当具有精准（Accuracy）、完整（Completeness）、真实（Reality）、易于理解（Understandabilty）、包含非功能需求（Non-function）等特征，被称为编写实例的ACRUN规则（图6-7）。

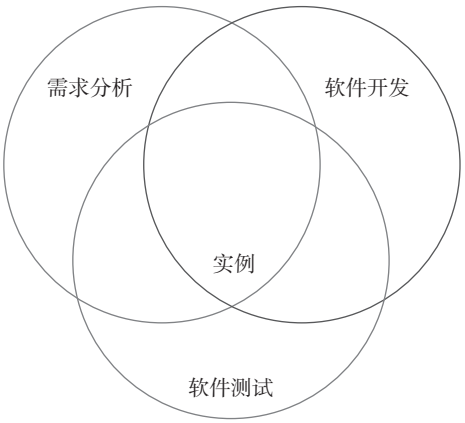

图 6-6　贯穿分析、开发、测试全过程的实例

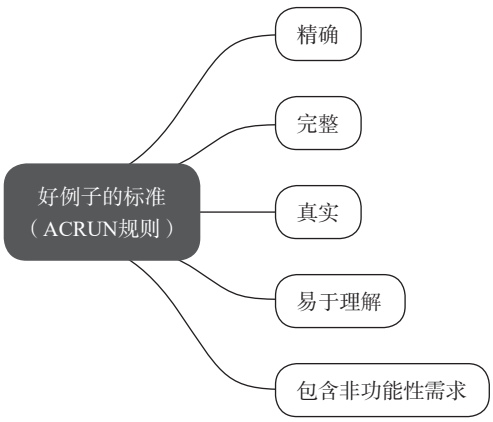

图 6-7　好实例的标准

1. 实例必须精确（Accuracy，A）

引入实例描述需求的目的是避免模棱两可。要做到这一点，实例自身必须首先做到让读者完全不会误解。因此，实例之外，还要清晰地定义出它的应用上下文。

对于“验证账号”用例，UserA、UserB是它的两个实例，只有用户名这两个实例是否精准呢？显然不是。UserA、UserB是否同一业务系统的账号、是否相同的角色等都不知道。为了更加准确，在实例的描述中需要补充这些更加详细的内容。

实例1：UserA是一个购物网站的顾客账号。

实例2：UserB是一个购物网站的商家账号。

是不是准确多了，其实，还有一点不准确的地方，在购物网站部分，是同一个网站还是不同的网站？所以，最好进一步精确化。

实例1：UserA是一个购物网站（www.abc.com）的顾客账号。

实例2：UserB是一个购物网站（www.abc.com）的商家账号。

2. 实例必须完整（Completeness，C）

实例完整包含两方面的内容。一方面是实例自身的描述内容必须完整。针对“验证账号”用例，账号登录至少需要用户名/密码信息，上述的两个实例缺失了密码的内容，需要补充。

实例1：UserA是一个购物网站（www.abc.com）顾客的账号，密码为Password_A。

实例2：UserB是一个购物网站（www.abc.com）商家的账号，密码为Password_B。

另一方面是实例集包含的所有实例是否能够完整地覆盖某个功能的所有场景。购物网站的用户主要是顾客和商家，这两个实例涵盖了这些类型。

在实践中，用主要业务场景中的业务对象作为初始实例是很好的入手点，然后再补充其他的实例，形成系统的完整实例集。以下是关于如何从一组初始实例进行扩展，最终获得覆盖所有需求的实用方法。

1）用实际数据验证实例完整性

当认为已经拥有了一组很完整的实例集后，可以用实际的数据（业务场景）来验证是否有这些实例没有覆盖的内容。这有助于发现可能漏掉的需求内容，能够使实例集更健壮、完整。在协作制定需求时，测试人员更擅长寻找此类实例。他们拥有相关知识和方法来找出漏掉的内容。

UserA和UserB都是系统中的真实用户，但是对于一个购物网站，当用实际数据验证时，会发现还有一类用户在这两类用户之外：仲裁和处理顾客与商家之间纠纷的用户。这类用户也真实存在，因此，需要一个代表他们的实例，这里称他们为仲裁员，用UserC作为用户名。

实例3：UserC是一个购物网站（www.abc.com）的仲裁员账号，密码为Password_C。

用数据做试验的一个风险是：输出太多没有显著区别的实例。这也是为什么下一个步骤“精细化需求”如此重要的原因。

2）使用替代方法来检验实例

面对复杂的老旧系统时，为了测试某个用户故事是否有一组定义良好的实例，可以要求客户使用一个替代方案来验证最终的实现。“你还会怎么对它进行测试”是开始这类讨论的一个好问题。要求客户使用替代方法来检测功能，也有助于软件团队找出自动化验证的最佳切入点。

3. 实例必须真实反映需求内容（Reality，R）

实例着重讨论真实的案例而非抽象的内容。为了更好地发挥实例的功效，实例必须真实。任何编造、简化或者抽象的实例都不会有足够多的细节，也无法展示足够多的变化。

为了确保实例的真实性，在获取实例过程中，应当注意以下两点。

1）避免虚构自己的数据

使用真实的数据十分重要，因为许多处理逻辑不同就是由于数据细微的变化和差异。如果客户和软件团队双方都使用虚构的实例来说明功能，而没有去寻找真实的数据，可能导致需求没有完全反映实际的需要。特别是当项目涉及遗留的老旧系统时，这类问题会更严重，虽然老旧系统的遗留数据使用起来比较复杂，但是，它们是对业务逻辑的最真实体现。为了减少遗留数据在迭代后期给团队带来的意外风险，需要尝试在实例中使用来自于老旧系统的真实数据，而非使用全新构造的实例。

2）直接从客户那里获得基本的实例

对于研发商业软件的团队，无法奢求会有客户（或他们的代表）参与到协作制定需求的工作坊中。产品经理会从不同的客户收集需求，并决定发布计划。软件团队可能会有非

常精确和清楚的实例，但它们可能并没有捕获到客户真正所需要的，这会导致歧义和误解的发生。

为了避免产品经理对客户需求的理解和客户真正的需要之间存在模糊和误解，要确保用来说明需求的实例是真实的，需要包含来自客户的真实数据。可以在需求工作坊中利用这些实例展开讨论。这些实例还必须包含在最终的可执行需求里，以便确保客户的期望得到满足。需要坚持使用实例来与客户沟通，以验证这些实例是否来自实际场景。

4. 实例应该易于理解（Understandability，U）

相对于抽象的描述，使用实例来展现需求更易于理解，也更容易察觉到与软件功能实现之间的不一致性和冲突。在了解一个特定功能所有实例的情况下，实例越易于理解，就越容易发现缺失的情况。如果实例不易理解，将无法评估它们的完整性和一致性。从另一方面看，如果实例过于捕获业务的细枝末节，也会使实例变得错综复杂，使用这样过于烦琐的实例来描述需求，也会增加评估需求一致性和完整性的难度。

下面是一些既能避免此类问题，又能够保持实例易于理解的方法。

1）避免探讨所有可能的组合

当团队刚开始使用实例描述需求时，测试人员常常会误解该过程的目的，并且坚持覆盖所有可能组合。但是，在举例说明需求的初期，应该寻找那些可以推进讨论并增进理解的实例。关键场景是理所当然的关注焦点，而边界场景通常是容易出现认知误解的地方，也需要关注。

“验证账号”用例已有的3个实例都代表它的关键场景，分别代表三类不同的用户类型（假如系统中只有这三类用户），不存在重复的情况。由于用户类型之间边界清晰，所以对这个用例而言，就不需要增加相应的实例了。

实例1：UserA是一个购物网站（www.abc.com）的顾客账号，密码为Password_A。

实例2：UserB是一个购物网站（www.abc.com）的商家账号，密码为Password_B。

实例3：UserC是一个购物网站（www.abc.com）的仲裁员账号，密码为Password_C。

2）发现隐含的概念

如果为了描述某个功能使用了非常多的实例，或者使用的实例特别复杂，往往是由于实例层级设计不合理造成的。分类明确、层次合理的实例集会更清晰、简洁，使需求更容易理解。通过仔细梳理特别复杂的实例，发现其中隐含的内容，并将这些内容单独定义成新的实例，能够促进实例合理分层。

使用层次清晰、精确真实的实例来描述需求，有助于读者理解所需功能的内涵。因此，要仔细地审视实例，看看是否有遗落或者隐藏的概念没有被覆盖。同时，信息充足的实例，不但能够使开发人员和测试人员从中获得足够多支持他们工作的信息，而且也可以用于需求、开发以及测试等的验收标准设计。

回顾“验证用户”用例已经抽取出来的三个实例，看看它们是不是已经代表了这个用例的全部场景，有没有需要补充的信息？编程经验丰富的开发人员可能会马上想到：账号的用户名设置没有要求么？密码没有强度要求么？最好还是设计一个实例，覆盖这样的场景。

实例4：D是一个购物网站（www.abc.com）的顾客账号，密码为d。

实际上，4.5.2节曾经提到密码的强度要求：

- 密码至少8个字符。
- 至少包括大小写字母、数字和特殊字符中的两种。

为了完整覆盖这些场景，还需要设计相应的实例。

实例5：UserE是一个购物网站（www.abc.com）的顾客账号，密码为Password，登录成功。

实例6：UserF是一个购物网站（www.abc.com）的顾客账号，密码为Pass_F，登录失败。

实例7：UserG是一个购物网站（www.abc.com）的顾客账号，密码为password，登录失败。

对于这个用例的功能需求部分，实例设计是不是就完整了呢？再回忆一下，在4.5.2节还提到“登录需要双因素认证”，这个隐含的信息在现在的实例集中也没有包含。为了在实例中体现“双因素”认证，在前面的7个实例中还应当加入相应的描述（假定以手机短信实现双因素认证）：

实例1：UserA是一个购物网站（www.abc.com）的顾客账号，密码为Password_A，手机号为139********，收到验证码并输入系统，登录成功。

5. 包含非功能性需求（Non-function，N）

非功能需求是需求不可分割的组成部分，也是对软件系统的简单易用、稳定运行、安全可靠等整体性表现的要求。虽然与描述功能需求的实例可以直接从工作场景、数据获得相比，描述非功能需求的实例获得相对间接，但是，经过深入分析与整体思考，获得高质量的非功能需求实例也非难事。

在5.3.1节可知，和“验证用户”用例相关的非功能需求如下。

QN1：为了防止密码泄露，在用户名/密码认证过程中，从输入页面传输到服务端需要加密传输。

QN2：认证过程不能过长，需要在3秒内完成。

QN3：为了规避恶意撞库，认证失败5次后，锁定登录账号10分钟。

为了覆盖这些非功能需求，要么应当丰富前面的实例，要么需要增加新的实例，例如QN1、QN2应当在原有实例上再增加一些内容。

实例1：UserA是一个购物网站（www.abc.com）的顾客账号，密码为Password_A，手机号为139********，收到验证码并输入系统，登录成功，对账号和密码进行加密传输，记录每次验证所需的时间，超出3秒的，在记录中增加“超时”标志。

而对于QN3就需要增加新的实例。

实例8：UserH是一个购物网站（www.abc.com）的顾客账号，密码为Password_H，手机号为139********，利用Password_H1、Password_H2、Password_H3、Password_H4、Password_H5尝试登录不成功后，再以Password_H尝试登录，系统应当返回“账号已锁定，请××分钟后尝试”的信息。

相对于功能需求实例，非功能性需求实例获取相对困难一些，下面是一些获取非功能需求实例的建议。

1）定义精确的非功能需求

非功能需求定义要精确，例如对于性能需求，“要比现有系统快”不是一个好的性能需求，要告诉人们具体需要多快，以及用何种方式实现。清晰地指定性能条件和举例说明有助于建立共识，并可以给开发人员提供清楚的实现目标。

2）为用户提供原型

因为客户是使用用户界面进行工作的，如果没有用户界面，客户通常会觉得难以深入理解软件系统。如果在前期的沟通中客户无法看到用户界面，很可能在和研发团队沟通中存在误解。往往研发团队以为已经澄清的内容，实际上还存在巨大的理解偏差，经常会出现客户看到真实的用户界面时，才发现系统功能并不是自己真正需要的。但用户界面的布局和易用性无法通过我们常用的真值表或自动化测试实例来体现，需要通过创建交互式原型来与客户探索模糊的需求。针对不同类型的软件系统，创建不同的保真度界面原型，有助于进行需求澄清。

3）讨论时使用检查清单

在讨论非功能需求时，客户通常会使用一个全局性的通用描述，例如，“所有页面要在1秒内加载完成”。多数情况下，这样的需求（以及其他类似的全局性需求）是有前提条件的，只不过客户通常没有那么严谨或者没有意识到。例如，对于软件系统，一般包括业务处理和报表统计两类功能，像上面的这个要求，应该指的是业务处理部分而不是报表统计部分。为了规避这种全面无差别需求的干扰，建立一个检查清单非常有用，将这些需求都加入一个清单中，为每个功能创建此类非功能性条件的验收标准。在评审一个功能实例时，按照检查清单判断这些非功能需求是否都需要满足，既保障了用户的非功能需求不被忽略，又防止了所谓的“全局通用”非功能需求问题。

4）建立一个参照的实例

非功能需求，特别是用户使用感觉方面的非功能需求，例如有用性、可信性、愉悦性等，冰冷的数字很难反映用户的真实感觉。因此，最好的方法是建立一个参照的实例，有助于客户和软件团队对系统预期达成共识。

6.2.4　精化需求

通过前面的过程，对于“验证用户”用例，现在我们已经有了两份描述客户需要的文档，一份是在敏捷开发中常用的用户故事和非功能需求，另一份是基于实际业务场景的8个需求实例。

（1）用户故事。

为了完成业务工作，作为用户认证与授权系统的最终用户，通过业务系统，使用用户认证与授权系统提供的认证功能，输入用户名/密码等认证信息后，成功登录业务系统，业务系统能够提供最终用户需要的业务功能。

（2）非功能需求。

QN1：为了防止密码泄露，用户名/密码认证过程中，从输入页面传输到服务端需要加密传输。

QN2：认证过程不能过长，需要在3秒内完成。

QN3：为了规避恶意撞库，认证失败5次后，锁定登录账号10分钟。

（3）8个需求实例。

- **实例1**：UserA是一个购物网站（www.abc.com）的顾客账号，密码为Password_A，手机号为139********，收到验证码并输入系统，登录成功，对账号和密码进行加密传输，记录每次验证所需的时间，超出3秒的在记录中增加“超时”标志。
- **实例2**：UserB是一个购物网站（www.abc.com）的商户账号，密码为Password_B，手机号为139********，收到验证码并输入系统，登录成功，对账号和密码进行加密传输，记录每次验证所需的时间，超出3秒的在记录中增加“超时”标志。
- **实例3**：UserC是一个购物网站（www.abc.com）的仲裁员账号，密码为Password_C，手机号为139********，收到验证码并输入系统，登录成功，对账号和密码进行加密传输，记录每次验证所需的时间，超出3秒的在记录中增加“超时”标志。
- **实例4**：D是一个购物网站（www.abc.com）的顾客账号，密码为d，登录失败。
- **实例5**：UserE是一个购物网站（www.abc.com）的顾客账号，密码为Password，139********，收到验证码并输入系统，登录成功，对账号和密码进行加密传输，记录每次验证所需的时间，超出3秒的在记录中增加“超时”标志。
- **实例6**：UserF是一个购物网站（www.abc.com）的顾客账号，密码为Pass_F，登录失败。
- **实例7**：UserG是一个购物网站（www.abc.com）的顾客账号，密码为password，登录失败。
- **实例8**：UserH是一个购物网站（www.abc.com）的顾客账号，密码为Password_H，手机号为139********，利用Password_H1、Password_H2、Password_H3、Password_H4、Password_H5尝试登录不成功后，再以Password_H尝试登录，系统应当返回“账号已锁定，请××分钟后尝试”的信息。

这8个实例包含了用户故事和非功能需求的内容，是它们具象化的展示，实例是比用户故事和非功能需求更具体、更容易理解的需求。但是，从上面的文字看，后者的内容重复很多，反而更容易引起混乱。因此，要想充分发挥实例化需求的潜力，一方面需要吸收实例具体、形象、贴近实际的优点，另一方面也要吸收传统需求精练、抽象度高的优点，才能精炼出高质量的实例化需求。精炼过程不仅要求提炼出核心要点，使需求表述既精确又简洁，还需要通过恰当的润色，确保无论是对当前需求的讨论还是未来可能的迭代开发，都能提供清晰无误的指导。在精细化需求时，需要综合考虑各方面的要求、原则和方法（图6-8）。

1. 需求要精确可测

实例化需求不仅是需求，也是用来验证软件系统是否按需求实现的客观标准，因此必须包含必要的信息来对系统进行验证。这些精确可测的需求，由于定义了精确的输入参数和预期的输出结果，可以很方便地转换为验收的标准。

图 6-8　精细化需求的原则和方法

2. 验证脚本不是需求

需求的目标是阐明软件系统做什么，描述的对象是业务概念，是以“人”为阅读对象的，特别要强调的是，技术人员和非技术人员都是它的读者，因此，需求要以非专业的语言表达。验证脚本的目的是验证软件系统能做什么，而不是说明系统到底应该具备什么样的功能，描述的是技术概念，是以“计算机”为阅读对象的，是技术人员使用的文档，随着自动化验证的大规模应用，更多的脚本是通过软件代码实现的。

3. 不要使用流程式的描述

除非是描述一个真正的处理流程，否则不要使用流程式的描述。使用流程式的描述是脚本化的标记。实例化需求不应该是关于系统如何工作的描述，而应该是说明系统应该做什么。

4. 需求应关注业务功能，而不是软件设计

原则上，需求不应该涉及软件设计。实例化需求只解释业务功能，对软件系统采用哪个算法、如何实现要保持开放性。这样做有两个优势：一是专业的人做专业的事，需求人员只专注业务功能，软件设计由开发人员根据所用技术的特点选用最佳解决方案；二是便于开发人员改善设计，设计方案的变动不会影响需求，并且仍然可以使用实例化需求进行业务验收。

5. 避免编写与代码实现紧密耦合的需求

软件系统的功能实现可能有不同的方式，例如，对一组对象的排序，有冒泡法、插入法、选择法、堆排序等多种，如果实例化需求与代码算法实现紧密耦合，会对需求的业务属性产生较大的影响，可能导致在软件功能没有进行任何变动时，仅仅由于软件算法实现的变化，需求实例就会失效。而且需求与算法紧耦合一方面会使需求的独立性大大降低，另一方面也会给实例化需求添加额外的维护成本，体现不出实例化需求的作用。

6. 不要在需求中引入技术难点的临时解决方案

在处理遗留老旧系统时，经常会遇到一些特殊的技术问题，并且非常难以修改。在进行实例化需求时，需要特别注意区分实际的业务流程和临时的技术解决方案。

理论上讲，需求归需求，技术归技术。在需求中，不需要涉及技术难点的问题。因为如果把需求绑定到技术问题上，会破坏需求独立性，可能导致对代码的细小改动，都需要大量的时间来更新需求。解决问题的方法是把技术难题放到技术攻关中解决，不要试图在需求中解决。这样做会使变更和改进系统更加容易。在进行技术改进时，需求也不会受到影响。

实际上，在解决老旧系统改造中的一些技术难题时，很难做到和需求无关，大多数情况需要引入额外的系统功能，以隔离或引流原来功能的访问流量。比较好的做法是保持这部分需求内容和真正业务需求之间松耦合，把这些需求隔离为单独的需求集，当完成系统改造后，把这个需求集丢弃。

7. 不要陷入用户界面的细节里

实例化需求的目的是通过引入实例的方法增强对需求的理解，减少不同团队之间沟通的信息衰减，而不是为了实现系统的用户界面。在制定实例化需求阶段，需要关注的重点是软件系统如何对外界输出所需的价值、提升所要支持的业务流程运转效率、降低工作人员劳动强度等重要的内容。用户界面的设计，有很多与用户体验相关的专业性要求，这应该是UX专业人员进行的工作（在4.5.2节简要介绍了设计界面的“四步法”，要设计高质量的用户应用界面，需要专业人员更多的投入）。

8. 需求应清晰易懂

在需求完成之初，由于是现有团队中成员协作完成的，团队成员（可能有部分是中间加入）都能理解它。但随着时间推移、团队中成员变迁，需求初创时期的成员可能大部分已经离开，如果需求不是清晰易懂，就可能出现信息理解不一致的情况。

9. 使用叙述性标题并使用短篇幅文字阐释目标

需求不能只包含输入与预期的输出，否则读者需要从输入输出中反推业务规则，使需求变得晦涩难懂。应当为需求定一个含义明确的标题，并在开篇就解释需求的目标和结构。这样的需求会更加友好，方便读者阅读。标题应总结出需求的意图，当读者搜索某个功能的解释时，可以很容易通过标题找到适当的需求。

10. 展示给别人看并保持沉默

如果无法做到协作制定需求，只能独自编写时，可以用下面的小技巧来检查需求是否清晰易懂。当需求编写完成后，可以拿给其他人看，在什么都不解释的情况下，看对方是否能够理解。如果发现必须要给对方进行解释时，就把解释的内容放在需求的开篇描述中。通过增加这样的内容，会使需求更加易于理解。

11. 不要过度定义需求

不要对实例进行过度的填充，试图覆盖所有输入参数的组合。过多的实例会淡化关键实例的价值，使实例变得难以理解，不再清晰易懂，定义3个良好的关键实例比定义100个不好的实例更加有用。关键实例包括那些描述重要业务功能、重要技术边界条件、业务容

易出现问题（比如以前导致了缺陷）的实例等。

12. 从简单的需求入手，然后逐步展开

对于需要使用许多参数组合来描述业务规则的需求，为了不让需求过于复杂，从编写高层次的实例化需求开始，详细的内容可以逐步补充。先从基础的主流程入手，然后根据业务重要性逐步增加其他内容。

13. 需求要专注

一个需求应该单独描述一件事情。专注的需求相对简短，比同时定义有多个关注点的需求更容易理解，容易达到“清晰易懂”。并且由于“专注”，需求会更易于维护，受其他方面的影响较小，基于此需求的验证也不会被频繁破坏，并且出现问题后更容易定位问题。

14. 在需求中使用“假定－当－那么”

为了让需求更容易理解，需求应该声明应用上下文，指定一个单一的动作，然后定义预期的后置条件。可以使用“假定（Given）-当（When）-那么（Then）”来进行描述。

```
假定一个前提
当某个行为发生时
那么后置条件就会得到满足
```

其实这也是Gherkin语法（参见7.2.2节）的格式，是业界广泛应用的实例化需求描述方式。对于使用表格、基于关键字或者自由文本系统的工具，也可以参照这样的格式来组织需求。

15. 不要在需求中明确建立所有依赖

在那些需要复杂配置的数据驱动软件系统中，不要把所有先决条件的配置和设置都放入需求中。虽然这会让需求看起来完整，但会让需求变得更加复杂，难以阅读和理解。即使和要描述的业务规则没有直接关系，配置中任何相关对象或属性的修改也会影响需求的验证。需求只需要专注于重要的属性和对象，将其他所有和需求目标不相干的依赖全部移到后续实现中。

16. 需求应使用领域语言

需求面向用户、需求人员、测试人员、开发人员以及任何想要了解系统的人，需要用每个人都能理解的语言编写。文档中使用的语言必须一致，可以减少不同人员阅读带来误解的可能性。

这16条规则和方法是众多实例化需求应用者在实践中总结出来的、能够提升需求质量的有效方法。根据这些原则和方法，从用户故事、非功能需求和最初的8个实例中，可以进一步精细化，得出3个更加清晰、易懂的实例化需求。

实例化需求 1：不同类型用户登录成功的需求

作为购物网站（www.abc.com）的用户，用户已经在网站中建立了自己的账号UserA（用户）、UserB（商户）、UserC（仲裁员），其中包括进行双因素认证的手机号，现在需要应用账号，并通过多因素认证登录系统，获得自己个性化的配置页面，完成自己的工作。

```
假定：我通过浏览器，输入www.abc.com网址，网站返回系统登录界面
接着：在登录页面的<用户名>编辑框中输入用户名，在<密码>编辑框中输入密码
当：单击“提交”按钮后
那么：验证成功时，向设定的<手机号>发送验证码
接着：记录提交到发出验证码的时间间隔
接着：把设定的<手机号>收到的验证码输入<验证码>编辑框中
当：单击“提交”按钮后
那么：返回“成功”表示登录成功，“失败”表示登录失败
实例：
| 用户名 | 密码          | 手机号          |
| UserA  | Password_A    | 139********    |
| UserB  | Password_C    | 139********    |
| UserC  | Password_C    | 139********    |
```

实例化需求 2：密码约束条件实现需求

作为购物网站（www.abc.com）的用户，用户已经在网站中建立了自己的账号D（密码长度不足）、UserE（可成功登录）、UserF（密码长度不足）、UserG（密码复杂度不足），其中包括进行双因素认证的手机号，现在需要使用账号测试密码的约束规则（密码至少8个字符，至少包括大小写字母、数字和特殊字符中的两种）是否正确实现。

```
假定：在浏览器中输入www.abc.com网址，网站返回系统登录界面
接着：在登录页面的<用户名>编辑框中输入用户名，在<密码>编辑框中输入密码
当：单击“提交”按钮后
那么：验证成功，向设定<手机号>发送验证码，验证失败，返回“失败”信息
实例：
| 用户名  | 密码         | 手机号          | 返回信息 |
| D       | D            | 139********    | 失败     |
| UserE   | Password     | 139********    |          |
| UserF   | Pass_F       | 139********    | 失败     |
| UserG   | password     | 139********    | 失败     |
```

实例化需求 3：多次登录尝试失败，锁定账号需求

作为购物网站（www.abc.com）的用户，用户已经在网站中建立了自己的账号UserH（多次使用错误密码登录），其中包括进行双因素认证的手机号，现在需要使用账号测试密码多次尝试登录约束规则（为了规避恶意撞库，认证失败5次后，锁定登录账号10分钟）是否正确实现。

假定：在浏览器中输入www.abc.com网址，网站返回系统登录界面
接着：在登录页面的<用户名>编辑框中输入用户名，在<密码>编辑框中输入密码
当：单击“提交”按钮后
那么：验证失败，返回“密码错误，还剩<×次>尝试机会”或“账号已锁定，请10分钟后尝试”信息
实例：

用户名	密码	次	返回信息
UserH	Password_H1	1	密码错误，还剩4次尝试机会
UserH	Password_H2	2	密码错误，还剩3次尝试机会
UserH	Password_H3	3	密码错误，还剩2次尝试机会
UserH	Password_H4	4	密码错误，还剩1次尝试机会
UserH	Password_H5	5	密码错误，还剩0次尝试机会
UserH	Password_H	6	账号已锁定，请10分钟后尝试

以上需求既是内容的描述，又是实际例子的展示，这样的描述是否更清晰，使开发人员和测试人员对于需求内涵的把握更准确呢？但是，这还不是实例化需求的全部。回想一下引入实例化需求的原因：在需求、开发、测试等环节使用相同的实例，以解决在不同角色间信息传递衰减的问题。因此，实例化需求一方面指导研发团队实现软件系统的需求，另一方面也包含需求验收标准的信息，可以转化为自动化测试用例，验证软件系统实现是否体现了需求的内容（图6-9）。

前面讲述的内容是如何通过精细化典型实例得到实例化需求，核心关注点是生成实例化需求。在后续内容中，将进入实例化需求在软件系统研发过程中的应用，也是其对提升软件系统研发效率最有价值的部分。

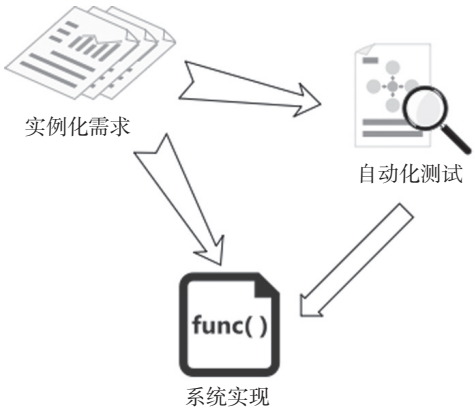

图 6-9　实例化需求作用

6.2.5　在不修改需求的情况下进行自动化验证

经过从目标中获取范畴，软件系统研发找到了正确的业务目标；通过协作制定需求，保证团队内部建立统一共识；通过举例说明，实现了既能避免歧义又能进行准确沟通的好方法；经过精细化需求，得到了足够优秀的实例化需求，它一方面以更清晰、更易理解的方式描述软件系统需要实现的目标，另一方面，由于实例的引入，实例化需求中也包含了所要实现功能的验证标准。在软件系统研发时，可以使用实例化需求来验证功能是否正确实现；在系统发生变更时，也可以使用实例化需求验证原有功能是否依然有效。

由于实例化需求内容详细，包含了需求的验证信息，所以非常适合自动化验证的进行。但是，如果在自动化过程中必须大量地调整需求内容，那么将再次陷入“传话游戏”的困境，并且还会失去实例化需求带来的价值。因此，为了不扭曲任何信息，需要在不修改需求的情况下进行系统的自动化验证。

本书非常强调自动化验证的作用，因为非自动化验证必须由人手工执行，从机制上，验证效果就不可避免地会受到人员能力的影响；相对而言，自动化验证从理论上消除了执行人员的影响，验证效果易于保持客观、一致。并且就软件系统研发的众多实践而言，自动化验证在提升软件质量，减少需求与开发、测试等角色间误解方面有巨大潜力。对于大型团队而言，自动化验证是确保客观评价工作成果、进行全面持续验证的基础。从长远来看，自动化验证还将扮演构建活文档的关键角色，确保需求的准确性与可读性，会为团队带来持久的收益。

因此，我们不应回避自动化验证带来的挑战，而应积极寻求解决问题的有效策略。实例化需求与自动化工具是开展自动化验证的基础，经过不断演进，现在已经成为推动项目成功的强大动力。

自动化工具的技术实现将在第7章进行详细介绍，这里着重介绍如何有效展开自动化验证的方法。

1. 从头开始自动化验证

在不修改需求的情况下进行自动化验证，其中包含的一个重要原则是“测试先行”，而能够自动化验证是“测试先行”的基础。因此，从软件系统实现开始就要坚持自动化验证。

1）为了学习工具，可以先尝试一个简单的工程

简单的工程可以最小化风险，无须处理复杂的集成与业务规则，可以专注于学习如何使用工具，并且不会对正在进行的开发工作带来太大风险。

2）事先规划自动化策略

在自动化验证的导入期，生产效率可能会有所下降。即使不考虑学习使用新工具带来的消耗，自动化验证在初期还是会给项目增加显著的开销。首先，自动化验证需要前期准备，开始自动化之验证前有大量的工作必须完成，包括创建基础的自动化组件，确定可执行需求的最佳格式及与系统集成的最佳方式，解决测试稳定性与专用环境的问题等诸多事项。其次，进行自动化测试，需要在测试脚本编写方面投入大量的精力和资源。

3）不要拖延自动化验证或将其委派他人

如果没有一个客观的自动化验证标准，当开发人员把需求标识为已完成时，将无法进行及时确认，可能会导致在最终验收时才暴露前期的问题，研发团队不得不在较长时间后修复这些问题，导致需要投入更多的资源。因此不要因为自动化测试的开销而推迟自动化，因为这只是权宜之计。虽然推迟自动化验证仿佛会加快交付需求的速度，但返工往往会在更大程度上影响系统的研发进度。

当自动化验证与开发同时进行时，能够增加开发人员对系统可测试性的关注。但当自动化验证延迟或委派给他人时，开发人员就不会再关心这个问题了，这将导致自动化验证更难、花费更高。

4）避免根据原有的手动验证脚本进行自动化验证

手动验证与自动化验证的制约条件是完全不一样的。在手动验证中，准备上下文环境是最耗时的环节，会倾向于重用验证的上下文环境。而在自动化验证中，时间主要花在寻找验证失败的原因上。因此在自动化验证中，更多是相对于较小并且更专注的验证。如果在单一脚本中同时检查多件不同的事情，因为各种不同部分的代码都会影响它的结果，容易出错，并且不容易定位错误。使用一组较小、专注且独立的验证脚本，会让验证更具弹

性，并且可以降低维护成本，同时也有助于更快地找出问题，会显著降低自动化验证的开销与维护成本。

5）通过用户界面验证赢得信任

对刚开始使用实例化需求的团队，当团队成员质疑可执行需求时，可以尝试让其通过用户界面来执行需求验证。“端到端”的验证能够较快地让软件系统研发团队收获自动化验证的成果，给软件研发团队带来以下收益。首先，需求人员能够较早看到软件系统运行的效果；其次，研发人员能够减少与需求人员之间没完没了的需求澄清，可专注于软件系统实现；第三，验证人员不用再猜测需求人员想要什么，研发人员实现了什么，它们之间是否存在“鸿沟”；第四，“端到端”验证规避了不同模块之间长期的不匹配，减少只有在系统集成时才暴露出大量问题的风险。

以上讲解的几个如何引入自动化验证的方法可能比较抽象，下面用一个“实例”来进行验证。当前存在多个技术框架，能够支持基于Gherkin语法书写的可执行需求（见7.2.1节），下面就用Gherkin语法把实例化需求1重构成可执行的需求。

```
特性：不同类型用户登录成功需求
作为购物网站（www.abc.com）的用户，用户已经在网站中建立了自己的账号UserA（用户）、UserB（商户）、UserC（仲裁员），其中包括进行双因素认证的手机号，现在需要应用账号，并通过多因素认证快速登录系统，获得个性化的配置页面，完成自己的工作。
场景大纲：用户登录过程
假定：通过浏览器输入www.abc.com网址，网站返回系统登录界面
那么：在登录页面的<用户名>编辑框中输入用户名，在<密码>编辑框中输入密码
当：单击“提交”按钮提交后
那么：验证成功时，向设定的<手机号>发送验证码
那么：记录提交到发出验证码的时间
那么：把设定的<手机号>收到的验证码输入到<验证码>编辑框中
当：单击“提交”按钮后
那么：返回“成功”表示登录成功，“失败”表示登录失败
例子：
| 用户名 | 密码          | 手机号          |
| UserA  | Password_A    | 139********    |
| UserB  | Password_C    | 139********    |
| UserC  | Password_C    | 139********    |
```

注意这个可执行需求中的粗体字部分，它们是和实例化需求1描述有区别的地方，如果没有编写过实例化需求，一定感到非常惊奇：这样的描述就能够执行了？是的，本书的例子程序中有大量这样的可执行需求定义文件，并且都是可执行的，读者可以尝试一下。这也是为什么实例化需求引入的可执行需求能够成为非技术人员与技术人员沟通桥梁的原因所在：以最小的格式变化，既实现非常容易读懂的文档，也实现可执行的需求。

2. 有效管理自动化验证代码

也许你还没有从实例化需求1重构为可执行需求的错愕中回过神来，那么就让我们看看这一魔术后面的奥妙：并非上面的“特性”描述能够直接编译成可执行的计算机“0101”代码集，它的可执行是通过和程序代码双向绑定间接实现的。在实例化需求领域，应用最广泛的是Cucumber框架，它用正则表达式实现“场景大纲”中“假定、那么、当”步骤描述和验证用例（这里以JUnit为例）绑定，实现它的可执行。例如：

步骤描述：当单击“提交”按钮提交后

正则表达式：@当("单击‘提交’按钮{string}后")

绑定函数：public void stepDefFunction(String str)

不要纠结这些是如何实现的，有大量的可用框架已经完成了上述功能，作为需求人员，只要知道并遵循实例化需求的书写格式就可以了。需求人员的关注焦点在业务，而不是这些技术实现。

在实例化需求中，自动化验证是它的核心价值之一，相对于传统的需求模式，需要额外投入一部分努力。但是，任何一项技术的引入，其优劣最终的评价标准都是收益是否大于投入，而对自动化验证代码的有效管理是降低实例化需求引入成本的有效手段。下面是几个能够提升自动化验证质量的最佳实践。

1）别把自动化验证代码当作二等公民

要提升对自动化验证代码的重视程度，把自动化验证代码本身当成可执行需求的一部分。软件系统研发要取得成功，“做正确的事”比“正确地做事”更重要。实例化需求的自动化验证代码可能比业务逻辑实现代码的生命周期更长。特别是在进行业务逻辑实现代码重构时，如果有自动化验证用例，研发人员的幸福指数会呈几何级数增长。

2）在自动化验证代码中细化验证过程

“需求归需求、代码归代码”是软件系统工程专业化分工的原则。需求关注的是软件系统运行的结果而不是实现过程。例如，从需求角度看，“那么：在登录页面的<用户名>编辑框中输入用户名，在<密码>编辑框中输入密码”和“当：单击‘提交’按钮提交后”这两步紧密相连，但是，从代码实现角度看，在两个步骤之间至少还有如何从页面拿到用户名和密码、如何把它们传输给后台服务、如何实现用户匹配等多个环节。在实例化需求实现中，把这些验证过程步骤交给了自动化验证代码，从而把需求和具体的实现细节隔离，使实例化需求的业务属性更强、关注内容更加聚焦。

3）不要在自动化验证代码里掺入业务逻辑

在自动化验证用例中模仿软件系统的部分业务流程或逻辑，虽然可以使验证更容易通过，但是，不要忘记：自动化验证代码是可执行需求的扩展，属于需求的范畴，不是软件系统逻辑实现的部分。如果把某个问题的解决逻辑掺入自动化验证代码当中，会造成这个自动化验证执行成功，但在真实系统中执行业务时却失败，因为这些逻辑不在系统的实现代码中。这样自动化验证就失去了其存在的意义。

3. 对用户界面进行自动化验证

从用户界面进行自动化验证是非常直接的验证方式。但是，在传统上，特别是富客户端流行的年代，对用户界面直接进行自动化验证曾经给人们留下很多痛苦的记忆：辛辛苦

苦、夜以继日、长时间录制的界面自动化验证脚本，由于界面上一个不起眼的微小变化就全部作废，需要重新录制。

但是，随着B/S架构的流行，HTML成为描述用户界面的主要工具，对于网页的自动化验证已经从可能走向成熟。当然，这种可测试性来源于对用户界面的精心设计。

1）标识每一个界面元素

在4.6.3节介绍借鉴网址URL（Uniform Resource Locator）编码方法的结构化文本标签编号方法时提到，高质量的功能编号不但能够标识功能，还能够被用来作为软件系统实现界面元素的标识，从而把需求和实现它的软件系统元素有机地联系起来，有助于系统功能的完整性和可测试性。

在HTML实现的网页用户界面中，它的每一个元素，包括编辑框、按钮、菜单都有一个ID属性，这个属性是一个全局变量，可以在整个网站的海量元素中唯一定位到特定的元素，这个ID与元素的其他属性，包括位置、形状、状态等都无关。页面元素的ID属性可以由系统自动生成，也可以在页面设计时提前赋值。例如，在用户认证与授权系统中，把登录时输入用户名需求的编号AAS.UserApp.Login.username作为ID赋予登录界面的用户名输入编辑框，从而在用户登录的可执行需求的自动化验证代码中，直接引用这个ID给用户名输入编辑框赋值：

```
@FindBy(id = "AAS.UserApp.Login.username")
private WebElement username;
```

通过高质量的系统用户界面设计和相关技术框架的结合，能够一劳永逸地实现针对页面自动化验证的代码。

2）引入Mock程序

“端到端”验证所要解决的另外一个问题是：如何在系统逻辑还没有完全实现时，使自动化验证用例能够运行。在软件系统研发的实践中，引入Mock程序是解决这一问题的有效方法。通过Mock程序实现软件系统不同模块之间的隔离，可以有针对性地自动化验证当前关注的模块，而不受其他部分的影响。

4. 管理验证数据

可执行的需求中包含所有对业务逻辑重要的数据，用来对功能进行举例说明，同时也忽略了一些其他不重要的信息。但当软件系统执行时，需要完整的数据支撑，这就需要把忽略的数据补充完整，才能实现该系统的所有实例自动化验证；下一个要关注的点是当有多个验证需要运行时，某一个验证可能会更改另一个验证所需的数据，使得验证结果并不可靠；还有一点是为了获得快速反馈，也不能在每一个验证里删除和恢复整个数据库。

为了解决以上问题，需要对数据做一些有针对性的处理。

1）避免使用预填充数据

自动化测试用例中最好只使用自己准备的数据，而不重用数据库中符合其目的的已有数据，因为这样会使数据库中的数据成为自动化上下文的一部分。虽然可以让需求更容易自动化，却增加了它们的理解难度。如果阅读此类需求的人不知道数据库中的内容，就很难理解需求的业务逻辑。

2）在验证用例中说明使用了哪些公共数据

在众多的企业中，会有一些包括自己核心业务对象（如客户、产品）等的主数据，这些数据不需要每个验证用例自己准备，可以引用现存的主数据。但是，要在需求的前置条件中描述清楚引用了哪些外部数据。

3）针对遗留系统从实际数据中获取验证用例

对遗留系统，如果领域非常复杂，从头开始建立验证数据集可能是一项复杂且容易出错的工作。在这种情况下，可以从它的实际数据中找出有代表性的实例，并依照其属性来准备自动化验证。使用这种方法来快速创建验证数据集，还可以简化在可执行的需求中为相关数据做准备的工作。

6.2.6 持续验证

基于软件系统无形性的本质特征，软件工程在演进过程中突破了传统工程实践（包括建筑、机械等，它们也是软件工程理论最初的来源）的限制，特别是在检测、验证方面，不再采用传统工程中的抽样、固定点模式，而是应用全系统的、持续不间断的验证模式。这一模式在软件工程中称为持续集成（Continuous Integration，CI），是软件开发过程中被广泛应用的一种最佳实践。自动化验证是持续集成的技术保证，在每次代码合并，甚至每次代码修改提交时，都能触发自动化的构建和验证，以确保这些变更按照需求进行，而且没有破坏已有的功能。持续集成的目标是尽早发现并修复问题，从而避免将问题传递到后续集成阶段。

当软件开发团队偏离了“持续构建、持续交付”的核心理念时，持续集成机制就会发出预警信号，促使团队纠偏。持续集成最好能有专用的、类似生产的环境，从而确保软件不仅能在开发者的本地环境中稳定运行，还能够展示出在预期的运行环境也具有良好的兼容性与可靠性。持续集成可以确保软件系统不但能够被正确地构建，而且也能够保持正确运行。通过可执行需求的持续验证，可确保软件系统研发时刻不偏离正确的方向。

想要持续验证可执行需求，需要克服以下三个问题。

（1）验证用例稳定性不足。

可以稳定执行验证的实例化需求才具备价值。如果验证过程不稳定，会减弱软件团队实例化需求应用的信心。如果频繁出现不是由于真正问题导致的失败，不但会浪费大量时间，开发人员也将不会再去查看验证出现的问题，哪怕是验证出了真正的问题也可能会被忽视。这样整个持续验证将变得没有意义。

（2）验证用例反馈较慢。

需求验证用例由于具有一定的集成验证性质，所以运行速度通常比单元测试慢很多。即使无法解决整体反馈慢的问题，我们也需要想出一种解决方案，做好验证用例的规划，尽量按照软件系统的模块划分验证，减少不必要的模块间交叉调用，降低验证的复杂度，从而提升反馈的效率。

（3）验证用例大量失败。

如果有大量的、依赖于多个模块的验证用例，会造成许多验证失败。大多数情况下，团队需要去管理失败的验证用例而不是直接进行修复。

为了实现可执行需求持续验证的目标，需要在提高稳定性、获得更快的反馈和管理失败的验证用例三方面进行不间断的努力（图6-10）。

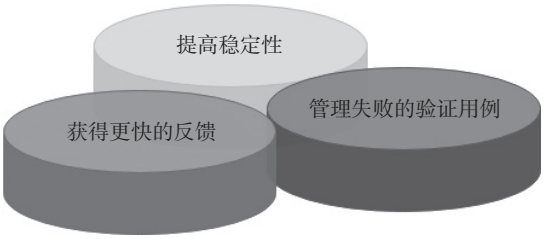

图 6-10　助力持续验证可执行需求的方法

1. 提高稳定性

可执行需求高度依赖稳定的运行环境，大部分非预期的验证用例失败与环境变化有关。因此，为了能够实现稳定的自动化验证，不但需要提升验证用例的质量水平，还要提高环境、外部连接、数据等的稳定性和可靠性。

1）在提升验证用例水平方面

（1）找出最大的问题并将其解决，然后不停地重复。

当自动化验证支持还不够好时，不要一次性引入过多验证用例，这样会使研发团队应接不暇，更坏的情况是影响交付进度，从而导致研发团队放弃自动化验证。欲速则不达，最有效的策略是一个迭代一个迭代地引入，每次都先找出影响最大的问题并解决，然后再解决另一个影响大的问题。通过不断的重复，一个迭代一个迭代地提高自动化验证的稳定性。这种方法也可以让研发团队在提高系统可测性的过程中不断学习并适应。

（2）利用持续集成验证历史信息，找到不稳定的验证用例。

对于一个不具备自动化验证能力的老旧遗留系统，由于存在太多导致验证不稳定的地方，往往难以找到最不稳定的验证所在。有一个方法有助于找到目标：如果可执行需求是在持续构建系统中运行，可以从验证用例的运行历史中查找哪些最不稳定，如果历史的构建是通过手工完成的，可以去查找它的历史测试报告，看看哪些地方的测试轮次最多，大概率是最不稳定的地方。通过这种方法，找到最大的问题并将其解决，然后不断重复这个过程，直到持续集成稳定运行。

2）在提升验证环境方面

（1）搭建专用的持续验证环境。

持续验证必须以可重现的方式进行，而保证可重现性的一个关键是确保专用的部署环境。如果软件团队将一个环境用作不同的用途，很多人会对环境做修改，往往会由于环境原因导致验证不通过。没有专用的环境，很难确定验证用例失败的真正原因是因为系统缺陷，还是因为对验证环境做了改变或者是系统不稳定。专用的环境可以排除计划外的改变，并降低环境不稳定的干扰。

（2）使用全自动部署。

不可靠的部署是验证结果不稳定的第二个最常见的原因。软件团队不但需要专用的环境，还要确保软件以可重复的方式部署。使用全自动部署，可以确保部署升级只有唯一的标准流程，可以大幅提高持续验证的可靠性。

3）在外部连接处理方面

（1）为外部系统创建较简单的验证替代品。

大型企业往往有着复杂的系统集成关系，软件团队负责的可能只是大型系统的一部分。验证系统需要和其他团队的验证系统通信，但是，无法保证其他团队的验证服务器始终可用、可靠并且正确。针对这种情况，最好创建Mock，用来模拟与其他系统的交互。使用这种方法也有潜在风险，那就是真实系统会随着时间演化，如果Mock没有及时更新，会无法反映真实的功能。为避免这个问题，要确保及时对验证进行一致性检查，保持Mock可以像真实系统一样工作。

（2）尝试多级验证。

对于大型的复杂集成系统，可以尝试使用多级验证。每个团队使用自己专用的持续验证环境，保证自己的变更得到验证。只有当验证在自己的环境中通过后，才可以把修改推送到持续验证的中心环境，与其他团队的变更进行集成。这样可以防止一个团队的问题影响到其他团队。同时，即使中心环境出现问题不可用，各个团队仍然可以使用自己的环境来验证自己所做的变更。

4）在验证数据管理方面

大多数情况下，引用公司的主数据是可执行需求能够运行的基础。任何对引用数据的变更都有可能破坏验证。专门设置一组完全独立的、可快速运行的用例来验证引用数据是否完整、正确。可以先运行验证，如果失败，就没有必要再运行其他验证了。

2. 获得更快的反馈

当包含大量验证时，执行所有验证，并从中获得反馈可能会很慢（像本书的例子用户认证与授权系统，系统很小，但是，每次完整地运行所有的可执行需求也需要以小时计）。每次系统变更后，全部执行所有的可执行需求会造成效率低下。为了能够提升开发效率及反馈速度，需要对可执行需求的验证进行精心设计。

1）验证模块化设计

一个可执行需求往往需要大量的验证步骤，特别是当可执行需求比较复杂、处于整个系统（如大型的集成系统）的顶端时，验证可能需要跨多个系统执行。这样的验证不但执行起来耗时较多，而且也容易失败，如果处理不好，就会发现“准备环境”花费了技术人员大量的时间。因此，验证需要进行精心设计，特别是要进行模块化，这样可以控制每个验证的涉及范畴，使开发人员有更多精力专注于解决真正的业务需要。

2）验证分类

基于需求的复杂度，有些用例需要验证较多的内容，有些用例只需要验证少量的内容。最好的办法是把不同类型的验证用例分类，一方面，在修改不涉及复杂用例的程序时，可以在修改过程中尽量少执行；另一方面，当执行不可规避时，技术人员可以做好规划，在这些验证用例执行的同时去完成其他工作。

3）使用定时任务

高质量的可执行需求以自动化验证用例为基础。在技术人员的工作时间里，只运行与自己的修改直接相关的验证，并且保持只提交通过验证的修改。那么在技术人员下班后，有大量的时间可以用来进行系统核心验证或系统全部验证的自动化运行。

3. 管理失败的验证

持续验证能够及时发现许多问题，当系统复杂、需求变更多时，验证失败的数目也会很大。这些验证失败所代表的问题有可能得到及时解决，也有可能需要客户进一步澄清，或者因为优先级不够而排在现在开发的功能之后。因此，持续验证中会有持续失败的验证。对这些失败的验证不能置之不理，要对它们进行有效的管理。

1）创建独立验证集

当发现不需要马上进行修复的失败验证时，为了不影响正常工作，可以先把它们放到一个独立的验证集中。放到一个单独的验证集中并不意味着置之不理，而是要持续跟踪此类问题，防止出现问题堆积。有限的几个并不重要的问题可能对产品发布没有影响，但是如果这个集合数量过多，就应该停下来做整顿，以免场面失控。如果有的验证失败了很长时间也没有人关注，那就说明它所代表的功能应该剔除，因为已经没有人关心这个功能了。

2）自动检查被禁用的验证

有时没有把失败的验证放到单独的集合中，而是禁用了这些验证，这样也可以达到已知的失败验证不会再破坏整个验证过程的目的。这种方法的问题是很容易遗忘这些被禁用的验证。因此，需要自动监控及定期评审这些验证，防止遗忘。

6.2.7　演化出一个活文档系统

一个长期运营的软件系统，不仅包括从代码编译出来的可执行程序，还包括供各种利益相关方使用的高质量文档系统。传统上，这些文档由研发团队中的不同人员产出，供其他利益相关方使用。受不同人员对于系统的理解程度、自身能力和文档质量控制等因素的影响，这些文档之间总是或多或少地存在信息不一致的地方，并且随着时间的推移，这种不一致变得越来越多。实例化需求成功地将需求信息“直接”输入到代码中，而代码中本来就存在系统的架构、设计等信息，那么能不能从代码中直接生成软件消费者需要的文档呢？实例化需求为这种想法提供了一种解决思路，而行为驱动开发框架为这种想法提供了技术支撑，促进了“以代码为中心”的“活文档系统”的产生（图6-11）。

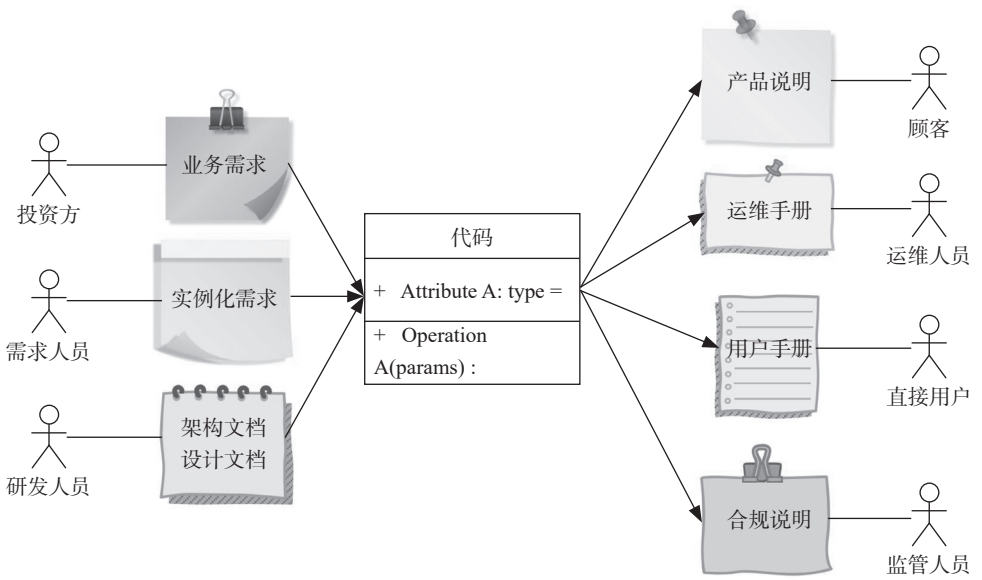

图 6-11　以代码为中心的活文档系统

活文档系统的核心在于其动态性和自解释性，它要求文档内容能够随着项目的进展而不断更新，并且每个部分都具备足够的独立性和完整性，以便在任何时候都能为相关人员提供准确、全面的信息。因此，在构建活文档系统时，不仅要关注需求的准确性和完整性，还要注重其组织结构和呈现方式，确保它们能够真正成为项目推进过程中不可或缺的辅助工具。

1. 实例化需求是活文档信息的重要来源

实例化需求第一次把需求变成了代码的一部分，把需求的内容“直接”输入到代码中，在没有降低代码严谨性的情况下，增加了程序代码中的信息含量。也为从代码中直接生成软件系统消费者需要的各种文档（例如产品说明、用户手册、运维手册）提供了重要的信息来源，并打下了坚实的基础。

2. 活文档及时可靠

由于活文档系统中的各类文档，无论是为了生成代码输入的可执行需求，还是由代码生成供消费者使用的用户手册、运维手册等内容，都实现了和代码严格对应，从不同角度、以不同的格式展现了同一份代码的含义，并且要么能够自动生成代码，要么自动从代码中生成，因此都能及时、可靠地反映软件系统的逻辑实现。

“活文档”作为解决软件系统算法、数据、文档三大组成部分中传统文档系统信息不一致的重要途径，将在第7章中讲解，这里不再赘述。

总结

实例化需求就是用实例来精细化需求，并通过引入新的技术框架，使需求在实例的加持下具有可执行的能力，从而成为解决需求频繁变更、需求传递过程中信息衰减等问题的有效方法。实现实例化需求并非一蹴而就的工作，需要一组过程模式来帮助团队转变。本章重点介绍了实例化需求的概念，实例化需求的“从目标获取范畴”“协作制定需求”“举例说明”“精细化需求”“在不修改需求情况下自动化验证”“持续验证”“演化出一个活文档系统”七个关键流程模式来协助需求人员熟练掌握如何使用实例化需求方法撰写高质量的软件系统需求。