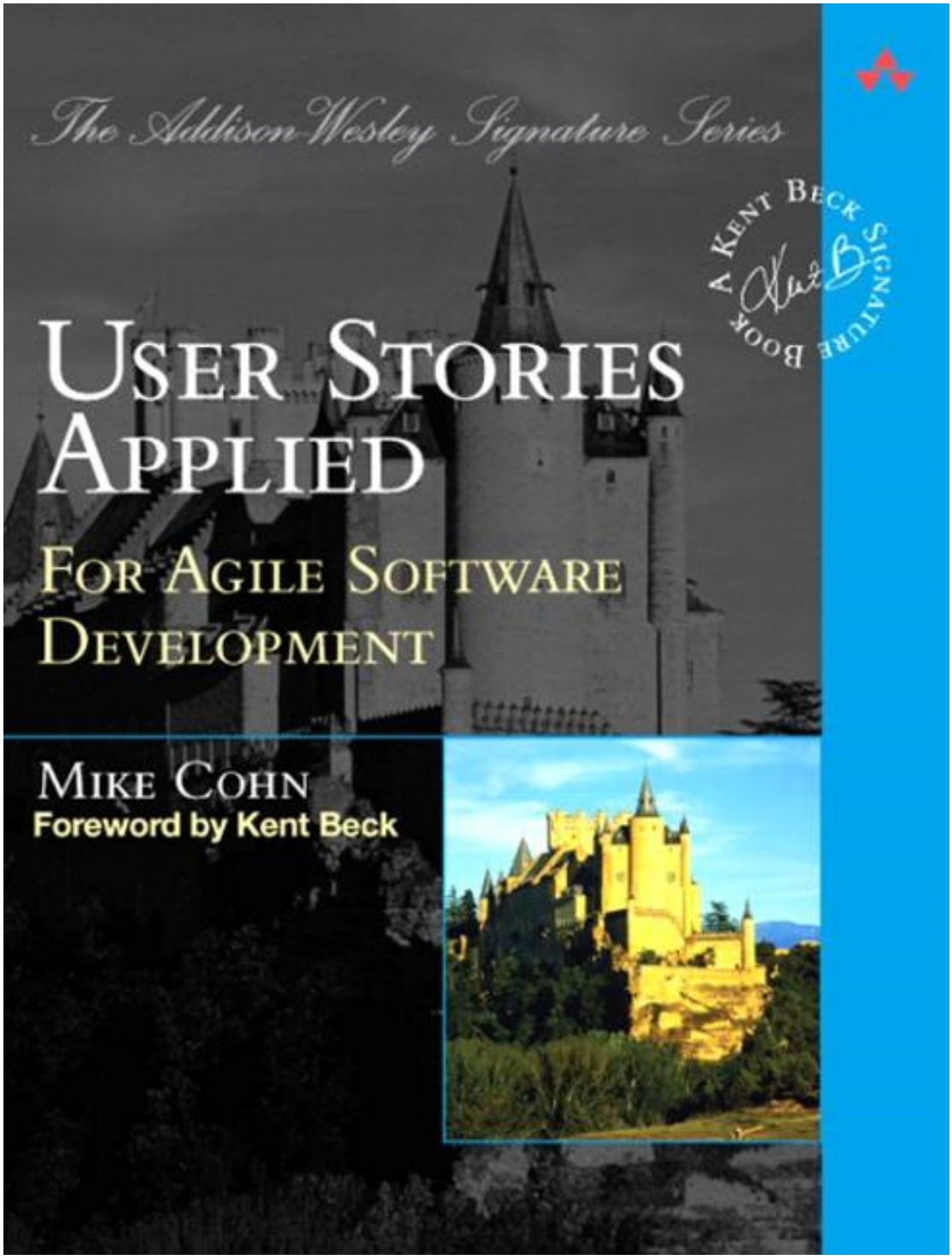

前言

你如何来确定一个软件系统会被设想成什么样子？又如何和众多相关人员沟通这个决定？这个复杂的问题是本书的中心。这个问题很难，因为每个角色都有不同的需求。项目管理者需要跟踪进度；开发者需要完成系统实现；产品管理者需要灵活性[①]。测试人员需要度量。用户需要一个可用的系统。从这些观点出发经过卓有成效的讨论，最后形成大家都可以接受的一致决议并在项目周期内维持这个平衡，这就是所有的难题。

表面看来，Mike Cohn 在本书中提出的方法——用户故事和一直以来试图解决这个问题的需求，用户用例和场景等办法如出一辙。是什么如此复杂？就是你写下自己的需求然后实现它。解决方案的扩展表明问题并不像看起来那么简单。变化影响了你该写些什么和你应该在什么时候写下这些东西。

在为用户故事写下这两方面内容后它就进入流程了，它们分别是：系统必须满足的每个目标和实现这些目标的大概花费。尽管它只有很少的描述，却传递了其他方法所不能展现的信息。根据“最终责任时刻[②]”规则，团队需要推迟写下特性的大部分细节直到特性完成前。

这种简单的时间变化有两个主要作用。第一，在其他特性还比较模糊时，团队可以非常简单的在开发周期中尽早开始实现“最丰满”的特性。指定了每个特性细节的自动测试确保了早期的特性可以像你添加新特性时指定的那样运行。第二，尽早重视特性鼓励从项目开始时就安排故事的优先级，而不是在交付时面对项目失败的景象惊慌失措。

根据 Mike 在用户故事方面的经验，这本书中有很多在开发团队中应用用户故事的实践建议。我希望读者们能清醒、自信地开发。

Kent Beck

Three Rivers Institute

鸣谢

本书从许多书评人的评论中获得了很大的帮助。我要特别感谢 Macro Abis，Dave Astels，Steve Bannerman，Steve Berczuk，Lynn Bain，Dance Brown，Laura Cohn，Ron Crocker，Ward Cunningham，Rachel Davies，Robert Ellsworth,Doris Ford,Hohn Gilman,Sven Gorts,Deb Hartmann,Chris Leslie,Chin Keong Ling,Phlip,Keith Ray,Michele Sliger,Jeff Tatelman,Anko Tijman,Trond Wingard,Jason Yip,和一些匿名书评人。

我还要由衷地感谢我的正式书评人：Ron Jeffries,Tom Poppendieck 和 Bill Wake。Ron 让我保持诚实和敏捷、Tom 的许多想法让我大开眼界。Bill 让我正确前行，还和我分享了他的 INVEST。本书还从一些自由工作者的建议中得到了很多帮助，我很自豪曾经和他们一起工作。

我同样感谢《Testing Extreme Programming》的作者 Lisa Crispin，她在 Addision-Wesley 愉快的经历激励我写这本书，没有她的鼓励，我根本就没办法开始。

我所知道的大部分知识都来自这 9 年来和 Tod Golding 争论。Tod 和我的想法大多时候都是一致的，但是我仍然从我们的争论中学到了一些东西。我很感激 Tod，这些年来他教给我很多东西。这本书的许多部分都因为我和他的交流而丰富。

感谢 Alex Viggio 以及让我产生这本书中许多理念原型的 XP Denver 的各位。也感谢你们，Mark Mosholder 和 J.B.Rainsberger，他们告诉我他们是如何使用软件代替记事卡的。感谢 Kenny Rubin，他和 Adele Goldberg 一同完成了《Succeeding With Objects》一书，他在这本书中彰显的自信让我在数年之后有了再写书的愿望。

衷心感谢 Mark 和 Dance Gutrich，Fast401k 的创立者，他们始终全心全意地支持用户故事和 Scrum。感谢在 Fast401k 的每个合作者，我们将继续朝着成为科罗拉多最优秀团队的目标努力。

我一定要感谢我的家人，因为那段日子网没有和他们在一起。谢谢你们，我最可爱的女儿和公主，Savannah 和 Delancy。我要特别感谢我最棒的漂亮的妻子，Laura，在我没时间时她做了很多。

我还欠了 addison-wesley 一个很大的人情。Paul Petralia 让整个写作过程自始至终都很愉快。Michele Vincenti 推动着整个过程。Lisa Iarkowski 在排版方面给了我很大帮助。Gail Cocker 让我的插图更富观赏价值。Nick Radhuber 最后把它们集中到了一起。

最后，但这远远不是最后，同样感谢 Kent Beck，他微妙的见解，付出，并把这本书纳入他签名的系列。

简介

我在上世纪 90 年代中期经常感到内疚。我为一个公司工作，帮他们每年并购一个新公司。每次并购一家新的公司我都会被任命去运营他们的软件开发部门。每个并购的开发部门都会带来壮观的，出色的、长篇的需求文档。我一定会感到内疚，因为我的团队并没有产出这么出色的需求描述文档。然而，我的团队在软件开发方面一如既往地取得了成功，比被收购的那些更出色的成功。

我明白我们的工作都做了什么。但是我有一个烦恼，那就是如果我们也写了这么巨大，长篇的需求文档我们会更成功。毕竟那是我当时所读的书和文章中所推崇的。如果一个成功的软件开发团队写了壮观的需求文档，那么看起来我们应该也是一样。但是我们没有时间。我们的项目一直都很重要而缺少时间不得不在项目开始时推迟写文档。

由于我们根本没有时间写漂亮的长篇的需求文档，我们发明了一种新的方法，就是和用户直接交流。与其把他们都写下来，自始至终传递，在超期时再协商，还不如直接交流。我们在纸上画一个显示器的例子，有时我们需要原型，我们经常只编写一点代码，然后把他们展示给用户，告诉他们我做了什么。至少每个月我们都会选取一个用户的需求的集合并把我们编码完全呈现给用户。由于和用户保持了近距离接触，并小规模地向用户呈现我们的进展，我们找到了一条不需要漂亮的需求文档的成功之路。

然而，我依然感到内疚，我们没有在我们认定的路上做什么事情。

1999 年 Kent Beck 出版了具有改革意义的书《敏捷编程解析：拥抱变化》。我的内疚顷刻间烟消云散。这时候有人说，比起先写文档再协商再写更多的文档，直接和客户交流是可行的。Kent 阐述了很多问题，也教会了我很多新的工作方法。最重要的是，他证明了我从自己的经验中学到的东西是正确的。

大量的前期需求聚集和文档可以从很多方面扼杀一个项目。其中最普遍的就是当需求文档也变成目标的时候。只有当需求文档对发布软件的真实目标有所帮助时它才应该被写下来。

另外一方面书写文档时不准确的用语也能扼杀项目。我还记得多年前的一个小孩洗澡的故事。小孩的父亲在浴缸里放满了水，帮助孩子走进浴缸。大概两三岁的小孩，脚尖倾在水里迅速地动了一下，然后告诉她父亲：“热一点”。父亲把自己的手放进水里，惊讶地发现，水已经很热了，比平时都热，而不太冷。

在简单思考了孩子的问题后，父亲发现他们的交流出现了误解，相通的语言却表达了不同的意思。小孩说“热一点”在成人理解来就是“升高温度”，而对于一个孩子来说，“热一点”的意思就是“和我认为暖和的温度接近一点”。

词语，特别是写下来的词语，用来表达像软件这么复杂的需求时是很贫乏的。由于这些词语很容易被理解为不同的含义，所以需要开发人员，客户和用户一起来讨论来代替它们。用户故事给我们提供了一种方法，让我们只把不能忽略的东西和在这次交流中能估计和计划的东西记录下来。

在读完第一部分后，你就要准备好改掉为每个正在进行的需求细节严格书写文档的习惯。在读完这本书后你就会知道在自己的项目中实施故事驱动开发必须的每件事。本书分为 4 部分和 2 个附录：

第一部分：启程——描述了要马上开始书写故事你必须要知道的所有事。用户故事的一个目标就是让大家讨论而不是写文档。第一部分的目标就是让大家尽可能地讨论。第一章是

概述了什么是用户故事和如何使用这些故事。接下来的一章介绍了一些书写故事的细节，如从用户角色模型中收集故事；在不能和最终用户交流的情况下书写故事；测试用户故事等。最后一章提出了一些准则，它们可以帮助你改进故事。

第二部分：估计和计划——包含了一系列用户故事。通常我们首先需要回答一个问题“用户故事需要开发多长时间”？第二部分的章节包括如何用故事点估计故事，如何计划一个持续 3 到 6 个月的版本发布，如何根据更多细节确定迭代计划，最后如何度量进度并评估进度是否和预期的相同。

第三部分：经常讨论关键问题。第三部分开始描述了故事和用例，软件需求说明书和交互设计场景说明书等其他需求分析方法的区别。接下来的章节描述了用户故事独有的一些优势，出现问题时如何沟通，如何在 Scrum 敏捷过程中使用用户故事。最后一章介绍了一些小问题诸如要把故事写在纸质卡片上还是在软件系统中和如何识别非功能需求。

第四部分：例子——希望这个泛泛的例子可以把之前介绍的都串起来。如果我们要求开发人员通过故事完全理解用户需求，那么在一个例子中用一个延伸的故事展示用户故事的各个方面来结束本书是很重要的。

第五部分：附录——用户故事是极限编程的产物，但你不需要为了阅读本书而去学习极限编程。附录 A 中有对极限编程的主要介绍。附录 B 中是每章结束时的习题答案。

第一部分 启程

在第一部分，我们将从什么是用户故事和如何应用它们说起。接着我们将看到更多的细节：关于如何书写用户故事、如何使用系统的用户模型帮助识别故事、在不能直接访问用户时如何和充当用户角色的人协同工作，如何编写用例来验证故事已经被成功开发完成了。最后我们将以一些良好的故事所共有的标准来结束第一部分。

通过这个部分学习，足以让你开始识别、书写和测试你自己的故事。你也要准备好学习如何用故事估计和计划，这将是第二部分的主要内容。

第一章　概述

软件需求是一个沟通问题。需要新软件的人（用户或销售商）必须和开发新软件的人沟通。要想项目成功，需要依赖来自不同领域决策者信息：一方面是客户、用户，也可以是分析师领域专家和项目发起人，另一方面是技术团队。

一旦任意一方支配了这种沟通，这个项目就会失败。如果业务团队占据了优势，交流过程就会聚焦于功能和交付日期，而忽视开发者会同时面对这两个问题，和开发者能否完全理解需求。如果开发者占据了主动，技术术语将代替业务语言，同时开发者也失去了倾听和学习需求的机会。

我们需要的是一种能让大家协同工作的方法，因此任意一方支配沟通、感情用事和资源分配的行政问题都会变成共性的问题。当资源分配向某一方完全倾斜时项目就会失败。如果开发人员承担了这些问题(通常是被告知“我不管你如何实现它但是必须在 6 月全部做完”)，他们很可能会用质量来交换那些附加的特性的进度，可能只会部分实现一个特性，也可能独自做出一些本该由客户和用户一同参与的决定。如果客户和用户承担了这部分重负，在项目开始前我们通常会看见一系列繁琐的讨论，直到特性一步步被从项目中裁剪掉。然后，当软件最终发布时，只剩下了很少的一部分功能，远比裁剪掉的少。

至此我们已经知道不能很完美的预测一个软件开发项目。当用户看到早期的软件版本时，他们总会有新的想法或改变以前的想法。由于软件的不确定性，大多数开发人员在估计故事持续的时间时都会面临困难。由于这些和那些因素，我们不能将项目中所有要做的事情都呈现在一张完美的 PERT 图[①]上。

那我们该怎么做呢？

我们只依靠我们手边已有的信息做出决定。并且我们经常这样做。与其在项目开始时做好所有的决定还不如将决定的过程分散在整个项目周期。要做到这样我们必须能频繁地尽早地获取信息。这就是用户故事的来源。

什么是用户故事？

用户故事描述了对系统或软件的每个用户或购买者都有价值的功能。用户故事包括三方面内容：

- 书写故事的描述，用来做计划以及提醒
- 讨论故事来丰富故事的细节
- 测试描述和记载的细节，用来确定故事何时完成

由于用户故事的描述都以传统地手写在故事卡上，Ron Jeffries 巧妙地将这三点命名为 3 个相同字母开始单词：卡片（Card）、沟通（Conversation）和确认（Confirmation）。卡片可能是用户故事最明显的表现，但这不是最重要的。Rachel Davies 曾说卡片“描述的用户需求好于文档”。这是理解用户故事的完美方法：卡片包含了故事的文字描述，沟通中得出的细节和确认时的记录。

如卡片 1.1 所示的用户故事的例子，它来自于我们假定的一个名为 BigMoneyJobs 的登记和搜索职位的网站。

Story Card 1.1. An initial user story written on a note card.

> A user can post her resume to the website.

为了保持一致性，本书中后续的例子也都来自 BigMoneyJobs。其他的例子可能包括：

- 用户可以搜索职位。
- 企业可以注册新的工作职位。
- 用户可以限定简历的浏览者。

由于用户故事描述了对用户有价值的功能，因此对于这个系统，下述的例子不是好的用户故事：

- 软件将用 C++来编写。
- 程序将使用连接池来连接数据库。

第一例子之所以不是好的用户故事是因为用户不会关心使用什么样的编程语言。但是，如果这是一个 API，系统用户（他自己也是开发者）最好将“软件要用 C++编写”写下来。

第二个例子不是好的用户故事的原因在于系统的用户不会关心像软件如何连接数据库这样的技术细节。

可能你看完这些故事后会惊讶道“请等一下——使用连接池在我的系统是一个需求”。如果是这样，请想一下，关键在于故事之所以是故事是因为他们体现了对于用户的价值。还有很多方法可以像这样描述故事，说出他们对用户价值。我们将在第二章“书写故事”中看到这样的例子。

细节从哪来？

一方面是要说明“用户可以搜索职位”，另一方面要能够像说明书一样单独引导编码和测试。细节是从哪里来的呢？下面这些没有答案的问题怎么样？

- 用户会按照什么来搜索？州？城市？职位简述？关键字？
- 用户必须成为网站的会员吗？
- 搜索参数可以保存吗？
- 找到职位后要显示什么样的信息？

很多这样的细节都可以作为新增的故事。其实多个合适粒度的故事要好于很庞大的故事。例如，整个 BigMoneyJobs 网站可以只用两个故事来描述：

- 用户可以搜索职位。
- 企业可以注册空缺职位。

很明显这两个故事太庞大包含了太多用途。第二章“书写故事”将完全针对故事的粒度问题，但是作为开始时的故事最好能由一个或一对开发人员用半天到两星期的时间编写和测试完成。公平地说，以上的两个故事可以简单的覆盖 BigMoneyJobs 的主要功能所以每个故事都需要大多数开发者一同工作一星期以上。

如果一个故事太庞大就会像史诗一样。这样的史诗可以被划分为 2 个或多个更小的故事。例如，故事“用户可以搜索职位”可以被划分为如下的故事：

- 用户可以按照地址，薪酬范围，职位，企业名称和职位的发布时间。
- 用户可以查看匹配到的职位的信息。
- 用户可以查看发布职位的企业详细信息。

但是，当一个故事包含了所有最终的细节时它就不能再分解了。例如，故事“用户可以查看匹配到的职位的信息”是一个合理的现实的故事。我们没有必要再把它划分成如下的样

子：

- 用户可以查看职位描述。
- 用户可以查看薪酬范围。
- 用户可以查看工作地点。

同样的，用户故事不需要再按照传统需求文档的形式扩展，像下面这样：

4.6 用户可以查看匹配到的职位信息

4.6.1 用户可以查看职位

4.6.2 用户可以查看薪酬范围

4.6.3 用户可以查看工作地点

比起将这些细节写成故事，更好的方法是开发团队和客户一起讨论这些细节。当这些细节变得重要时就需要讨论这些细节。在讨论后也可以在故事卡上做些批注，如故事卡片 1.2。但是，关键是沟通，而不是在故事卡上的注释。无论是开发者还是客户都不能在三个月后再指着故事卡说“瞧，这里我说的很对吧”。故事不是契约式的责任。就像我们看到底，讨论的结果要以测试用例的形式记录下来，以便证明故事的开发过程一直正确进行。

Story Card 1.2. A story card with a note.

> Users can view information about each job that is matched by a search.
>
> Marco says show description, salary, and location.

“它需要多长时间？”

在高中文化课上，只要老师留了作业，我总会问“需要多长时间完成”。老师一直都不喜欢这个问题但是它是一个很好的问题因为它可以告诉我老师的期望。理解项目用户的期望是很重要的。这些期望最好通过验收测试用例的形式记录下来。

如果你使用纸质故事卡，你可以翻过卡片把这些期望都记录下来。写下来的这些期望将像备忘录，指导如何来测试故事，如故事卡 1.3 所示。如果你使用了电子系统，它可能已经支持记录验收测试用例。

Story Card 1.3. The back of a story card holds reminders about how to test the story.

> Try it with an empty job description.
> Try it with a really long job description.
> Try it with a missing salary.
> Try it with a six-digit salary.

测试细节是简单而概括的。可以在任何时候被增加或移除测试。测试的目的就是表达故事中附加的信息，因此开发者会了解他们什么时候能完成这些。老师的期望对于我了解何时要写完 Moby Dick 论文很有用，同样的，客户的期望对于开发人员了解他们应该何时完成故

事也是很有用的。

客户团队

一个理想的项目中应该有人专职为开发者的工作制订优先级，解答软件最终用户的所有疑问，书写所有故事。我们一直对这些期望了太多，所以我们组建了客户团队。客户团队中包括了能确定软件的用户需求的人。这意味着客户团队需要测试人员，产品主管，真正用户和交互设计师。

用户故事的流程是什么样子？

应用用户故事的项目拥有和以往不同的感觉和节奏。使用传统的瀑布流程定义了一个书写全部需求文档、分析需求、设计解决方案、实现解决方案然后测试方案的周期。在这个过程中，客户和用户在项目开始时书写需求文档在结束时验收软件，但是在需求设计和验收之间的活动中用户和客户几乎消失了。现在，我们已经知道这样是不能工作的。

在故事驱动的项目中你应该了解的第一件事是客户和用户要全程参与。他们不希望错过项目当中的任何一个环节。无论团队是否实践 XP（见附录 A），任何形式的敏捷统一过程这都是真的，如 Scrum，home-grown，故事驱动的敏捷过程。

客户和新软件的最终用户要准备好在书写用户故事时起到积极的作用，特别是使用 XP 的时候。故事书写过程最好由由系统的最终用户开始。例如，你要搭建一个旅行预订网站，你可能的用户有频繁的传单，假期计划人等等。客户段对应该包括尽可能多的实际用户代表。但是如果这不能实现，那用户角色模型可能完成这个任务。

为什么由用户来编写故事？

相对于开发者，客户团队来编写用户故事有两个主要原因。第一，每个故事必须用业务语言编写，而不是技术术语，这样客户团队才能确定迭代或版本发布中故事的优先级。第二，作为产品功能最主要的设想者，客户团队最适合描述产品的行为。

尽管故事可以在项目中的任何时候编写，但项目的第一个故事通常故事讨论会议中编写。在故事编写讨论会议中，大家头脑风暴，写出尽可能多的故事。准备好了开始的故事集，开发人员就可以估计每个故事的大小了。

同时，客户团队和开发者可以选择一个合适的迭代周期，大概一至四周。在项目过程中每个迭代的周期都相同，在每轮迭代结束的时候开发者负责交付部分应用的全部可用代码。在迭代过程中，客户团队依然要积极参与，和开发者讨论这个迭代要完成的故事。在迭代中客户团队还要编写测试用例，然后和开发者一同完成自动化测试。另外，客户团队要确定项目一直按照产品设计交付。

一旦迭代的周期确定，开发人员就可以估计每个迭代可以完成多少工作了。我们称之为速率。团队第一次估计的速率往往是不正确的，因为没有办法知道将来的速率。但是我们可以用第一次估计的速率做一个粗略草图或发布计划，需要多少迭代，来表明每个迭代需要做多少工作和整个项目需要多少迭代。

如果是做版本发布计划，我们要将故事排成若干堆，每个迭代完成其中一堆。每堆故事总的估计值不能超过估计的速率。最高优先级的故事应该被排到第一堆。当这堆故事已经足够多的时候，剩下的优先级最高的故事就会被排到第二堆（在第二轮迭代完成）。这样的循环一直持续到完成了几乎全部在项目周期中可以完成的故事或目前完成的故事足以发布版本（更多的内容将才第九章和第十章中讲述）。

在每个迭代开始之前客户团队可以为计划做中途校正。在迭代完成时，我们就可以知道开发团队真正的速率，用真正的速率代替之前的估计值。这就意味着每堆故事都需要适当增删来调整。同时，有些故事完成的速率会比预期的快，这就意味着团队需要在本轮迭代增加一些额外的故事。也有时候有些故事会比预期的艰难，这就需要移除一部分故事到下一个迭代或者完全从发布版本中移除。

发布计划和迭代计划

一次发布由一个或多个迭代组成。发布计划是指在项目时间表和需求功能集之间确定一种平衡。迭代计划是指选择在一轮迭代中要完成的故事。客户团队和开发者共同参与发布计划和迭代计划的制定。

要制订发布计划，客户团队要从确定故事优先级开始。在确定优先级时他们需要考虑：

- 特性设计是针对普通用户或客户的
- 特性设计是针对少数重要用户或客户的
- 故事之间的相关性。例如，一个“缩小”的故事本身的优先级可能并不高，但是它和高优先级的“放大”故事是互补的，我们必须像对待高优先级的故事一样对待它。

开发者对许多故事有不同的优先级。他们会建议根据技术困难或故事的相关性修改故事的优先级。客户团队会考虑他们的意见，但会从最大化集体价值出发制定优先级。

故事的优先级不能在没有估计成本时就确定。去年夏天，我确定了花费后才决定去塔希提岛度假。在这方面其他地方话费提升了它的优先级。确定优先级的主要因素就是每个故事的成本。故事的成本由开发者给出，每个故事用故事点估计。因此可以认为，一个 4 个故事点长度的故事的成本是一个长为 2 个故事点故事的 2 倍。

把故事分发到版本发布涵盖的迭代中去就形成了发布计划。开发者说明他们期望的速率，也就是他们认为可以在每个迭代中完成的故事点数，客户将故事分配到迭代中，确保分配到每个迭代中的故事点不会超过团队期望的速率。

例如，假设表 1.1 列出了你的项目中的所有故事，他们按照优先级降序排列。团队估计的速率是每轮迭代 13 个故事点。故事将如表 1.2 所示被分配到迭代中。

Table 1.1. Sample stories and their costs.

Story	Story Points
Story A	3
Story B	5
Story C	5
Story D	3
Story E	1
Story F	8
Story G	5
Story H	5
Story I	5
Story J	2

团队期望的速率是 13，因此每个故事都不能计划超过 13 个故事点。这就意味着第二轮和第三轮迭代都只能计划 12 个故事点。别担心，这些并不是很准确的估计并不是很重要。如果开发者的速率超过了预计的速率，他们可以要求增加其他一两个故事。必须注意的是，在第三轮迭代时客户团队选择了故事 J 代替具有更高优先级的故事 I。这是因为故事 I 的长度为 5 个故事点，对于第三轮迭代来说它太大了。

Table 1.2. A release plan for the stories of Table 1.1.

Iteration	Stories	Story Points
Iteration 1	A, B, C	13
Iteration 2	D, E, F	12
Iteration 3	G, H, J	12
Iteration 4	I	5

除了临时跳过一个大故事而把它替换成一个小故事，还有另外一个办法就是把大故事划分成 2 个小故事。假设 5 个故事点长的故事 I 可以被划分成故事 Y（3 故事点）和故事 Z（2 故事点）。故事 Y 包含了原来故事 I 中的最重要的部分，现在它可以在第三轮迭代完成，如表 1.3 所示。关于何时、如何划分故事将在第二章和第七章介绍。

Table 1.3. Splitting a story to create a better release plan.

Iteration	Stories	Story Points
Iteration 1	A, B, C	13
Iteration 2	D, E, F	12
Iteration 3	G, H, Y	13
Iteration 4	J, Z	4

什么是验收测试？

验收用例是用来验证开发完的故事能否完全按照客户团队期望的那样运行的过程。一旦一轮迭代开始，开发者开始编码，客户团队也开始编写测试用例。这还依赖于客户团队对技术的熟练程度，也就是说把所有写到故事卡上的用例输入到自动测试工具中。客户团队中应该有一个积极并且有能力的测试人员，因为他们对于这些任务他们更有技术。

在迭代中测试用例应尽早完成（如果事先可以估计到迭代中所要做的事情甚至可以在迭代开始前就编写测试用例）。尽早完成测试用例非常有帮助，这可以使客户团队的很多假设和期望尽早传递给开发者。例如，假设你写了这样一个故事“用户可以用信用卡为他购物车里的商品结账”。然后你在故事卡的背面写了这些简单的测试用例：

- 使用 VISA，万事达信用卡和美国运通卡测试（成功）；
- 使用速食店会员卡测试（失败）；
- 使用 Visa 借记卡测试（成功）；
- 使用正常的卡，损坏的卡或背面没有编号的卡测试；
- 使用过期的卡测试；

- 使用不同支付额账户测试（包括超过卡的限额）；

这些测试能验证用户希望系统只识别 Visa，万事达信用卡和美国运通卡，而不允许使用其他卡支付的需求。在将这些测试交给开发者之前，客户团队不但要表明自己的期望，还可能要提醒开发者那些他们容易忘记的情况。例如，开发者可能忘记考虑过期的卡。在他开始编码前将这个需求以测试用例的形式写在故事 card 上，这会帮助他节约时间。更多的关于编写故事验收用例的描述将在第六章呈现。

为什么会有变化？

这个时候你可能会问为什么改变？为什么要写故事卡？为什么需要所有的这些沟通？为什么不直接编写需求文档或使用用例？用户故事比起其他方法来有很多优势。很多的细节将在第十三章中叙述，但是其中的一部分原因如下：

- 用户故事更强调口头交流而不是文字交流；
- 用户故事是你和开发者都可以理解的；
- 用户故事的粒度更适合做计划；
- 用户故事为迭代开发服务；
- 用户故事鼓励推迟细节设计直到你对你真正需要的东西有了充分的理解。

因为用户故事注重交流而不是文档，重要的决定不再在那些很可能根本不会被阅读的文档中体现。故事中重要的情节都由验收测试来体现，这些用例会被频繁运行。而且我们拒绝在文档中愚蠢地写下这样的话：

系统必须保存一个地址和业务电话或移动电话。

这句话是什么意思？它说明系统必须保存如下一种：

（地址和业务电话）或移动电话；

地址和（业务电话或移动电话）；

因为用户故事并不需要技术术语（记住，用户故事是由客户团队来编写的），它们需要开发者和客户团队都理解。

每个用户故事描述一个独立的功能，也就是说，在某一特定环境下用户想做的一件事。这样用户故事就可以当作一个计划工具来使用。你可以更好的估计在发布版本之间切换故事的价值，而不是估计取消一个或多个“系统应该……”描述的影响。

一个迭代过程就是一个持续改进的进步过程。一个开发团队在开始接触一个系统时会知道一些或很多还不完善或者有缺陷的地方。然后他们持续改进这些方面直到产品满足要求。每个迭代中软件都会因为有更多细节加入而更完善。因为迭代开发很可能遍历故事，所以故事在迭代中会表现的很好。对于你确实需要但现在又不重要的特性，你可以先写一个粗略的故事（史诗）。当你准备把这个故事纳入系统的时候你可以拆分这个粗略的故事，用一些更简单更容易实现的故事来代替它。

遍历一个故事集的特性允许故事推迟细节设计。因为我们目前可以只写一个粗略的故事来标记这个特性，在开发这个特性前没有必要为系统的每个部分都写好故事。延迟细节设计很重要，因为允许我们在确定我们需要某个特性之前不花费时间思考它。故事不鼓励提前假装把所有的事都做好。它鼓励软件在客户和开发者的讨论中持续改进。

总结

故事卡中应简述对用户或用户有价值的功能。

故事卡是故事的可视部分，而更重要的部分在于客户和开发者关于故事的讨论。

客户团队应包括能确定软件最终用户需求的人。可能包括测试者，产品管理者，真实用户和交互设计师。

客户团队要编写故事，因为他们处于描述需求的最佳位置，也因为随后他们需要和开发者一同设计出故事细节并确定故事优先级。

按照故事对客户的价值确定客户的优先级。

通过把故事安排带到迭代中来制订迭代和发布计划。

速率是开发者在一个迭代中能完成的工作。

处于一个迭代中估计的故事的估计值总和不能超过开发者预计的本轮迭代的速率。

如果某个故事不适合放在当前迭代中，你可以把它拆分为两个或两个以上更小的故事。

验收测试用来验证开发者实现的这个故事的功能是否和客户团队在编写故事时的设想一致。

用户故事是值得实践的，原因在于它强调口头交流，它能够让你和你的开发者有相同的理解，它能用来做迭代计划，在迭代开发过程中表现良好，它鼓励延迟细节设计。

问题

1.1 用户故事的三个部分分别是什么？

1.2 客户团队中应该包括哪些人？

1.3 下面哪些是精彩的故事？为什么？

a. 用户可以在 Windows XP 和 Linux 系统中运行程序。
b. 将使用第三方程序来处理所有图表。
c. 用户可以撤销执行过的 50 条命令。
d. 软件将在 7 月 30 日发布。
e. 软件将用 JAVA 实现。
f. 用户可以从一个下拉列表中选择他的国家。
g. 系统将使用 Log4J 将所有的错误记录到一个文件中。
h. 如果用户 15 分钟后没有保存她的工作系统将给出提示。
i. 用户可以选择“导出成 XML”功能。
j. 用户可以将数据导出成 XML 格式。

1.4 需求沟通比较需求文档有什么优势？

1.5 为什么要在故事卡的后面写下测试用例？