第 25 章

如何充分估算和确定优先顺序的案例

Estimating and Prioritizing Stories

一个敏捷开发的团队必须从商业价值的角度来定义所开发的软件的功能，然后计划好如何实现这些功能，以便毫不拖延地实现这些价值的目标。在本章中，我们将介绍如何通过案例卡片帮助开发团队界定这些功能，并且说明如何对相关工作的实施进行恰当的估算，并且确定其优先顺序。本章为第 26 章进行了重要的铺垫，第 26 章将介绍计划敏捷项目的方法。

25.1 与客户一起工作的案例
Working with Customer Stories

客户案例提供了一种方法来确定一个软件产品中的各个分散的具有商业价值的功能。因此敏捷开发团队可以通过使用案例来确定和计划开发工作，并且在实现软件的商业价值过程中衡量其进度。通常你会希望安排一个两人小组的开发人员来执行这种含有少量编程片断的案例。

> 注：虽然客户案例这个概念源自于极限编程（XP）[1]，但是这个词也经常用于其他类型的敏捷开发项目。相比之下，我们更喜欢用团队客户案例这个术语（或合适案例），不喜欢用客户案例这个术语，因为这样可以避免与用例这个术语产生任何可能的混淆[2]。

[1] Beck, Kent. Extreme Programming Explained, First Edition (Addison-Wesley, 2000).

[2] Jacobson, Ivar. “Object-oriented development in an industrial environment” (OOPSLA, 1987).

概述
Overview

我们最先在第 3 章介绍过客户案例这个词，当我们解释这个词的时候，讲过这个术语有 3 个基本的部分：卡片、交谈并确认[3]。

图 25-1 所示为一个案例的卡片部分。卡片正面是这个案例交谈的摘要，卡片背面是这个案例的确认部分的描述和一个用户测试。案例产生于开发人员与客户之间的简短交谈，这种交谈是在项目的生命周期内作为案例实践的结论出现的。

因此，关于这个功能的大多数信息并没有被记录下来，所以请注意，你不能仅仅孤立地看卡片上的内容。例如，图 25-1 中的案例概括了 OSPACS 团队中的两个成员：Luke（开发人员）和 Sally（客户）之间的一次交谈。乍一看，这个案例实现起来非常简单，但是如果你能够与 Luke 交谈的话，就会很快发现，这实际上代表了大量的工作，因为它涉及要访问一个外部数据库，还须要对多个字段的内容进行匹配，而且这些不同格式的数据的添加方法也不一致（见图 24-4）。

> 注：小尺寸的卡片意味着只能捕捉到开发人员与客户之间讨论的很少的一小部分结果。具体的细节包含在其后由客户写的测试内容中（见第 7 节）。

512

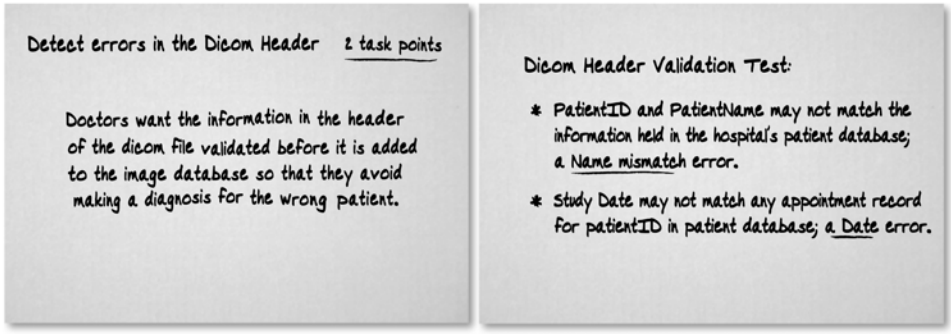


图 25-1 来自 OSPACS 项目的案例卡片的正面和背面

实践案例

案例实践要求开发团队使用卡片来计划一个项目，卡片用于对工作进行总结、评估，

[3] Jeffries, Ron. “Essential XP: Card, Conversation, Confirmation” (www.xprogramming.com).

以及确定优先顺序。

开发人员根据与客户进行的关于业务上需要软件做些什么的简短讨论生成一个案例卡片。开发人员将讨论的内容总结到卡片上的报告表格中。如“一个<用户类型>希望<能力>实现<商业价值>”[4]。他还可以在卡片上写上案例的名称，这里须要慎重选择一个简短而有足够意义的名称，使项目团队能够在以后的讨论中使用这个名字。

在卡片写完后，项目团队马上就要估算实现这个案例有哪些必要的工作。这就可以给客户留下实现此案例需要多少成本的概念，这就有助于决定这些工作的优先顺序，指出它们在堆积如山的等待执行的案例中的位置。在每个迭代开始的时候，开发团队很简单地就可以制定工作计划，他们只须从最上层的案例卡片拿起，直到拿出的卡片累积到与上一次迭代完成的厚度处于同一高度即可。这样的话，项目就要根据开发团队的实际进度和当前业务的优先顺序适当调整交付目标，然后定期重新计划。

513

敏捷开发团队专注的是有价值的功能的交付，而不是完成任务，这意味着开发团队可以采用更简单更有效率的方式进行计划。因此团队成员并不须要在项目开始的时候就确定所有的需求，这样在项目完成之前，对业务环境出现的任何变化，都能更灵活地响应。这就使得敏捷开发团队相对于传统开发团队有一个显著的优势，即传统开发团队须要详细地制定计划进度，并且依赖于正式的文档，通常这样的限制会使传统开发团队在项目的开始时要制定更复杂的静态计划。从以往交付软件须要符合实际业务须求的经验来看,常规计划往往不如敏捷计划有效。

> 注：迭代总是持续进行一个固定的时期。在某些项目中，会持续 1 周，但是在其他项目中可能会持续 2 周甚至 3 周。

生成案例
Generating Stories

当客户和开发人员最终坐下来讨论一个案例的时候，可能只需要几分钟时间将案例写在卡片上，并且给这个案例确定一个合适的名字。由于开发人员可能最终不会落实这个案例，因此在此阶段不须要进行非常具体的工作。重要的是只须要考虑那些与估算项目规模有关的细节。例如，业务规则的数量和复杂性、用户界面的性质、所需存储的数据，等等。

[4] [USA] Cohn, Mike. User Stories Applied (Addison-Wesley, 2004).

开发人员可以在案例卡片的背面记录这些细节的摘要，也可以记录到自己的记事本上，他必须从中提炼概括的条目，因为正如我们先前所说的，以后再获取案例的全部细节，即在
514 客户测试的时候来写这些内容。

通过实践，人们在如何产生客户案例方面逐渐精通，所以你应该看看这个例子中提供的 Mike Cohn 的书——《User Stories Applied》[5]，并且去试着照搬相同的做法。然而，由 Bill Wake 创始[6]的 INVEST（缩写）方法提供了一个有用的记住一个好的客户案例的属性，并可能帮你避免一些常见错误：

- 独立性——一个案例不必依赖于其他的执行案例，因此你可以优先考虑它们的业务原因，而不是技术原因。
- 可协商——不论是客户还是开发人员都必须站在自己的立场上记录案例的条款。
- 有价值——如果一个功能不能提供任何直接的价值或给业务带来利润，那么我们凭什么希望软件去执行这个功能呢？
- 可估算——将大的案例（长篇故事）分解成较小的案例，并且调查那些未知的问题，直到你可以放心地进行估算，这个时候，就可以落实这些案例了。
- 小型的——分解或重做这些案例，直到一个开发人员能够在几天内完成。但是也不要让这些案例太小，以至于开发人员只需要几分钟就完成了代码，而不是几个小时完成。
- 可测试——你必须对案例能够进行测试，以确保有一个都认可的案例结束的评判标准。这也防止你为非功能性需求生成一些案例，如易用性、可靠性，等等。

提示：不要尝试写一个程序试图以代码方式来管理客户案例，因为这将失去与其他人
515 讨论时用物理卡片处理记录的重要价值。

25.2 充分估算
Estimating

如果案例的估算由曾经做过的有经验的人员进行，结果就更为可靠，因此，应由开发

[5] [USA] Cohn, Mike. User Stories Applied (Addison-Wesley, 2004).
[6] Wake, Bill. “INVEST in Good Stories, and SMART Tasks” (http://xp123.com/xplor/xp0308/index.shtml).

人员进行案例估算。这有 3 种通用的估算案例大小的方法。

- 直觉——花几分钟思考一下这个案例之后，你就会产生一个基于你曾做过的相似工作的估算。如果你具备相关的技能并熟知开发团队的情况，你的直觉可能会令人惊讶地准确。
- 类比——你通过对比开发团队已经完成的类似的工作来生成对案例大小的相关估算。当你能够将工作划分得比任务“A”多一点，比任务“B”少一点的时候，就是最佳估算了。
- 分解工作——将一个大的任务分解为一系列的子任务，你会发现这比直觉或类比的估算更容易些，然后汇总这些子任务的所有的估算值。有一个好办法可以处理那些大大超过其余任务的工作，但是当你将工作分得太细就会变得不准确。

不管你使用什么方法，也不管你付出多少努力，要估算得正确是一个老大难的问题。事实上，往往你的最初十分钟的直觉，并不比项目团队花费数小时分解工作，并且思考与其他工作进行恰当的类比之后，最终生成的估算要差。

度量案例大小
Sizing Stories

案例描述了业务需要哪些功能，但是开发人员如何实现这些功能并没有描述。因此，当估算一个案例的大小的时候，首先要考虑与其他开发人员之间的替代方法，使你可以完成工作。开发人员谁先写案例卡片，谁就必须带领大家讨论如何描述客户设想的各种各样 516
的测试，因为这样做通常可以剖析出大量的这个案例中的本质信息。

在讨论五到十分钟后，小组必须试着就任务的规模大小问题达成一致意见，这样将会引导项目成为最有希望的解决方案。虽然你最终会把大的案例（长篇故事）分割为更小的案例，使得你能够在一个迭代中就独立实现这个案例，但是又不总是须要在最初的时候分割这样的长篇案例。更为重要的是，要在一个较高的水平上持续讨论这个案例的估算，并且在考虑任务的大小方面，集中精力进行具有重要意义的选择。你不得不试图找出任务的各个部分的信息，单独测量它们的规模大小，然后生成一个估算值作为所有部分的汇总。可能站在咖啡机旁边的时候，你就可以仅花费五到十分钟，通过与你的同事的讨论，来获得对任务大小的感觉。详细的任务计划要来得晚得多，在实施之前才会有，详细内容请看第 26 章。

提示：当测算案例大小的时候，可借这个机会来确定你是否已准备好应对可能造成一个迭代中无法完成这个案例的问题。这样一来，在这个工作实施之前，开发团队就可以确保这个工作的完成。请看第 26 章内标题为“任务计划”的小节。

绝对值还是相对值的估算
Absolute Values versus Relative Values for Estimation

一种常见的方法是规定任务的大小，给出一个数字作为完成任务可能需要的时间，但是这样会产生很多问题。一个有 10 名开发人员的开发团队，一个项目中确定的各项任务的组成将需要花费 360 个人-周的有价值工作。因此，他们预测将能够在 36 周之内完成这个项目：36=360/10。4 个星期之后，开发团队的进度落后于计划：在这个时候开发人员只完成了 20 个人-周的有价值工作，而不是他们原先预计能够完成的 40 个。因此，他们必须在项目计划中修改人-周的估算，以纠正他们缓慢的项目进度，并且重新计算他们的交
517 货时间：72=（360×2）/10。

将绝对时间加入到项目计划中意味着无论开发团队的进度状态发生了什么样的变化，你都必须更新每个估算。为此，不但要耗费大量的时间变更所有的估算，而且在几个月后，人们开始对数字失去信心，因为你总是在修改。尽管同样是标志正确的进展速度，但是采用绝对时间是一个比较差的初始化估算值的选择。

让我们来考虑一下我们如何能够改进一下用相对量代替绝对值来表示进度情况。

一个有 10 名开发人员的项目团队在一个完整项目中的开发速度确定在 0～9 的范围内，项目总共有 2880 个任务点。他们希望每周能完成 80 个任务点，并且预计项目在 36 周内完成：36=2880/80。不幸的是，一个月以后，他们只完成了 40 个任务点的价值。但是，他们不必改变任何项目计划的估算，因为它们的相对大小仍然是正确的。相反，他们只是简单地修改了功能点的数量，他们希望每周交付的功能点为 40 个，并且重新计算了交付日期：72=2880/40。

用相对值表示估算值比每天不断更新项目计划容易得多，因为单独的数据不会经常变更。事实上，只有在你须要调整一个单独任务的估算时，相对大小的错误在计划中才变得很明显。例如，如果你估算创建所有的对话框需要一个任务点，但是后来你发现这个任务更现实的大小是 2 个任务点，这就须要调整所有其余的估算来完成对话框的创建任务。真正重要的是必须要保持协调，使得每个任务的大小是按比例与所有的其他估算相对应的。

> 注：项目计划会由于开发团队的进展速度每个月都超出估算而不断调整，改变绝对估算所需的工作使得你的项目计划在这方面的调整变得很困难。但是使用相对估算的话，你可以频繁地调整计划，当进行动态计划的时候，这是一个重要的考虑因素。

518

相对估算尺度
Relative Estimate Scales

大多数的人发现比较事情的大小比产生一个绝对数据的大小容易。例如，估计你的鞋比你的朋友的鞋大一些似乎比估计你的鞋是 10.5 英寸要更可靠些。不过重要的是进行类似比较的时候，要使用合适的尺度。比如，说明搜集到的鞋子的尺寸的时候，采用 1 到 100 尺度范围的尺寸，如 72，81，82，69，实际上就不如采用 1 到 10 范围内的尺寸，如 7，8，8，7，准确。Mike Cohn[7]的报告说明了相当多的成功使用非线性尺度的估算案例：

0，1，2，3，5，8，13，20，40，100

这个尺度范围包含了 0，对于任务来说作为计划的目的，会认为这个数据太小，但是无论如何都需要完成。给出这样的任务作为明确的记分，当用来计算在近一个时期已经交付了多少个点的时候，会对进度产生错误的印象。但是在它们集体变得有意义之前，你可以在整个任务期间预期只完成这么多的零点任务。在 Mike 的尺度范围中，大于 8 的数据对于案例来说是有问题的，大大超出了正常，对于大多数案例而言，你的估算会是一个相对尺寸，1，2，3，5 或 8。使用非线性系列，迫使你更精确地比较尺度，比如它们要大一点，因此一定要大于 3，但是一定不能大于 8，就变成了 5。理想情况下，你的对比可能会降低到 1，2 或 3 个任务点——一个足够小的区间范围，这对于所有开发人员来说在整个项目开发过程中都能够一致接受。

任务点和案例价值估算
Task Points and Story Cost Estimation

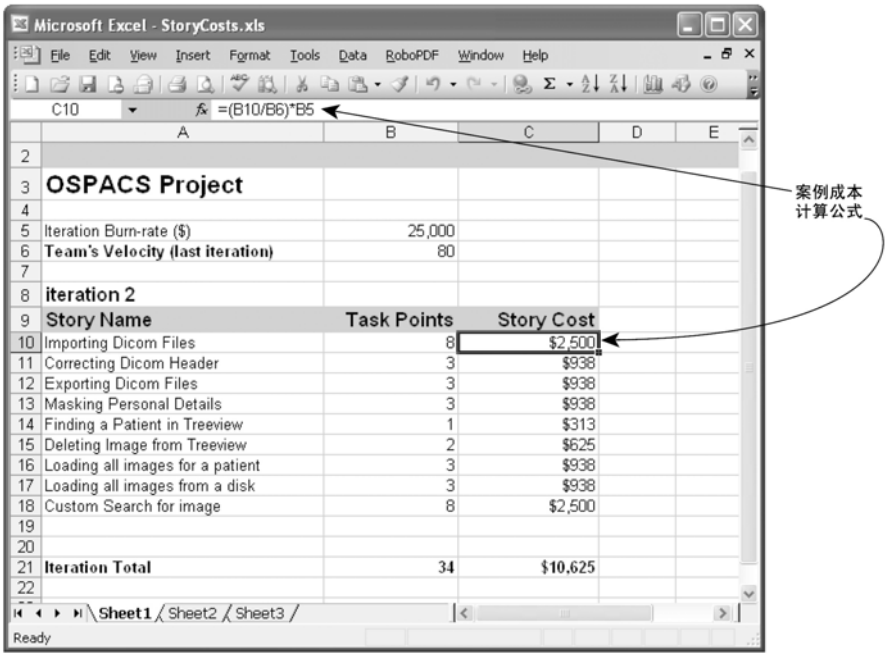
使用相对量的估算主要的缺点是会使得数据更加难以被理解。当开发人员之间互相讨论估算的时候，这不是问题，因为每个人都很快的对 1、2 或 3 任务点的不同有直观的认识。

519

[7] [AEP] Cohn, Mike. Agile Estimating and Planning (Addison-Wesley, 2006).

但是，向没参与估算过程的人解释这种差异可能比较困难。为此，我们建议你通过图 25-2 所示的简单计算将任务点的估算转换为案例价值，我们将使用如下术语。

- 任务点是开发人员做估算用的。是一些工作任务的相对尺度，采用非线性值衡量：0，1，2，3，5，8，13，20，40，100。
- 开发速度用于衡量团队开发进度。是在一次迭代中，案例中成功实现的那些任务点的合计值。
- 迭代消耗率是指项目进行一次迭代的运行费用。
- 案例费用是指客户的预算值。这表示正在执行的案例是以美元计算的。客户通过用案例任务点的总数除以团队的速度，然后乘以迭代消耗率就可以算出这个值。



520

图 25-2 扩展表格中显示的是任务点和案例价值

注：项目的消耗率不一定是在一次迭代中的项目实际运行成本，但是你应该使这个数字现实些，因为这能帮助客户和开发人员都来关注这个问题，以便提供更合理的资金投入。

预算
Budgeting

很容易由于预算的改变来调整一个项目，因为对开发团队而言，要实施这些案例可能的迭代的总数量取决于以下简单公式算出的预算（价格）：迭代的总数量=项目的总价格/迭代消耗率。因此，如果你有 250 000 美元，并且平均消耗率是每次迭代 25 000 美元，这个项目团队将有十次迭代的可能来实施这些案例，或者说如果你假设每次迭代的时间周期是一周，就能坚持十周。预算减低 50 000 美元的话，项目团队就减去 2 次迭代，这将导致一些低优先级的案例被取消，并且将向前调整 2 次迭代的最后发布日期。

> **警告**：我们必须提醒你的是，在增加预算的时候，你就须要加快团队的速度，因为如 Brooks 规律[8]所说，加入新的开发人员到项目中越迟，只能使项目推迟得越久。反之，接受你的速度的现实，就要要求客户重新排定优先顺序。

25.3 优先顺序
Prioritizing

案例的优先级决定了你应该实现的案例的执行顺序。决定优先顺序是客户的一项主要工作，因为他们有责任决定最终须要交付哪些业务及何时交付。不过，优先级还须要建立在对技术业务问题的响应上，因此，开发人员在过程中必须提供一些输入功能。 521

价值
Value

价值开始于软件发布给业务使用之后的时刻，所以按照优先顺序来提供最大的价值是明智的，可以使得利润能够在项目中尽可能早地得到积累（请见第 10 节）。你可以将现金流量折现、增值分析等财务上的术语进行量化，然后使用数字来设置优先顺序。不过，你也可以在非财务术语中指定一个值来评估出相对理想的案例和用户的满意程度，这是由实施的结果决定的。你对有代表性的客户（和用户）进行访问，基于一些基准为案例打分，这些基准如 DSDM 的 MoSCoW 规则[9]：必须有，应该有，可能有和会不会有。Sam

[8] [MMM] Brooks, Frederick P. The Mythical Man-Month (Addison-Wesley, 1975).

[9] DSDM: Dynamic Systems Development Method Web site (www.dsdm.org).

Guckenheimer[10]在他的书《Software Engineering with Visual Studio Team System》中对这种技术作了比较好的总体介绍，然后简单地进行这类分析。

> **提示：**要有一个会计计算出项目所有的财务上的数据，这将使你的分析具有权威性，并且能防止计划由于一些不合适的财务设想而遭到质疑。

业务风险
Business Risk

从业务的角度来看，业务风险是业务中存在的潜在失败，导致你不能从投资中获得最
522 好的价值。这种事可能会由于软件的延期交付而发生，因此商业机会不能得到完全的利用，或者可能导致软件的开发比预期耗费了更多的资源，从而减少了商业利润。不过，最主要的业务风险通常是交付的软件不能满足当前的业务需求。

> **注：**带有商业背景的客户通常善于管理各类风险（如出现项目延期交付或预算超支），因此，让客户施展这些技能，使他们有办法控制项目是明智的做法，具体描述请见第 26 章。

出现交付软件不能满足当前业务需求的风险，一般是由于外部的环境发生了变化或开发人员并没有完全正确理解他们必须要交付什么样的功能。因此，为了管理业务风险，客户必须降低那些受外部环境影响大的案例的优先级，提高那些有点难以解释的案例的优先级。这样一来，项目团队就会尽可能迟地交付那些有可能受到外部影响的案例，这样就能尽可能减少实施和交付之间发生变化的时间。这也意味着开发团队能尽可能早地交付所有复杂的案例，因此任何误解都可以在尽可能早的情况下被解决，以此来不断地实现商业的价值。

技术风险
Technical Risk

当技术风险的上升是由于项目团队的开发人员不知道应该如何实现这个案例的时候，可能须要适当地增加这个案例的优先级。这是因为你对一个案例开展工作时所产生的知

[10] [SETS] Guckenheimer, Sam, and Juan Perez. Software Engineering with Microsoft Visual Studio Team System (Addison-Wesley, 2006).

识，通常会消除案例实现中其余部分的不确定性因素，从而降低技术风险。在项目的早期处理能更好承担各种风险，因为这样你有更多的时间来找到一个技术解决方案，并且防止制造出一个不能发布的产品（因为它缺乏一些关键的功能）。 523

> 注：MSF 敏捷开发过程模板允许为任何已经确定的风险创建工作项，这样你就可以追踪并且报告它们的情况。但是，对一个小的团队来说，在案例卡片上写注释来提醒人们注意哪些问题更合适。

移除依赖关系
Removing Dependencies

显然，当案例之间有从属关系存在的时候，你的客户不能有效地确定基于风险与价值基础上的优先顺序，例如，案例 A 只能在案例 B 完成之后进行。所以，你应该尽量避免在案例之间产生任何依赖关系，并且在试图设定优先顺序之前，移除任何可能已经出现的依赖关系。

通常情况下，在开发人员估算案例的时候出现依赖关系要好过在客户制定案例时出现依赖关系。这是因为，在估算的过程中，开发人员思考过如何执行这个案例，所以他们可以试着将这个案例中的任务分离为一连串的子任务，然后声明其中的一些子任务作为其他案例的公用子任务，以此来优化开发任务。

当一个任务涉及不只一个案例的时候，就会在案例之间产生依赖关系，因此要消除这个问题，你只须确保每个任务都只涉及一个案例，并接受这样做可能导致一定数量的任务重叠的事实。有些重叠的任务可以在开发人员开始优化前实现，请见第 26 章中的任务计划一节。

> 注：偶尔，案例的估算主要取决于案例的运行。在这种情况下，你必须提醒客户在确定案例的优先顺序的时候注意这个问题，同样地，要警告客户注意技术风险。

应该避免将工作分解得太细，这是因为你要开始思考如何处理在不同的案例中间共享任务，以及因此产生的附带风险来优化工作。因此，案例之间存在的从属关系通常意味着在估算开发人员的任务时，开发人员介入了太多的细节。正因为如此，许多敏捷开发团队在估算期间都避免直接讨论任务，而是直接将一定大小的案例作为“案例点”，按照一定的单元，直接写在相关的案例卡片上。我们认为，为人们将要做的“任务”提供大小尺度 524

比为他们想要的“案例”提供大小尺度要好，这是再自然不过的了。正因为如此，我们喜欢使用任务点这个术语。

> 注：传统的敏捷开发的案例大小的度量是理想的天数[11]，但是你可能会发现一些项目中用了其他的单位，如案例点[12]。当你在处理相对数据的时候，这些单位的名称是不相干的。要紧的是比较任务间的大小。

25.4 总结 CONCLUSION

在本章，当你要对案例进行创建、估算、确定优先顺序的时候，我们确定了以下几个要点供你考虑。

- 在生成卡片期间，开发人员和客户一起开发案例，互相交谈，确定了如何测试作为某种形式的确认。缩写 INVEST 描述的是一个好的案例的属性，它是独立的、可协商的、有价值的、有能力进行估算的、小的和可测试的。
- 开发人员以卡片为单位，通过在卡片上写入相对大小的数据（如任务点）来估算案例。这样你可以调整计划来修改进度而无须变更每个案例的估算。相关的估算往往
525 也更准确。
- 你应该使用一个非线性的尺度，如 0，1，2，3，5，8，13，20，40，100 来表示任务点的大小；但是大多数的案例将只须要一个小范围内数量的任务点（1，2，或 3 个）。
- 客户通过整理大量等待执行的卡片，确定案例的优先顺序，使得最高优先的案例放在最上面。他们按照商业的价值和风险来设定优先顺序，例如那些提供最高价值的、商业风险最小的案例会被最先实现。
- 开发人员也在设定案例的优先顺序中发挥一部分作用，因为技术风险的处理宜早不宜迟，但是必须避免案例之间的技术互相依赖，因为这会影响设置业务的优先顺序。

> 注：正确地估算和确定优先顺序是控制好任何项目的先决条件，因为将垃圾信息加入
526 到计划过程中必然会导致一无所获的结果，所以垃圾信息必须被清理出来。

[11] Beck, Kent. Extreme Programming Explained, First Edition (Addison-Wesley, 2000).
[12] Beck, Kent. Extreme Programming Explained, First Edition (Addison-Wesley, 2000).