第12章　通过原型法减少项目风险

我最近遇到一个软件开发组，他们有一个令人不悦的经历：用户以不适合为理由拒绝了他们开发的整个产品。在产品发布之前，用户并没有见过用户界面，他们发现界面和潜在的需求都存在问题。软件原型是一种技术，你可以利用这种技术减少客户对产品不满意的风险。来自用户的早期反馈可以使开发小组正确理解需求，并知道如何最好地实现这些需求。

即使你完成了在前几章所叙述的需求获取、分析和编写规格说明，你的需求中还有一部分对客户或开发者仍然不明确或不清晰。如果不解决这些问题，那么必然在用户产品视图和开发者对于开发什么产品的理解之间存在期望差距。通过阅读文本需求或研究分析模型，很难想象软件产品在特定的环境中如何运行。原型可以使新产品实在化，为使用实例带来生机，并消除你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。

原型有多种含义，并且参与建立原型的人可以有不同的期望。例如，一个飞机原型实际上可以飞翔 —— 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。本章将研究各种类型的软件原型、它们在需求开发中的应用以及如何使原型成为软件开发过程中有效的组成部分（ Wood and Kang 1992）。

12.1　原型是“什么”和“为什么”要原型

一个软件原型是所提出的新产品的部分实现。使用原型有三个主要目的：

- 明确并完善需求　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。用户对原型的评价可以指出需求中的许多问题，在你开发真正产品之前，可以最低的费用来解决这些问题。
- 探索设计选择方案　原型作为一种设计工具，用它可以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。
- 发展为最终的产品原型　作为一种构造工具，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，你可以完成整个产品的开发。

建立原型的主要原因是为了解决在产品开发的早期阶段不确定的问题。利用这些不确定性来判断系统中哪一部分需要建立原型和希望从用户对原型的评价中获得什么。对于发现和解决需求中的二义性，原型也是一种很好的方法。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。用户、经理和其他非技术项目风险承担者发现在确定和开发产品时，原型可以使他们的想象更具体化。原型比开发者常用的技术术语更易于理解。

12.2　水平和垂直的原型

当人们谈到“软件原型”时，他们所想到的通常是可能的用户界面的“水平原型”

(horizontal prototype)。水平原型也叫做“行为原型”(behavioral prototype)或“模型”(mock-up)。它可以让你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，原型使用户所探讨的问题更加具体化。需要注意的是，这种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正面像，可能允许这些界面之间的一些导航，但是它仅包含少量的功能并没有真正实现所有的功能。想像你心目中的美国西部：牛仔走进酒店，而后从马房中走出来，然而，他并没有喝酒，也没见到一匹马，因为，在虚假的建筑物之后什么也不存在。

一个模型(mock-up)展示给用户的是在原型化屏幕上可用的功能和导航选择。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的内容的信息。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。虽然原型看起来似乎可以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特定的使用实例的一种观念。用户对原型的评价可以指出使用实例的可选过程，遗漏的过程步骤，或原先没有发现的异常情况。

当你在相当抽象的级别上建立原型时，用户可以把注意力集中在需求和工作流问题上，而不会被精细的外形或屏幕上元素的位置所干扰（Constantine 1998）。在澄清了需求并确定了界面中的框架之后，你可以建立更详细的原型来探索用户界面的设计。还可以使用不同的屏幕设计工具或甚至使用纸和铅笔来建立水平原型，这将在以后讨论。

垂直原型（vertical prototype），也叫做结构化原型或概念的证明，实现了一部分应用功能。当你不能确信所提出的构造软件的方法是否完善或者当你需要优化算法，评价一个数据库的图表或测试临界时间需求时，你就要开发一个垂直原型。垂直原型通常用在生产运行环境中的生产工具构造，使其结果一目了然（更有意义）。比起在软件的需求开发阶段，垂直原型更常用于软件的设计阶段以减少风险。

我曾经参与一项需要实现一个特殊的客户 /服务器体系结构的项目，并作为从以主机为中心的环境到基于网络化的 Unix服务器和工作站的应用环境的转换策略的一部分（ Thompson and Wiegers 1995）。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，这可以使我们评估所提出体系结构的通信组件、性能和可靠性。实验是成功的，我们基于那一体系结构的实施也是成功的。

12.3 抛弃型原型或进化型原型

在构造一个原型以前，需要充分与客户交流，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。你可以建立一个抛弃型原型(throwaway prototype)或探索型原型(exploratory prototype）来回答这些问题，解决不可测性并提高需求质量（Davis 1993）。因为你打算在原型达到预期目的以后将它抛弃，所以可以花最小的代价尽快地建立该原型。如果你在原型上付出的努力越多，那么项目的参与者就越不愿意将它抛弃。

当你建立抛弃型原型时，你忽略了很多具体的软件构造技术。而强调在健壮性、可靠性、性能和长期的可维护原则下迅速实现软件并易于维护。基于这一原因，你不能将抛弃型原型

中的代码移植到你的产品系统中，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。

当你遇到需求中的不确定性、二义性、不完整性或含糊性时，最合适的方法是建立抛弃式模型。你需要解决这些问题以减少在继续开发时存在的风险。原型可帮助用户和开发者想象如何实现需求和可以发现需求中的漏洞。它还可以使用户判断出这些需求是否可以完成必要的业务过程。

图12-1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。每一个使用实例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（见第 10章）。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使用实例描述的改变（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，你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。比起直接从使用实例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。

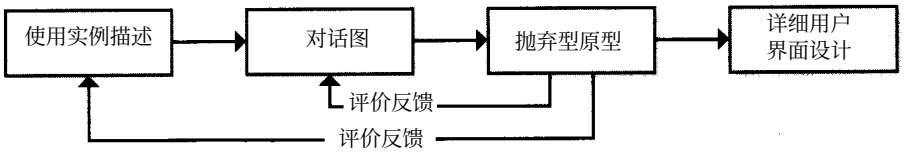

图12-1 利用抛弃型原型从用户任务到界面设计的活动序列

与抛弃型原型相对应的是进化型原型 (evolutionary prototype)，在已经清楚地定义了需求的情况下，进化型原型为开发渐增式产品提供了坚实的构造基础。进化型原型是螺旋式软件开发生存周期模型的一部分（ Boehm1988）,也是一些面向对象软件开发过程的一部分（Kruchten 1996）。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。一个进化型原型必须设计为易于升级和优化的，因此，你必须重视软件系统性和完整性的设计原则。要达到进化型原型的质量要求并没有捷径。

我们应该考虑进化型原型的第一次演变，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发展为实现最终完整的产品。这种原型提供了可以使用户快速获得有用功能的方法。

演化式模型适用于Web开发项目。在我曾经主持的一个 Web项目中，我们根据从使用实例分析中得出的需求，建立了四个原型序列。许多用户对每一个原型进行了评价，根据他们对我们提出的问题的回答，我们对原型进行了修正。对第四个原型修改之后，产生了我们的Web站点。

图12-2描述了在软件开发过程中，可以综合使用多种原型的许多方法。例如，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，可以渐增式地实现需求。贯穿图 12-2的一条可选路径在最终设计用户界面之前，将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。你所不能实现的是：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。表 12-1总结了抛弃式、演化式、水平和垂直原型的一些典型应用。

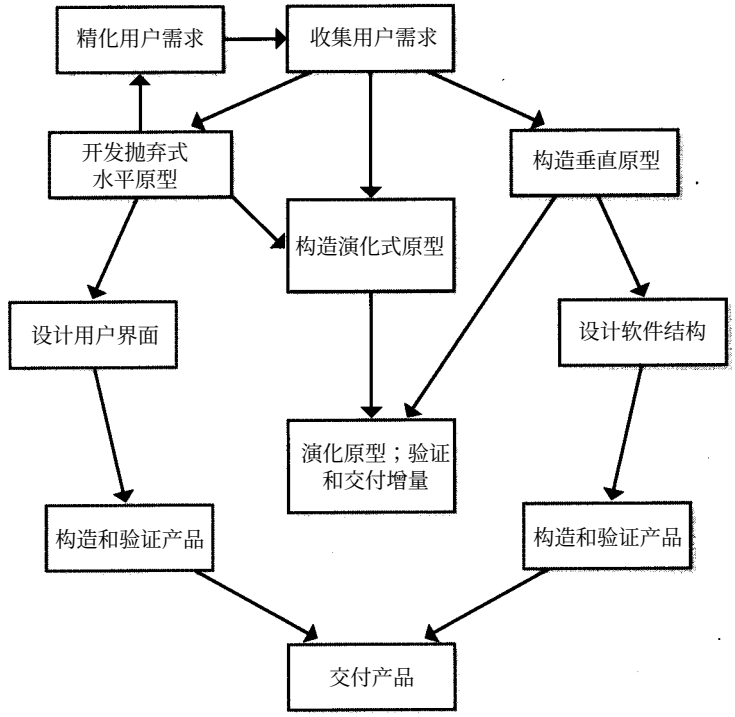

图12-2 在软件开发过程中使用原型法的一些可能的方法

表12-1 软件原型的典型应用

	抛 弃 型	演 化 型
水平	• 澄清并精化使用实例和功能需求 • 查明遗漏的功能 • 探索用户界面方法	• 实现核心的使用实例 • 根据优先级，实现附加的使用实例 • 开发并精化Web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

12.4 书面原型和电子原型

在许多情况下，一个可执行的原型未必可以获取所需的用于解决关于需求不确定性的信息。书面原型（paper prototype）（有时也叫低保真原型）是一种廉价、快速并且不涉及高技术的方法，它可以把一个系统某部分是如何实现的呈现在用户面前（ Rettig 1994; Hohmann 1997）。书面原型有助于判断用户和开发者在需求上是否达成共识。他们可以使你在开发产品代码前，对各种可能的解决方案进行试验性的并且低风险的尝试。

书面原型所包括的工具仅仅是纸张、索引卡、粘贴纸、塑料板、白板和标记器。你可以对屏幕布局进行构思，而并不必关心那些按钮和小装饰物应该出现在什么位置上。用户愿意提供反馈，这将引起多页书面原型极充分的改变。有时，他们并不急于评论一个基于计算机的可爱的原型，因为该原型凝结了开发者的许多辛劳。开发者也是经常不愿意对精心制作的

电子原型（electronic prototype）做重大更改。

有了“低保真”原型，当用户评价原型时，一个人可以充当计算机的角色。用户最初的动作是高呼他想在特定的屏幕上做什么：“我需要在File菜单中选择Print Preview选项。”当用户进行这一“操作”时，模仿计算机的人就会把关于显示方面的纸张和索引卡给用户看。用户就可以判断这些界面是否是所期望的响应，并且还可以判断所显示的项是否正确。如果有错误，只要用一张新纸或索引卡，重画一张就可以了。

不管你建立原型的工具多么高效，在纸张上勾画界面是最快的。书面原型方便了原型的快速反复性，而在需求开发中反复性是一个关键的成功因素。在运用自动化工具建立详细用户界面原型，构造一个进化型原型或者从事传统设计和构造活动之前，书面原型对于精化需求是一种优秀的技术，它还提供了一个管理客户期望的有用工具。

如果你决定建立一个电子抛弃型原型，那么就有许多工具可以使用（ Andriole 1996）。这些工具包括：

- 编程语言，例如：Microsoft Visual Basic, IBM VisualAge Smalltalk 和 Inprise Delphi.
- 脚本语言，例如：Perl, Python 和 Rexx
- 商品化的建立原型的工具包、屏幕绘图器和图形用户界面构筑师。

基于Web使用可以快速修改的HTML页（超文本标注语言），它对于建立澄清需求而不去探索特定的用户界面设计的原型是很有用的。合适的工具可以使你迅速地实现并更改用户界面组件，而不管在界面后面的代码效率的高低。当然，如果你正在建立一个演化式模型，那么你必须一开始就使用产品开发工具。

12.5 原型评价

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以获取所需要的信息，这样你可以提高原型评价的有效性。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。你可以从使用实例和原型描述的功能中获得评价脚本。这一脚本可以让用户执行特定的任务并且指导他们评价你觉得最不确定的原型部分。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。评价组必须从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。

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，亲自观察用户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信息。要注意用户所指出那些原型部分。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，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。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，你要避免诱导用户用设计好的特定方法执行一些功能。

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，那么把这些结论和你如何实现都记录下来。没有用户想法参与的决策，就必须不断地回溯，将造成不必要的时间浪费。对于一个垂直原型，记录好所实施的评价以及评价结果，从而做出关于所探索的不同技术方法可行性的决策。

12.6 原型法的最大风险

原型法是一种减少软件项目失败风险的技术。然而，原型法又引入了自身的风险。最大的风险是用户或者经理看到一个正在运行的原型从而以为产品即将完成。“哦，这看起来好像差不多了！”充满热情的原型评价者说：“这看起来真的很好，你能把它完成后交给我吗？”

一句话：不行！如果你正在演示或评价一个抛弃型原型，无论它与真正的产品是如何相像，它决不会达到产品的使用程度。它仅是一个模型，一种模拟或一次实验。处理风险承担者的期望是成功原型法的一个关键因素，因此要保证那些见到原型的人理解为什么要建立原型并且怎样建立原型。决不能把抛弃型原型当作可交付的产品。交付原型可导致项目的延期完成，因为那些设计和编码并没有考虑到软件质量和容错性。

不要因为害怕提交不成熟产品的压力而阻碍你建立原型，但是你必须让见到原型的人明白你不会交付原型，甚至不会将它称之为软件。控制这种风险的一种方法是利用书面原型而不是电子原型。评价书面原型的人决不会误认为产品已经完成开发并可以交付了。另一种可能的方法是使用不同于在真正开发时所用的原型法工具，这将有助于你抵抗“已完成”原型开发并可把它当作产品交付的压力。

在原型评价期间，继续处理那些期望。如果评价者看到原型可以对一个模拟的数据库查询响应甚快，那么他们可能期望在最终的软件产品中也具有同样惊人的性能。在对最终产品的行为进行模拟时，要考虑现实中的时间延迟（这可以使原型不易被看作可即将交付的产品）。

12.7 原型法成功的因素

软件原型法提供了一套强有力的技术，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错误和用户界面的缺陷。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稽。

下一步：

- 查明你的项目中引起需求混乱的部分（如使用实例）。用书面原型勾画出代表你对需求的理解和如何实现的可能用户界面。让一些用户利用你的原型模拟执行一个任务或使用实例。确定对需求的最初理解中不完整和不正确的部分。相应地更改原型并重新检验。
- 给你的原型评价者递交一份本章信息的总结。这将有助于他们理解原型背后的原理，并且使之期望原型法结果的现实。