第11章　软件的质量属性

许多年前，我参加了一项工程，在该项目中用新的应用程序替换许多已有的主机（ main frame）应用程序。根据用户的要求，开发组设计了一个基于窗口的用户界面并定义了新的数据文件，其容量是旧文件的两倍。虽然新系统满足了技术上的规范，但并没有达到客户可接受的程度。用户总是抱怨用户界面运行缓慢，并且新的数据文件所占用的磁盘空间太大。

用户没有陈述对新产品的一些特性的期望，这就不能在他们所提出的功能需求中体现出来。糟糕的是，开发者和用户没有详细地讨论新技术方法所牵涉到可能的性能，从而导致了用户期望与产品实际性能之间的期望差异。比起仅仅满足客户所要求的功能，软件的成功似乎更为重要。

11.1　非功能需求

用户总是强调确定他们的功能、行为或需求 —— 软件让他们做的事情。除此之外，用户对产品如何良好地运转抱有许多期望。这些特性包括：产品的易用程度如何，执行速度如何，可靠性如何，当发生异常情况时，系统如何处理。这些被称为软件质量属性 (或质量因素)的特性是系统非功能（也叫非行为）部分的需求。

质量属性是很难定义的，并且他们经常造成开发者设计的产品和客户满意的产品之间的差异。就像Robert Charette(1990)指出的那样：“ 真正的现实系统中，在决定系统的成功或失败的因素中，满足非功能需求往往比满足功能需求更为重要 ”。优秀的软件产品反映了这些竞争性质量特性的优化平衡。如果你在需求的获取阶段不去探索客户对质量的期望，那么产品满足了他们的要求，这只能说你很幸运。但更多的可能是客户失望和开发者沮丧。

虽然，在需求获取阶段客户所提出的信息中包含提供了一些关于重要质量特性的线索，但客户通常不能主动提出他们的非功能期望。用户说软件必须 “ 健壮 ”, “ 可靠 ” 或 “ 高效 ” 时，这是很技巧地指出他们所想要的东西。从多方面考虑，质量必须由客户和那些构造测试和维护软件的人员来定义。探索用户隐含期望的问题可以导致对质量目标的描述，并且制定可以帮助开发者创建完美产品的标准。

11.2　质量属性

虽然有许多产品特性可以称为质量属性 (Quality Attribute)，但是在许多系统中需要认真考虑的仅是其中的一小部分。如果开发者知道哪些特性对项目的成功至关重要，那么他们就能选择软件工程方法来达到特定的质量目标（ Glass 1992; DeGrace and Stahl 1993）。根据不同的设计可以把质量属性分类（Boehm 1976; DeGrace and Stahl 1993）。一种属性分类的方法是把在运行时可识别的特性与那些不可识别的特性区分开（ Bass, Clements and Kazman 1998）。另一种方法是把对用户很重要的可见特性与对开发者和维护者很重要的不可见特性区分开。那些对开发者具有重要意义的属性使产品易于更改、验证，并易于移植到新的平台上，从而可以间接地满足客户的需要。

在表11-1中，分两类来描述每个项目都要考虑的质量属性；还有其它许多属性（ Charette 1990）。一些属性对于嵌入式系统是很重要的（高效性和可靠性），而其它的属性则用于主机应用程序（有效性和可维护性）或桌面系统（互操作性和可用性）。在一个理想的范围中，每一个系统总是最大限度地展示所有这些属性的可能价值。系统将随时可用，决不会崩溃，可立即提供结果，并且易于使用。因为理想环境是不可得到的，因此，你必须知道表 11-1中那些属性的子集对项目的成功至关重要。然后，根据这些基本属性来定义用户和开发者的目标，从而产品的设计者可以作出合适的选择。

表11-1 软件质量属性

对用户最重要的属性	对开发者最重要的属性
有效性（availability）	可维护性 (maintainability)
高效性 (efficiency)	可移植性 (portability)
灵活性 (flexibility)	可重用性 (reusability)
完整性 (integrity)	可测试性 (testability)
互操作性(interoperability)	
可靠性 (reliability)	
健壮性 (robustness)	
可用性 (usability)	

产品的不同部分与所期望的质量特性有着不同的组合。高效性可能对某些部分是很重要的，而可用性对其它部分则很重要。把应用于整个产品的质量特性与特定某些部分、某些用户类或特殊使用环境的质量属性要区分开。把任何全局属性需求记录到在第 9章所示的软件需求规格说明的第e.4部分中，并把特定的目标和列在软件需求规格说明第 d部分的特性、使用实例或功能需求相联系起来。

11.3 定义质量属性

你必须根据用户对系统的期望来确定质量属性。定量地确定重要属性提供了对用户期望的清晰理解，这将有助于设计者提出最合理的解决方案（ Gilb 1988）。然而，大多数用户并不知道如何回答诸如“ 互操作性对你的重要性如何？ ”或者“ 软件应该具有怎样的可靠性？ ”等问题。在一个项目中，分析员想出了对于不同的用户类可能很重要的属性，并根据每一个属性设计出许多问题。他们利用这些问题询问每一个用户类的代表，可以把每个属性分成一级（不必多加考虑的属性）到五级（极其重要的属性）。这些问题的回答有助于分析员决定哪些质量特性用作设计标准是最重要的。

然后，分析员与用户一起为每一属性确定特定的、可测量的和可验证的需求（ Robertson and Robertson 1997）。如果质量目标不可验证，那么就说不清你是否达到这些目标。在合适的地方为每一个属性或目标指定级别或测量单位，以及最大和最小值。如果你不能定量地确定某些对你的项目很重要的属性，那么至少应该确定其优先级。 IEEE关于软件质量度量方法的标准提出了一个在综合质量度量基准体系下定义软件质量需求的方法（ IEEE 1992）。

另一个定义属性的方法是确定任何与质量期望相冲突的系统行为（ Voas 1999）。通过定义不悦人意行为 —— 一种反向需求 —— 你可以设计出强制系统表现出那些行为的测试用例。如果你不能强制系统，那么你可能达到了你的属性目标。这种方法最适用于要求安全性能很高的

应用程序，在这些应用程序中，系统的差错可能会导致生命危险。

这一节的剩余部分将简要地介绍列在表 11-1中的每一个质量属性，并提出一些有助于用户陈述他们对质量属性期望的问题。虽然这些例子有些简单，但提供了由“化学制品跟踪系统”或其他项目总结出的样本目标的陈述。你将需要选择最好的方法来表达每一个质量属性需求，这样可以指导开发者进行设计选择。

11.3.1 对用户重要的属性

1) 有效性 有效性指的是在预定的启动时间中，系统真正可用并且完全运行时间所占的百分比。更正式地说，有效性等于系统的平均故障时间（ MTTF）除以平均故障时间与故障修复时间之和。有些任务比起其它任务具有更严格的时间要求，此时，当用户要执行一个任务但系统在那一时刻不可用时，用户会感到很沮丧。询问用户需要多高的有效性，并且是否在任何时间，对满足业务或安全目标有效性都是必须的。一个有效性需求可能这样说明：“工作日期间，在当地时间早上 6点到午夜，系统的有效性至少达到 99.5%，在下午4点到6点，系统的有效性至少可达到99.95%。

2) 效率 效率是用来衡量系统如何优化处理器、磁盘空间或通信带宽的（ Davis 1993）。如果系统用完了所有可用的资源，那么用户遇到的将是性能的下降，这是效率降低的一个表现。拙劣的系统性能可激怒等待数据库查询结果的用户，或者可能对系统安全性造成威胁，就像一个实时处理系统超负荷一样。为了在不可预料的条件下允许安全缓冲，你可以这样定义：“在预计的高峰负载条件下， 10%处理器能力和15%系统可用内存必须留出备用。”在定义性能、能力和效率目标时，考虑硬件的最小配置是很重要的。

3) 灵活性 就像我们所知道的可扩充性、增加性、可延伸性和可扩展性一样，灵活性表明了在产品中增加新功能时所需工作量的大小。如果开发者预料到系统的扩展性，那么他们可以选择合适的方法来最大限度地增大系统的灵活性。灵活性对于通过一系列连续的发行版本，并采用渐增型和重复型方式开发的产品是很重要的。在我曾经参与的一个图形工程中，灵活性目标是如下设定的：“一个至少具有 6个月产品支持经验的软件维护程序员可以在一个小时之内为系统添加一个新的可支持硬拷贝的输出设备。”

4) 完整性 完整性（或安全性）主要涉及：防止非法访问系统功能、防止数据丢失、防止病毒入侵并防止私人数据进入系统。完整性对于通过 WWW执行的软件已成为一个重要的议题。电子商务系统的用户关心的是保护信用卡信息， Web的浏览者不愿意那些私人信息或他们所访问过的站点记录被非法使用。完整性的需求不能犯任何错误，即数据和访问必须通过特定的方法完全保护起来。用明确的术语陈述完整性的需求，如身份验证、用户特权级别、访问约束或者需要保护的精确数据。一个完整性的需求样本可以这样描述：“只有拥有查账员访问特权的用户才可以查看客户交易历史。”

5) 互操作性 互操作性表明了产品与其它系统交换数据和服务的难易程度。为了评估互操作性是否达到要求的程度，你必须知道用户使用其它哪一种应用程序与你的产品相连接，还要知道他们要交换什么数据。“化学制品跟踪系统”的用户习惯于使用一些商业工具绘制化学制品的结构图，所以他们提出如下的互操作性需求：“化学制品跟踪系统应该能够从ChemiDraw和Chem-Struct工具中导入任何有效化学制品结构图。”

6) 可靠性 可靠性是软件无故障执行一段时间的概率（Musa, Iannino and Okumoto 1987）。

健壮性和有效性有时可看成是可靠性的一部分。衡量软件可靠性的方法包括正确执行操作所占的比例，在发现新缺陷之前系统运行的时间长度和缺陷出现的密度。根据如果发生故障对系统有多大影响和对于最大的可靠性的费用是否合理，来定量地确定可靠性需求。如果软件满足了它的可靠性需求，那么即使该软件还存在缺陷，也可认为达到其可靠性目标。要求高可靠性的系统也是为高可测试性系统设计的。

我的开发组曾经开发过一个用于控制实验室设备的软件，这些设备全天工作并且使用稀有的、昂贵的化学制品。用户要求真正与实验相关的那部分软件要高可靠性，而其它系统功能，例如周期性地记录温度数据，则对可靠性要求不高。对于该系统的一个可靠性需求说明如下：“由于软件失效引起实验失败的概率应不超过 5‰”。

7) 健壮性　健壮性指的是当系统或其组成部分遇到非法输入数据、相关软件或硬件组成部分的缺陷或异常的操作情况时，能继续正确运行功能的程度。健壮的软件可以从发生问题的环境中完好地恢复并且可容忍用户的错误。当从用户那里获取健壮性的目标时，询问系统可能遇到的错误条件并且要了解用户想让系统如何响应。

我曾经主持过一个叫作图形引擎的可重用软件组件的开发，该图形引擎具有描述图形规划的数据文件，并且把这一规划传送到指定的输出设备上（Wiegers 1996b）。许多需要产生规划的应用程序就要请求调用图形引擎。由于在图形引擎中，我们将无法控制这些应用程序的数据，所以此时健壮性就成为必不可少的质量属性。我们的一个健壮性需求是这样说明的：“所有的规划参数都要指定一个缺省值，当输入数据丢失或无效时，就使用缺省值数据。”这个例子反映了对一个“用户”是另一个软件应用程序的产品，其健壮性设计的方法。

8) 可用性　可用性也称为“易用性”和“人类工程”，它所描述的是许多组成“用户友好”的因素。可用性衡量准备输入、操作和理解产品输出所花费的努力。你必须权衡易用性和学习如何操纵产品的简易性。“化学制品跟踪系统”的分析员询问用户这样的问题：“你能快速、简单地请求化学制品并浏览其它信息，这对你有多重要？”和“你请求一种化学制品大概需花多少时间？”对于定义使软件易于使用的许多特性而言，这只是一个简单的起点。对于可用性的讨论可以得出可测量的目标，例如“一个培训过的用户应该可以在平均 3分钟或最多5分钟时间以内，完成从供应商目录表中请求一种化学制品的操作。”

同样，调查新系统是否一定要与任何用户界面标准或常规的相符合，或者其用户界面是否一定要与其它常用系统的用户界面相一致。这里有一个可用性需求的例子：“在文件菜单中的所有功能都必须定义快捷键，该快捷键是由 Ctrl键和其它键组合实现的。出现在 Microsoft Word 2000中的菜单命令必须与Word使用相同的快捷键”。

可用性还包括对于新用户或不常使用产品的用户在学习使用产品时的简易程度。易学程度的目标可以经常定量地测量，例如，“一个新用户用不到30分钟时间适应环境后，就应该可以对一个化学制品提出请求”，或者“新的操作员在一天的培训学习之后，就应该可以正确执行他们所要求的任务的95%”。当你定义可用性或可学性的需求时，应考虑到在判断产品是否达到需求而对产品进行测试的费用。

11.3.2 对开发者重要的属性

1) 可维护性　可维护性表明了在软件中纠正一个缺陷或做一次更改的简易程度。可维护性取决于理解软件、更改软件和测试软件的简易程度，可维护性与灵活性密切相关。高可维

护性对于那些经历周期性更改的产品或快速开发的产品很重要。你可以根据修复（ fix）一个问题所花的平均时间和修复正确的百分比来衡量可维护性。

“ 化学制品跟踪系统 ”包括如下的可维护性需求：“ 在接到来自联邦政府修订的化学制品报告的规定后，对于现有报表的更改操作必须在一周内完成。”在图形引擎工程中，我们知道，我们必须不断更新软件以满足用户日益发展的需要，因此，我们确定了设计标准以增强系统总的可维护性：“ 函数调用不能超过两层深度 “ ，并且 “ 每一个软件模块中，注释与源代码语句的比例至少为1：2。”认真并精确地描述设计目标，以防止开发者做出与预定目标不相符的愚蠢行为。

2) 可移植性　可移植性是度量把一个软件从一种运行环境转移到另一种运行环境中所花费的工作量。软件可移植的设计方法与软件可重用的设计方法相似（ Glass 1992）。可移植性对于工程的成功是不重要的，对工程的结果也无关紧要。可以移植的目标必须陈述产品中可以移植到其它环境的那一部分，并确定相应的目标环境。于是，开发者就能选择设计和编码方法以适当提高产品的可移植性。

3) 可重用性　从软件开发的长远目标上看，可重用性表明了一个软件组件除了在最初开发的系统中使用之外，还可以在其它应用程序中使用的程度。比起创建一个你打算只在一个应用程序中使用的组件，开发可重用软件的费用会更大些。可重用软件必须标准化、资料齐全、不依赖于特定的应用程序和运行环境，并具有一般性（ DeGrace and Stahl 1993）。确定新系统中哪些元素需要用方便于代码重用的方法设计，或者规定作为项目副产品的可重用性组件库。

4) 可测试性　可测试性指的是测试软件组件或集成产品时查找缺陷的简易程度。如果产品中包含复杂的算法和逻辑，或如果具有复杂的功能性的相互关系，那么对于可测试性的设计就很重要。如果经常更改产品，那么可测试性也是很重要的，因为将经常对产品进行回归测试来判断更改是否破坏了现有的功能性。

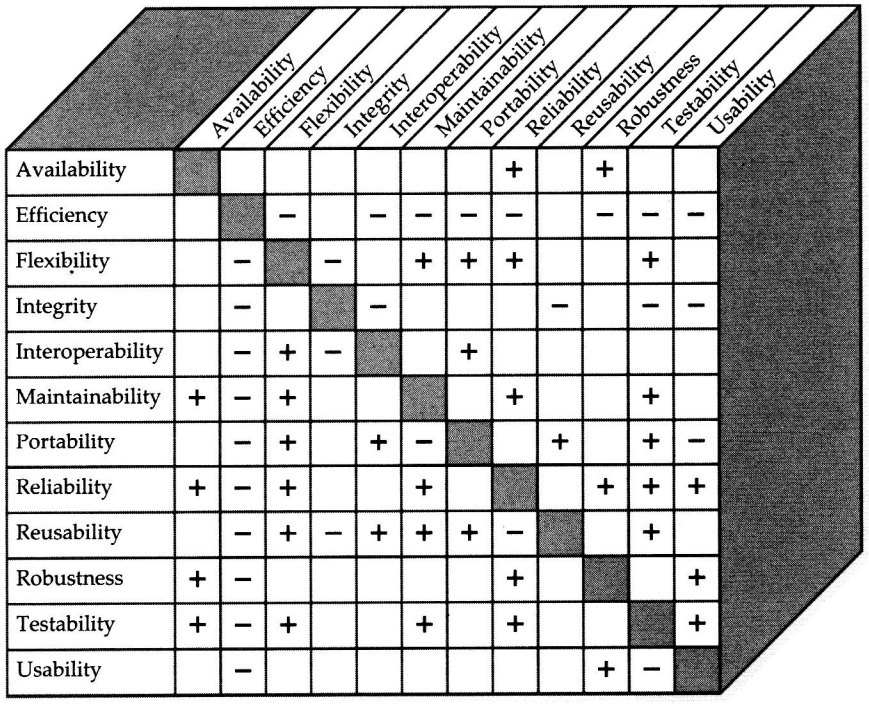
因为我们知道随着图形引擎功能的不断增强，我们需要对它进行多次测试，所以作出了如下的设计目标：“ 一个模块的最大循环复杂度不能超过 20。”循环复杂度度量一个模块源代码中逻辑分支数目（McCabe 1982）。在一个模块中加入过多的分支和循环将使该模块难于测试、理解和维护。如果一些模块的循环复杂度大于 20，这并不会导致整个项目的失败，但指定这样的设计标准有助于开发者达到一个令人满意的质量目标。

11.4 属性的取舍

有时，不可避免地要对一些特定的属性对进行取舍。用户和开发者必须确定哪些属性比其它属性更为重要，并定出优先级。在他们作决策时，要始终遵照那些优先级。图 11-1描述了来自表11-1的质量属性之间一些典型的相互关系，当然你也可能会遇到一些例外（ Charette 1990；IEEE 1992; Glass 1993）。一个单元格中的加号表明单元格所在行的属性增加了对其所在列的属性的积极影响。例如，增强软件可重用性的设计方法也可以使软件变得灵活、更易于与其它软件组件相连接、更易于维护、更易于移植并且更易于测试。

一个单元格中的减号表明单元格所在行的属性增加了对其所在列的属性的不利影响。高效性对其它许多属性具有消极影响。如果你编写最紧凑，最快的代码，并使用一种特殊的预编译器和操作系统，那么这将不易移植到其它环境，而且还难于维护和改进软件。类似地，一些优化操作者易用性的系统或企图具有灵活性、可用性并且可以与其它软硬件相互操作的

系统将付出性能方面的代价。例如，比起使用具有完整的制定图形代码的旧应用系统，使用外部的通用图形引擎工具生成图形规划将大大降低性能。你必须在性能代价和你所提出的解决方案的预期利益之间作出权衡，以确保作出合理的取舍。



	Availability	Efficiency	Flexibility	Integrity	Interoperability	Maintainability	Portability	Reliability	Reusability	Robustness	Testability	Usability
Availability							+		+			
Efficiency			−		−	−	−		−	−	−	
Flexibility		−		−		+	+	+			+	
Integrity		−			−				−		−	−
Interoperability		−	+	−			+					
Maintainability	+	−	+					+			+	
Portability		−	+		+	−			+		+	−
Reliability	+	−	+			+				+	+	+
Reusability		−	+	−	+	+	+	−			+	
Robustness	+	−						+				+
Testability	+	−	+			+		+				+
Usability		−								+	−	

图11-1 选择的质量属性之间的正负关系

为了达到产品特性的最佳平衡，你必须在需求获取阶段识别、确定相关的质量属性，并且为之确定优先级。当你为项目定义重要的质量属性时，利用图 11-1可以防止发生与目标冲突的行为。以下是一些例子：

- 如果软件要在多平台下运行（可移植性），那么就不要对可用性抱有乐观态度。
- 可重用软件能普遍适用于多种环境中，因此，不能达到特定的容错（可靠性）或完整性目标。
- 对于高安全的系统，很难完全测试其完整性需求；可重用的类组件或与其它应用程序的互操作可能会破坏其安全机制。

在软件中，其自身不能实现质量特性的合理平衡。在需求获取的过程中，加入对质量属性期望的讨论，并把你所了解的写入软件需求规格说明中。这样，才有可能提供使所有项目风险承担者满意的产品。

下一步：

- 从表11-1中确定一些对于你目前项目的用户至关重要的质量属性。系统地陈述每个属性的一两个问题，这将有助于用户说清他们的期望。基于用户的回答，为每一个重要属性写出一两个明确的目标。