质量属性的代价和之间约束的个人理解

质量属性的代价

为了方便思考和编写，这里只讨论可用性、可靠性、有效性和完整性。

1. 可用性（Availability）

可用性的定义是在要求的外部资源得到保证的前提下，产品在规定的条件下和规定的时刻或时间区间内处于可执行规定功能状态的能力。它是产品可靠性、维修性和维修保障性的综合反映。

关于Availability的计算公式，MTBF / (MTBF + MTTR)，其中平均修复时间（Mean Time To Repair，MTTR）是描述产品由故障状态转为工作状态时修理时间的平均值。在工程学，MTTR是衡量产品维修性的值，在维护合约里很常见，并以之作为服务收费的准则，而平均故障间隔时间（Mean Time Between Failure， MTBF）是指相邻两次故障之间的平均工作时间，是衡量一个产品的可靠性指标。通常大家习惯用N个9来表征系统可用性，比如99.9%（3-nines availability），99.999%（5-nines availability）。

代价

将可靠性或可用性等质量属性指定为100%时，一定要谨慎，因为这是几乎不可能实现的，而且要努力达到这一目标的成本也是很高的。有一家公司对它的商店地板制造系统提出了100%的可用性需求、要求一年365天，一天24小时，系统都可用。为了设法达到这一严格的可用性、公司安装了两套独立的计算机系统，这样就可以在不处于运行状态的备用计算机上安装升级软件。对司用性需求，这是一个花费很高的解决方案。

2. 可靠性(Reliability)

可靠性是软件无故障执行一段时间的概率。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。

代价

相对于软件可用性来讲，可靠性是指软件能够精确执行需求要求的操作的能力。提升可靠性与提升可用性有部分重叠，比如在高并发的情况下，缓存系统可能出现缓存雪崩，这时缓存系统崩溃，使得数据库直接暴露了出来，所有数据只能直接存储于数据库。结果不能提供软件可靠性，此问题的解决方法有很多，但又必须付出不小的代价。

3.有效性(efficiency)

有效性是用来衡量系统在利用处理器的处理能力、磁盘空间或通信带宽等方面的能力。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。

代价

出于对有效性的要求，系统要在保留一部分硬件能力用于缓冲的情况下，充分利用其能力来进行系统的各种操作。比如CPU的处理能力、内存有效利用率或通信带宽等。充分提升利用硬件的能力，可以降低用户或服务器对硬件的要求，从而扩大了用户范围或降低了用户对硬件性能的进一步需要。但对于某些场景下，我们并不需要对有效性有太大的要求，比如某些需要对用户层面的计算，我们可以采用胖客户端的设计，来降低服务器对硬件的需求，间接地降低了有效性的要求；但对于某些计算频繁的情形，比如实时的场景渲染或数据库的设计，有效性是有很高的要求。

4. 完整性 (integrity)

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、保护软件免受病毒入侵,以及保护输入到系统的数据的保密性和安全性,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。

代价

对完整性的要求，我们需要考虑多方面的安全问题，比如网络安全，软件本身的安全（防止非法修改内存等）。为了防止安全问题的出现，我们可能需要增加用户的冗余操作或牺牲一些软件性能。对于引入用户冗余操作的场景，几乎用户对帐号的所有操作都需要冗余操作，如登录帐号时必须要使用密码，或通过手机号登录需要短信验证。完整性要求对用户的易用性也提出了一些约束。

质量属性之间的约束

1. 高可用性对其他属性的约束

为了实现高可用性，我们往往需要对高频度高性能要求的业务进行优化。解决方法是通常有两种，其一是对硬件或运行环境的优化，如加入缓存、对分布式架构的软件系统增加主机、数据库查询语句的优化等；其二是对软件本身的优化，如算法优化等。这些优化措施对其他属性有着较高要求，比如对灵活性而言，由于优化总是具有强烈的针对性，一个功能的改变常常使得优化被整个推翻；再比如可维护性，分布式的系统对与运维人员来说是一个很大的挑战，每一个主机的运行状态如何？每一个性能指标是否达到？等等问题。

2. 完整性对其他属性的约束

对于高完整性系统，其可测试性往往难以面面俱到，且很多隐藏问题的场景难以复现；用户的易用性可能也会降低。比如在一个嘈杂且高噪音高频率的环境下，由于硬盘磁头和盘片的固有频率与环境相似，导致共振现象产生，读写头失灵从而宕机同时丢失大量数据。再比如很多社交软件为了防止大量爬取用户个人信息等，提出了严格的访问时间和访问平台的限制，如淘宝京东等电商平台甚至限制linux平台上的低频访问，大大降低linux用户的易用性。但我认为这种损失是非常值得的。

3. 可移植性对其他属性的约束

可移植性对易用性作出了不小的约束。比如对pc端web页面设计，我们可以降大量功能作为可点击的组件摆放在不同位置。但若使用相同的页面移植到移动端时，由于屏幕太小和用户常误触等问题，界面设计往往需要有针对性的设计，否则将大量降低用户的易用性。