第2章 软件质量属性

内容

- **2.1 软件质量属性概念**
- 2.2 设计时质量属性
- 2.3 运行时质量属性
- 2.4 系统质量属性
- 2.5 用户质量属性
- 2.6 其他质量属性

Quality Attribute

- A **quality attribute (QA)** is a **measurable** or **testable** property of a system that is used to indicate **how well the system satisfies the needs** of its stakeholders. You can think of a quality attribute as measuring the "**goodness**" of a product along some dimension of interest to a stakeholder.
- 质量原意是指好的程度，与目标吻合的程度，在软件工程领域，目标自然就是需求。
- 软件质量属性指的是一个系统的可度量、可测试的属性，这些属性会影响到系统的运行时行为、系统设计方式以及用户的体验等。质量属性的优劣程度反映了设计是否成功以及软件系统的整体质量。

Architecture and Requirements

- 1. ***Functional requirements***. These requirements state what the system **must do**, and how it must behave or react to runtime stimuli.
- 2. ***Quality attribute requirements***. These requirements are **qualifications**（限制性条件） of the **functional requirements** or of the **overall product**. A qualification of a functional requirement is an item such as **how fast** the function must be performed, or **how resilient**(恢复活力的) it must be to erroneous input. A qualification of the overall product is an item such as the **time to deploy** the product or a limitation on **operational costs**.

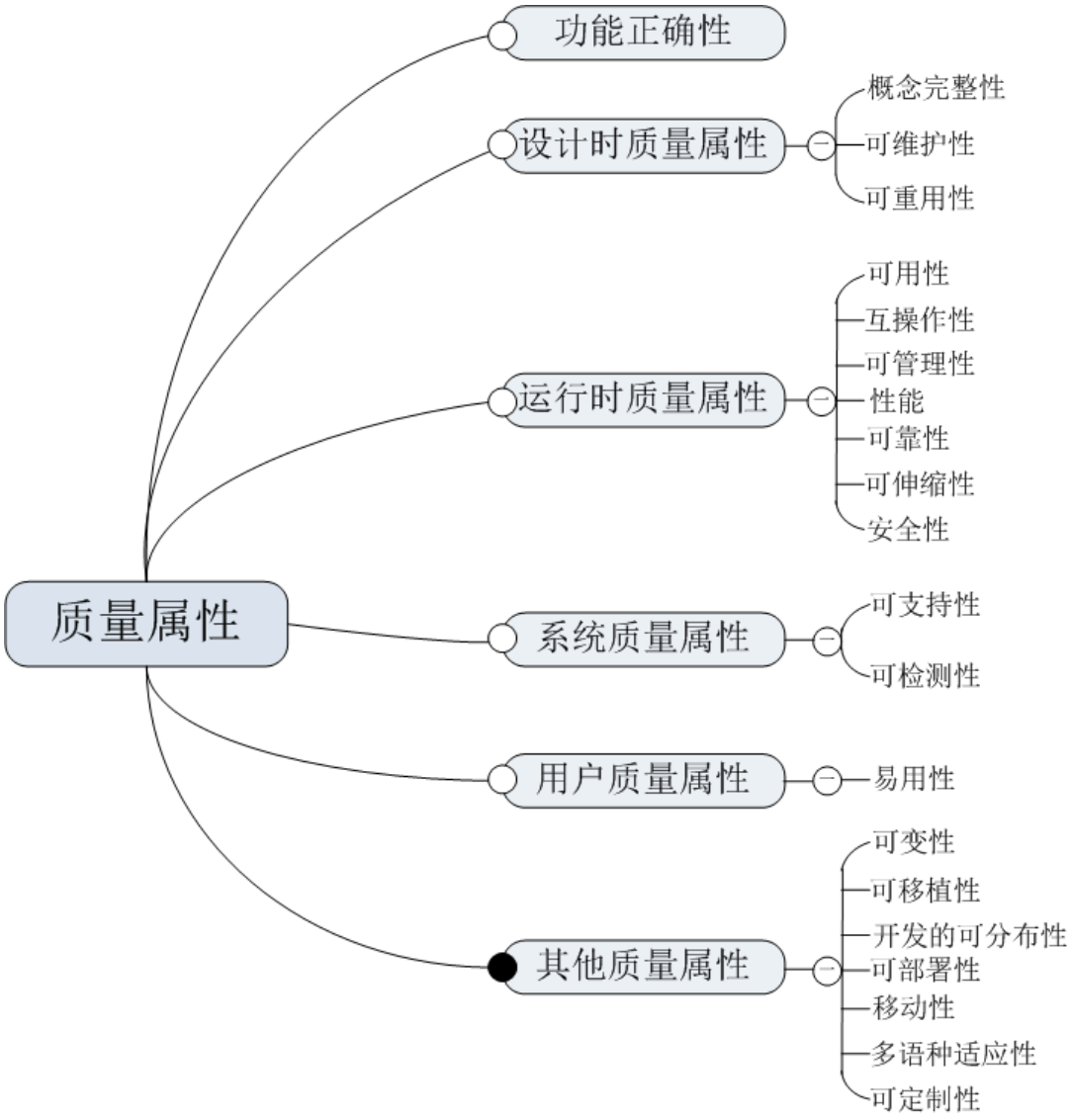
Architecture and Requirements（Cont.）

- 3. ***Constraints***. A constraint is a **design decision with zero degrees of freedom**. That is, it's a design decision that's already been made.
- Examples include the requirement to use a certain **programming language** or to **reuse a certain existing module**, or a **management fiat** to make your system service oriented.

"Response" of architecture requirements

- 1. ***Functional requirements*** are satisfied by **assigning an appropriate sequence of responsibilities throughout the design**. As we will see later in this course, **assigning responsibilities to architectural elements is a fundamental architectural design decision**.
- 2. ***Quality attribute requirements*** are satisfied by the **various structures designed into the architecture**, and the **behaviors** and **interactions** of the elements that populate those structures.
- 3. ***Constraints*** are satisfied by **accepting the design decision** and reconciling（调解） it with other affected design decisions.

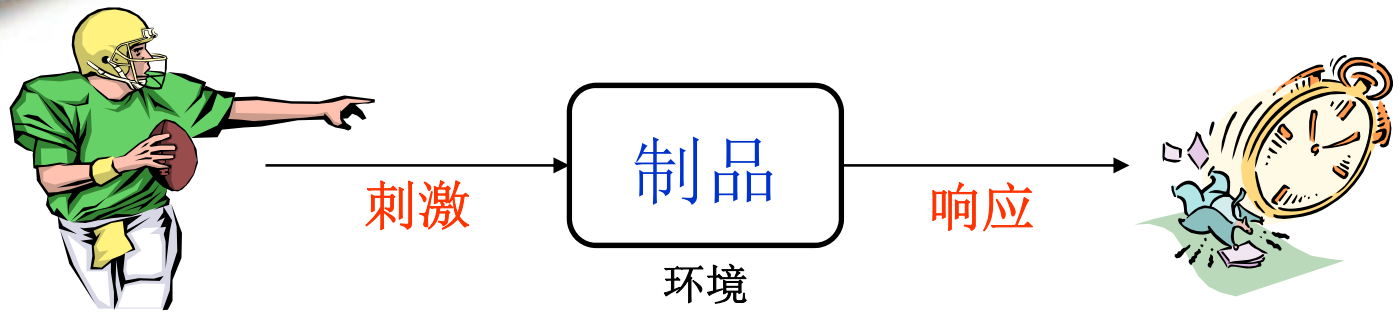
软件质量属性分类


质量属性
功能正确性
设计时质量属性
概念完整性
可维护性
可重用性
运行时质量属性
可用性
互操作性
可管理性
性能
可靠性
可伸缩性
安全性
系统质量属性
可支持性
可检测性
用户质量属性
易用性
其他质量属性
可变性
可移植性
开发的可分布性
可部署性
移动性
多语种适应性
可定制性

功能的正确性

- 对任何系统而言，能按照功能需求正确执行应是对其**最基本的要求**。
- 正确性是指软件按照需求正确执行任务的能力，这无疑是**第一重要的软件质量属性**。

质量属性需求说明



质量属性场景（Scenario）的6个组成部分

- 刺激源：生成刺激的实体(人、系统或其它)
- 刺激：当到达时引起系统进行响应的条件
- 环境：刺激到达时，系统所处的状态，如系统可能处于过载，或者正在运行或处于其他情况；或指该刺激在系统的某些条件内发生
- 制品：被刺激的事物。可能是整个系统或系统的一部分
- 响应：刺激到达时所采取的行动
- 响应度量：当响应发生时，应该能够以某种方式进行度量，以便对需求进行测试，并明确质量属性需求

内容

- **2.1 软件质量属性概念**
- **2.2 设计时质量属性**
- 2.3 运行时质量属性
- 2.4 系统质量属性
- 2.5 用户质量属性
- 2.6 其他质量属性

概念完整性
（Conceptual Integrity）

- **Conceptual integrity** is the quality of a system where all the concepts and their relationships with each other are applied in a **consistent** way throughout the system.
- 设计应该表现出整体的协调、一致和可预测性，也就是概念完整性（Conceptual Integrity）。即，不论多少人参与了设计和实施，系统结构应该反映为统一的。

概念完整性

场景元素	可能的值	示例
刺激源	开发人员，设计人员，架构师	开发人员
刺激	重构、迁移、新成员、小组沟通	在实时定位系统中，对系统某一定位算法功能进行重构
制品	组件、代码、文档、系统	代码，设计文档
环境	在设计、开发、编译、部署或运行时	系统正常运行
响应	开发人员根据一致的概念重构代码；将系统迁移到新的平台运行；新成员遵循编码规范进行编码；建立协调和统一的沟通方式	移除原有算法，将重构后代码集成到系统中
响应度量	开发人员重构的时间；系统迁移所需工作量等	系统重构所修改的代码行数

概念完整性常见问题及解决方案

- 缺乏一致的开发过程。应该考虑引入应用生命周期管理（ALM）评估，以及使用经过验证和测试的开发工具和方法学。
- 软件生命周期中参与的不同小组之间缺乏协作和沟通。应该考虑创建统一开发过程并整合一些工具来推动处理流程、沟通和写作的协调统一。
- 缺乏设计和编码标准。应该考虑发布一套有关设计和编码规则的指导原则，以及为开发过程引入代码审查机制，以确保大家都遵循这些指导原则。
- 存在一些已有（遗留）的系统，使得重构和迁移到新平台或新规范难以进行。需要考虑创建遗留技术的升级之路，并将应用与外部依赖隔离。例如，使用外观设计模式来和遗留系统进行整合。

可维护性
（Maintainability）

- **Maintainability** is the ability of a system to **undergo changes** to its **components**, **services**, **features**, and **interfaces** as may be required when **adding** or **changing functionality**, **fixing bugs,and meeting new business requirements**.
- Maintainability can be measured in terms of **the time** it takes **to restore the system to its operational status** following a failure or removal from operation for upgrading. Improving system maintainability will increase **efficiency** and **reduce run-time defects**.

可维护性一般场景

场景元素	可能的值	示例
刺激源	系统内部、外部，设计人员，开发人员	开发人员
刺激	组件替换、系统测试、系统更新等	更新报表系统的界面
制品	组件、类、系统、使用文档等	组件
环境	设计、开发、编译、部署或运行时	系统正常运行
响应	开发人员替换系统组件；测试人员进行回归测试等	分离出原来的界面，并将新的界面集成到系统中
响应度量	替换系统组件所需的工作量；是否有系统使用文档；系统更新难易程度等	1人在1个工作日完成

可维护性常见问题及解决方案（1）

- 组件和层之间过度的依赖，以及对具体类不恰当的耦合都会增加替换、更新和修改的难度，并且使得修改具体类会同时牵连整个系统。可以考虑把系统设计为定义明确的层或关注点，以分离系统的UI、业务过程和数据访问功能。也可以考虑通过使用抽象（比如抽象类或接口），而不是具体的类作为跨层的依赖，并且最小化组件和层之间的依赖。
- 使用直接的通信方式，使得组件和层的物理部署进行改动变得困难。需要选择合适的通信模型、格式和协议。以及考虑通过设计一些使用插件模块或适配器的接口，实现插件式架构，这样可以很方便地进行升级和维护，并且也可以提高可测试性，以最大化灵活性和可扩展性。

可维护性常见问题及解决方案（2）

- 对于诸如身份验证和授权等特性，如果依赖自定义的实现，那么会阻碍重用性和可维护性。应该尽可能使用平台内置的功能和特性来避免这个问题。
- 组件和片段逻辑代码不够内聚，使得替换变得困难，并且造成了对其它组件不必要的依赖。需要将组件设计成“高内聚，低耦合”，这样可以最大化灵活性并且推动可替换性和重用性。
- 缺乏文档，会阻碍重用、管理以及将来的升级。应该确保提供了文档，至少也应该介绍应用程序的整体结构。

可重用性
（Reusability）

- **Reusability** is the probability that a component will **be used in other components or scenarios** to add new functionalities **with little or no change**. Reusability minimizes the duplication of components and also the implementation time. **Identifying the common attributes between various components** is the first step in building small reusable components of a larger system.

可重用性一般场景

场景元素	可能的值	示例
刺激源	系统，设计人员，开发人员	开发人员
刺激	使用可重用组件，调用接口等	使用系统中的数据访问模块
制品	代码，类，组件，子系统等	代码
环境	设计、开发、编译或部署时	设计时
响应	通过可重用组件搭建领域相关的新系统，通过调用接口实现系统的某一新功能等	通过调用封装好的数据库访问模块（DAO）实现对数据库的透明访问
响应度量	搭建系统所编写代码量；实现新功能所需代码量等	访问数据库编写的代码量

可重用性常见问题及解决方案

- 在多个地方使用不同的代码或组件实现相同的功能。要检查应用程序设计并确定公共功能，在独立的组件中实现这些功能以便重用。检查应用程序设计确定诸如验证、日志和身份验证等横切（cross）关注点，在独立组件中实现这些功能。
- 使用了多个相似的方法来实现相似的任务。应该使用一个方法，通过参数的变化来实现行为的变化。
- 使用了几个系统来实现相同特性或功能，而不是在另外一个系统中，或是跨多个系统、跨应用程序中的不同子系统来共享或重用功能。考虑通过服务接口从组件、层以及子系统暴露功能，供其他层和系统使用。考虑使用平台无关的数据类型和数据结构，这样不同的平台都可以访问和识别。

内容

- **2.1 软件质量属性概念**
- **2.2 设计时质量属性**
- **2.3 运行时质量属性**
- 2.4 系统质量属性
- 2.5 用户质量属性
- 2.6 其他质量属性

可用性 (Availability)

- **Availability** defines **the proportion of time that the system is functional and working**. It can be measured as a percentage of the total system downtime over a predefined period. Availability will be affected by system errors, infrastructure problems, malicious(恶意的) attacks, and system load.
- 定义了系统正常运转并发挥功能的时间比例
- 通过总的系统失败时间在预定义周期中的百分比来衡量。
- 系统错误、基础结构问题、恶意攻击及系统负荷都会影响可用性。

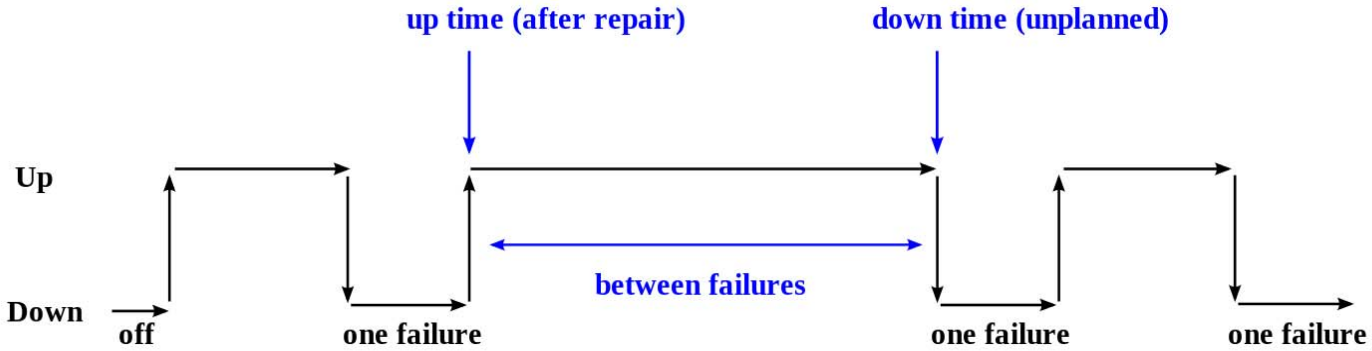
可用性

可用性与系统故障及其后果相关。可表示为：

$$\alpha = \frac{MTBF}{MTBF + MTTR}$$

MTBF: Mean Time Between Failure, 平均故障间隔时间

MTTR: Mean Time To Repair, 平均修复时间



Time Between Failures = { down time - up time}

$$\text{Mean time between failures} = \text{MTBF} = \frac{\sum(\text{start of downtime} - \text{start of uptime})}{\text{number of failures}}.$$

可用性（案例）

Availability	Downtime/90 Days	Downtime/Year
99.0%	21 hours, 36 minutes	3 days, 15.6 hours
99.9%	2 hours, 10 minutes	8 hours, 0 minutes, 46 seconds
99.99%	12 minutes, 58 seconds	52 minutes, 34 seconds
99.999%	1 minute, 18 seconds	5 minutes, 15 seconds
99.9999%	8 seconds	32 seconds

可用性涉及的关键问题

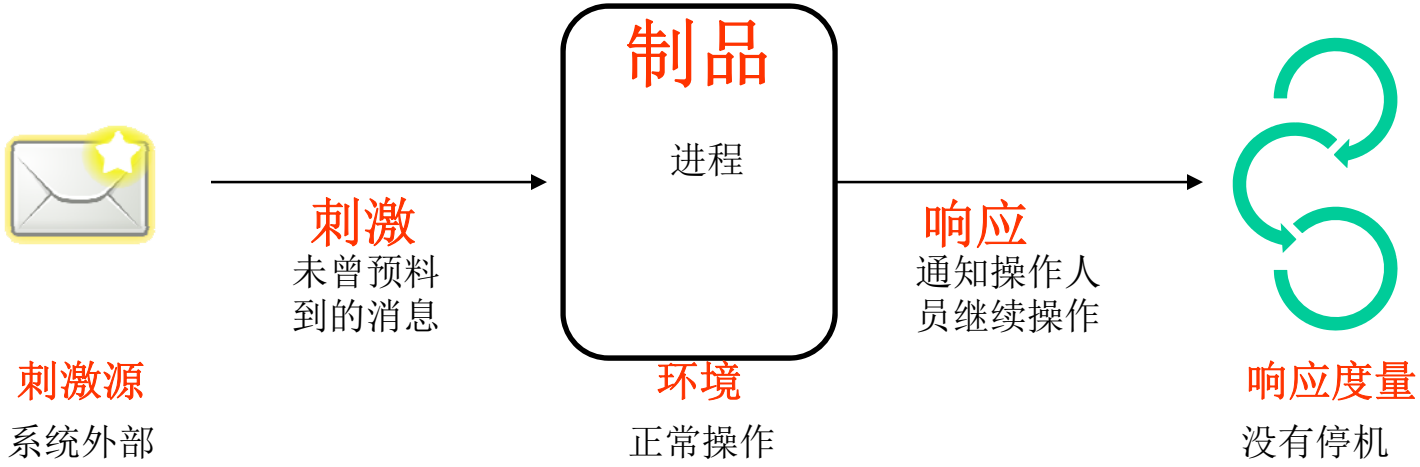
- 物理层宕机或不响应可能会引起整个系统宕机，例如数据库服务器或者应用服务器。需要考虑如何为系统物理层设计故障转移策略。
- 拒绝服务（Dos）攻击会阻止授权用户正常访问系统，如果因为需要处理时间不够或网络配置和网络阻塞等原因，系统不能及时处理大量请求，那么可能会中断操作。
- 不正确的使用资源会降低可用性。
- 应用程序中的bug或错误会导致系统出现异常。
- 频繁升级，比如添加安全补丁或对用户应用程序升级会降低系统可用性。
- 网络故障会导致应用程序不可用。

可用性一般场景

场景元素	可能的值	示例
源	系统内部/外部：人、软/硬件、物理环境	校园网和办公系统故障
刺激	遗漏、崩溃、不恰当的时间、不正确的响应	无法登录系统
制品	系统的处理器、通信通道、持久存储器、进程	通信通道
环境	正常操作，启动，关闭，修复模式，降级操作，过载操作	降级模式
响应	检测和隔离错误：记录错误，通知适当的人或系统； 从错误中恢复：禁止导致错误或故障的事件源；使得系统在指定时间内不可用；运行于降级模式（提供有限容量和功能）	通知网络中心工作人员，通过公告告知用户故障情况
响应度量	可用性百分比 故障检测时间，修复时间，系统在降级模式下运行的时间间隔，系统可用的时间间隔	80.00%？

可用性场景样例

“在正常操作期间，进程收到了一个未曾预料到的消息。该进程通知操作人员收到了这一消息，并继续操作(没有停机)”



可用性计算例题

一个系统一年（按 365 天计算）的运行情况如下

运行	宕机	运行时间	宕机时间
0	708.2	708.2	
708.2	711.7		3.5
711.7	754.1	42.4	
754.1	754.7		0.6
754.7	1867.5	1112.8	
1867.5	1887.4		19.9
1887.4	2336.8	449.4	
2336.8	2348.9		12.1
2348.9	4447.2	2098.3	
4447.2	4452		4.8
4452	4559.6	107.6	
4559.6	4561.1		1.5
4561.1	5443.9	882.8	
5443.9	5450.1		6.2
5450.1	5629.4	179.3	
5629.4	5658.1		28.7
5658.1	7108.7	1450.6	
7108.7	7116.5		7.8
7116.5	7375.2	258.7	
7375.2	7384.9		9.7
7384.9	7952.3	567.4	
7952.3	7967.5		15.2
7967.5	8315.3	347.8	
8315.3	8317.8		2.5
	Total＝	8205.3	112.5
	MTBM＝	683.8	
	MTTR		9.4

$$MTBF = \frac{8205.3}{12} = 683.775$$

$$MTTR = \frac{112.5}{12} = 9.4$$

$$\alpha = \frac{683.775}{683.775 + 9.4} = 98.6\%$$

互操作性
(Interoperability)

- **Interoperability** is the ability of diverse components of a system or different systems to operate successfully by **exchanging information**, often by using services. An interoperable system allows you to **exchange and reuse information** internally as well as externally.
- **Communication protocols**, **interfaces**, and **data formats** are the key considerations for interoperability. **Standardization** is also an important aspect to be considered when designing an interoperable system.

互操作性一般场景

场景元素	可能的值	示例
源	一个要向其他系统进行互操作请求的系统	国家气象局天气预报JSON接口
刺激	系统间交换信息的请求	请求天气预报信息
制品	接收请求的系统	目标系统
环境	运行前，或者运行中	运行中
响应	以下情况中的一个或多个： •请求被拒绝，通知适当的实体（人或系统） •请求被接受，信息成功交互 •请求被一个或多个相关系统记录	系统接受国家气象局天气预报接口返回的JSON数据，并在系统中显示
响应度量	以下情况中的一个或多个： 正确交换信息的百分比 正确拒绝信息的百分比	正确交换率99.99%

影响互操作性的问题

- 与使用不同数据格式的外部系统或遗留系统进行互操作。可使用编制和适配器模式来连接外部系统或遗留系统，并且在系统之间转换数据格式，或使用正则数据模型来处理大量异构数据格式的交互
- 边界模糊使得不同系统的环境相互融合。考虑如何使用服务接口和映射层来隔离系统。如使用基于XML或标准类型的接口来暴露服务以支持和其他系统的互操作
- 缺少对标准的支持。了解所从事领域已经有的正式标准，使用既有标准而非重新创建一份自己的标准。

可管理性（Manageability）

- Design your application to be easy to manage, by exposing sufficient and useful instrumentation（监测手段） for use in **monitoring systems** and for **debugging** and **performance tuning**
- 可管理性是指查看和修改指定系统或软件状态的能力，它描述了系统管理员管理应用程序的难易程度
- 可以通过暴露足够多有用的指示器，来监控系统和调试以及优化性能，让应用程序易于管理

可管理性的四个方面

- 系统**配置**：软件能够**方便的安装、部署、配置和集成**，并提供机制对整个过程进行有效的规划、监督和控制。
- 系统**优化**：通过**调整软件自身参数**、属性、行为以适应外界不同的计算环境和应用需求，使软件功能和效能得到最大的发挥。
- 系统**诊断**：软件出现故障或潜在隐患时，能够具备某种手段**对问题进行定位、报警**，并依照一定的策略在必要时采取措施补救。
- 系统**防护**：当软件运行要素被无意的破坏或被恶意的攻击时，软件应能够对其加以**识别和采取可能的应对方案**，并及时的**修复和恢复**。

分析某款软件，如Microsoft SQL Server 20XX 的可管理性如何？

可管理性一般场景

场景元素	可能的值	示例
源	系统管理员，以及系统自身	系统管理员
刺激	系统配置：安装、部署或者集成系统 系统优化：调整软件参数 系统诊断：对故障进行定位、报警 系统防护：恶意攻击或破坏	提高实时定位系统中位置更新频率
制品	目标系统	实时定位软件
环境	进行管理操作时，系统应该是在运行中	定位平台运行中
响应	改变受管理组件运行行为 从特定组件中捕获用于报告和通知的运行时事件和历史事件 对于故障能够及时修复和恢复	定位引擎向定位监控软件传输定位数据的频率提高
响应度量	管理操作花费的时间，成本等	在定位监控软件设置修改参数即可，2秒钟之内定位引擎提高传输频率

性能
（Performance）

- **Performance** is an indication of the **responsiveness**（响应能力） of a system to execute specific actions in a given time interval.
- It can be measured in terms of **latency(Response Time) or throughput**. Latency is the time taken to respond to any event. Throughput is the number of events that take place in a given amount of time. Factors affecting system performance include the demand for a specific action and the system's response to the demand.

性能的一般场景

场景元素	可能的值	示例
源	大量的独立源中的一个，可能来自系统内部	游戏玩家
刺激	定期事件到达；随机事件到达；偶然事件到达	通过敲击键盘，向服务器发送游戏指令
制品	目标系统	游戏服务器
环境	正常模式；超载模式	正常模式
响应	处理刺激；改变服务级别	进行数据处理，并转发到其他玩家
响应度量	延迟、期限、吞吐量、抖动、缺失率、数据丢失	延迟时间为32毫秒

影响性能的关键问题

- 客户端响应时间的增加，吞吐量的减少和服务器资源的过度使用。
- 内存消耗增加导致性能降低，大量缓存失效，数据源访问增加。因此要设计有效及正确的缓存策略。
- 数据库服务器处理的增加导致吞吐量下降。因此要确保选择有效率的事务类型、锁、线程和队列的实现方式。
- 网络带宽使用的增加导致响应时间延迟以及客户端和服务器负载的增加。
- 当仅需要一部分数据时却要从服务器获取全部数据，从而引起不必要的服务器负载。
- 不恰当的资源管理策略导致应用程序创建多重资源实例，进而增加响应时间。

可靠性（Reliability）

- **Reliability** is the ability of a system to **continue operating as expected over time**.
- Reliability is measured as the probability that a system will not fail and that it will perform its intended function for a specified time interval. Improving the reliability of a system may lead to a more secure system because it helps to prevent the types of failures that a malicious user may exploit.

可靠性函数：

$$R(t) = e^{-(t/\theta)}$$

θ可以是MTBF或MTTF(mean time to failure，平均失效前时间)

可靠性

- 软件缺陷->软件错误->软件故障
- 软件缺陷：常常又被叫做Bug
- 软件错误：软件缺陷在一定条件下暴露并导致系统在运行中出现可感知的不正常、不正确、不按规范执行的内部状态，则认为软件出现“错误”，简称出错
- 软件故障：系统输出不满足用户提供的正式文件或双方协议的条款上指明的要求。

内在原因：开发者失误　外在原因：环境异常+错误操作

可靠性计算例题

一个系统一年（按 365 天计算）的运行情况如下

运行	宕机	运行时间	宕机时间
0	708.2	708.2	
708.2	711.7		3.5
711.7	754.1	42.4	
754.1	754.7		0.6
754.7	1867.5	1112.8	
1867.5	1887.4		19.9
1887.4	2336.8	449.4	
2336.8	2348.9		12.1
2348.9	4447.2	2098.3	
4447.2	4452		4.8
4452	4559.6	107.6	
4559.6	4561.1		1.5
4561.1	5443.9	882.8	
5443.9	5450.1		6.2
5450.1	5629.4	179.3	
5629.4	5658.1		28.7
5658.1	7108.7	1450.6	
7108.7	7116.5		7.8
7116.5	7375.2	258.7	
7375.2	7384.9		9.7
7384.9	7952.3	567.4	
7952.3	7967.5		15.2
7967.5	8315.3	347.8	
8315.3	8317.8		2.5
	Total＝	8205.3	112.5
	MTBM＝	683.8	
	MTTR		9.4

$$t = 365 \times 24 = 8760$$

$$\theta = MTBF = 683.8$$

$$R(t) = e^{-(t/\Theta)} = e^{-(8760/683.8)} = 0.00027\%$$

Mean Time Between Maintenance (MTBM)
平均维护间隔时间

可用性 VS. 可靠性

- 可靠性通常低于可用性，因为可靠性要求系统在[0,t]的整个时间段内需正常（注意是“连续”！）运行；
- 可用性大于或等于可靠性，对于可用性，要求就没有那么高，系统可以发生故障，然后在时间段[0,t]内修复。修复以后，只要系统能够正常运行，它仍然计入系统的可用性。

可靠性的一般场景

场景元素	可能的值	示例
源	用户，系统	具有Wi-Fi指纹定位功能手机
刺激	发生软件故障，导致系统失效运行	无线信号变弱
制品	目标系统	定位系统
环境	正常运行，或降级运行	降级运行
响应	在出现系统故障时，有很多可能的反应，包括记录故障、通知选择的用户或其他系统、切换到降级模式（容量较小或功能较少）、关闭外部系统或在修复期间变得不可用。	定位精度降低或定位失败
响应度量	响应度量可以指定可靠性函数R(t)，失效率等来度量	MTBF、MTTF

影响可靠性的关键问题

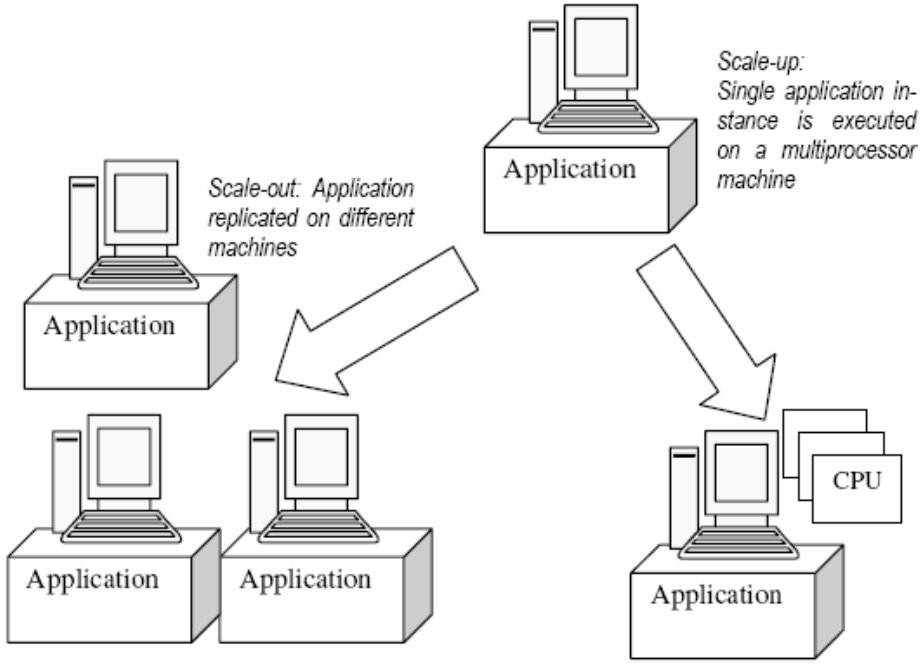
- 系统崩溃或无响应。要确定一些方式来检测故障并且自动触发故障转移，或重定向负载到后备系统。
- 输出不一致性。要实现诸如事件和性能计数器之类的指示器，来检测性能问题或是发送到外部系统的失败请求，然后通过诸如事件日志、Trace文件或WMI等标准系统来暴露信息。
- 诸如系统、网络和数据库等客观条件故障引起的系统故障。要找出一些办法来处理不可靠的系统、通信故障或是交易失败。

可伸缩性（Scalability）

- **Scalability** is an attribute of a system that displays the ability to function well even with change in demand. Typically, the system should be able to **handle increases in size or volume**.
- The aim is to maintain the system's availability, reliability, and performance even when the load（**Simultaneous connections**, **Data size**, and **Deployment**） increases.
- 可伸缩性是一种衡量软件系统适应系统规模（用户数量、数据量、网络节点）增长的能力，目标是在负载增加情况下维护系统的可用性、可靠性和性能。

可伸缩性（Scalability）

- There are two methods for improving scalability: **scaling vertically（垂直伸缩）**, and **scaling horizontally（水平伸缩）**. You add more resources such as CPU, memory, disk, etc. to a single system to scale vertically. You add more machines, for serving the application, to scale horizontally.



可伸缩性一般场景

场景元素	可能的值	示例
源	用户	用户
刺激	用户数量增加或者业务调整	用户数量增加
制品	软件系统	电子商务网站
环境	正常运行，超载运行	正常运行
响应	硬件方面，添加新的硬件设备，或提升硬件性能；软件方面，支持分布式的计算集群，支持添加新的计算设备	通过水平扩展的方式，增加新的服务器
响应度量	增加的计算/存储资源容量，扩展的节点/服务器数，系统的可用性、可靠性和性能	增加1台应用服务器，性能提升

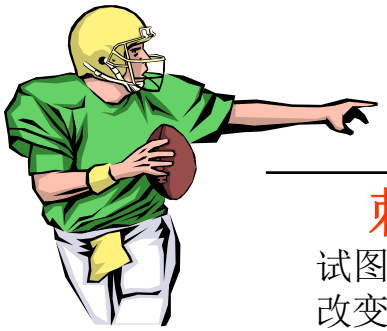
安全性
（Security）

- **Security** is an attribute of a system that needs to be protected from **disclosure**(泄密) or **loss of information**(信息丢失). Securing a system aims to **protect assets and unauthorized modification of information**(未授权的信息修改). The factors affecting system security are confidentiality（机密性）, integrity, and availability.
- **Authentication**（身份验证）, **encryption**（加密）, and **auditing**（审计）and **logging**（注册登录）are the features used for securing systems..

安全性（Security）

- 安全性指系统防止恶意行为或系统设计的使用方式之外行为，以及防止信息泄露和丢失的能力
- 目标是保护资产并防止未经授权对信息进行修改
- 提高安全性还可以增加系统的可靠性，因为安全性提高了，攻击成功的可能性以及攻击对系统造成的损害就降低了。
- 影响系统安全的因素是机密性、完整性和可用性，用于让系统更安全的特性包括身份验证、加密、审计和日志。
- 试图突破安全防线的行为被称为攻击，可以有很多形式
  - 未经授权试图访问数据或服务，或试图修改数据；
  - 试图使系统拒绝向合法用户提供服务

安全性一般场景



刺激源

正确识别、非正确识别或身份未知的个人或系统(来自内或外部)；经过了授权/未经授权而访问了有限/大量的资源

刺激

试图显示数据、改变/删除数据、访问系统服务、降低系统服务的可用性

制品

系统服务，系统中断的数据

环境

在线/离线、联网/断网、连接有防火墙/直接连接到网络

响应

对用户进行身份验证；隐藏用户身份；阻止/允许对数据和/或服务的访问；授权或收回对数据和/或服务的许可；

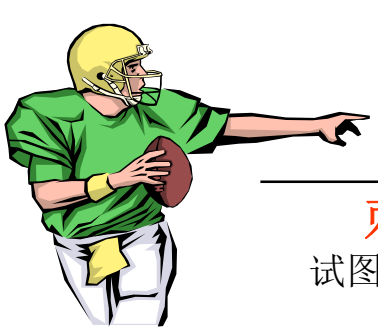
根据身份记录访问/修改或试图访问/修改数据或服务；以一种不可读的格式存储数据；识别无法解释的对服务的高需求；通知用户或另外一个系统并限制服务的可用性

响应度量

用成功的概率表示、避免安全防范措施所需要的时间/努力/资源；检测到攻击的可能性；确定攻击或访问/修改数据和/或服务的个人的可能性；在拒绝服务攻击的情况下仍然可以获得服务的百分比；恢复数据服务；被破坏的数据/服务和/或被拒绝的合法访问的范围

安全性场景样例

“一个经过身份验证的个人，试图从外部站点修改系统数据；系统维持了一个审核跟踪，并在一天内恢复了正确的数据”



刺激源

通过了身份验证的个人

刺激

试图修改信息

制品

系统中的数据

环境

在正常操作下

响应

系统保持审核跟踪

响应度量

在一天内恢复校正数据

影响安全性的关键问题

- 伪造用户身份。使用授权和身份验证来防止伪造用户身份。
- 由诸如**SQL注入以及跨站脚本等恶意输入**引起的危害。通过确保验证输入的长度、范围、格式以及类型，并且使用约束、拒绝以及过滤原则来防止产生此类危害。
- 数据篡改。把网站分成匿名用户、标识用户和经过身份验证的用户。使用应用程序指示器记录日志及暴露可监控的行为。
- 用户行为的否认。对于一些重要的应用程序行为。可以使用一些方式来审计和记录所有用户行为。
- 信息泄漏及敏感数据的丢失。需要让应用程序的各个方面都能防止敏感系统信息和敏感应用程序信息的被访问和暴露。
- 诸如拒绝服务（**DoS**）之类的攻击会导致服务中断。考虑减少会话超时间，以及通过代码或硬件来检测此类的恶意攻击。

内容

- **2.1 软件质量属性概念**
- **2.2 设计时质量属性**
- **2.3 运行时质量属性**
- **2.4 系统质量属性**
- 2.5 用户质量属性
- 2.6 其他质量属性

可支持性（Supportability）

- 可支持性是系统在不正常工作的情况下提供信息以确定和解决问题的能力。
- 涉及的关键问题包括：
  - 1.缺乏调试信息
  - 2.缺乏故障排查工具
  - 3.缺乏跟踪工具
  - 4.缺乏健康监控

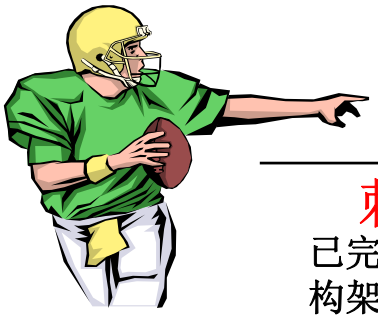
可支持性的通用场景

场景元素	可能的值	示例
源	系统内部	游戏软件
刺激	系统发生异常或故障，导致系统无法正常运行	出现内存泄露异常
制品	目标系统	游戏软件
环境	非正常运行	非正常运行
响应	提供系统调试信息，系统状态信息，以及其他的监控信息	弹出警告对话框，同时在日志文件中输出相关的异常信息
响应度量	提供支持信息与出现故障次数的比例	90%

可测试性（Testability）

- 软件可测试性是指通过测试（通常是基于运行的测试）揭示软件缺陷的容易程度。
- 可测试性衡量的是为系统及其组件创建测试标准，以及执行这些测试来确定是否满足标准的要求。

可测试性的一般场景



刺激源

单元开发人员；增量集成人员，系统验证人员，客户验收测试人员，系统用户

刺激

已完成的分析、构架、设计、类和子系统集成；所交付的系统

制品

设计过程，一段代码或完整的应用

环境

设计、开发、编译或部署时

响应

提供对状态值的访问、提供所计算的值、准备测试环境

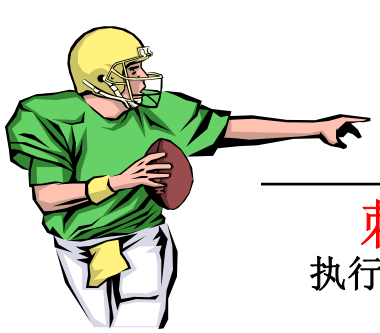
响应度量

已执行的可测试语句的百分比；如果存在缺陷出现故障的概率；执行测试的时间；测试中最长依赖链的长度；准备测试环境的时间

可测试性的一般场景的可能取值

可测试性场景样例

“单元测试人员在一个已完成的系统构件上执行单元测试，该构件为控制其行为和观察其输出提供了一个接口；在3小时内测试了85%的路径”



刺激源

单元测试人员

刺激

执行单元测试

制品

系统的构件

环境

在构件完成时

响应

构件具有控制行为的接口，并且构件的输出是可观察的

响应度量

在3小时内测试了85%的路径

内容

- 2.1 软件质量属性概念
- 2.2 设计时质量属性
- 2.3 运行时质量属性
- 2.4 系统质量属性
- **2.5 用户质量属性**
- 2.6 其他质量属性

易用性（Usability）

- 易用性是一种有关用户体验（User Experience）质量的描述
- 易用性关注的是对用户来说完成某个期望任务的容易程度和系统所提供的用户支持的种类。
- 易用性是把用户作为开发过程的中心，即“以用户为中心进行设计”
- 易用性包含：
  - 易理解性：文档、功能名称、图标、提示信息等
  - 易学习性：文档详尽、操作一目了然
  - 易操作性：人机界面友好、界面设计科学合理以及操作简单等

易用性一般场景

刺激源

最终用户

刺激

想要学习的系统特性、有效使用系统、使错误的影响最低、系统适应用户需求、对系统满意

制品

系统

环境

在运行时或配置时

响应

系统提供一个或多个响应来支持“学习系统特性”：帮助系统与环境联系紧密；界面为用户所熟悉；在不熟悉的环境下界面是可以使用

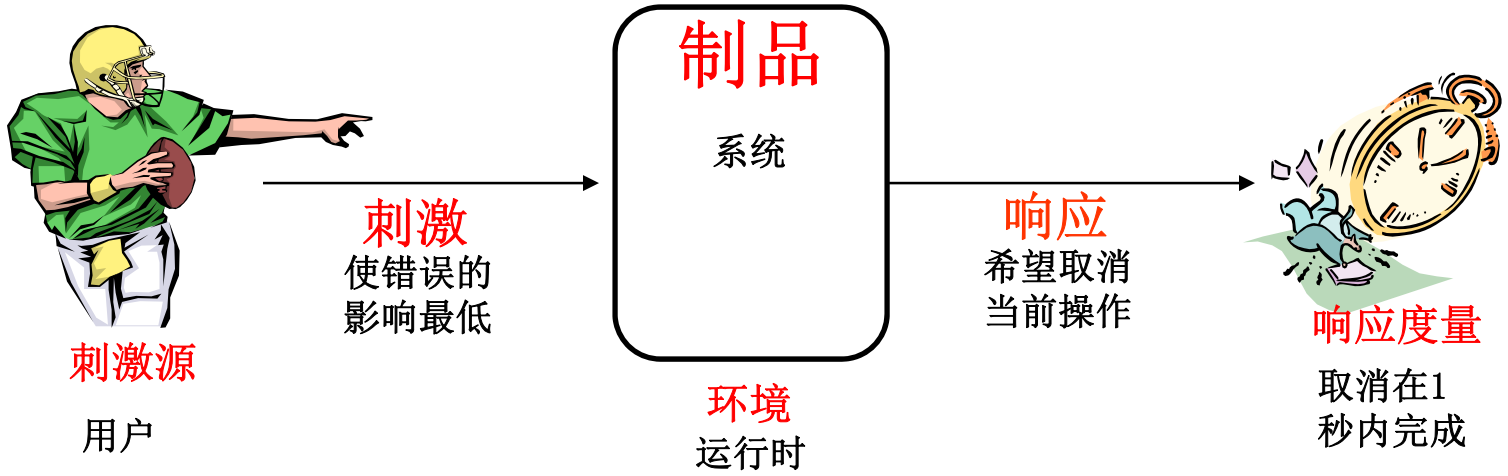
响应度量

任务时间、错误数量、解决问题数量、用户满意度、用户知识的获取、成功操作在总操作中的比例、损失的时间/丢失的数据量

易用性一般场景的可能取值

易用性场景样例

“想把错误的影响降到最低的用户，希望在运行时取消用户的操作，取消在1秒内发生”



影响易用性的关键问题

- 进行某项任务需要太多的交互（大量鼠标点击）。（确保设计了必要的屏幕和输入流程及用户交互模式来最大化易用性）
- 不正确的多步界面流程。（考虑尽可能使用工作流来简化多步操作）
- 数据元素和控件分组不合适。（选择合适的控件类型，使用被大家接收的UI设计模式）
- 没有提供良好的用户反馈，特别是应用程序在遇到错误和异常后不能响应。

服务设计（Service Design）——用户体验设计的新趋势

- 服务设计是有效的计划和组织一项服务中所涉及的人、基础设施、通信交流以及物料等相关因素，从而提高用户体验和服务质量的设计活动
- 关键是“用户为先＋追踪体验流程＋涉及所有接触点＋致力于打造完美的用户体验”

接触点指产品终端客户接触到的产品相关特性

参考资料：
http://baike.baidu.com/link?url=mZufW8DIyEGvy4lbbxS__cBqDodY_T-fdRb5YfAqr9FEVNy4fV7wC_a-Cwk1z4SAaPzy1PgrWFa6XY7SuSMyIq
https://en.wikipedia.org/wiki/Service_design

内容

- 2.1 软件质量属性概念
- 2.2 设计时质量属性
- 2.3 运行时质量属性
- 2.4 系统质量属性
- 2.5 用户质量属性
- 2.6 其他质量属性（自学）

作 业

- 1.软件可管理性主要表现在哪些方面，并以一款软件为例分析其可管理性如何。
- 2.系统非功能需求是什么含义？与其对应质量属性有哪些？
- 3.功能需求、质量属性需求、约束分别对软件架构产生哪些影响？
- 4.为提高系统运行时质量属性如可用性、性能、可伸缩性，您认为可采取哪些设计策略？
- 5.对于您曾经开发或目前正在开发的系统来说，最重要的质量属性需求是什么？试着使用质量属性场景描述这些需求（给出图、表中的一种）。