来自林连南的第二次作业题目布置的回答情况：

答案均来自各位的回答，希望大家见怪不怪

## 第四章：

|  |  |
| --- | --- |
| **（1）原文** | What is the relationship between a use case and a quality attribute scenario? If you wanted to add quality attribute information to a use case, how would you do it? |
| **（2）翻译** | 用例和质量属性场景之间有什么关系？ 如果要将质量属性信息添加到用例中，您将如何进行？ |
| **（3）解答** | 用例和质量属性场景：相同事情的不同方法  在软件工程中，用例描述用户想要实现的功能，而质量属性方案包含功能在设计的应用程序中如何运行的描述。功能和质量属性是正交的，如果要将质量属性信息添加到用例，那就添加新的用例 |

|  |  |
| --- | --- |
| **（1）原文** | Do you suppose that the set of tactics for a quality attribute is finite or infinite? Why? |
| **（2）翻译** | 你认为质量属性的一套策略是有限的还是无限的？为什么呢？ |
| **（3）解答** | 经过我们讨论，是无限的，策略的重点是单一质量属性响应。在一种策略中，没有考虑权衡。必须由设计者明确考虑和控制权衡。策略将重叠，经常可以选择多种策略来改善特定的质量属性。选择使用哪种策略取决于诸如其他质量属性之间的权衡和实施成本等因素。这些考虑因素超越了对特定质量属性的策略的讨论。我们提出的策略可以而且应该得到改进。考虑性能：计划资源是一种常见的性能策略。但是，为了特定目的，需要将此策略细化为特定的调度策略，例如最短作业优先级，循环法等。使用中介是一种可修改的策略。但是有多种类型的中介（层，代理和代理，仅举几例）。因此，架构师将采用改进来制作每种具体的策略。 |

|  |  |
| --- | --- |
| **（1）原文** | Consider the choice between synchronous and asynchronius comunication(a choice in the coordination mechanism category) .What quality attribute requirement might lead you to choose one over the other? |
| **（2）翻译** | 请考虑在同步和异步通信机制之间进行选择（在协调机制下选择）。是什么质量属性要求导致你选择这一个而不是另一个？ |
| **（3）解答** | 同步通信效率更高，更加适合对速度要求高的传输，在性能的质量属性要求下我会选择同步通信；异步通信点对点，对时序要求低，在易用性质量属性要求下我选择异步通信。 |

|  |  |
| --- | --- |
| **（1）原文** | Consider the choice between statfeful and stateless comunication(a choice in the coordination mechanism category) .What quality attribute requirement might lead you to choose one over the other? |
| **（2）翻译** | 请考虑在有状态和无状态通信机制之间进行选择（在协调机制下选择）。是什么质量属性要求导致你选择这一个而不是另一个？ |
| **（3）解答** | 有状态通信无需额外的调用，低延迟的优点，在性能和可用性的质量属性要求下我会选择有状态通信；无状态通信有防止数据丢失的优点，在安全性的质量属性要求下我会选择无状态通信。 |

|  |  |
| --- | --- |
| **（1）原文** | Think about the screens that your favorite automatic teller machine uses.What do those screens tell you about binding time decisions reflected in the architechture? |
| **（2）翻译** | 想想你最喜欢的自动柜员机使用的屏幕。这些屏幕所说的绑定时间决策如何反映在架构上？ |
| **（3）解答** | 通过推迟绑定时间，来使设计者（银行维护者）在维护方面来做到即使用户做出了修改，也不会影响到数据的改变。在屏幕上设置注册界面，使得用户有插入银行卡类似的注册动作。 |

|  |  |
| --- | --- |
| **（1）原文** | Most peer-to-peer atchitechture employs late binding of the topology.What quality attributes does this promote or inhibit? |
| **（2）翻译** | 大多数p2p架构使用后期绑定的拓扑，什么质量属性促进还是抑制 ？ |
| **（3）解答** | 对等网络，即对等计算机网络，是一种在对等者之间分配任务和工作负载的分布式应用架构，是对等计算模型在应用层形成的一种组网或网络形式。  其可以定义为：网络的参与者共享他们所拥有的一部分硬件资源（处理能力、存储能力、网络连接能力、打印机等），  这些共享资源通过网络提供服务和内容，能被其它对等节点直接访问而无需经过中间实体。  在此网络中的参与者既是资源、服务和内容的提供者，又是资源、服务和内容的获取者，也就是同一资源客户越多，共享就越快。 |

|  |  |
| --- | --- |
| **（1）原文** | Discuss the choice of programming language (an example of choice of technology) and its relation to architecture in general, and the design decisions in the other six categories? For instance, how can certain programming languages enable or inhibit the choice of particular coordination models? |
| **（2）翻译** | 讨论编程语言的选择（例如选择一门技术）及其与架构的大体关系，以及其他六个类别的设计决策？ 例如，某些编程语言如何允许或禁止选择特定的协作模型？ |
| **（3）解答** | 一个好的软件架构能够满足系统的品质，能够支持计划编制过程，对系统开发的指导性，能够有效的管理复杂性，为复用奠定了基础。绝大多数架构或者编程语言的产生都是来源于项目。比如php言语属于开源言语，利于学习，运用普遍。只能用来开发网站，无法用来开发软件或其他的一些应用，局限性比拟大。普通运用php开发的企业网站，通常会运用网上现成的开源CMS来搭建程序。 |

|  |  |
| --- | --- |
| **（1）原文** | We will be using the automatic teller machine as an example throughout the chapters on quality attributes. Enumerate the set of responsibilities that an automatic teller machine should support and propose an initial design to accommodate that set of responsibilities. Justify your proposal. |
| **（2）翻译** | 我们将在整个章节中使用ATM作为质量属性的示例。 列举自动柜员机应支持的一系列功能，并提出初始设计以适应这组功能。 证明你的提议是正确的。 |
| **（3）解答** | 刺激源 用户  刺激 办理业务  制品 系统  环境 正常运行  响应 请求被处理  响应度量 响应时间不超过5秒  4.安全性  刺激源 非授权用户  刺激 试图非法操作  制品 系统中的数据  环境 正常操作下  响应 对用户验证，阻止操作  响应度量 操作被拒绝  5.可测试性  刺激源 单元开发人员  刺激 已完成构架和子系统的集成  制品 代码段  环境 开发时  响应 准备集成环境  响应度量 执行测试的时间  6.易用性  刺激源 最终用户  刺激 办理业务  制品 系统  环境 在运行时  响应 根据操作处理相应业务  响应度量 95%的用户对结果满意 |

## 第五章

|  |  |
| --- | --- |
| **（1）原文** | 1. Write a set of concrete scenarios for availability using each of the possible responses in the general scenario.  2. Write a concrete availability scenario for the software for a (hypothetical) pilotless passenger aircraft. |
| **（2）翻译** | 1. 可用性的通用场景下有多个可能的回应，为每个回应写一组概念场景。  2. 为 (假设的) 无人驾驶客机编写具体的软件可用性场景。 |
| **（3）解答** | 1. A. Log the fault：把错误记录到日志里。比如硬盘检测到坏块时会把相关信息记录到日志里。  B. Notify appropriate entities (people or systems)：通知恰当的实体（人或系统）。如编译失败时编译器显示错误信息。  C. Disable source of events causing the fault：使导致错误的事件源失效。如过滤不正确的输入。  D. Be temporarily unavailable while repair is being effected：修复时暂停服务。操作系统重启时修复系统错误。  E. Fix or mask the fault/failure or contain the damage it causes：修复或屏蔽错误，或者控制损失。比如屏蔽发生错误的硬件。  F. Operate in a degraded mode while repair is being effected：修复时在降级模式下操作。如错误率过高时无线网卡降低传输速率。  2. A. Log the fault：把错误记录到日志里。如把速度传感器的错误信息记录到黑匣子中。  B. Notify appropriate entities (people or systems)：通知恰当的实体（人或系统）。如油料不足时通知地面控制中心。  C. Disable source of events causing the fault：使导致错误的事件源失效。如定时除冰使速度传感器不被冰覆盖。  D. Be temporarily unavailable while repair is being effected：修复时暂停服务。如发生机械故障时自动迫降。  E. Fix or mask the fault/failure or contain the damage it causes：修复或屏蔽错误，或者控制损失。如电路故障时停止照明。  F. Operate in a degraded mode while repair is being effected：修复时在降级模式下操作。如机舱泄露时到低空飞行，就近迫降。 |

|  |  |
| --- | --- |
| **（1）原文** | 3. Write a concrete availability scenario for a program like Microsoft Word.  4. Redundancy is often cited as a key strategy for achieving high availability. Look at the tactics presented in this chapter and decide how many of them exploit some form of redundancy and how many do not. |
| **（2）翻译** | 3.为类似于Microsoft Word的程序编写具体的可用性场景  4. 冗余通常被认为是实现高可用性的关键策略。 看看本章介绍的策略，并决定其中有多少采用某种形式的冗余，有多少则没有。 |
| **（3）解答** | 3.A. Log the fault：把错误记录到日志里。记录使用word时可能产生的错误信息（无法打开文档的问题，格式转换的问题，图片显示问题，单词拼写问题，语法问题等）。  B. Notify appropriate entities (people or systems)：通知恰当的实体（人或系统）。如无法打开文档的问题会通知给开发者，单词拼写问题、语法问题会显示在屏幕上通知给使用者。  C. Disable source of events causing the fault：使导致错误的事件源失效。如单词拼写问题会用红线标注给与警示。  D. Be temporarily unavailable while repair is being effected：修复时暂停服务。如无法打开文档则会提醒错误并停止尝试打开。  E. Fix or mask the fault/failure or contain the damage it causes：修复或屏蔽错误，或者控制损失。如无法转换格式则会自动保存原格式。  F. Operate in a degraded mode while repair is being effected：修复时在降级模式下操作。如无法显示图片则会预留图片空间，待使用者解决。  4. 在Detect Faults（Ping/Echo, Monitor, Heartbeat, Timestamp, Sanity Checking, Condition Monitoring, Voting, Exception Detection, Self-Test）和Prevent Faults (Removal from Service, Transactions, Predictive Model, Exception Prevention, Increase Competence Set)的过程中没有采用冗余.  在Recover from Faults的过程中：  Active redundancy: 采取主动冗余  Passive redundancy: 采取被动冗余  Spare: 采取被动冗余  Exception Handing: 没有采用冗余  Rollback: 采取主动冗余和被动冗余结合  Software Upgrade: 采取主动冗余或者被动冗余二者都可  Retry: 没有采用冗余  Ignore Faulty Behavior: 没有采用冗余  Degradation: 没有采用冗余  Reconfiguration: 没有采用冗余  Shadow: 没有采用冗余  State Resynchronization: 没有采用冗余,但是工作的时候和主动冗余以及被动冗余合作  Escalating Restart: 采用了被动冗余(Level 0)  Non-Stop Forwarding (NSF): 采取主动冗余 |

|  |  |
| --- | --- |
| **（1）原文** | 5.How does availability trade off against modifiability ? How would you make a change to a system that is required to have “24/7” availability (no scheduled or unscheduled downtime , ever)? |
| **（2）翻译** | 5.可用性与可修改性如何进行权衡？当需要具有“24/7”可用性时应该如何更改系统（没有计划或非计划的停机时间）？ |
| **（3）解答** | 可用性指标是架构设计的重要指标，对外是服务承诺，对内是考核指标。将系统分层、分割的主要目的是增加系统可维护性，以及对后期发展提供更好的功能扩展能力、并发处理能力。在进行权衡时需要根据具体的使用场景以及需求来决定，比如在项目会有多次进行迭代的情况，而产品在迭代的过程中本身会暂时影响了正常使用，初期阶段重视可修改性会对项目进程有很好的帮助。而当项目趋向于稳定时更应该重视可用性以便提供更好的客户体验。  对于需要’24/7’可用的情况可以采用镜像的方式来确保业务的连续性。 |

|  |  |
| --- | --- |
| **（1）原文** | 6. Create a fault tree for an automatic teller machine. Include faults dealing with hardware component failure, communications failure, software failure, running out of supplies, user errors, and security attacks. How would you modify your automatic teller machine design to accommodate these faults? |
| **（2）翻译** | 6. 为自动出纳机构造故障树。包括处理硬件组件故障，通信故障，软件故障，资源贫乏，用户错误和安全攻击等故障。 您如何修改自动出纳机设计以适应这些故障？ |
| **（3）解答** | F:\最后的TIPS\1213246931\Image\C2C\AA10A7B3BF27CA0893E812B990EA41C3.jpg  在自动出纳机程序编写的时候增加相应的查错机制，同时删除一些长久不需要存储的、无关紧要的数据，对自动出纳机进行定期维修，检查软硬件使用情况。 |

|  |  |
| --- | --- |
| **（1）原文** | 7.Consider the fault detection tactics(ping/echo, heartbeat , system monitor, voting, and exception detection). What are the performance implications of using these tactics？ |
| **（2）翻译** | 7.考虑故障检测策略（响应机制，心跳机制，系统监控，投票和异常检测）。 使用这些策略有什么性能影响？ |
| **（3）解答** | Ping/echo（响应）  通过系统监视器发出Ping请求来探测服务的通信路径是不是通的，某个组件是不是还在工作。  监视(Monitor)  通过使用一个监视器的组件来检测其他组件的健康状况。  心跳(Heartbeat)  通过组建定区发出一个心跳信息，来主动的告诉系统它正在工作。  投票(Voting)  使用冗余的组件做同一件事情，以相同的输入，如果产生不同的输出，则忽略少数，采纳多数的结果。  性能：性能是指软件系统及时提供相应服务的能力。包括速度、吞吐量和持续高速性三方面的要求。  以上的可用性策略都需发出请求并通过返回的信息判断组件的可用性，在一定程度上会增加系统负载，提高系统响应时间，  但也提高了服务可用时间段，例如，一年内90%可用性的系统最多有36.5天的停机时间，使用可用性策略，可使系统可用性达到95%或以上  使用可用性策略或多或少都会影响系统的响应速度，即使它可以增加服务时间，你需要决定性能或可用性在当前环境下哪个更关键一点 |

## 第六章

|  |  |
| --- | --- |
| **（1）原文** | Find a web service mashup. Write several concrete interoperability scenarios for this system. |
| **（2）翻译** | 找一个web 服务的复合应用程序实例，并写出几个这个系统的具体的互操作场景。 |
| **（3）解答** | 使用外部系统登录   |  |  | | --- | --- | | 互操作性 | | | 刺激源（Source of stimulus） | 用户 | | 刺激（Stimulus） | 使用外部系统如QQ、微信等方式登录本系统 | | 制品（Artifact） | 使用外部系统登录后的系统 | | 环境（Environment） | 正常操作下 | | 响应（Response） | 登录成功 | | 响应度量（Response measure） | 99.9%登陆成功 |   导入facebook好友   |  |  | | --- | --- | | 互操作性 | | | 刺激源（Source of stimulus） | 用户 | | 刺激（Stimulus） | 点击导入facebook好友按钮 | | 制品（Artifact） | 显示好友照片、状态信息的页面 | | 环境（Environment） | 正常操作下 | | 响应（Response） | 导入成功 | | 响应度量（Response measure） | 99.9%导入成功 |   添加新浪微博数据源   |  |  | | --- | --- | | 互操作性 | | | 刺激源（Source of stimulus） | 用户 | | 刺激（Stimulus） | 添加第三方账户（如新浪微博账号等） | | 制品（Artifact） | 可使用新浪微博数据源的系统 | | 环境（Environment） | 网络连接正常下 | | 响应（Response） | 添加成功 | | 响应度量（Response measure） | 99.9%添加成功 | |
|  |  |

|  |  |
| --- | --- |
| **（1）原文** | What is the relationship between interoperability and the other quality attributes highlighted in this book? For example, if two systems fail to exchange information properly, could a security flaw result? What other quality attributes seem strongly related (at least potentially) to interoperability? |
| **（2）翻译** | 互操作性和这本书里面强调的其他的质量属性的关系怎么样？例如，如果两个系统不能正常地交换信息，可能会导致一个安全瑕疵（隐患）吗？哪些别的质量属性看上去和互操作性有紧密的关系（至少有可能是）？ |
| **（3）解答** | 互操作性指的是系统内或者系统之间不同的组件可以有效地进行信息交换，通常是以服务（Service）的形式来进行的。互操作性的关键因素包括通信协议，接口定义，数据格式的定义等等，而标准化是实现互操作性的重要手段。互操作性与其他质量属性关系如下：   1. 可用性Availability   互操作性对可用性具有良好的保证，因为对于可用性来说，用户对于操作可能会出现各种的问题，而互操作性会在一些服务场景上降低错误的发生，在很大的程度上增加了补救措施的种类。   1. 可修改性Modifiability   在使用具有互操作性强度高的系统时，系统的可修改性会提高很多。在进行本系统的更新时，系统自身的可修改性会因互操作性的提高而降低开发成本。   1. 性能Performance   由于互操作性的介入，系统的性能可能会随着功能的完善，导致性能的提高。由于互操作性的介入会使很多板块的灵活性增加，届时使得整个系统更加具备高内聚、低耦合的特征，同时使得整个系统可以更加有效的得到优化，提高运行效率。   1. 安全性Security   安全性可能会降低，比如一个论坛如果安全性较低，那么和他具有良好互操作性的一些外部系统的安全性将会受到极大的不确定性保证。   1. 可测试性Testability   互操作性较高的情况下，可测试性将会降低。介于安全性的考虑因素，某些外部系统对于测试会有高度敏感性。因此具有良好互操作性的系统在可测试性上会具有相对不好的表现。   1. 易用性Usability   在一定应用场景下，高度互操作性的软件会使得软件的易用性提高。具备不同的系统之间的交互，在一定情况下会降低用户的学习成本和使用成本，因为用户可以借由其他系统的信息进行合适的了解性操作。例如我们使用QQ登录一些论坛时，就降低了我们的注册成本或者是输入账号密码的登录成本。  如果两个系统不能正常的交换信息，可能会是一个安全隐患导致的。安全性，可用性，可修改性，看起来和互操作性关系很紧密。 |

|  |  |
| --- | --- |
| **（1）原文** | Is a service-oriented system a system of systems? If so, describe a service-oriented system that is directed, one that is acknowledged, one that is collaborative, and one that is virtual. |
| **（2）翻译** | 一个面向服务的系统是系统的系统吗？如果是这样，描述面向服务的系统，一个有指导的系统，一个受认可的系统，一个协作的系统和一个虚拟的系统。 |
| **（3）解答** | 面向服务的系统都是SoS，系统的系统SoS是指一组去实现相同目标的系统的总集合。  Directed和Acknowledged这两类SoS是在很明显地去尝试去创建一个SoS。两者的关键区别在于前者更加具有SoS划分等级的一个管理，而后者则是保持有高度的自治性。  对于Collaborative和Virtual这两类，对于Collaborative来说，每个SoS可能有着自己的管理和资金权限，而没有对每个SoS的总体管理。Virtual包含更大的系统，而且更加具备一种临时性，而对于整个系统则没有管理授权。  Directed：未来战斗系统Future Combat Systems，FCS 共有十四个单独的子系统，包括:无人照看的地面传感器 (Unattended Ground Sensors, UGS)，超视距发射系统 (Non-Line-of-Sight launch System, NLOS-LS)，两款分别与每排和每旅士兵有机结合的无人空中载具 (Unmanned Aerial Vehicle, UAV)，两款无人地面载具的变种:小型无人地面载具 (Small Unmanned Ground Vehicle, SUGV) 和多功能功用/后勤和装备载具 (Multifunctional Utility Logistics Equipment, MULE)，八种有人地面载具 (Manned Ground Vehicle, MGV)，外加联系所有这些子系统的网络 (14+1)，以及士兵本身 (14+1+1)。  Acknowledged：学校，具有一定的隶属关系。但是并不是完全的集权管制。例如，华南理工大学共可以分为4大块，学校党政职能部门，学院，直属单位，附属医院。其中学校党政职能部门下面可以分为32个小部门，而学院共有29所，直属单位有16个，附属医院3所。  Collaborative：Internet。各个互联网内部的系统具有相对松散自由的协作性关系，而不会因此会有过大的约束力。  Virtual：美国超过3000所的电子公司。美国每个州都有一个公用事业委员会，来去监督公用事业公司在自己州的运转，并且联邦能源部提供一些政策指导。在这个内部的许多系统必须互操作，但是整个系统没有一个主管理权限。 |

|  |  |
| --- | --- |
| **（1）原文** | Universal description, Discovery, and Integration(UDDI) was touted as a discovery service, but commercial support for UDDI is being withdrawn. Why do you suppose this is? Does it have anything to do with the quality attributes delivered or not delivered by UDDI solutions? |
| **（2）翻译** | UDDI（描述、发现、集成）技术被视为一个发现性服务，但是对UDDI的商业支持正在被撤回。你认为为什么会这样？它有什么处理这个由UDDI传递与否的质量属性的方法吗？ |
| **（3）解答** | UDDI被吹捧为一项探索服务，但是他得到的商业支持正在减少。你认为这是为什么？有什么办法能够处理UDDI解决方案所实现的质量属性吗？  UDDI是  （1）一个广泛的，开放的行业计划，它使得商业实体能够彼此发现，定义他们怎样在internet上互相作用，并在一个全球的注册体系架构中共享信息。  （2）一种基础的系统构筑模块，他使商业实体能够快速，方便地使用他们自身的企业应用软件来发现合适的商业对等实体，并与其实施电子化的商业贸易。  （3）Web服务集成的一个体系框架。它包含了服务描述与发现的标准规范。  由于UDDI本身的一些缺陷，现在IBM、HP等开始推荐自己的标准，如IBM的 WSRR（IBM WebSphere Service registry/repository )，但是同时声明：WSRR包括同步结构，将能够支持WSRR与分开的UDDI注册表之间合理的共存,即WSRR是兼容UDDI的。目前投递到UDDI注册服务的大部分内容在格式上都是很糟糕的，使这种注册服务目前的有效性大打折扣。可能需要为UDDI的使用制定更多的标准并不断进行更新。 |

|  |  |
| --- | --- |
| **（1）原文** | Why has the importance of orchestration grown in recent years? |
| **（2）翻译** | 为什么orchestration的重要性近些年在增强？ |
| **（3）解答** | 随着Web服务技术的日益成熟，Web Service已成为一种将大规模集成客户，供应商和业务合作伙伴应用程序的实用方法。 为了使Web服务构建的投资更有利可图，需要采用标准的开放式集成方法。 Orchestration流程编制使用一个可执行的中心流程来协同内部及外部的Web Service交互。通过中心流程来控制总体的目标，涉及的操作，服务调用顺序。这种集中化管理使Web服务能够在不了解彼此影响的情况下进行添加和删除，还允许在出现错误和异常的情况下进行补偿。其结果可以看作一个新的Web Service，可以执行，只是执行的过程需要调用别的Web服务。  Orchestration虽然是采用统一控制的方式，但是编制的状态为由里向外的方式来反映，视角集中在具体的参与者的活动。Orchestration让我们可以更好的理解每个个体需要完成的任务。其对于建立动态的，可扩展的流程是非常重要的。因此，作为企业连接的标准解决方案，orchestration近年来变得越来越重要。 |

|  |  |
| --- | --- |
| **（1）原文** | If you are a technology producer, what are the advantages and disadvantages of adhering to interoperability standards? Why would a producer not adhere to a standard? |
| **（2）翻译** | 如果你是一个技术生产者，坚持互操作性准则的优缺点有哪些？为什么生产者不会遵守一个准则？ |
| **（3）解答** | 优点:提供已有功能给更多应用，获取其他功能，更好的可用性，可修改性  缺点:降低产品竞争力，快速发展的技术打破操作闭环，安全性降低  理由:产品独立性，保持闭环，避免竞争，难以平衡多个准则 |

|  |  |
| --- | --- |
| **（1）原文** | With what other system will an automatic teller machine need to interoperate? How would you change your automatic teller system design to accommodate these other system? |
| **（2）翻译** | 一个ATM需要与什么其他系统交互？你会怎样改变你的ATM系统设计来适应其他系统？ |
| **（3）解答** | 互操作性指的是这样的一种能力,使得分布的控制系统设备通过相关信息的数字交换,能够协调工作,从而达到一个共同的目标。对于自动柜员机，其互操作性将更注重安全性，主要的应用场景是控制中心、用户管理系统与子柜员机之间的互操作。 |

另外一组:

|  |  |
| --- | --- |
| **（1）原文** | 1. Find a web service mashup. Write several concrete interoperability scenarios for this system. |
| **（2）翻译** | 列举一种混搭式的万维网服务，并围绕这一系统写几个具体的应用了互操作性的场景。 |
| **（3）解答** | 通过上网搜索，我找到一种叫做“web混搭平台”的基于信息聚合技术的解决方案。这一解决方案能完整的实现服务生命周期的各个环节：服务建模及开发、服务发现、服务部署、服务管理与监控等。它的体系结构分应用层、中间层、基础层、硬件层四层，其功能结构分信息接入功能模块和聚合信息展现模块两个模块，并包括四个平台：管理平台、开发平台、运行平台和安全策略平台。  互操作性的战术分locate、manage interfaces 两个种类，locate战术只有一种策略，即discover service。Web混搭平台具有应用推荐功能，在用户编辑应用的时候，可以根据用户的编辑目的、编辑过程推荐相类似的Mashup应用，加快Mashup应用提供者的开发速度，同理还可以推荐数据源，这两个功能可以通过基于语义的服务检索功能实现，这就是课本上所说的“Locate a service through searching a known directory service”，亦即“Discover service”，一个应用了Locate（定位）的应用场景。  Manage interfaces战术下有两种策略：orchestrate 和tailor interface。  Web混搭平台具有应用推荐管理功能，负责用户相关数据的维护，同时还配置（协调）和管理聚合服务的推荐规则，使之有序化，因为这些预先定义的规则是用来设定推荐Mashup应用的策略；此外，还有应用生命周期管理功能，管理聚合应用的整个生命周期过程，包括应用的部署、激活、停止、删除以及对聚合应用可见性和可用性的控制，这有点类似于课本上“workflow engines”的例子，都是用了某种控制机制来协调、管理、有序化那些特定服务的调用，都是用于系统为了完成一项复杂的任务而用一种复杂的方式来进行交互的时候。简单地说，就是“给操作加下标（orchestration scripts the interaction）”。这是应用了orchestration的场景。至于最后一种tailor interface策略，是指为一个接口增加或者删除功能。常见的功能（Capabilities）有translation, adding buffering, smoothing data等。显然，这个策略应用很广、很常见。比如混搭平台的权限管理功能，对用户进行权限设置，在用户调用Mashup应用的时候，负责根据应用的执行环境的请求，进行权限的检查，判断用户是否有权进行应用的订阅以及相应的订阅规则设置，对于不可信的用户可以考虑隐藏一些特定的功能。这是tailor interface的一个典型应用场景。  **参考文献：Web混搭平台的研究与设计.李敏敬.2016(7)** |

|  |  |
| --- | --- |
| **（1）原文** | 2. What is the relationship between interoperability and the other quality attributes highlighted in this book? For example, if two systems fail to exchange information properly, could a security flaw result? What other quality attributes seem strongly related (at least potentially) to interoperability. |
| **（2）翻译** | 互操作性与这本书中重点标记的其他质量属性的关系是什么？举个例子，如果两个系统没有正确的交换信息，会有安全隐患吗？还有哪些质量属性强依赖或者潜在依赖与互操作性？ |
| **（3）解答** | 互操作性依赖于可用性，互操作性要求在多个系统中可靠的交换信息，要求系统是可用的，系统的可用性要有保证。  互操作性会影响安全性，如果交换了错误的信息，就会造成安全隐患。  可修改性潜在依赖于互操作性，软件可能会添加多个系统相互协作的功能。  互操作性要求多个系统之间信息交换，系统多了可测试性会降低。  易用性依赖于互操作性。好的易用性要求软件完成用户渴望的任务，若该任务要求在几个系统之间传递信息，则互操作性就是该功能的基础。  其他属性有兼容性和共存性与互操作性有依赖关系。 |

|  |  |
| --- | --- |
| **（1）原文** | 3. Is a service-oriented system a system of systems? If so, describe a service-oriented system that is directed, one that is acknowledged, one that is collaborative, and one that is virtual. |
| **（2）翻译** | 面向服务的系统是否是系统的系统？ 如果是，请各描述一个指定面向服务的系统，其中一个被公认的，一个是协作性的，一个是虚拟化的。 |
| **（3）解答** | 面向服务的系统是系统的“系统”  1．公认的  淘宝。众所周知，淘宝的后端实现了各种各样的业务功能，例如浏览商品和查看物流两个功能，如果把这所有的业务功能写成一段代码，淘宝这样的企业级的代码肯定结构混乱，且不易于维护。如果能将这些各种各样的业务功能，也就是服务，分为最小粒度，封装进不同的模块，那么要修改哪种服务的功能，只要修改对应模块的代码就可以了，而且当这些服务共同运作的时候，如果有一个服务出现故障，其他服务依旧能够使用。  2．协作性  SpringCloud框架。SOA服务治理是有效实现服务间协作性的实践操作。当服务越来越多，调用方也越来越多的时候，它们之间的关系就变得非常混乱，需要对这些关系进行管理。这时候，就可以应用一些框架，比如SpringCloud，通过它，我们就能清晰地看到服务被谁调用，谁调用了哪些服务，哪些服务是热点服务需要配置服务器集群，通过调用分析和压力检测，还可以实现服务集群的负载均衡。  3. 虚拟化  虚拟机。硬件虚拟化包括在一个硬件设备中，以虚拟机的方式运行多个版本的操作系统。这将为在数据中心运行的内部应用程序提供更低的成本、更大的灵活性和风险管理的好处。 |

|  |  |
| --- | --- |
| **（1）原文** | 1. Universal Description, Disco, and Integration(UDDI) was touted as a discovery service, but commercial support for UDDI is being withdrawn. Why do you suppose this is? Does it have anything to do with the quality attributes delivered or not delivered by UDDI solutions? |
| **（2）翻译** | 统一描述、发现和集成（UDDI）被吹捧为发现服务，但UDDI的商业支持正在被撤销。 你认为这是为什么？ 是否与UDDI解决方案提供或未提供的质量属性有关？ |
| **（3）解答** | UDDI的衰落是必然的。  UDDI在提出时听起来是个好主意，他是一种基于[XML](https://zh.wikipedia.org/wiki/XML" \o "XML)的跨平台的描述规范，可以使世界范围内的企业在[互联网](https://zh.wikipedia.org/wiki/%E4%BA%92%E8%81%94%E7%BD%91" \o "互联网)上发布自己所提供的服务。具有如下好处：   * 强移植性，任何规模的行业或企业都能得益于 UDDI。 * 高性能，在 UDDI 之前，还不存在一种 Internet 标准，可以供企业为它们的企业和伙伴提供有关其产品和服务的信息。也不存在一种方法，来集成到彼此的系统和进程中。 * 强扩展性，定义一旦首选的企业被发现后如何启动商业 * 扩展新客户并增加对目前客户的访问 * 满足用户驱动的需要，为在全球 Internet 经济中快速合作的促进来清除障碍   UDDI的出现为web service提供了良好的理论基础和技术抽象，但是在实际使用中，它缺失了某些重要的软件质量属性：   * 过于复杂，低易用性。大型组织应该拥有其Web服务的目录，但这可以像维基页面一样简单。 * 忽略安全性。如果UDDI代理为用户动态选择服务提供商，用户就没有机会对服务的安全性进行任何尽职调查。 * 管理和收集小额支付的困难仍然存在。 * Web服务通常用于防火墙后面，用于SOA，将应用程序与业务伙伴集成，以及调用众所周知的API。对于这些目的，UDDI是完全多余的。 |

|  |  |
| --- | --- |
| **（1）原文** | Why has the importance of orchestration grown in recent years? |
| **（2）翻译** | 5.为什么近几年编排越来越重要？ |
| **（3）解答** | Because the orchestrate is a tactic that uses a control mechanism to coordinate and manage and sequence the invocation of particular services, its total purpose is to meet the needs which concern the logic of the business. It is really important to use orchestration to balance the invocation of particular services to avoid the collision. |

|  |  |
| --- | --- |
| **（1）原文** | If you are technology producer, what are the advantages and disadvantages of adhering to interoperability standards? Why would a producer not adhere to a standard? |
| **（2）翻译** | 6.如果你是一个技术开发者，遵守互操作性标准有什么利与弊？为什么开发者不遵守标准？ |
| **（3）解答** | Advantages:  The standards provide huge benefits to the interoperation between different systems.  Disadvantages:   1. Standards, when incorporated into products, tools, and services, undergo customizations and extensions because every vendor wants to create a unique selling point as a competitive advantage. 2. Standards also have a life cycle of their, but deciding when to adopt a new or revised standard is a critical decision for organizations. 3. Standards are often deliberately open-ended and provide extension points. 4. Engineers with opinions have their own standard which is a problem.   Producers would not adhere to a standard because they can get more profits as their competitive advantages in social market. |

|  |  |
| --- | --- |
| **（1）原文** | With what other systems will an automatic teller machine need to interoperate? How would you change your automatic teller system design to accomodate these other systems? |
| **（2）翻译** | 自动柜员机需要和其他哪些系统进行交互操作？你会如何改进你的自动柜员机系统的设计来使它和其他系统和适应？ |
| **（3）解答** | （1）自动柜员机需要用到REST技术，对数据进行CURD操作，所以我认为它需要和本地或者远程数据库系统进行交互，这就要求能正确、准时、安全的完成相应的操作；（2）它是一种终端设备，所以要和人进行交互（如果把人看作一个系统的话），这要求它能提供友好的见面，傻瓜式的操作方法，简单易用；（3）它可能与电子货币（虚拟货币、银行卡、二维码、指纹、人脸等）有关，所以要和支付系统、安全保障系统进行交互（这里的支付系统、安全系统可能与之前的数据库系统有交集，但不完全相同），这就需要它可靠、稳定，出错率极低，出故障后恢复快等等。  综上，自动柜员机需要和数据库系统、支付系统、安全保障系统以及人进行交互。  改进措施：1. 准确而快速的连接数据库。可以提高数据存储设备的性能、容量，优化存储方式，及时清理垃圾数据，减少冗余，优化查询机制等等。这同时考虑到了互操作性质量属性中Discovery、Handing of the response两个重要的方面； 2. 简化操作，美化界面，以便用户能快速流畅的使用这个自动柜员机。要做好这方面，必须理解用户需求，做好调研、市场走向，做好充分的测试，将bug最小化。这主要考虑到了互操作性的Discovery方面；3. 力求搭建一条快速又安全的绿色支付通道。要充分与第三方（如果存在）进行沟通，与相关系统有一个完美的衔接过程，使得每一笔交易都能走好、走准属于它们的那一条路，并完善故障机制处理，增强系统的健壮性。这主要考虑到了互操作性的响应处理机制方面。 |

## 第七章

|  |  |
| --- | --- |
| **（1）原文** | Modifiability comes in many flavors and is known by many names. Find one of the IEEE or ISO standards dealing with quality attributes and compile a list of quality attributes that refer to some form of modifiability. Discuss the differences. |
| **（2）翻译** | 可修改性有许多种类，并且有许多名称。 在IEEE或ISO标准中找到其中一个处理质量属性的标准，并解释一系列涉及某种形式的可修改性的质量属性，并讨论其差异。 |
| **（3）解答** | ISO/IEC 25010：  https://edisciplinas.usp.br/pluginfile.php/294901/mod\_resource/content/1/ISO%2025010%20-%20Quality%20Model.pdf  The quality attributes that refer to some form of modifiability:   1. Modularity：degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components. 2. Reusability：degree to which an asset can be used in more than one system, or in building other assets. 3. Interoperability: degree to which two or more systems, products or components can exchange information and use the information that has been exchanged. |

|  |  |
| --- | --- |
| **（1）原文** | For each quality attribute that you discovered as a result of the previous question, write a modifiability scenario that expresses it. |
| **（2）翻译** | 对于你在上一个问题中发现的每个质量属性，请编写表达它的可修改方案。 |
| **（3）解答** | 1.modularity:There is a source stimulus scenario made by the developer. Modularity can make the architecture being modularity can make module independent with each other, so when one module has been changed the other won’t need to be changed. It can make modify easier.  2.reusability: There is a source stimulus scenario made by the developer. Reusability can make the component of the module reusable and don’t need to change the whole module  3.interoperability: There is a artifact scenario made by another system with which it interoperates. When system A has an change, it just need to make its API unchangeable then the systems exchange information with it won’t need to be modify. |

|  |  |
| --- | --- |
| **（1）原文** | 3.In a certain metropolitan subway system, the ticket machine accept cash but do not give change. These is a separate machine that dispenses change but does not sell tickets. In an average station there are six or eight ticket machines for every change machine. What modifiability tactics do you see at work in this arrangement? What can you say about availability? |
| **（2）翻译** | 在某大都会地铁系统中，售票机接受现金但不会找零钱。有一台独立的机器，它提供零钱但不卖票。平均每个车站有六台或八台售票机搭配一台换置零钱的机器。在这种安排下，你在工作中看到什么样的可适应性策略？关于可用性你能说些什么？ |
| **（3）解答** | * 我看到的策略主要有1）Reduce the Size of Module(减小模块规模)，卖票和找零属于不同的操作，将这两个模块分离开，方便了未来对系统的修改2）Increase Cohesion(增加内聚)，每一个模块只负责一个功能，不夹杂其他操作，实现了模块化，有利于整个系统的开发和运作，也方便调试系统，未来增加模块直接增加即可，不需要修改之前的模块 * Availability（可用性）是指能够满足用户的需求，在这里面由于模块化做的很好，这大大降低了系统出现问题的可能性，找零系统跟售票系统不属于同一个模块，当其他模块出问题的时候剩余模块还可以继续工作，这个系统属于一个High Availability的系统。 |

|  |  |
| --- | --- |
| **（1）原文** | 4. For the subway system in the previous question, describe the specific form of modifiability (using a modifiability scenario) that seems to be the aim of arranging the ticket and change machines as described. |
| **（2）翻译** | 对于前面问题中的地铁系统，描述可修改性的具体形式（使用可修改性场景）,该可修改性似乎是按照描述的方式安排票和兑换机的目的 |
| **（3）解答** | * 如上所述，在这里的可修改性具体体现形式在于分割（split module）模块和增加内聚（increase cohesion）。在此系统中，有六到八台售票机器搭配一台找零机器，这两种机器的要完成的功能不一样，售票机只售票，找零机只找零，根据他们不同的职责就可以将他们划分为两个模块，在划分模块的过程中既实现了分割模块的目的又实现了增加内聚的目的 |

|  |  |
| --- | --- |
| **（1）原文** | 5.A wrapper is a common aid to modifiability. A wrapper for a component is the only element allowed to use that component; every other piece of software uses the component’s services by going through the wrapper. The wrapper transforms the data or control information fro the component it wraps. For example, a component may expert input using English measures but find itself in a system in which all of the other components produce metric measures. A wrapper could by employed to translate. What modifiability tactics does a wrapper embody? |
| **（2）翻译** | 5.包装是可修改性的常见辅助。组件的包装器是允许使用该组件的唯一元素; 每个其他软件都通过包装器使用组件的服务。包装器将数据或控制信息转换为它包装的组件。例如，组件可以使用英语测量的专家输入，但发现自己处于所有其他组件产生度量测量的系统中。可以使用包装器进行翻译。 包装器体现了什么可修改性策略？ |
| **（3）解答** | Wrapper embody 体现了cohesion 和 coupling 两个策略。首先原本系统中的每个组件都是相互独立的个体，因为包装器是唯一可以使用组件的元素，所以通过包装器将各个部件连接在了一起，这里体现了cohesion策略。其次，组件期望用英语度量作为输入，这里包装器就可以充当英语翻译的中介，让所有组件的输出都由包装器进行翻译，这样子，就不用每个组件自己进行翻译，减少了组件的coupling. |

|  |  |
| --- | --- |
| **（1）原文** | 6. Once an intermediary has been introduced into an architecture, some modules may attempt to circumvent it, either inadvertently (because they are not aware of the intermediary) or intentionally (for performance, for convenience, or out of habit). Discuss some architectural means to prevent inadvertent circumvention of an intermediary. |
| **（2）翻译** | 6.一旦将中介引入到体系结构中，某些模块可能会无意中（因为他们不了解中间人）或故意（为了方便，为了方便或出于习惯）而试图绕过它。 讨论一些建筑手段，以防止无意中规避中间人。 |
| **（3）解答** | 1.increase cohesion and reduce coupling ,通过这样使各个模块成为耦合度较低的的各部分，然后就要形成整体运作的时候就必须通过中介的作用将他们cohesion在一起。  2. split module, encapsulate, abstract common services等方法都可以。 |

|  |  |
| --- | --- |
| **（1）原文** | 7. In some project, deployability is an important quality attribute that measures how easy it is to get a new version of the system into the hands of its users. This might mean a trip to your auto dealer or transmitting updates it arrives. In project that measures deployability separately, should the cost of a modification stop when the new version is ready to ship？justify your answer. |
| **（2）翻译** | 在某些项目中，可部署性是一个重要的质量属性，用于衡量将新版本的系统交到用户手中的难易程度。 这可能意味着您的汽车经销商旅行或传输它到达的更新。在独立测量可部署性的项目中，如果新版本准备好发布，用来修改的开销是否应该停止？ |
| **（3）解答** | Answer: 是，在一个独立测量软件系统的可部署性系统，软件的基本功能能够完成即可，对于用户获取软件的便利程度才是最重要。在新版本发布之际，此时的修改会带来极大的不确定性，所以此时应尽量减少修改的开销，尽量不修改。 |

|  |  |
| --- | --- |
| **（1）原文** | 8. The abstract common services tactic is intended to reduce coupling, but it also might reduce cohesion. Discuss. |
| **（2）翻译** | 抽象公共服务策略旨在减少耦合度，但也可能降低内聚度，请讨论。 |
| **（3）解答** | Answer: 软件设计中通常用耦合度和内聚度作为衡量模块独立程度的标准。划分摸块的一个准则就是高内聚低耦合。耦合度从以下几个方面衡量：  （1）一个模块对另一个模块的调用；  （2）一个模块向另一个模块传递的数据量；  （3）一个模块施加到另一个模块的控制的多少；  （4）模块之间接口的复杂程度。  抽象公共服务是将两个模块中类似功能进行抽象统一接口，通过不同参数类型，提供不同服务。通过这种策略，代码量直接减少，模块间共用代码，直接降低耦合度。但是这也意味着多了一层抽象，致于类之间的关系变得模糊，引来了内聚度降低的问题。 |

|  |  |
| --- | --- |
| **（1）原文** | For each quality attribute that you discovered as a result of the previous question, write a modifiability scenario that expresses it. |
| **（2）翻译** | 对于你在上一个问题中发现的每个质量属性，请编写表达它的可修改方案。 |
| **（3）解答** | 1.modularity:There is a source stimulus scenario made by the developer. Modularity can make the architecture being modularity can make module independent with each other, so when one module has been changed the other won’t need to be changed. It can make modify easier.  2.reusability: There is a source stimulus scenario made by the developer. Reusability can make the component of the module reusable and don’t need to change the whole module  3.interoperability: There is a artifact scenario made by another system with which it interoperates. When system A has an change, it just need to make its API unchangeable then the systems exchange information with it won’t need to be modify. |

另外一组：

|  |  |
| --- | --- |
| **（1）原文** | Modifiability comes in many flavors and is known by many names. Find one of the IEEE or ISO standards dealing with quality attributes and compile a list of quality attributes that refer to some form of modifiability. Discuss the differences. |
| **（2）翻译** | 可修改性有多种风格，并且有许多名称。找到处理质量属性的IEEE或ISO标准之一，并编制一系列质量属性，这些属性涉及某种形式的可修改性。并讨论它们的差异。 |
| **（3）解答** | 在课本194页图12.1展示了一个ISO标准( The ISO/IEC FCD 25010 product quality standard )。  涉及可修改性的质量属性：可用性(Availability), 可测试性(Testability)，性能效率(Performance efficiency),安全性(Security)  可用性指的是系统在考察时间内是否能够正常运行。  可测试性指的是发现软件系统错误，并且设计测试、执行测试的能力。  性能一般指的是系统对于用户操作的响应时间的快慢。  安全性是指系统保护数据和信息不被未授权访问所获取，而且提供数据信息给已授权的访问操作。 |

|  |  |
| --- | --- |
| **（1）原文** | For each quality attribute that you discovered as a result of the previous question, write a modifiability scenario that expresses it. |
| **（2）翻译** | 对于您在上一个问题中发现的每个质量属性，请编写一个表达它的可修改场景。 |
| **（3）解答** | **Availability:**  Source: Heartbeat Monitor  Stimulus: Server Unresponsive  Environment: Normal Operation  Response: Inform Operator Continue to Operate  Response Measure: No Downtime  **Testability:**  Source: Unit Tester  Stimulus: Code Unit Completed  Environment: Development  Response: Results Captured  Response Measure: 85% path Coverage in Three Hours  **Performance:**  Source: Users  Stimulus: Initiate Transactions  Environment: Normal Operation  Response: Transactions Are Processed  Response Measure: Average Latency of Two Seconds  **Security:**  Source: Disgruntled Employee from Remote Location  Stimulus: Attempts to Modify Pay Rate  Environment: Normal Operations  Response: System Maintains Audit Trail  Response Measure: Correct Data is Restored within a Day and Source of Tampering Identified. |

|  |  |
| --- | --- |
| **（1）原文** | In a certain metropolitan subway system, the ticket machines accept cash but do not give change. There is a separate machine that dispenses change but does not sell tickets. In an average station there are six or eight ticket machines for every change machine. What modifiability tactics do you see at work in this arrangement? What can you say about availability? |
| **（2）翻译** | 在某个大城市地铁系统中，售票机接受现金但不出零。同时有一个独立机器换零但不售票。在一般地铁站中有6或8个售票机对应1个零钱机。在这种安排工作中你看到了哪些可修改性战术？对于可用性你可以说些什么？ |
| **（3）解答** | 对于可修改性，体现的战术有：减小模块大小，提高内聚，减小耦合。  减少模块大小-分离模块：将售票机和零钱机按功能分隔，以便当故障发生时减少平均开销。每个模块的功能保持了简单化和可用性。  提高内聚：在地铁系统中，售票机和零钱机具有不同的功能且不交叉。  减少耦合：功能封装在系统的不同模块中，二者互不影响。  对于可用性，当有一台售票机出故障时，多台售票机可以分摊多余负担保证系统可用性；同时系统满足了原子性，一致性，隔离性和持久性，使售票组件和找零组件运行时不会产生冲突或其他错误。 |

|  |  |
| --- | --- |
| **（1）原文** | For the subway system in the previous question, describe the specific form of modifiability (using a modifiability scenario) that seems to be the aim of arranging the ticket and change machines as described. |
| **（2）翻译** | 对于上个问题中的地铁系统，描述可修改性的具体形式（使用可修改性的场景），使它实现如描述安排售票机和零钱机的目的。 |
| **（3）解答** | Source：终端用户，地铁系统管理员，开发人员  Stimulus：买票或找零  Artifact：数据，功能组件，resources，configuration  Environment：compile time，initiation time，design time  Response：出售车票；找回等价零钱  Response measure：maybe in 3 minutes；多台机器同时维护 |

|  |  |
| --- | --- |
| **（1）原文** | A wrapper is a common aid to modifiability. A wrapper for a component is the only element allowed to use that component; every other piece of software uses the component's services by going through the wrapper. The wrapper transforms the data or control information for the component it wraps. For example, a component may expect input using English measures but find itself in a system in which all of the other components produce metric measures. A wrapper could be employed to translate. What modifiability tactics does a wrapper embody? |
| **（2）翻译** | 包装器通常用来辅助可修改性。每个组件的包装器是允许使用该组件的唯一组件，所有其他软件都通过包装器使用该组件的服务。包装器为它包装的组件转换数据或控制信息。例如，组件可能期望使用英制进行输入，但发现自己处于一个系统中，其中所有其他组件都使用公制。此时就可以使用包装器进行翻译。包装器体现了什么可修改性策略？ |
| **（3）解答** | 小组讨论认为包装器体现了3种可修改性策略：  封装，使用中介和限制依赖项  所有策略都用于减少耦合。 |

|  |  |
| --- | --- |
| **（1）原文** | Once an intermediary has been introduced into an architecture, some modules may attempt to circumvent it, either inadvertently (because they are not aware of the intermediary) or intentionally (for performance, for convenience, or out of habit). Discuss some architectural means to prevent inadvertent circumvention of an intermediary. |
| **（2）翻译** | 一旦将中介引入到体系结构中，一些模块可能会无意（因为他们不知道该中介）或故意（为了性能，为了方便或出于习惯）地试图绕过它。讨论一些架构方案，以防止无意地规避中介结构。 |
| **（3）解答** | 目前找到两种来阻止无意识规避中介的方法：   1. 写出详细的文档，函数注释来提醒开发者中介的存在，同时增加直接调用模块的难度（如增大模块接口的隐藏性，但同时会增加故意避免中介的难度）。 2. 调用模块的函数默认是通过中介的，如果要避免中介需要在客户端调用函数时将是否使用中介位置为false。 |

|  |  |
| --- | --- |
| **（1）原文** | In some projects, deployability is an important quality attribute that measures how easy it is to get a new version of the system into the hands of its users. This might mean a trip to your auto dealer or transmitting updates over the Internet. It also includes the time it takes to install the update once it arrives. In projects that measure deployability separately, should the cost of a modification stop when the new version is ready to ship? Justify your answer. |
| **（2）翻译** | 在某些项目中，可部署性是一个重要的质量属性，用于衡量将新版本的系统置于其用户手中的难易程度。这可能意味着一次访问自动分发功能或通过互联网传输更新。它还包括一旦到达时安装更新所需的时间。 在分别衡量可部署性的项目中，如果新版本准备好发布，修改的成本是否应该停止？证明你的答案。 |
| **（3）解答** | It is not necessary that we should stop the cost of a modification when the new version is ready to ship. In modern development process, version control system plays a very important role. With the help from VCS, we can maintain a non-linear workflow. That is when we make a modification, we do not have to publish it in the next version. We can commit it to a branch, test it. Until it is stabilized, we don’t publish this modification to customers. However, when we publish a new version, more work on integration and tests is needed. For this reason, we may need to reduce the cost of a modification. |

|  |  |
| --- | --- |
| **（1）原文** | The abstract common services tactic is intended to reduce coupling, but it also might reduce cohesion. Discuss. |
| **（2）翻译** | 抽象公共服务策略旨在减少耦合，但它也可能降低内聚。讨论一下。 |
| **（3）解答** | 抽象公共服务的初衷是如果之后需要修改的话，只要在一处进行修改就可以了。但如果抽象得不好，产生的公共服务里可能需要为各种使用情况实现特例进行处理，这样就降低了公共服务本身的内聚，即同一个功能模块要处理各种不同的指责 |

|  |  |
| --- | --- |
| **（1）原文** | Identify particular change scenarios for an automatic teller machine. What modifications would you make to your automatic teller machine design to accommodate these changes? |
| **（2）翻译** | 为自动取款机识别特定变化的情景。您对自动取款机的设计进行了哪些修改以适应这些变化？ |
| **（3）解答** | **Reduce size of a module:**  Split the automatic teller machine’s module as deposit module,withdrawal module,account management module,user login module and so on.  **Increase cohesion:**  To make the two module which have different responsibilities break,we can make a subdivision to some module and make reading bank card module,get cash module split.  **Reduce coupling:**  To accommodate the scenarios that providing service to different country’s people,we can split the language service by different languages.  **Defer binding:**  We can bind the ATM to more influencing factors and add more parameters. To make a better service for any clients,we can collect a few parameters,such as client’s age,nationality and using purpose then provide its financial service. |

## 第八章

|  |  |
| --- | --- |
| **（1）原文** | 1.“Every system has real-time performance constraints.” Discuss. Or provide a counterexample. |
| **（2）翻译** | “每个系统都会受到实时的性能约束”，请讨论这句话，或者给出一个反例。 |
| **（3）解答** | As is shown at the beginning of Chapter 8, performance is all about time and the software system’s ability to meet timing requirements. When events occur, like interrupts, messages, requests from users or other systems, or clock events marking the passage of time, the system, or some element of the system, must respond to them in time. For example, in complex measurement and control applications, real-time operating systems that require very high real-time performance must be used. Besides, performance depends on two aspects, the size of system and the ability of your system to compute. The size of system is fixed on development, and the ability to compute is fixed on deployment. In real-time running, every event will compete for the resources, or a interrupt occurred to occupy a portion of resources. The competition for resource is very common in a system. When the resources is occupied by some events, then other events can not fetch a resource, that is the bottleneck of a system. An extensively discussed example in the software design is the 12306 china railway system. The peak value of daily click in the system is about 157 billion times, occurred on January 24, 2018. However, 3 years ago, without a great design as now, facing to 36 billion peak value, about a quarter in 2018, the 12306 system are of high delay, hard to connect, showing inconsistent number of rest tickets and so on. This is the real-time performance. |

|  |  |
| --- | --- |
| **（1）原文** | 1. write a performance scenario that describes the average on-time flight arrival performance for an airline. |
| **（2）翻译** | 编写一个性能场景，描述航空公司平均情况下航班准时抵达的性能。 |
| **（3）解答** | |  |  |  |  |  | | --- | --- | --- | --- | --- | | Source: | stimulus: | artifact: | response: | response measure: | | Planes | Arrival or leaving | Schedule system | Schedule the flights | worst-case latency of 10min  average-case latency of 1min  Throughput of 5  Miss rate of 1% | |  |  | Environment: |  |  | |  |  | Normal operation |  |  |   In this scenarios, the planes arrival or leaving under normal operation. The schedule system schedules the flights with the worst-case latency of 10min, average-case latency of 1min, throughput of 5, miss rate of 1% |

|  |  |
| --- | --- |
| **（1）原文** | 1. Write several performance scenarios for an automatic teller machine. Think about whether your major concern is worst-case latency, average-case latency, throughput, or some other response measure. How would you modify your automatic teller machine design to accommodate these scenarios? |
| **（2）翻译** | 为ATM编写几种性能场景。考虑您的主要问题是最坏情况延迟，平均情况延迟，吞吐量或者其他一些响应措施。您如何修改ATM设计以适应这些情况？ |
| **（3）解答** | |  |  |  |  |  | | --- | --- | --- | --- | --- | | Source: | stimulus: | artifact: | response: | response measure: | | User | check balance  Transfer  Withdraw  Save | ATM system | Display the balance  Transfer are processed  Withdraw are processed  Save are processed | worst-case latency of 10s  average-case latency of 5s  Throughput of 1  jitter of 2s  Miss rate of 0 | |  |  | Environment: |  |  | |  |  | Normal operation |  |  |   In this scenarios, user ask for check balance, transfer or withdraw/save money under normal operation. The ATM system processes these requires with worst-case latency of 10s, average-case latency of 5s, Throughput of 1, jitter of 2s and Miss rate of 0.  The case for worst-case latency: improve the processing efficiency of the ATM  The case for average-case latency: use high efficiency algorithm  The case for miss rate: design the ATM to check the process result multiple times |

|  |  |
| --- | --- |
| **（1）原文** | 4.Web-based systems often use proxy servers, which are the first element of the system to receive a request from a client(such as your browser).Proxy servers are able to serve up often-requested web pages, such as a company’s home page, without bothering the real application servers that carry out transactions. There may be many proxy servers, and they are often located geographically close to large user communities, to decrease response time for routine requests. What performance tactics do you see at work here? |
| **（2）翻译** | 基于网络的系统经常使用代理服务器作为最先从客户端接受请求的构建。代理服务器可以在不增加处理业务的真实应用服务器负载的情况下，完成常用页面，例如公司主页的访问请求。许多代理服务器被部署在靠近用户社区的地方来降低访问延迟。在这个例子中用到了什么样的性能战术？ |
| **（3）解答** | 1. Reduce overhead tactic for control resource demand. For requesting the resources from the real serve, which needs a longer path also needs more network transmission overhead and time delay overhead. But if we set a neighbor server for the close user we can reduce these overhead and get the response faster. 2. Maintain multiple copies of data for manage resources. As a matter of fact, the resources maintained in the real server and the proxy server are the same, which is redundant. By using more memory space, we can reduce the requests of each resource and get our response faster. 3. Increase the resources. In this case we increase many proxy servers, which has the similar processing ability to the real server. So this tactic increase the resources. Different requests are distributed to different servers and the load of each server decrease. As the result, the total process time of the system decrease. |

|  |  |
| --- | --- |
| **（1）原文** | 5.A fundamental difference between coordination mechanisms is whether interaction is synchronous or asynchronous. Discuss the advantages and disadvantages of each with respect to each of the performance responses: latency, deadline, throughput, jitter, miss rate, data loss, or any other required performance-related response you may be used to. |
| **（2）翻译** | 协调机制之间的根本区别在于交互是同步还是异步。讨论这两种协调机制在以下几种性能响应方面的优缺点：延迟，截止时限，吞吐量，抖动，未命中率，数据丢失或您可能使用过的任何其他所需的性能相关响应。 |
| **（3）解答** | （1）Latency: Latency has more impact on synchronous interaction than asynchronous. For synchronous interaction, if the event couldn’t be processed by system, it will affect the operations which is waiting for the results of the process, while it has less impact on asynchronous interaction because the results of the process don’t have to be delivered back to where it comes from.  （2）Deadline: The impact resulted from deadline is similar to the one with latency. For synchronous interaction, it requires the system to response as soon as possible so as to improve the performance. For example, a client send a request about purchasing an item, if the server cannot respond in a limited deadline, then the client will probably get mad and cancel the transaction, which will also cause the merchants to loss. On the other hand, deadline has less impact on asynchronous interaction for the reason that the results may not that important with respect to time constrains.  （3）Throughput: Throughput is equally important to both types of interaction. Throughput is measured by the number of transactions the system can process in a unit of time. Either the synchronous interaction or the asynchronous interaction can improve performance by improving the throughput.  （4）Jitter: Jitter means the allowable variation in latency. Now that as is mentioned above about the latency, jitter has more impact on synchronous interaction than asynchronous. Asynchronous interaction allows a relatively larger range of time, while synchronous interaction won’t be likely to accept variation in latency.  （5）Miss rate: High miss rate is unacceptable to both synchronous interaction and asynchronous interaction. As a measure of performance response, the miss instead of right response will affect the performance, no matter what interaction it is.  （6）Data Loss: The situation about data loss is similar to miss rate. As a measure of performance response, the miss of arrival of events or responses from system will affect the performance, no matter what interaction it is. |

|  |  |
| --- | --- |
| **（1）原文** | 6.Find real-world (that is, non-software) examples of applying each of the manage-resources tactics. For example, suppose you were managing a brick-and-mortar big-box retail store. How would you get people through the checkout lines faster using these tactics? |
| **（2）翻译** | 请列举在非软件场景下应用每一个资源管理战术的例子。例如，假设你在管理大型实体零售店，你会如何使用这些战术让人们更快地埋单？ |
| **（3）解答** | 1. Increase resources. We will hire skilled cashiers rather than a green hand. For a skilled cashier, it need less time to server a consumer, so the line moves faster. 2. Introduce concurrency. Generally, in a brick-and mortar big-box retail store, there are many checkout counters. So we can checkout more consumers at the same time. For the consumers that wait in lines, the total time for waiting decrease. 3. Schedule resources. Make a suggestion for the consumers about which line they should wait in. We can estimate the waiting time and recommend the fastest line. 4. Bound queue sizes. If the store contains more people than the threshold, we should forbid the people coming continually to prove the serve quality. |

|  |  |
| --- | --- |
| **（1）原文** | 7.User interface frameworks typically are single-threaded. Why is this so and what are the performance implications of this single-threading? |
| **（2）翻译** | 用户界面框架通常是单线程的。为什么会这样？这种单线程的性能影响是什么？ |
| **（3）解答** | Generally operations on UI widgets are not thread-safe, which means that when multi-threads access concurrently, lock mechanism will result in the situations such as operations on UI widgets becomes more complex, dead lock and low-efficiency. UI frameworks with multi-threads will easily under the influence of deadlock, partially for the reason that input event process has occasional interaction with any object model behind the GUI widgets. For example, operations issued by users will pass from the OS to specific application in the way of bubbling. The process is that firstly OS monitor and detect a mouse click, it transforms it to an event of MOUSE CLICK and eventually the event becomes a high-level event (such as an event named “buttonpressed”) and be transferred to a monitor of an application. On the other hand, the application will issue an new operation such as changing the background color of a widget which will be delivered back to the OS. This will be transferred to a specific widget class and then transferred to the OS to render. The two operations described above have an access to the same GUI object in two totally different way, it must guarantee that every single object is thread-safe in principle, but it leads to deadlock due to the inconsistent order of locks. Although multi-threads can improve the concurrency to improve performance, its disadvantages make it unreliable on performance. Thus it will lower the performance of UI frameworks after trade-off. However, Because the UI thread should react to user input, and it is also responsible to render all the graphics and animation on the view, sometimes the single thread is not enough. We may introduce another thread to help the main UI thread to render visuals. In summary, for convenience and based on what’s discussed above, UI frameworks are typically single-threaded. |

另外一组：

|  |  |
| --- | --- |
| **（1）原文** | Find real-world(that is, nonsoftware) example of applying each of the manage-resources tactics. For example, suppose you were managing a brick-and-mortar big-box retail store.How would you get people through the checkout lines faster using these tactics |
| **（2）翻译** | 为每一个资源管理策略找一个现实世界的例子。比如，假设你正在管理一个实体仓储式零售店。你怎么运用策略让顾客更快地排完队完成购物？ |
| **（3）解答** | 增长资源：提高零售店的硬件质量：如：增大面积，更新设备  支持并发：增加收银台的数量，以此提升同时服务多位顾客的能力  持有computation的复制：设置应急人员来应对客流高峰期  持有数据的复制：大量进货某种特定的热门商品，防止供不应求  限制访问规模：限制每位顾客每次可以购买的商品数  调度资源：设立一个调解人，在顾客发生争执时负责调解顾客 |

|  |  |
| --- | --- |
| **（1）原文** | User interface framework typically are single-threaded, Why is this so and what are the performance implication of this single threading |
| **（2）翻译** | 用户界面框架总是单线程的，为何如此？单线程的性能效果如何？ |
| **（3）解答** | 多线程的缺点：需要占用更多内存  程序更加复杂，出现BUG的概率会增大  有出现死锁的可能性  单线程在速度上不如多线程。  在实际使用中，用户即使运行多个程序，但同一时间也只会操作其中一个程序的界面；在操作界面时，人的反应速度远不如计算机的运算速度，单线程的运算速度足以应对数据的输入输出。 |

|  |  |
| --- | --- |
| **（1）原文** | Web-based systems often use proxy servers, which are the first element of the system to receive a request from a client (such as your browser). Proxy servers are able serve up often-requested web pages, such as a company’s home page, without bothering the real application servers that carry out transaction. There may be many proxy servers, and they are often located geographically closed to large user communities, to decrease response time for routine requests. What performance tactics do you see at work here? |
| **（2）翻译** | 基于Web的系统通常使用代理服务器，它是系统从客户端（例如您的浏览器）接收请求的第一个元素。 代理服务器能够提供经常请求的网页，例如公司的主页，而不会打扰执行任务的真实应用服务器。 可能存在许多代理服务器，并且它们通常位于地理位置上离大型用户社区较近的位置，以减少常规请求的响应时间。 你在其中看到了哪些性能策略? |
| **（3）解答** | 减少开销：在地理位置接近的情况下减少一般请求的响应时间。  提高效率：提供经常使用的网页（例如，主页），而不会干扰实际应用服务器。  引入并发：多个代理服务器提供并发性。  增加资源：多个代理服务器增加资源。  保存多份数据的副本：代理服务器能够提供经常请求的网页，例如公司的主页。 |

|  |  |
| --- | --- |
| **（1）原文** | A fundamental difference between coordination mechanisms is whether interaction is synchronous or asynchronous. Discuss the advantages and disadvantages of each with respect to each of the performance responses: latency, deadline, throughput, jitter, miss rate, data loss, or any other required performance-related response you may be used to. |
| **（2）翻译** | 协调机制之间的根本区别在于交互是同步还是异步。讨论每种性能响应的优缺点：延迟，截止时间，吞吐量，抖动，丢包率，数据丢失或你可能熟悉的任何其他所需的性能相关的响应。 |
| **（3）解答** | 1.Latency：信息的传输和处理需要一定的时间，因此交互中必定存在延迟。延迟会影响同步交互，并且对异步交互的影响较小。  2.Deadline：任何事件都应在一定时限内处理。Deadline有助于避免死锁并促进系统恢复。但将处理周期设置得太长或太短都会影响系统的正常运行。 3.Throughout：通常以系统可以在单位时间内处理的事务数量给出。吞吐量反映了系统的并行性受带宽的影响，因此有时会成为影响系统效率的瓶颈。 4.Jitter：延迟允许的变化。延迟是由多种原因引起的，因此没有固定的延迟，换句话说，延迟中存在一些抖动。允许延迟更改允许系统适应更多情况并提高性能。但是，确定抖动范围更加困难。 5.Miss rate：当系统太忙而无法响应时，会发生丢包。丢包率反映了数据传输。 6.数据丢失：系统发生丢包时会发生数据丢失。数据丢失的主要原因是物理线路故障，设备故障，病毒攻击和路由信息错误。数据丢失会在一定程度上影响系统的正常运行。但UCP协议允许某些数据丢失。 其他与性能相关的响应：可用性，例如网络是否正常通信以及路径是否可访问。 |

|  |  |
| --- | --- |
| **（1）原文** | “Every system has real-time performance constrains.”Discuss it. Or provide a counterexample |
| **（2）翻译** | “每个系统都有实时性能限制。”讨论这句话或举出反例。 |
| **（3）解答** | 不一定。比如邮件传输可以存在一定延迟。 |

|  |  |
| --- | --- |
| **（1）原文** | Write a performance scenario that describes the average on-time flight arrival performance for an airline. |
| **（2）翻译** | 编写一个性能场景来描述航空公司的飞机平均准时抵达性能。 |
| **（3）解答** | 源：内部：飞机状况；外部：天气  刺激：飞机起飞  器具：机场，飞机  环境：一般  反馈：降落地点，抵达时间  反馈衡量：准时与否，是否在指定位置着陆，延迟量 |

|  |  |
| --- | --- |
| **（1）原文** | Write several performance scenarios for an automatic teller machine.Think about whether your major concern is worst-case latency,average-case latency,throughput , or some other response measure. How would you modify your automatic teller machine design to accommodate these scenarios? |
| **（2）翻译** | 为自动出纳机写几个性能场景。思考以下哪些反馈衡量最值得关心：最大延迟，平均延迟，吞吐量，还是其他。你要怎么通过修改自动出纳机的设计来适应这些场景。 |
| **（3）解答** | 取款，存款：考虑到用户的耐心有限，需要注意的是延迟，为了不耗尽用户的耐心，最应该关心的是最大延迟。而错误率也是关键，绝对不能出错。可以适当牺牲吞吐量和平均延迟来确保错误率和最大延迟 |

## 第九章

|  |  |
| --- | --- |
| **（1）原文** | Write a set of concrete scenarios for security for an automatic teller machine. How would you modify your design for the automatic teller machine to satisfy these scenarios? |
| **（2）翻译** | 请描述一组ATM（自动柜员机）的具体安全场景。并回答你将如何修改ATM的设计以满足这些安全需求？ |
| **（3）解答** | 见下表 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Source** | **Stimulus** | **Artifact** | **Environment** | **Response** | **Response Measure** | **Modify Plan** |
| Thief | Get money | Break the machine | Normal state | The bank will broadcast the alerts, and videotape the whole process | Guards get to the scene within 5 minutes | Lock the data in ATM when it is being damaged and Lock the bank’s door to avoid the criminal from escaping. |
| Thief | Get money | The card insert in the machine | When clients are drawing out money from ATM | System maintains the previous service | Drawback the card within 5 minutes | Add a detective sensor to distinguish foreign materials besides cards in the bayonet |
| Thief | Get money | The password of clients cards | When clients are drawing out money from ATM | / | / | Control unilateral visual angle within 30 degrees; Install a reflector on the ATM such that the clients know what’s happening behind them. |
| Thief | Get money | Get the money that clients have drawn out | When clients are drawing out money from ATM | / | / | Detect foreign materials in the banknote mouth. |
| Virus | Get access to and modify the data in the backend | Data within the system | When the machine is connected to the bank ‘s production network | System remains Audit Trail | Correct data is restored within a day and source of abnormal modification is detected | Add virus detection and port protection in the system; Control the execution of non-whitelist files to achieve security protection, while assisted by virus detection and killing, as well as virus traceability and site-specific clearance. |
| Virus | Change data | Data in the current clients’ account | Normal state | / | / | Add port protection and process protection in the system. |

|  |  |
| --- | --- |
| **（1）原文** | One of the most sophisticated attacks on record was carried out by a virus known as Stuxnet. Stuxnet first appeared in 2009 but became widely known in 2011 when it was revealed that it had apparently severely damaged or incapacitated the high-speed centrifuges involved in Iran's uranium enrichment program. Read about Stuxnet and see if you can devise a defense strategy against it based on the tactics in this chapter. |
| **（2）翻译** | 自有记录以来，最复杂的安全攻击之一是由一个名为Stuxnet的病毒引发的。Stuxnet于2009年首次出现，但当Stuxnet在2011年让伊朗铀浓缩计划中的高速离心机严重损坏或者说失效时，Stuxnet才变得广为人知。阅读Stuxnet相关资料，看看你是否可以根据本章的策略来设计一份Stuxnet防御策略。 |
| **（3）解答** | Stuxnet is the world's first destructive virus specifically for industrial control systems, capable of exploiting seven vulnerabilities in Windows and Siemens SIMATIC WinCC systems. The virus is mainly spread through U disk and LAN.  Stuxnet and its variants are worms that exploit the latest Windows Shell vulnerabilities to spread malicious files. The cause of this vulnerability is that Windows incorrectly parses shortcuts, which can execute malicious code when a user clicks on a display icon for a specially crafted shortcut.  Before I talk about how to defense against it, we should know the principle of infecting. Stuxnet attack system mainly by infecting PLC, which is called programmable logic controller. Besides, it uses special codes called Rootkit to prevent being detected by PLC manager. The purpose of Stuxnet is to modify the behavior of the industrial production control system by modifying the PLC, including intercepting the read/write request sent to the PLC to determine whether the system is a potential attack target; modifying the existing PLC code block and going to the PLC Write new code blocks; use the Rootkit function to hide PLC infections, avoid detection by PLC administrators or programmers.  Now we know how Stuxnet works roughly. We can use tactics of security to defense against it.  Detect intrusion and verify service integrity: If anything is downloaded without authentication, it must be deleted. Besides, if a desktop occurs a link, system must scan it and detect its source file.  Identify actors, authentication actors, authorize actors, limit access, limit exposure, encrypt data: before execution a file, system must notify the administrator. Besides, system can notify the administrator how much permission it gives application. Data must be encrypted so that the virus can’t use it.  Revoke access, lock computer and inform actors: If the system is infecting, it can limit application install recently. It can lock computer and stop data transmission. System can notify the administrator.  Maintain audit trail and restore: System can make a backup periodic. System can trace all application executing so that it can stop unusual application. System can recover from backup. |

|  |  |
| --- | --- |
| **（1）原文** | Some say that inserting security awareness into the software development life cycle is at least as important as designing software with security countermeasures. What are some examples of software development processes that can lead to more-secure systems? |
| **（2）翻译** | 有人说，将安全意识纳入软件开发生命周期的重要性不亚于将安全对策纳入软件设计中。请问有哪些软件开发过程可以带来更安全的系统？ |
| **（3）解答** | Software development processes with security is a software design process that targets secure software products, including secure schematic design or solution design, security coding, security testing, and security deployment.  Take Janusic SDL SaaS as an example. The simplest security development process includes three phases (development test phase, product release phase, operation and maintenance phase). The standard application development process includes project planning, requirements analysis, summary design, and development. Testing, product release, acceptance, operation and maintenance, etc.  Each stage has a corresponding quality control task, especially a safety task.  **Project planning stage, Demand analysis phase, Summary design phase, Development test phase, Product release phase, Acceptance phase, Operation and maintenance stage.** |

|  |  |
| --- | --- |
| **（1）原文** | Security and usability are often seen to be at odds with each other. Security often imposes procedures and processes that seem like needless overhead to the casual user. But some say that security and usability go (or should go) hand in hand and argue that making the system easy to use securely is the best way to promote security to the user. Discuss. |
| **（2）翻译** | 人们通常认为安全性和可用性是相互矛盾的。安全性通常会给临时用户增加一些程序或步骤，而这些程序或步骤往往被被看作不必要的开销。但是另有一些人说安全性和可用性是（或应该是）相互配合的，他们还认为让系统的安全使用轻松起来是促进用户安全的最佳方式。请就以上问题进行讨论。 |
| **（3）解答** | Software usability refers to the ability of software to perform specified functions within specified conditions and within specified time. Software usability, one of the inherent features of software products, demonstrates the ability of the system to perform its functions in accordance with user requirements and design goals. Among them, the specified conditions include the software operating environment and software operation profile. The operating environment refers to the hard and software environment in which the software runs, including the operating system, application, compilation system, database system, etc. of the software running; the hardware environment includes the CPU memory of the computer, input/output, and so on.  The main factors affecting software usability: (1) software scale (2) software internal structure (3) software development method (4) software development tools (5) software testing and investment  There are many expressions of understanding and definition of safety. For example, in ISO8402, the term Safety is defined as: limiting the risk of injury (to humans) or damage to an acceptable state. Ways to improve the security of the software are: (1) security management (2) software risk analysis (3) software requirements description and design verification (4) using fault-tolerant technology:  The relationship between usability and security is very tight, and the parameters that affect the relationship between them are mainly to change the deviation between the program state environment and the state. To this end, we can use the change of mutual relationship to further study the transition law between them, that is to say: use the changes of the environment and the deviation loss parameters to convert some usability problems into security problems, using security related Technology to deal with usability issues; it is also possible to translate some security issues into usability issues and deal with security issues with usability -related technologies. |

|  |  |
| --- | --- |
| **（1）原文** | List some examples of critical resources for security that might become exhausted. |
| **（2）翻译** | 有一些关键的安全资源可能被耗尽，请举出相关例子。 |
| **（3）解答** | Server resources are exhausted. There are generally several situations:  1） Suffering from CC attacks, people are maliciously using the "zombie machine" to access a large amount of instant, occupying your server resources, causing the CPU and memory to run out, causing the server to crash.  2）suffered from DDoS and other traffic attacks, resulting in exhausted server bandwidth resources, resulting in the website, applications can not open.  3）the server has some kind of self-replicating virus or Trojan, causing the hard disk resources and memory resources to be exhausted and causing the server to crash. |

|  |  |
| --- | --- |
| **（1）原文** | List an example of a mapping of architectural elements that has strong security implications. Hint: think of where data is stored. |
| **（2）翻译** | 绘制一个具有重大安全隐患的架构元素的制图  提示：考虑数据的存储位置。 |
| **（3）解答** | Memory. Determine how alternative mappings of architectural elements that are under consideration may change how an individual or system may read, write, or modify data; access system services or resources; or reduce availability to system services or resources. Determine how alternative mappings may affect the recording of access to data, services or resources and the recognition of unexpectedly high demands for resources. |

|  |  |
| --- | --- |
| **（1）原文** | Which of the tactics in our list will protect against an insider threat? Can you think of any that should be added? |
| **（2）翻译** | 我们列表中有哪些策略可以防范内部威胁？你有补充的策略吗？ |
| **（3）解答** | Allocation of Responsibilities, Coordination Model, Data Model, Mapping among Architectural Elements, Resource Management, Binding Time, Choice of Technology.  Add: sensitive access credentials control, the nature of those privileged users. |

|  |  |
| --- | --- |
| **（1）原文** | In the United States, Facebook can account for more than 5 percent of all Internet traffic in a given week. How would you recognize a denial-of-service attack on Facebook.com? |
| **（2）翻译** | 在美国，在给定的一周内，Facebook的流量可占全网的5％以上。  你如何识别Facebook.com上的拒绝服务攻击？ |
| **（3）解答** | Large companies like Facebook are very likely to suffer DDOS attack from hackers or competitors. Therefore, the detection of DDOS attack is very important. Nowadays, Soft Computing or Artificial Intelligence-based methods are applied extensively for the attack detection. On the basis of analysis methods, detection approaches are classified into Signature-based, Anomaly-based and Hybrid detection.   1. Signature-based detection   Signature-based detection is also known as Misuse detection, Pattern detection,Knowledge-based or Rule-based detection. This approach captures the required behavior from the available datasets (such as protocol stipulations, network traffic occurrences) and also collects facts about various attacks and system exposures. It uses the acquired expertise to identify the occurrence of anomalous events and produces an alarm if an attack is identified. It uses an index of ‘signatures or patterns’ of public attacks and matches the incoming traffic with the stored patterns to identify the attack instances. Expert systems is a specific example, this approach builds a set of rules to specify the well-known attacks and draw some conclusions from the rules and facts. Then the incoming traffic instances are matched against the rules to check whether any rule is satisfied and detect the inconsistent behavior of the system. But this approach requires re-building the rule frequently to assist the newly discovered vulnerabilities.   1. Anomaly-based detection   Anomaly-based detection approach (also known as novelty detection, outlier detection, behavior based or one-class learning scheme) is capable of detecting new, unknown and novel (unidentified) attacks. This approach mirrors the standard network behavior and compares it with the incoming data instances. When the divergence between an observed and expected behavior surpasses a predefined threshold, the detection system generates an anomaly alarm; hence an attack is disclosed. Specifically, point anomaly-based detection is a common approach. If a single data instance is considered as an anomaly as compared to the remaining dataset, then the approach is known as Point anomaly-based detection. Nowadays, it is the most significant and interesting field of the research on anomaly-based detection.   1. Hybrid detection   A mixture of above two approaches, also the most efficient way. |

|  |  |
| --- | --- |
| **（1）原文** | The public disclosure of vulnerabilities in production systems is a matter of controversy. Discuss why this is so and the pros and cons of public disclosure of vulnerabilities. |
| **（2）翻译** | 生产系统中的漏洞是否需要公开是一个有争议性的问题。讨论为什么是否公开生产系统中的漏洞是一个有争议性的问题以及公开漏洞的利弊。 |
| **（3）解答** | Vulnerability disclosure is the practice of reporting security flaws in computer software or hardware. Vulnerability disclosure and how it is performed can be a controversy issue because vendors prefer to keep the vulnerability under wraps until they have a patch ready to distribute to users. Conversely, researchers and security professionals, as well as enterprises whose data or systems may be at risk, prefer that disclosures be made public sooner.  **Pros:**   1. If the vendors know that complete vulnerability details have been, or soon will be, made public they hurry up creating patches. There is however a risk that the vendor will be stressed to release a patch before it is really thought through and tested. The patch may then not fix the problem completely, or cause compatibility problems. 2. If an administrator knows that there are complete vulnerability details made public, this increases the chances that he/she will take the problem seriously and really apply the provided patches. There are many reasons for an administrator not to apply all available patches. They include worries that the patches will introduce new errors into the system, a high work load, plain laziness, and that patches for example the OS are not fully supported by application program vendors. Knowledge about the fact that vulnerability details are in circulation out there also gives the administrator an argument against management/vendors for more resources in security issues. 3. Those who create security scanners need as detailed descriptions of new vulnerabilities as necessary. 4. Those who do penetration testing need as detailed descriptions of new vulnerabilities as necessary. 5. If the vulnerability details are published, all the developers in the world can learn from them and try not to make the same mistakes again. 6. An attacker usually has a lot of spare time to find out the details of a new vulnerability, but an over-worked administrator usually doesn’t. Full disclosure will in this case be more of benefit to the administrator than to the attacker. 7. An administrator might use vulnerability details to find similar vulnerabilities in other systems. If the same administrator only has a patch that “does something to increase security”, he/she will not have a chance to notice that a similar vulnerability exists in another system.   **Cons:**   1. If the vulnerability details had not been published, there might not have been a single attack performed using the specific vulnerability. The problem is that you can never be sure of that until there are no more affected systems in use anywhere. There is also a great risk that the really dangerous attackers are never noticed, because they do not deface web sites or other obvious things. 2. A lot of script-kiddies without any detailed knowledge can now exploit the vulnerabilities. If the details had been kept secret, they would not have the resources to do it. 3. The vendors may be stressed to provide patches that are not really thought through and sufficiently tested. That may even be worse than providing no patches at all in some cases. 4. When vulnerability details are first published, there will be a flood of attacks against affected systems. |

## 第十章

|  |  |
| --- | --- |
| **（1）原文** | A testable system is one that gives up its faults easily. That is, if a system contains a fault, then it doesn’t take long or much effort to make that fault show up. On the other hand, fault tolerance is all about designing systems that jealously hide their faults; there, the whole idea is to make it very difficult for a system to reveal its faults. Is it possible to design a system that is both highly testable and highly fault tolerant, or are these two design goals inherently incompatible? Discuss. |
| **（2）翻译** | 可测试性系统可以很容易的在问题出现时中断，也就是说，当系统存在一个故障时，则不会花费很长时间或很多努力来显示故障。在另一方面，故障容错就是设计那些小心翼翼地隐藏故障的系统，从而使得整个的目标就是使系统很难记录自己的故障。那么设计一个有很高测试性又有很高容错性的系统可能吗？还是这两个设计目标固有就是相斥的？请讨论。 |
| **（3）解答** | 我们小组最后的讨论结果是可以的，讨论内容与过程均陈述在下面的阐述中，理由如下：  首先要解答这个问题，我们小组先对这两个名词进行了定义。软件的可测试性是指软件发现故障并隔离、定位其故障的能力特性，以及在一定的时间和成本前提下，进行测试设计、测试执行的能力。简单的说，可测试性就是软件会将自己的错误暴露在人们的面前。容错性，是指软件检测应用程序所运行的软件或硬件中发生的错误并从错误中恢复的能力。梁健恒同学在这时候说了很有意思的话：可测试性就是女生素颜朝天出门，容错性就是化了妆再出门。仔细一想，还觉得颇有道理。  在讨论过程，我认为可测试性更倾向与产品在发布之前，公司内部的测试，相当于一种自我检查。黄同学也提出测试在产品发布之后也同样存在，比如一个版本的更新也需要进行测试。同时他也指出容错性并不是指容许所有故障，而对一些系统可忽略或者可自行处理的故障进行容许。这个观点我们的组员均表示赞同。  接下来对于两者能不能兼容，梁同学提出的是反对意见，他认为两者本身就是矛盾的，一边希望系统把错误暴露出来，一边有希望系统能隐藏一些错误，这已经不是鱼和熊掌能不能兼得的问题了，他认为这问题就像是找一个数既是正的又是负的。黄同学认为两者可以兼得，他提出两者强调的不同的方面，可测试性的目的是为了让软件能正常的运作，容错性的目的也在于此。同时，容错性的定义中包含了让系统自行处理故障，比如死锁，让系统能正常工作。既然系统能自行解决这些故障，那么将这些故障报告出程序员也不是很需要，这些故障也不需要被可测试性所检出。我同时也提出，并不是可测试性系统所报出的故障都能被程序员所完美解决的，比如死锁，因为每台计算机的配置各部相同，内存不同，处理系统位数不同等等，所以每台计算机处理死锁的最佳方法也不尽相同，交给计算机自行处理或许会更好。然后我们组对死锁这个问题又进行了一番探讨，赵同学觉得我对死锁的认识是错的，然后我们又拿出上学期的操作系统，又仔细研读了一便死锁问题，最后我终于意识到死锁并不是本道问题的关键，才把话题拉回到正题上来。  随后我们交流的关键点在于对故障的分类，哪些是可自行解决的，哪些是人为介入的，哪些又是可测试性系统不需要报告的，随后我们做了如下总结。  可测试性和容错性是可以兼容的。系统之所以会容错，是因为那些故障能被系统自行解决，或者这些无视这些故障，也不会很大程度上影响系统的运行。而可测试性的目的是寻找那些会使系统崩溃或者错误输出的故障，从而进行认为修改和完善。两者所针对的故障类型不同，为了达到的目的是相同的，反正应该能相得益彰。 |

|  |  |
| --- | --- |
| **（1）原文** | “Once my system is in routine use by end users, it should not be highly testable, because if it still contains faults-and all systems probably do-then I don’t want them to be easily revealed.” Discuss. |
| **（2）翻译** | “当我的系统在被终端用户日常使用时，系统不应该具有很高的可测试性，因为就像大多数系统一样，系统会存在故障，而我并不希望故障能被轻易的揭示。”请进行讨论。 |
| **（3）解答** | 对于本道题，我们组的意见也有所分歧，有人认为这是一个不好的行为，是对用户不负责的行为。虽然表面上看起来，是为了让用户有更好的用户体验，但是可能实际上会对用户带来不小的损失，比如导致输出的错误，游戏进行到一半闪退导致没有存档或者无法重连等等。这是一种对用户非常不负责任的行为。  看到这个题目时，梁同学就特别激动，然后就跟讲述了自己在玩pc端吃鸡时的经历，他说：这蓝洞是真的不行，做一个游戏每天要么卡bug，要么游戏闪退，有一次我都在决赛圈了，然后突然游戏崩了，那时候我都想把电脑砸了。此时，赵同学表示深深的赞同，说有一次他玩的时候，卡在一个缝里，怎么也走不出来，队友用雷炸也不行，用车开到旁边来接也不行，最后活生生被毒死，他还强调那把他已经捡到了一把AWM，觉得吃到鸡已经十拿九稳了。然后我就问他们：你们说的和这道题有什么关系呢？  后面梁同学才意识到前面说的比较多余，随后点到正题说，游戏崩溃后，会弹出一个提示框，里面包含了错误信息，玩家可以选择“close with send”，就是将错误信息返回给设计商；以及“close without send”。梁同学认为设计商有了这样的反馈，就能更好的进行优化更新，使游戏更少的崩溃，这样才能真正提高玩家的用户体验。如果让故障隐藏起来，可能这游戏的完善道理就会变得更加艰难。  但是黄同学认为这是一个好的特性，应为现在仍有许多不法分子，会钻一些软件的漏洞，如果这个系统具有良好的可测试性，这就方便不发份子对软件进行测试，从而找出软件的漏洞，攻破系统的防线，对设计商造成不可估量的损失，也有可能对使用者造成一定的财产损失，就如最近的支付宝在appstore上被盗刷。  紧接着我们又进行了一些激烈的争论，但是最后谁都没有说服谁，得出的结论是这样的特性有两面性。总结如下：系统在被终端用户使用时，如果具有很高的可测试性，则用户的反馈可以提高系统完善的速度，提高系统的质量；但与此同时，也给不法分子可乘之机，尤其是那些运用智商犯罪的人，尤为可怕，带来的后果也不堪设想。所以就需要设计商和用户两方都多留一个心眼，共同推动系统的进步。 |

|  |  |
| --- | --- |
| **（1）原文** | Many of the tactics for testability are also useful for achieving modifiability. Why do you think that is? |
| **（2）翻译** | 许多用于实现可测试性的方法对实现可修改性也非常有用，你认为是为什么呢？ |
| **（3）解答** | 我们小组的成员都认为用于实现可测试性的方法比如控制、观察系统状态和限制复杂度和用于实现可修改性的方法有很多相似之处，比如说缩小模块的规模。张扬和赵亮认为实现可测试性的关键是迅速地找到程序为何发生错误，而实现可修改性的关键在与减少修改时的工作量，而限制复杂度可以让程序更为简洁明了，对于修改和找出错误有很大帮助。而黄旸珉认为可修改性要求的低内聚和高耦合目的是要将程序的各个部分拆分开，每个部分内部更集中，而这也符合可修改性中容易找出错误的目的。我认为能控制各部分来观察其对整个程序的影响可以允许程序员在发现错误后迅速地对错误代码作出修改，这也正好满足了可修改性的要求。其余三位组员同意我的观点的同时指出可修改性要求程序员能以较高的性价比对系统进行变更，因此程序的合理分割和尽可能地简化将发挥很重要的作用。但是我认为过度地简化将会影响程序功能的正常运行和结构的严密性。在经过讨论后我们达成共识：合适的简化能在保证程序功能正常运作的同时达到可修改性和可测试性。赵亮通过研究常见的软件可测试性检查表中元素：可操作性、可观察性、可控制性、可分解性、简单性、稳定性、易理解性。提出简单性中的“需要测试的内容越少，测试的速度越快”与可修改性缩减模块规模的理念相通，因此应用在实现可测试性的方法对于实现可修改性也很有帮助。同时，张扬指出可分解性对于测试范围和问题分解的要求也使软件要具有各部分易于拆分、同一部件高度内聚的特点，这与实现可修改性的低耦合、高内聚方法不谋而合。最后我们得出结论：因为可测试性与可修改性在要求、特点上的相似性决定了实现它们的方法对互相都有作用。 |

|  |  |
| --- | --- |
| **（1）原文** | Write some concrete testability scenarios for an automatic teller machine. How would you modify your design for the automatic teller machine to accommodate these scenarios? |
| **（2）翻译** | 为自动柜员机写一些具体的可测试性应用场景，你会怎样修改自动柜员机的设计以使它适用于那些场景？ |
| **（3）解答** | 应用场景1  Source:------------>Artifact:------------->Response Measure:  Unit Stimulus: Code Response: 90% Path Coverage in 2.5 Hours  Tester Code Unit Result  Unit Captured  Completed  应用场景2  Source:------------>Artifact:--------------->Response Measure:  End Stimulus: Whole Response: Successfully  Users Whole System Activity Withdraw  System Captured Within  Function 30s  我们在讨论后得出以下结果：要使ATM适应这些应用场景，需要加快ATM内部数据的传输，尽量减少验证银行卡所需的时间和验证密码所需的时间。张扬认为减少时间的关键在与使用新的密码验证方法，我同意他的观点并提出优化ATM机的交互界面也可以大幅度地减少存取所需要的时间。赵亮和黄旸珉在了解现在流行的ATM机系统后，提出当前ATM机受限于较旧的操作系统，存取速度往往不尽人意，因此操作系统的更新也是必不可少的。目前ATM机上装载的都是WindowsXP系统，鉴于自身经济价值以及质量稳定性的影响，ATM机更新换代的速度远低于软件的更新速度。我提出WindowsXP系统在当前显得比较老旧，有必要升级到Windows 7以上的版本，提高ATM机处理用户需求的速度。然而其他三位组员反对我的提议，虽然Windows 7应用在ATM机上不存在安全问题，但是很多设备根本不支持Windows 7运行，甚至没有Windows 7的驱动和接口。因此升级系统并不是一个合理的方案，在经过我们的讨论后，更新优化程序是比较合理的方案。目前的ATM机交互界面比较死板，对于用户交互并不友好，黄旸珉提出我们可以针对这点来对ATM系统进行优化，比如说重新设计界面等。另外张扬提出可以将ATM机从按键式改进到全触屏式，更加方便用户的使用。但是这个提议遭到了我的质疑，我认为这样的改动花销会很大，鉴于按键式ATM机的普及程度，更新ATM机会增加银行的成本。所以我们最后的讨论结果是从ATM机的UI着手，重新设计界面，提升用户友好度，以减少用户存取的时间。 |

|  |  |
| --- | --- |
| **（1）原文** | What other quality attributes do you think testability is most in conflict with? What other quality attributes do you think testability is most compatible with? |
| **（2）翻译** | 你认为有什么其他的质量属性是和测试属性最为冲突的吗？  你认为有什么其他的质量属性和测试属性是最为相容相似的吗？ |
| **（3）解答** | I think the most conflict attribute with testability is modifiability, because modifiability breaks the stabilization of the testability, and it means changing and extra work for testability and make it more difficult to realize, also, testability will require data and functions to be not “modifiable” because it will widen the range of the testability.  I think the most compatible attribute with testability is uasbility, because usability needs testability to provide and secure the usability for the user, also the testability aims at providing good usability for the users.  张扬认为和质量属性Testability最相容相似的应该是performance，因为Testability的质量决定了performance的表现，如果测试质量不佳，软件的performance表现也会不行。而梁健恒认为availability更相容相似一点，因为testability能够贯彻实现Availability的定义内容，避免不期望的事件发生。赵亮则比较赞同张扬的意见,因为testability的目的也包括了提升performance的表现。 |

|  |  |
| --- | --- |
| **（1）原文** | One of our tactics is to limit nondeterminism. One method is to use locking to enforce synchronization. What impact does the use of locks have on other quality attributes? |
| **（2）翻译** | 我们其中的一个策略就是限制不确定性。其中一个方法是使用锁来强制信息同步。使用锁会对其他质量属性有什么影响呢？ |
| **（3）解答** | I think locks will have impacts on these attributes: For Availability, it will reduce the errors due to the elimination of nondeterminism; For interoperability, locks will weaken it because of the forcing synchronization, all parts cannot work well together due to the forcing snchronization; for security, forcing synchronization eliminates the situation that changing after submission.  张扬认为使用Locking会影响到Availability的响应时间等，因为强制信息同步必然导致增加了同步时间等，使得Availability的响应时间增加；赵亮认为Locking会提升Performace的表现，因为减少数据丢失的可能性，但与此同时也会增加等待时间。梁健恒比较同意赵亮的意见，但认为Locking也影响了performance的等待，吞吐等表现。 |

|  |  |
| --- | --- |
| **（1）原文** | Suppose you’re building the next great social networking system. You anticipate that within a month of your debut, you will have half a million users. You can’t pay half a million people to test your system, and yet it has to be robust and easy to use when all half a million are banging away at it. What should you do? What tactics will help you? Write a testability scenario  for this social networking system. |
| **（2）翻译** | 假定你正在打造下一个伟大的社交网络系统。你预期在产品上线后的一个月内，你会拥有50万用户。你显然不能让这50万人有偿为你测试系统功能，但是你的系统必须具有鲁棒性且易于使用以应付50万人同时访问。你应该怎么做？什么样的策略会帮助你。为这个社交网络系统写一个可测试场景 |
| **（3）解答** | 在上线产品前，应该进行相应的软件测试。首先开发团队要做单元测试来判断软件的各个独立单元是否符合功能性需求以及集成测试来判断软件能否适应运行环境和基础设施，以及多个组件能否正常协同工作。因为我们致力于打造一个伟大的社交网络系统，软件的系统必然庞大，状态空间很多不利于观察，所以最好的策略还是限制软件复杂程度，控制架构复杂性和不确定性。  小组讨论：  ZY和HYM在开始都认为控制和观察系统状态是一个更好的策略来维护系统的可测试性，因为这是一个通用且易于实现的方法  但ZL和LJH认为我们的软件系统是一个功能繁多的复杂系统，如果不在初期设计让软件保持一个简单的架构，而是不断地通过更新来修修补补，对长期发展来言是不好的，庞杂的系统是很难测试和维护的。所以，限制软件复杂程度，控制架构复杂性和不确定性是一个更好的策略 |

|  |  |
| --- | --- |
| **（1）原文** | Suppose you use executable assertions to improve testability. Make a case for, and then a case against, allowing the assertions to run in the production system as opposed to removing them after testing. |
| **（2）翻译** | 假定你使用了一个可执行的断言来提高可测试性。 请分别提出一个充分的理由来支持和反对，“允许断言运行在产品系统中，而不是在测试之后删除它们”。 |
| **（3）解答** | 断言不应该运行在一个发布的商业产品中，因为断言是处理来自内部的错误数据的，是处理程序员的错误的，且频繁的调用会极大的影响程序的性能，增加额外的开销而一个优秀的商业产品是不应该出现这样的问题的。再好的产品也是会有bug的，程序员总是会犯错的，让断言运行在部署代码中，并且在字段中打开以利于调试。  小组讨论：  ZL和ZY提出的支持理由是，断言在经过测试后是没有意义的，一个Release版本应该专注于用户体验，而不是程序员调试。而且hit assert后，在 debug 版本里是触发 debug crash，能被 debugger 捕获到。在 release 版本里不能这么做。理想的做法是要建立一个服务，把 release assert 的信息收集起来。但是这种 infrastructure 的建立很费功夫，没有意义。  HYM和LJH提出的反对理由是, assert可以在release版本不启用，但不应该移除，而且现在支持release-enable版assert越来越多。人们也越来越重视软件安全问题，但软件发布周期也越来越短，几天一个小版本非常寻常，除非有很规范、成熟的自动化测试，否则测试的覆盖率比传统软件差远了，所以不得不用像assert这样的防御式编程，很多时候不得不靠运行时检查来保证不出严重错误。 |

## 第十一章

|  |  |
| --- | --- |
| **（1）原文** | Write a concrete usability scenario for your automobile that specifies how long it takes you to set your favorite radio stations? Now consider another part of the driver experience and create scenarios that test other aspects of the response measures from the general scenario table. |
| **（2）翻译** | 为你的汽车写一个具体的可用性场景并在其中指明设置自己最喜欢的广播电台需要多长时间？ 现在考虑司机体验的另一部分，并对一般场景表中提到的测试响应度量的其他方面创建具体的场景。 |
| **（3）解答** | First question:  After long driving, a driver was too tired so he stopped his car on the roadside and wanted to open radio to listen to his favorite station, which plays some beautiful music. He opened the power, then the radio, then started rotating switch to search all station one by one. Until he found his favorite one, the whole process lasted five minutes.  Second question:  Number of errors:  A driver found the radio fail to work, so he closed all the device and stalled the car, then he reopened it and tried it again, but unfortunately, he cannot start the car, causing it was out of gas.  Number of tasks accomplished:  The driver walked to the nearest gas station and bought gas, then he successfully started the car and radio, driving away with music in the air.  User satisfaction:  This driver was satisfied that the radio was not broken though he had bought the car for a very long time. The only unsatisfying point was it always spent him five minutes every time he found that radio station, and he wished what if the car can remember which station he liked best.  Gain of user knowledge:  He kept his thought and went to the manufacturer in reality. He suggested the boss add a button on the radio, which can directly find the most-listened radio channel. That man thanks him for providing this good idea and told him there was already a team working on it.  Ratio of successful operations to total operations:  Buying it such a long time, this car had a few problems. For example, it cannot successfully be started for the first try, the ratio is nearly 70%, and radio was nearly no reaction when opening it, with ratio only 20%.  Amount of time or data lost when an error occurs:  Every time when failing to start the car, driver needed to wait about three minutes to try again. And radio became more and more unclear even though it can find some station. |

|  |  |
| --- | --- |
| **（1）原文** | Write a concrete usability scenario for an automatic teller machine. How would your design be modified to satisfy these scenarios? |
| **（2）翻译** | 为自动取款机写一个具体的可用性场景。你会怎么修改自己的设计来满足这些场景？ |
| **（3）解答** | For example, one day, there came an old man whose sons and daughters were not nearby but his wife was sick on Saturday and he needed to get money from ATM. Unluckily, he did not know how to use and there was nobody nearby.  For this scenario above, I would modify my design in the following way. Elders are almost far away from high technology, so to make it easy for them, ATM can be designed more intelligent which can understand human language. In this way, elders just need to say what they want to do, the ATM would tell them step by step how to complete this task.  Again, assume one day, a boy was asked by his parents to get money from ATM by himself. This boy brought the credit card and knew the password, but he was not tall enough to put in the card and input the password.  For this new scenario, I would design ATM with lifting function. By putting a sensor at the top of ATM, when people come close, it would detect whether a adult or a child comes, then it would automatically adjust its height to make customers comfortable.  In another example, here came a handicapped people who, unfortunately, did not have arms and was dumb, and knew nothing about numbers. He came to get the money for his operation which can help him speaking.  This scenario is quite extreme. For such customers, I would modify ATM design in the following way. Considering this man cannot speak or take the credit card out of his trouser pocket, we can add an embedded system whose position is just at the same height as his pocket. He can walk close to the scanning plate, which can read the credit card though having clothes between them. After getting the data on the credit card, on the floor where he stands, using a pressure sensor, we can make those options on the screen under his feet. He can tread the function he wants to do and input password using feet, too. |

|  |  |
| --- | --- |
| **（1）原文** | How might usability trade off against security? How might it trade off against performance? |
| **（2）翻译** | 实用性和安全性之间应该如何权衡？同样的，在实用性与使用性能之间又该如何权衡呢？ |
| **（3）解答** | In terms of password management, in modern life, password management software is actually used by everyone, but many people do not know how to use it. At present, many password management software can be found in various application markets. If classified according to storage methods, they can be basically divided into two types: local storage and cloud storage. The advantage of local storage is that it is highly secure and has absolute control over data files. Because it is a local storage, the backup of the data is very important. In order to save the trouble of backup and synchronization, some people will choose password management software that supports cloud storage. The advantage of cloud storage-based password management software is that it is easy to use, basically no manual backup and synchronization, but the security is low, and the cloud data has the possibility of leakage. If you want to use cloud storage, it is recommended to use a complex master password to secure your data. |

|  |  |
| --- | --- |
| **（1）原文** | Pick a few of your favorite web sites that do similar things,such as social networking or online shopping.Now pick one or two appropriate responses from the usability general scenario (such as “achieve the task at hand”) and a correspondingly appropriate response measure. Using the response and response measure you choose, compare the web sites’ usability |
| **（2）翻译** | 选择一些你最喜欢做类似事情的网站，比如社交网络或网上购物。现在从可用性一般情景中选择一两个适当的响应（例如“实现手头的任务”）和相对应的响应措施。 使用您选择的响应和响应措施，比较网站的可用性。 |
| **（3）解答** | In my opinion, take Tmall and Jingdong self-employed, although there is no essential difference, there are still differences in the details of some responsive shopping processes. When Tmall makes a settlement, it will automatically select the matching offer option, and the response will get a relatively lowest price. Jingdong needs individuals to make a comparison of choices. From this point of view, the user experience is relatively low.terminated immediately, so that the user would not wait for his cancel command. |

|  |  |
| --- | --- |
| **（1）原文** | Specify the data model for a four-function calculator that allows undo. |
| **（2）翻译** | 为四则运算计算器提出允许撤销操作的数据模型。 |
| **（3）解答** | We can use a stack to implement undo. Every time the user raises a command for the four-function calculator, for example, add, subtract, multiply or divide a number, we push it into the stack, with operator and number. When the user wants to perform undo, we pop a previous command from the stack, and revert the operation. For instance, add become subtract, multiply become divide, and do the reverted operation on the current result, then we successfully undo it and the user would become satisfied. In order not to overflow the memory, we would limit the stack size, when the stack is full, we delete the most previous command in the stack. |

|  |  |
| --- | --- |
| **（1）原文** | Why it is that in so many systems, the cancel button in a dialog box appears to be unresponsive? What architectural principles do you think were ignored in these systems? |
| **（2）翻译** | 为什么在如此多的系统中，对话框中的取消按钮似乎并不能很快地响应?你认为这些系统中哪些架构原则被忽略了？ |
| **（3）解答** | In my opinion, the situation occurs mostly when a complex task is performing by the system, the task might be cut in to some parts and the cancel command could only be performed between two parts. Then the user would find the operation is unresponsive.  I think the cancel of Support User Initiative is ignored. The system must be listening for the user’s cancel command so that it would not be blocked by other actions. The command being canceled must be terminated immediately, so that the user would not wait for his cancel command. |

|  |  |
| --- | --- |
| **（1）原文** | Why do you think that progress bars frequently behave erratically, moving from 10 to 90 percent in one step and then getting stuck on 90 percent? |
| **（2）翻译** | 你认为为什么进度条经常表现得无规律性，比如进度一步就从10%到了90%，然后卡在了90%？ |
| **（3）解答** | Because in most cast, progress bars not reflect the real processing progress. They are just designed to satisfy users’ mental requirements. Therefore, after moving from 10 to 90 percent in one step, progress bars can make users feel the program is loaded pretty quickly. Actually, it’s far away from accomplishment. But in this way, users can get good response and prefer to wait a few seconds more so that the usability is improved compared to the normal progress bars, moving from 0 to 100 step by step.  Sometimes the progress bars represent the number of processing tasks not the time needed. It will also get to this circumstances because the last task is complex to finish. |

|  |  |
| --- | --- |
| **（1）原文** | Research the crash of Air France Flight 296 into the forest at Habsheim, France, on June 26, 1988. The pilots said they were unable to read the digital display of the radio altimeter or hear its audible readout. If they could have, do you believe the crash would have been averted？In this context, discuss the relationship between usability and safety. |
| **（2）翻译** | 调查研究1988年6月26日法国航空296航班坠毁在法国Habsheim森林中得原因。飞行员曾说他们无法读取无线高度计量表上的显示数字也无法听到其声音读书。试想如果他们当时可以，你是否相信事故可以避免？讨论分析这种环境下可用性和安全性之间的关系。 |
| **（3）解答** | I hold the view that if they could have read the digital display of the radio altimeter or hear its audible readout, the crash would have been averted.  Usability ensures safety to some degree. When it comes to some significant case like pilot operating system during flight, usability is crucial to ensure the public safety. Users should get timely and accurate response.  Safety regulates the level of usability. The usability of a program is defined by its safety. When the safety is important and has a great impact, according to which we have to improve the usability. |

## 第十二章

|  |  |
| --- | --- |
| **（1）原文** | 10. Relate monitorability to observability and controllability, the two parts of testability. Are they the same? If you want to make your system more of one, can you just optimize for the other?  1. The kingdom of Bhutan measures the happiness of its population, and government policy is formulated to increase Bhutan’s GNH (gross national happiness). Go read about how the GNH is measured (try [www.grossnationalhappiness.com](http://www.grossnationalhappiness.com)) and then sketch a general scenario for the quality attribute of *happiness* that will let you express concrete happiness requirements for a software system.  2. Choose a quality attribute not described in Chapters 5-11. For that quality attribute, assemble a set of specific scenarios that describe what you mean by it. Use that set of scenarios to construct a general scenario for it. |
| **（2）翻译** | 10. 对比可监测性和可测试性的两个部分，可观察性和可控性。它们是一样的吗？如果你希望你的程序同时有这两个质量属性，你能否优化另一个质量属性？  1. 不丹王国希望了解国民的幸福程度以制定能够增加国民幸福总值（GNH）的政策。有关GNH的信息（[www.grossnati](http://www.grossnationalhappiness.com)[onalhappiness.com](http://www.grossnationalhappiness.com)）。描绘一个有关幸福的质量属性，能让你对软件系统表达具体的幸福需求的场景。  2. 选择一个没有在5-11章中出现的质量属性。请用几个场景来说明这个质量属性。使用这几个场景构建通用的场景。 |
| **（3）解答** | 10. 不相同，可检测性应用于程序执行的期间，以便让服务器和程序的运维人员能够通过程序的执行的一些反馈信息采取相应的措施。而可观察性和可控制性则在程序的测试期间，测试人员需要程序测试期间的大量信息，以便减少程序的错误，让程序能够发布使用。可以在软件测试期间同时设计一个监测应用程序，并在应用程序代码中加入反馈数据给服务器的代码，以便监测应用程序使用。  1. 幸福质量属性衡量国民的幸福指数国民幸福总值（GNH）是一种全面和可持续的发展方式，它将物质和非物质价值观与人类想要寻求幸福的信念相平衡。GNH的目标是在生活的各个方面实现必要的平衡发展。该质量属性的四个重要支柱为，良好的治理，可持续的社会经济发展，文化的保护和促进，环境保护。幸福质量属性涉及到生活水平，教育，健康，环境，社区，时间，心理健康，社会治理，文化。刺激源：政府。刺激：国民幸福的需求。人工制品：国民幸福总值。环境：国家运行期间。反馈：人民的各种数据。应对措施：实施相应政策以提高国民信服指数。幸福质量属性策略：人民数据输入，政府收集处理数据，制定相应对策，输出新政策。  2. 可监测性。运维人员通过应用程序实时运行期间产生的错误进行收集，使开发组能开发更为优良的版本。应用程序收集用户数据，反馈给服务器，实现诸如个性化和推荐等功能。运维人员根据程序反馈，做出例如手动修改数据库，修改应用程序数据等操作。通用场景：刺激源：运维人员。刺激：应用程序返回数据。人工制品：返回的数据。环境：程序运行期间。反馈：程序行为可以被监测。应对措施：收集bug，开发新版本，能够即时处理的运维人员即时处理，个性化和推荐。 |

|  |  |
| --- | --- |
| **（1）原文** | 1. The Kingdom of Bhutan measures the happiness of its population, andgovernment policy is formulated to increase Bhutan’s GNH (gross nationalhappiness). Go read about how the GNH is measured (try www.grossnationalhappiness.com) and then sketch a general scenario for the qualityattribute of happiness that will let you express concrete happiness requirementsfor a software system.  2. Choose a quality attribute not described in Chapters 5–11. For that qualityattribute, assemble a set of specific scenarios that describe what you meanby it. Use that set of scenarios to construct a general scenario for it.  3. For the QA you chose for discussion question 2, assemble a set of designapproaches (patterns and tactics) that help you achieve it. |
| **（2）翻译** | 1. 不丹王国为衡量其人口的幸福，政府制定政策以提高不丹国民幸福总值。去了解GNH是如何测量的（请访问www.grossnational.com），然后勾画一个关于幸福质量的一般场景，能让你表达一个软件系统的具体幸福需求。  2. 选择第5章至第11章中未描述的质量属性。对于那个质量属性，组装一组特定的场景来描述你的意思。使用这组场景来构建它的一般场景。  3. 对于您为讨论问题2选择的QA，组装一组帮助您实现它的设计方法（模式和策略）。 |
| **（3）解答** | 1.  GNH（国民幸福指数）是指反映国民生活质量和幸福程度的指标。不丹的GNH（[国民幸福指数](https://www.baidu.com/s?wd=%E5%9B%BD%E6%B0%91%E5%B9%B8%E7%A6%8F%E6%8C%87%E6%95%B0&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)）测量，主要考察四个方面：可持续发展、环境保护、文化保护以及政府的有效管理。  幸福质量的一般场景：  场景的部分： 可能的值：  刺激源： 政府  刺激： 制定政策  环境： 不丹国家  制品: [国民幸福指数](https://www.baidu.com/s?wd=%E5%9B%BD%E6%B0%91%E5%B9%B8%E7%A6%8F%E6%8C%87%E6%95%B0&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)  响应： 政府指定政策并实施来提高国民幸福指数  响应度量： 政策执行后国民生产总值指数、社会健康指数、社会福利指数、社会文明指数和生态环境指数  2.  Scalability（可扩展性）  可扩展性可分为水平可扩展性和垂直可扩展性。  特定场景：可以是向服务器集群添加另一台服务器，或者向单个计算机添加更多的内存。这将出现如何有效地利用额外资源的问题。  一般场景：  场景的部分： 可能的值：  刺激源： 最终用户、开发人员  刺激： 添加或删除资源  环境： 运行时、构建时、设计时  制品: 系统用户平台、界面和系统服务器  响应： 处理添加或删除资源的影响  响应度量： 分配给现有和新资源的负载  3.  可扩展性是一种对软件系统计算处理能力的设计指标。可用Shared Space（共享空间）的设计模式，所有工作者线程通过共享空间共享数据，来协调各个线程的并发和数据分隔任务分隔问题。 |

|  |  |  |  |
| --- | --- | --- | --- |
| **（1）原文** | 1. For the QA you chose for discussion question 2, develop a design checklist for that quality attribute using the seven categories of guiding quality design decisions outlined in Chapter 4. | |  |
| **（2）翻译** | 对于你在讨论问题2中选择的QA，使用在第4章中概述的指导质量设计决策的七个类别，为该质量属性开发一个设计检查表。 | |  |
| **（3）解答** | Category | Checklist | | |  |
|  | Allocation of responsibilities（职责分配） | 通过考虑技术、法律、社会用途和客户的变化，确定哪些扩展是可能发生的：  -确定扩展的纵向或者横向的职责  -确定扩展后的影响的职责  -确定模块的职责分配 | | |  |
|  | Coordination model（协调模型） | 确定哪些功能或质量属性可以在运行时更改，以及如何影响协调，例如：在服务器集群中添加服务器会有什么变化，扩展要尽量影响少的模块  确定协调的设备、协议可能会扩展。 | | |  |
|  | Data model（数据模型） | 确定对数据资源抽象、其操作或其属性可能发生的更改，还要确定对这些数据抽象的扩展或扩展类别将涉及到创建、初始化。  对每次的扩展，确定扩展是由最终用户、系统管理愿或开发人员进行。  对于每个潜在的变化：  -确定需要修改的数据抽象  -确定对哪些数据抽象的创建、初始化、持久性、操作、转换或销毁是否有更改  -确保数据抽象的分配  设计数据模型，以便分配给数据模型的每个元素的项可能一起更改 | | |  |
|  | Management of resources（资源管理） | 确定职责或质量属性的修改将如何影响资源的使用  确定扩展对资源的添加或删除的影响，以及确保修改后的资源足以满足系统需求  封装所有资源管理器 | | |  |
|  | Mapping among architectural elements（在架构元素映射） | 确定对函数或质量属性扩展的范围，如：  -执行依赖关系  -处理添加或删除资源的影响  确保这些扩展是使用机制执行的 | | |  |
|  | Binding time decisions（绑定时间决定） | 决定最迟何时需要进行扩展  选择一种延迟绑定机制，在选择的时间交付适当的功能  确定这些扩展不会对系统造成不可接受的所害 | | |  |
|  | Choice of technology（技术选择） | 确定技术选择使哪种扩展更容易或更困难  -技术选择是否有助于进行、测试和部署修改  -修改技术的选择的容易程度  选择支持最可能修改的技术，如Shared Space | | |  |

|  |  |
| --- | --- |
| **（）原文** | 5. What might cause you to add a tactic or pattern to the sets of quality attributes already described in Chapters 5-11 (or any other quality attribute, for that matter)? |
| **（）翻译** | 什么可能会导致你将战术或模式添加到已经在第5-11章中描述过的质量属性集(或任何其他质量属性)? |
| **（）解答** | 该质量属性在新的场景中不适用，没有紧凑的知识体系；质量属性运用到新的模型中；实现特定质量属性时。 |

|  |  |
| --- | --- |
| **（）原文** | 6.According to slate.com and other sources, a teenage girl in Germany “went into hiding after she forgot to set her Facebook birthday invitation to private and accidentally invited the entire Internet. After 15,000 people confirmed they were coming, the girl’s parents canceled the party, notified police, and hired private security to guard their home.” Fifteen hundred people showed up anyway; several minor injuries ensued. Is Facebook “unsafe”? Discuss. |
| **（）翻译** | 据slate.com和其他消息来源称，德国一名十几岁的女孩“忘记把自己的Facebook生日邀请设为私人活动，不小心邀请了整个互联网，然后躲了起来”。在15000人确认他们会来之后，女孩的父母取消了聚会，通知警察，并雇佣私人保安来保护他们的家。反正还是有一千五百人来了;随后发生了几起轻伤事件。Facebook“不安全”吗? 请讨论。 |
| **（）解答** | 我们觉得Facebook就这个事件而言还是安全的，首先Facebook并无暴露用户信息，单纯是女孩错误操作而造成了一系列事故。就从发Facebook的角度来说，软件是无法准确识别操作者的主观意愿的，是否设置为私人并不在软件的安全范畴，纯属个人失误，不能怪这个软件不具备安全性。 |

|  |  |
| --- | --- |
| **（）原文** | 7.Author James Gleick C"A Bug and a Crash," www.around.com/ariane. html) writes that "It took the European Space Agency 10 years and S7 billion to produce Ariane 5, a giant rocket capable of hurling a pair of three ton satellites into orbit with each launch. ... All it took to explode that rocket less than a minute into its maiden voyage ... was a small computer program trying to stuff a 64bit number into a 16-bit space. One bug. one crash. Of all the careless lines of code recorded in the annals of computer science,this one may stand as the most devastatingly efficient." Write a safety scenario that addresses the Ariane 5 disaster and discuss tactics that might have prevented it.   1. Discuss how you think development distributability tends to "trade of" against the quality attributes of performance, availability, modifability, and interoperability.   Extra Credit: Close your eyes and, without peeking, spell dstributability Bonus points for successfully saying development distributability" three times as fast as you can.  9. What is the relationship between mobility and security? |
| **（）翻译** | 7.作者James Gleick 在《A Bug and a Crash》，www.around.com/ariane. html中写道：“欧洲航天局花了10年时间和7亿美元用于生产Ariane 5号，这是一种能够在每次发射时将一对3吨卫星投入轨道的巨型火箭......所有这一切都在火箭首航起飞不超过一分钟爆炸化为乌有......是一个小型计算机程序试图将64位数字填入16位空间。一个错误,引发了一个坠毁。在计算机科学史上记录的所有粗心的代码行中，这个可能会是最具破坏性的效率。“请写一个Ariane 5号灾难的关于安全的场景，并讨论可能阻止它的策略。  8.讨论您如何认为开发可分配性与质量属性性能，可用性，可修改性和互操作性的权衡。  提示：闭上眼睛，不用偷看，拼写可分发性有点，说“开发可分配性”三次。  9.移动性和安全性之间有什么关系？ |
| **（）解答** | The internal SRI software exception was caused during execution of a data conversion from a  64-bit floating-point number to a 16-bit signed integer value. The value of the floating-point  number was greater than what could be represented by a 16-bit signed integer. The result was an  operand error. The data conversion instructions (in Ada code) were not protected from causing  operand errors, although other conversions of comparable variables in the same place in the code were protected.  Safety  **Scenario**: The Ariane 5 disaster  **Source**: developer  **Stimulus**: a data conversion from a64-bit floating-point number to a 16-bit signed integer value  Environment: after liftoff  **Artifacts**: system  **Response**:avoid the conversion  **Response measure**:avoid the bug and make the rocket work as we expected.  Tactics:   1. Testability:Prepare a test facility including as much real equipment as technically feasible, inject realistic input data, and perform complete, closed-loop, system testing. Complete simulations must take place before any mission. A high test coverage has to be obtained   Review the test coverage of existing equipment and extend it where it is deemed necessary.   1. Safety:   Review all flight software (including embedded software), and in particular :   * Identify all implicit assumptions made by the code and its justification documents on the values of quantities provided by the equipment. Check these assumptions against the restrictions on use of the equipment. * Verify the range of values taken by any internal or communication variables in the software. * Solutions to potential problems in the on-board computer software, paying particular attention to on-board computer switch over, shall be proposed by the project team and reviewed by a group of external experts, who shall report to the on-board computer Qualification Board.   Development Distributability: is the quality of designing the software to support distributed software development.  Performance:It’s about time and the software system’s ability to meet timing requirements.  Availability:A property of software that it is there and ready to carry out its task when you need it to be.  Modifiability is about change, and our interest in it centers on the cost and risk of making changes.  Interoperability:The degree to which two or more systems can usefully exchange meaningful information via interfaces in a particular context.  Trade off: Distributed System(DS) is depend on the performance of the communication network, so the timing requirements of DS is limited. And the synchronize problem is a big problem, the Availability cannot be ensure. The logic behind system is hard to reason about so that modifiability is not so feasible. The distributability need many interface to guarantee, so interoperability need to be better.  Safety is about the software’s ability to avoid entering states that cause or lead to damage, injury, or loss of life to actors in the software’s environment, and to recover and limit the damage when it does enter into bad states.  Mobility deals with the problems of movement and affodances of a platform(e.g.. size, type of display, type of input devices, availability and volume of bandwidth, and battery life). Issues in mobility include battery management,reconnecting after a period of disconnection, and the number of different user interfaces necessary to support multiple platforms. Scenarios will deal with specifying the desired effects of mobility or the various affodances. Scenarios may also deal with variability, where the same software is deployed on multiple (perhaps radically different) platforms.  The Mobility makes the software’s environment more diverse, so that the safety problem is more capable to take places and it’s more difficult to guarantee safety. |