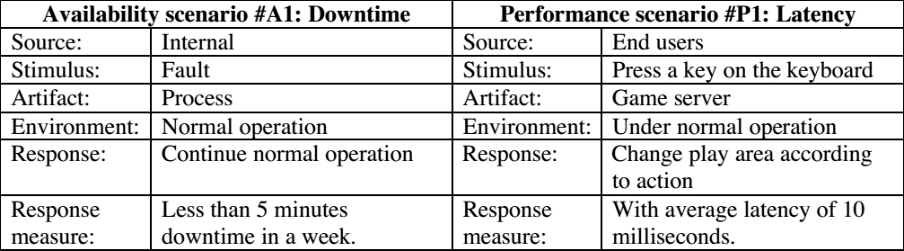
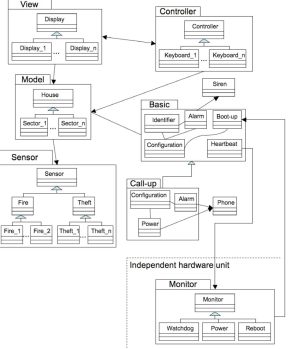
**06 1.列出应该在设计模式描述中的五个项目(部分)，并描述为什么这些项目(部分)在设计模式描述中非常重要**

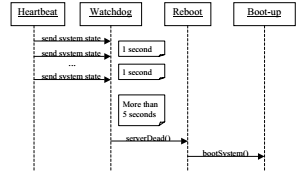
模式名称和分类，•意图:解决了什么，•适用性/上下文，什么地点时间可以用•实现:，•示例代码:， •结构:， •后果:使，也被称为:设计模式的其他名称，动机:合作；参与者

**06 2.Create one quality scenario for availability and one quality scenario for performance for the system described below.**

**06 3.5.1 Most important quality attribute(s) and architectural driversforthsystem:**

这个产品牵扯到多种配置，关心用户的安全，必须是可靠的。两个最重要的品质，可用性和可修改性。系统架构驱动程序:•必须提供高可用性，因为系统可能会挽救生命或者让人们远离危险。•提供能够处理各种类型传感器的接口。•架构必须能够支持各种类型的显示和键盘/按钮配置。.当没电的时候自动切换到电池供电；不同价格版本功能不同。

**5.2**选择并描述合适的建筑策略，并描述策略怎样影响质量属性？

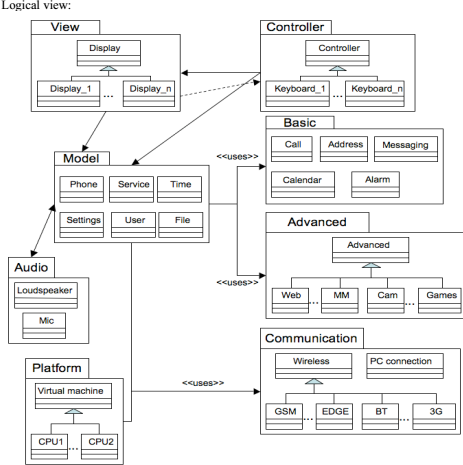
****他的解决方案将集中改进可用性和可修改性，**Heartbeat(可用性**):提供心跳功能以确保系统运行和存活(可用性)。系统由独立的硬件单元(监视器)组成，它有一个电池备份硬件。固定的时间间隔发送心跳信号。每秒钟到独立的心跳监听器单元。如果这个单位没有心跳信号在一段时间内(例如5秒)，主单元将重新启动。如果失败,主系统将切换到电池电源。**状态重同步(可用性):** 心跳信号从主要单位发送到监控单元由主单元的状态信息组成。监测单元能重新启动前一个状态的主单元。**异常(可用性):**异常可用于检测主单元中的故障应该被送到监护单元。主单元不稳定时，监控单元重启主单元。**使用MVC (可修改性)**:通过MVC方法，可以为各种显示和键盘/按钮配置提供支持在相同的架构。**使用中介(可修改性)：**为了支持传感器的变化**，**中介用来代表不同的传感器为接入系统的不同传感器提供相同的接口。**独立的核心功能和高级功能(可修改性):** 系统提供的功能可以根据系统的不同的价格区间而变化，所有系统的核心功能应该与更高级的系统提供的更高级的功能分离。**5.4 Create logical and process view:** 逻辑视图基于模型-视图-控制器模式，即视图模块为不同类型的显示提供变化，控制器模块提供支持针对不同类型的键盘/按钮。模型模块表示房子的状态的部门。一个叫做传感器的独立模块为各种传感器提供支持火灾和盗窃传感器。基本功能是在基本模块中提供的管理报警，配置报警系统，一个标识符部分来处理关机时的用户标识。Call-up用于危险发生联系安全、消防、业。监控模块（运行在独立的硬件上）负责从主报警系统采集心跳信号并监控可能的功率输出.监视主单元（看门狗）的状态，万一发生故障（软件或电源）可重新启动主单元。

**07 1.3题**、**Software for SoNewHenriksenTM Mobile Phones**5.1 Architectural drivers。

该系统的主要架构驱动程序是可修改性(硬件和软件)，另一个重要的架构驱动程序是可用性(尤其是主要功能，比如(电话管理)作为一种电话，应该可以在紧急情况下使用。性能也是一个重要的架构驱动，因为移动设备的资源有限就CPU、内存和存储而言。对于系统的用户来说，移动设备的usability很重要。与业务相关的，上市时间。

5.2 **Architectural tactics**主要关注点应该是**可修改性**，因为这是主要的架构驱动程序。-用MVC使用户输入，用户展示，逻辑分离。-使用可扩展子类的泛型类来支持键盘的变化，显示和沟通。-使用虚拟机支持各种处理器。

与**可用性**相关，关键的调用功能和通信模块应该得到在其他功能崩溃的情况下，在可靠性方面更优先考虑。软件应该有广泛的故障处理，可以重新启动软件，以防它挂起。

与**性能**相关，整个系统应该设计为内存和CPU高效避免性能下降

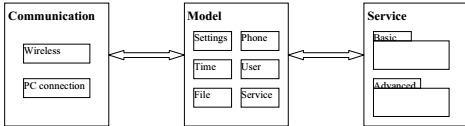
与**usability易用性**相关的是，MVC体系结构模式将提供一个灵活的可提供可更改的灵活的用户接口的解决方案。用户界面也应该如此通过维护用户的模型来提供对上下文的敏感，从而提供与用户正在做的事情有关的帮助功能。

**5.3 Architecture views**

逻辑视图通过使用模型-视图-控制器来反映对可修改性的关注模式和使用可变点显示，键盘，高级功能，中央处理器配置和通讯网络。内存和存储的差异不是在逻辑视图中可视化，因为预期操作系统会处理这一部分。该模型维护手机的逻辑表示(电话状态等)，设置(首选项)、用户(用于可用性)、文件管理和服务运行。的服务可利用的是基于两个模块Basic和Adva。

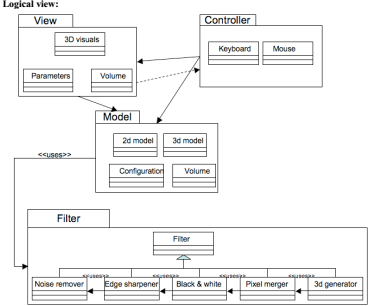
系统将有三个并行运行的**主要进程**(线程)。第一个过程,模型，负责维护系统运行时的雕像(电话状态，时间，服务状态，用户状态，设置，文件等)或协调电话上的其他进程。第二过程，沟通，负责沟通和检查

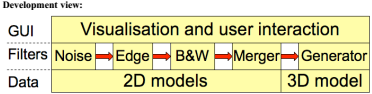
对于传入的调用、消息等。第三个流程，服务，是当前调用的服务用户可以是基本服务(如呼叫、地址、日历、消息或警报)，或高级服务(如网络、多媒体、相机或游戏)。

**08 1.Software for an Oil Reservoir Computation System**

**6.1 Identify the *architectural drivers* for the system described below**

系统的架构驱动程序是系统的每个部分都可以被替换(可修改性)并且在不同的部分之间应该保持相同的接口即使系统发生了变化(可修改性)。此外，性能将是对于这个系统非常重要，因为有很多计算正在进行。

**6.2 Choose and describe suitable *design and/or architectural patterns* for the problem。described below, and describe how the patterns affect the quality attributes**

最明显的架构模式是mvc模式，使其更容易实现更改GUI，管道-过滤模式来描述正在进行的数据过滤。可以使用各种各样的设计模式，但是一个候选者observer- pattern是用于实现mvc的。**6.3创建系统的体系结构视图。架构必须根据4 + 1视图模型在两个视图中描述:逻辑和开发** 逻辑视图基于两种架构模式:mvc模式和管道和过滤器模式。在2D中的模型模块保存2D扫描的表示模型，一个三维模型，来自用户的基于参数的系统的配置表示和容量表示。在filter模块中，过滤器类包含基本的管道和过滤器功能，用于将输入转换为某些输出。不同的过滤器的依赖关系使用“uses”原型来建模。

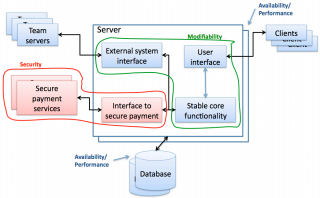
**接口：**2D模型和过滤器之间的接口是原始图形格式24比特(颜色)。过滤器之间的接口也是24位有色的原始格式之间的噪音，边缘和黑与白。B&W的输出是原始的2位格式(黑和白)，这是相同的格式合并模式输出。他输出接口的发电机滤波器是通用的3D格式(U3D)。过滤器和GUI层之间的接口是通用的3D格式。开发视图显示为分层的架构模式，分隔数据、筛选器和GUI。注意，在这个视图中，有些部分被省略了，比如volume和parameters。

**10 2.选择合适架构模型**

**1）**游戏引擎：层 **2）**一个可以在各种移动应用程序上运行的烹饪食谱的应用程序：MVC **3）**通过一组数据转换来分析天气数据的PC应用程序：管道过滤器**4）**交换电子名片的应用：p2p **5）**分布式协作应用程序，用于共享存储在A中的各种信息常见的数据库(存储库)：黑板

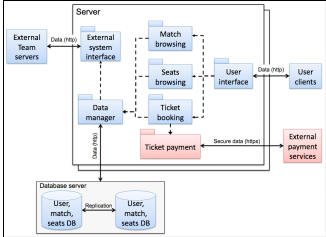
***10 5.Problem 5 Design a software architecture using ADD***

**Step 1. Choose module to decompose：The Tippeliga-Ticket System**

**Step 2a. Choose architectural drivers:** 1)该系统每周停机不能超过2分钟(可用性) .2) 系统应提供安全的电子支付(安全)**3)** 系统应该能够与不同的团队服务器通信(可修改性) **4)** 系统必须处理40,000个并发用户(性能)

**Step 2b. Choose architectural patterns: 1)** 安全策略:服务器防火墙、自动操作系统用户、验证用户、处理付款通过第三方和安全连接的使用(https).2) 可修改性策略:将功能划分为一致的单元，并计划可更改外部系统的接口。**3)** 性能策略:使用服务器的复制来确保对许多用户的支持从团队服务器缓存数据.**4）**可用性策略:使用服务器的复制来处理停机时间

**Architectural patterns:** 使用一种模型-视图控制器模式，视图和控制器被表示在用户界面部分，模型是数据库。在服务器和数据库上使用复制，以提高性能和可用性。关注分离是用来提供的可修改性(单独的核心功能和外部接口)，以及一个具有安全的计算和接口的单独部分

**Step 2c. Instantiate Modules and Allocate Functionality**：应用体系结构模式和实例化功能，用于匹配浏览、座位浏览和门票预订。还添加了一个数据管理器，它负责处理来自外部系统的数据缓存(存储在数据库中)

**Step 2d. Define Interfaces of the Child Modules：服务器与系统外部部分之间的接口**:（在用户界面和用户客户端之间:HTTP上的HTML，在外部系统接口和外部团队服务器之间:XML / HTTP，数据管理器和数据库服务器之间:SQL /HTTP，在票务支付和外部支付服务之间:加密的二进制通过HTTPS）**。服务器内部接口（**用户界面类提供了使用其他类的显示方法:（displayMatch()，displaySeats()，displayBooking()）**）票务支付类提供的方法payTicket(机票预订类使用)Data manager类提供了其他三个类使用的方法:（**getMatchInformation()，getSeatsInformation()，getSeatsInformation()**。外部系统接口类提供了一种更通用的访问信息的方法，从外部团队服务器:（**getInformation()**）**

**11年：6题选择合适模式**

**1）**用于温度传感器的系统软件（Control loop）**2）**从地震样本中提取原始数据的软件（管道，过滤器）

**3）**AI（Hierarchical task tree）**4）**软件代理系统经仓库（黑板）**5）**管理空中交通的系统（分层）

**7题Software for Surveillance and Monitoring of Patients (SMP)**

a)**架构驱动**: 最重要的架构驱动因素是可用性(非常重要的是设备被用来监控生命和死亡的情况)，和可修改性(传感器、功能、输入和输出的不同配置) 。**b)** **架构策略和模式:**使用策略和模式是非常重要的促进可修改性和可用性。为了确保单元正在运行，需要一些

来自外部硬件单元的ping - echo或heartbeat机制知道系统还在运行。如果检测到系统不工作，需要一种机制来重启系统，如果不能重启，要发出警报。

**QA**

**Performance Tactics：**控制资源需求-减少对资源的需求 管理资源-工作效率更高

通过管理资源需求来提高绩效:

管理采样率——减少生成的事件数

限制事件响应——使用排队系统(在最坏的情况下，如果队列满了，则抛出事件)

优先处理事件——忽略低优先级事件

提高资源效率,投资时间改进算法减少开销,如合并组件→减少延迟

负面影响可修改性

绑定执行时间——牺牲准确性以获得更高的性能，例如使用更快的算法来近似正确的行为

通过管理资源提高绩效:

增加资源——更快的处理器，额外的内存，…(垂直)

引入并行计算

维护计算的多个副本——不止一台机器进行计算(横向扩展)

维护多个数据副本，以获得更快的数据访问绑定队列大小——控制传入事件的数量

进度资源-引入资源调度(例如优先级)

**Usability Tactics：Support user initiative Support system initiative**

支持用户主动

取消——允许用户取消长期运行的操作撤消——证明是回滚状态

暂停/恢复-控制长期运行操作的执行

聚合-通过允许多个对象的批处理操作来防止微管理(例如，选择多个文件)

支持系统项目

**Reliability Tactics：****Detect faults， Recover from fault，s Prevent faults**

**1检测到故障**

监视-显式组件监视系统

**Ping/ echo**——通过异步测试消息测试组件的可达性

**心跳**-故障检测机制，通过周期性信息时间戳-检测不正确的消息序列

检查消息的有效性，例如对消息的校验和

投票- - - 3个系统做同样的事情来检测不一致

**异常检测**——例如，CPU寄存器的除数为零，读取未初始化内存的访问权限

自我测试——由组件本身进行故意的健康检查进入翻译页面

ping/ echo和心跳策略在不同的进程之间运行，异常策略在一个进程中运行。

**2从错误中恢复（**准备恢复，进行系统修复**）**

**主动冗余**—相同的系统运行备份，即时更新(热重启) c/s配置中用，例如数据库管理系统

**被动冗余**——相同的系统运行备份，定期更新(warm 重启/双冗余/三重冗余)用于控制系统

**备用：**重新引导到适当的软件配置，并在出现故障时初始化它的状态。用作备用客户端工作站

修复：影子操作，异常处理，回滚——回到较旧的(良好的)系统状态，软件升级-修补系统(或子系统)，重试——重复失败的动作，降级-减少功能的显式状态

**3防止错误**

**从服务中移除（**将组件脱机以防止(或重新启动)**），事务（**事务被用来防止任何数据受到影响，如果一个进程中的一个步骤失败，并且同时防止多个并发线程之间的冲突访问相同的数据**），进程监视器（**一旦检测到流程中的错误，监视过程可以删除不执行的流程并创建一个新的实例，并将其初始化为备用策略中的某个适当状态**），异常的预防增加权限集**——改进组件以应对异常状态

**Security Tactics:** **Resist attacks Detect attacks React to attacks Recover from attacks**

抵抗攻击

识别角色——识别外部输入身份验证者的来源

授权的演员

限制访问——限制访问资源，例如防火墙

限制暴露——限制系统加密数据访问点的数量——保护通信

单独的实体——在不同的机器上有组件(或虚拟机)

更改默认设置-强制用户更改系统的默认设置

检测到攻击

检测入侵——例如监视网络流量并识别恶意行为

检测服务拒绝——例如，比较服务请求模式和已知的拒绝服务攻击

验证消息完整性——例如校验和、哈希值

检测消息延迟——例如监视时间以传递消息

应对袭击

撤销访问——限制访问可能泄露的资源

锁帐户-例如，重复失败的登录，作为锁定帐户的触发器

通知行动者-意味着通知相关的行动者，例如邮件给系统管理员

从攻击中恢复过来

恢复-将系统(及其数据)回滚到已知的状态(在入侵之前)

审计跟踪-记录用户和系统操作，允许计算机取证

**Maintainability Tactics**减少元件的尺寸 增加内聚力 降低耦合 延迟绑定

Testability Tactics: 控制和观察系统状态——使测试更容易

限制复杂性——复杂的系统更难测试，从而使它们更简单

**非运行：**可维护性:您可以轻松地修复bug，并添加新特性的可演化性:您的系统如何容易地处理更改可测试性:您如何轻松地测试系统的正确性

**可维护性**：在设计和实现阶段，编码注释，面向对象的设计，连续的编码风格，文档。

策略：减少元件的尺寸，增加内聚力，降低耦合，延迟绑定

**可测试性策略**：控制和观察系统状态——使测试更容易限制复杂性——复杂的系统更难测试，从而使它们更简单。

**Configurability：**系统在不重新编译或重新安装的情况下改变其操作参数的能力，如选择适当的数据库驱动程序，配置网络参数

**Scalability：**系统在不重新编译或重新安装的情况下提高其容量的能力。如提供额外的Web页面意味着只将这些Web页面复制到Web服务器文件系统中

**7.1Performance**

**1、****Source of stimulus刺激源 Stimulus（**周期性的事件到达;零星的事件到达;随机事件到达**） Environment（正常，过载） Artifact工件 （系统） Response（**过程刺激;变化的服务水平**） Response measure （**延迟，截止日期，吞吐量，抖动，失当率，数据丢失**） Tactics**

**2、**性能策略的目标是对在某个时间限制条件下到达系统的事件生成响应。

**3**、**问题**：资源消耗。资源包括CPU、数据存储、网络通信带宽和内存，但它还可以包含由特定系统在设计中定义的实体。

阻塞的时间。由于资源不可用，或者因为计算依赖于其他尚未可用的计算结果，所以可以使用资源来阻止计算。资源争用。资源的可用性。依赖于其他计算

4、**策略**：:资源需求,资源管理和资源仲裁。

**资源需求**：减少延迟的一种策略是减少处理事件流所需的资源，提高计算效率，提高计算效率，限制执行时间，限制队列大小。

**资源管理**（用的最多）：引入并行，维护多个数据或计算的副本，增加可用资源。

**仲裁**：选择合适调度策略**，**调度的竞争标准包括最佳的资源使用、请求的重要性、最小化使用的资源数量、最小化延迟、最大化吞吐量、防止饥饿以确保公平等等。有**先进先出和固定优先级**

三种常见的优先级策略是：语义的重要性 截止时间单调。率单调

动态优先级调度:轮询，最早到期优先。

**7.2．1MODIFIABILITY**

**1、Source of stimulus（**终端用户、开发人员、系统管理员**） Stimulus（**希望添加/删除/修改/改变功能，质量属性，容量**） Environment（**在运行时，编译时间，构建时间，设计时间**） Artifact工件 （**系统用户界面、平台、环境;与目标系统相互作用的系统**） Response（**在架构中定位要修改的地方;在不影响其他功能的情况下进行修改;测试修改;部署修改**） Response measure （**成本按受影响因素的数量，努力，金钱;影响其他功能或质量属性的程度**） Tactics**

**2、策略目标：**控制可修改性的策略以控制时间和成本来控制实现、测试和部署变更的时间和成本

**3、策略**：本地化修改（保持语义一致性，预测预期的变化，推广模块，限制可能的选项）

防止连锁反应（隐藏信息，维持现有接口（增加接口，增加适配器，提供一个占位程序A，限制通信路径，**使用中介**））

**7.2.2 SECURITY（：**不可否认、机密性、完整性、保证、可用性和审核**）**

**1、Source of stimulus（**攻击的来源可能是人或其他系统。它可能已经被识别(正确或错误)，或者可能是目前未知的。攻击本身是未经授权的访问、修改或拒绝服务**） Stimulus（**尝试显示数据、更改/删除数据、访问系统服务、降低系统服务的可用性**） Environment（**无论是线上的还是离线的，连接的或断开的，防火的或开放的**） Artifact工件 （**系统服务;数据在系统**） Response（**对用户进行身份验证;隐藏用户的身份;阻止访问数据和/或服务;允许访问数据和/或服务;授予或收回访问数据和/或服务的许可;记录访问/修改或试图通过身份访问/修改数据/服务;以一种不可读的格式存储数据;认识到对服务的不可解释的高需求，并通知一个用户或另一个系统，并限制服务的可用性**） Response measure （**所需的时间/努力/资源，以规避与成功概率有关的安全措施;检测攻击的概率;确定负责攻击或访问/修改数据和/或服务的个人的可能性;在拒绝服务攻击下仍可使用的服务比例;恢复数据/服务;在多大程度上损坏和/或合法的访问被拒绝**）**

**2、Security Tactics：**对用户进行身份验证。授权用户。维护数据的机密性（加密，通信链路（虚拟私有网络(VPN)，安全套接字层(SSL)）），保持完整性（校验和

散列结果）。限制接触，限制访问（防火墙）

**7.3AVAILABILITY**

**1、Source of stimulus**（内部系统;外部系统）**，Stimulus（**遣漏，崩溃，不按时响应，不正确响应**），Artifact（**指定了需要高度可用的资源，例如处理器、通信通道、进程或存储**），**Environment**（**正常运行，降级模式**），Response（**记录，登录失败，通知选定的用户或其他系统，切换到降级模式，具有更少的容量或更少的功能，关闭外部系统，变得不可用在修复时**），Response measure（**响应度量可以指定可用性百分比，或者它可以指定修复的时间，也可以指定系统必须可用的时间，或者系统必须可用的时间**）**

**Availability Tactics：（目标）**防止错误成为失败或至少限制错误的影响并使修复成为可能。**Detect faults， Recover from fault，s Prevent faults**

**Partten**

**设计模式：创建型模式（工厂模式，抽象工厂模式，单例模式，**建造者模式，原型模式**）结构型模式（**适配器模式、装饰器模式、代理模式、外观模式、桥接模式、组合模式、享元模式**）行为型模式（**策略模式、模板方法模式、观察者模式、迭代子模式、责任链模式、命令模式、备忘录模式、状态模式、访问者模式、中介者模式、解释器模式**）**

**体系结构包括**:

**组件**:例如:过滤器、数据库、对象、客户机/服务器。**连接器**:程序调用，管道，事件广播

**属性**:例如:签名、pre /post conds、RT规范

**Benefits of Using Styles：**设计重用**，**代码重用，可理解性的系统组织，互操作性，支持的风格标准化，可视化

**Architectural Styles**：数据流方式（**管道和过滤器**，批处理顺序），数据中心架构（**黑板**，存储库）层架构（抽象层的架构，n层体系结构），通知架构（发布/订阅，点对点，基于事件），

网络中心风格（客户机-服务器，点对点）远程调用的架构（Web服务，代理，解释器），GUI架构(交互)（MVC，Presentation-­‐‑Abstraction-­‐‑Control），具有适应性的体系结构（微­内核，反射），过程控制，

**Data-flow architectures：：**数据流风格的特征是将系统视为连续的输入数据片段的一系列转换，数据输入系统，然后一次一个地流经组件，最后，将数据分配给一些最终目的地(输出或数据存储)。

**Pipes and filters：**过滤器不知道其他过滤器(组件之间松耦合)，理想情况下，他们是完全独立的，数据流在流中:很好地处理图像，音频，视频。**管道有助于可维护性，**易于添加和删除组件(有助于演化)，允许注入特殊部件以解决横切问题，例如监控吞吐量、日志记录，允许并发/并行执行(可扩展性)。

**缺点**：管道经常导致批量处理，因此不适合交互式应用程序在数据结构中，很难实现增量更新的最小公分母，每个过滤器都必须解析/解析数据(对性能有害)增加了每个组件的复杂性

**Batch Sequential：**一个降级版本的Pipers和过滤器样式。

批处理模型还强调数据在流程步骤之间转移，但是，数据是通过块传递的数据不会转移到下一个进程步骤，直到整个块被处理。每个进程步骤独立于任何其他步骤。

**Data-centered architectures：**通信手段区分了两种亚型:存储库和黑板存储库:客户机向系统发送请求，执行必要的操作(例如插入数据) 黑板:当感兴趣的数据发生变化时，系统向订阅者发送通知和数据，因此是活跃的。两种组件：（中央数据结构——黑板，在黑板上操作的部件），系统控制完全由黑板状态驱动，例子（人工智能，系统集成的软件环境，(例如，Interlisp)，编译器架构）

**Web architecture：**以数据为中心的架构的另一个例子是Web架构，有一个常见的数据模式(即Web的元结构)跟随超媒体数据模型，页面(节点)和它们之间的链接，以及寻址机制。数据操作不是直接在系统中，而是通过应用程序实现的，尽管HTTP协议有数据操作的方法

通信协议是HTTP，分析:完整性没有保证(404错误)，可伸缩

**Resource-Oriented Architecture：**这样的架构将Web看作是一个巨大的分布式数据库数据模型:通过URL和链接来寻址，数据操作:HTTP方法(GET、PUT、POST、DELETE)可伸缩、良好的性能、可用性等。

**Layered Architectures**：降低复杂性，提高模块化，可重用性，可维护性，不同的分层标准:最明显的是抽象。例子：操作系统（硬件服务、内核服务、系统服务、UI服务，依次向上）

虚拟机：从与操作系统非常相似的概念上指定编译器和真实机器之间的接口，提高可移植性

解释器：类似于虚拟机，但仅限于执行代码，如shell脚本，可以在系统组件之间共享状态，而脚本允许更大的灵活性，以牺牲性能为代价，可能有助于配置。

Peer-to-peer protocol stacks：分层架构的常见示例是网络协议栈，TCP/ IP协议栈四层，最低层处理两台计算机之间的通信，因特网层处理网络的数据包路由。传输层保证数据包是无错误的，而且它们的接收顺序与发送的相同

应用程序层支持特定于应用程序的协议，如HTTP,SMTP,FTP，层构成了一系列越来越抽象的操作，更高的层跨越了网络上计算机之间的虚拟连接

**层结构优点：**每一层都只与相邻的层相连接，帮助进化，因为一个人可以交换一个单一的层，限制对邻居层的修改，如果通信是标准化的，则可以通过多个系统(尤其是多个系统)来帮助重用，在实践中，每个层通常由专门的开发团队维护。

**缺点**：并不是所有的系统都能以层来组织，抽象可能会对性能产生负面影响，特定的优化不可能很难找到正确的抽象层次，在实践中，努力实现影响多个层的特性。

**N-tier architectures：：**n层架构是由业务应用程序发起的现代客户机-服务器架构，通常与Web应用程序相关。将架构概念化到表示、应用程序和数据存储层。

**2层**：富客户端（ui+逻辑+通信），服务器存储数据。每个客户机运行一个完整的应用程序。**缺点**:每个客户机必须知道如何与所有数据服务器通信可伸缩性被破坏，因为客户机与服务器紧密耦合

**3层：**从两层架构演化而来解决其缺陷，在客户端和数据服务器应用程序或业务逻辑层之间插入第三层:中间层。典型的中间层位于服务器端(但最近可能会在服务器和客户机之间进行分割)。客户端-应用服务器-数据服务器。

优点：更容易的客户端维护，因为中间层可以单独更新，更好地隔离数据存储的细节:可扩展性、可配置性，更好地利用网络，在中间层中添加额外的处理层的可能性，可以清晰地分离表示、应用程序逻辑和数据.  **表示层-应用层-数据层**。

**富客户端：**富客户端包含应用程序的全部知识，它可以访问平台上所有UI机制。**两种类型的富客户端**：使用标准协议(例如电子邮件客户端)的标准应用程序的客户端，客户端使用自定义协议来定制应用程序

**瘦客户端**：web客户端

**Notification architectures：**信息和活动通过通知机制传播的体系结构，gui中的事件处理，**发布-订阅**，**:监听器和回调。**类似于观察者模式，类似于黑板以数据为中心的体系结构

组件:发布者、订阅者、管理分配的代理，连接器：网络协议。质量产生高效的单向传播信息，而组件的耦合非常低。**当事件发生时**，通知组件将事件发送到所有已注册的组件(广播)另一种可能性是只将事件发送给感兴趣的组件。**例子：**数据库中触发器，编辑器中检查拼写检查

**优点**：松耦合，可进化。缺点：无法控制流程，不利测试

**网络中心风格：c/s**模式被其他架构风格所使用，它可以用来实现共享的存储库. 如以数据为中心的存储库模式。用于对单个共享数据结构进行操作的过滤器

**c/s扩展：**优先垂直扩展，不要系统知道。水平扩展需要系统支持。

**有状态**：服务器追踪应用状态。（瘦客户端）**无状态**：客户端追踪应用状态。（富客户端）无状态服务器的可伸缩性通常要比有状态服务器好。

**点对点**：高可扩性。缺点：高延迟，难以维护和测试。

**Remote Invocation Architectures：**远程调用架构涉及分布式处理组件

通常，客户端组件调用远程组件上的方法(函数)。**优点**：通过分布式计算提高性能。只有当网络是可靠和快速的，通信开销是可管理的。**缺点**：组件紧密耦合，管理寻址增加通信开销。

**broker（代理）：**将通信从应用程序功能支持的分布式系统中分离出来，代理隐藏来自系统组件的通信。代理可以被用来透明地将非分布式系统更改为分布式系统。**例如**:客户机通过代理与远程对象进行交互。**任务：**找到合适的服务器/服务，将请求转发到服务器。将结果(或错误消息/异常)报告给客户机。**优点**：组件的解耦(网络方面)有助灵活性、可维护性和可演化性，允许分布。

**缺点**：络通信可能会引入新的错误类型，由于网络延迟和带宽有限，行为可能会发生变化。

**Interceptor：**允许在不更改核心组件的情况下集成其他服务，由组件提供的接口，该组件允许另一个组件进行注册，在特定的事件中被调用。**例如**:日志记录，系统由两个组件组成:客户机和服务器，当某些事件发生时，每个组件都提供一个回调接口，日志组件在每个组件上注册自己，日志组件将日志存储到共享存储库中。**优点：**可重用性，灵活性，可维护性，可扩展性。

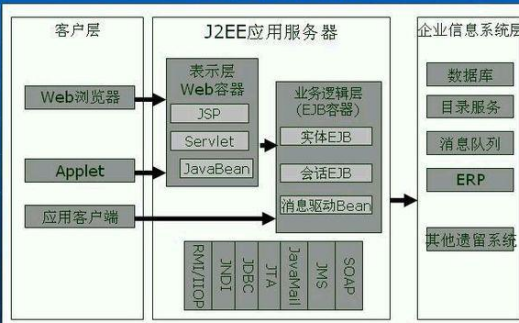
缺点：变复杂，级联回调，无尽循环，核心组件需要提供回调接口，可测性差。

**MVC：**模型:封装应用程序域的行为和数据，视图:呈现模型，从模型请求更新，为控制器准备用户界面，通常是多个视图。控制器:对用户输入作出反应，修改模型和分派到视图，操纵模型触发应用程序行为。控制器和视图都依赖于模型控制器，视图是UI的一部分。**模型-视图-控制器模式不能替代n层架构，**模型是n层模式和MVC模式的一部分，该模型与较低的抽象层进行通信，模型可以使用通知模式来通知更改的视图。**优点：**分离关注点，帮助可重用性，多个用户界面，不改变模型，例如移动和网络，帮助可配置性(由于接口更改更容易，与应用程序逻辑的更改相比，预期的副作用更少)。**缺点：**增加复杂性，不是所有系统都适应，基于通知的更新，可用性差

**Difference between MVC and PAC：**MVC的控制器专注于输入和输出到视图，PAC的控制器则侧重于代理和内部代理之间的沟通和协调。PAC将系统划分为分层的松散耦合代理，

而MVC则侧重于模型和视图的分离，其中没有分层的代理。

**异种架构：**Web有一个2层架构:浏览器和服务器，浏览器本身有一个通知架构来处理用户事件。基于网络的搜索引擎，概念上:以数据为中心，分层，三层结构:分层(网络)，三层，通知，执行:分布式的，面向服务的，带有回调的通知



水平扩展more machines、faster hardware vertically