目录

[一. 软件构架 2](#_Toc71665442)

[二. 软件设计 3](#_Toc71665443)

[三. 体系结构的目标 3](#_Toc71665444)

[五. 软件架构师的角色 4](#_Toc71665445)

[设计专长 4](#_Toc71665446)

[领域专长 4](#_Toc71665447)

[技术专长 4](#_Toc71665448)

[方法学专长 4](#_Toc71665449)

[软件架构师的隐藏角色 5](#_Toc71665450)

[体系结构工程师的任务 5](#_Toc71665451)

[六. 质量属性 5](#_Toc71665452)

[6.1 静态质量属性 5](#_Toc71665453)

[6.2 动态质量属性 5](#_Toc71665454)

[七. 质量方案 6](#_Toc71665455)

[通用质量属性 6](#_Toc71665456)

[八. 体系结构的设计风格 7](#_Toc71665457)

[九. 普通结构设计 8](#_Toc71665458)

[十. 体系结构类型 9](#_Toc71665459)

[十一. 体系结构设计过程 9](#_Toc71665460)

[了解问题 9](#_Toc71665461)

[识别设计元素及其关系 9](#_Toc71665462)

[评估架构设计 10](#_Toc71665463)

[十二. 关键架构原则 10](#_Toc71665464)

[为改变而建，而不是为之持久 10](#_Toc71665465)

[降低风险并进行模型分析 10](#_Toc71665466)

[将模型和可视化用作沟通和协作工具 11](#_Toc71665467)

[使用增量迭代方法 11](#_Toc71665468)

[十二. 关键设计原则 11](#_Toc71665469)

[关注点分离 11](#_Toc71665470)

[单一责任原则 11](#_Toc71665471)

[最少知识原理 11](#_Toc71665472)

[最大限度地减少前期的大型设计 11](#_Toc71665473)

[不要重复功能 12](#_Toc71665474)

[重用功能时要优先考虑组合（而不是继承） 12](#_Toc71665475)

[识别组件并将它们分组在逻辑层中 12](#_Toc71665476)

[定义层之间的通信协议 12](#_Toc71665477)

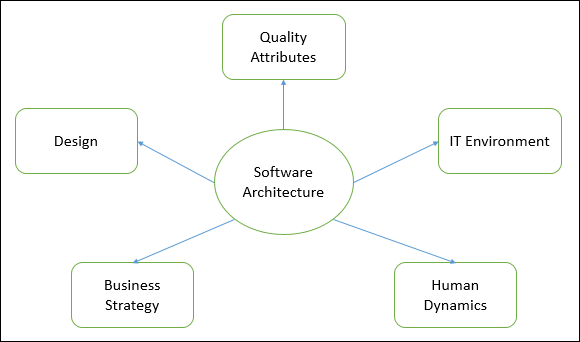
[定义图层的数据格式 12](#_Toc71665478)

[系统服务组件应该是抽象的 12](#_Toc71665479)

[设计异常和异常处理机制 12](#_Toc71665480)

[命名约定 13](#_Toc71665481)

软件系统的体系结构描述了其主要组件，它们之间的关系（结构）以及它们之间如何交互。软件体系结构和设计包括几个促成因素，例如业务战略，质量属性，人员动态，设计和IT环境。



我们可以将软件体系结构和设计分为两个不同的阶段：软件体系结构和软件设计。在**体系结构中**，非功能性决策由功能性需求强制转换和分离。在设计中，功能要求得以实现。

一. 软件构架

体系结构是**系统**的**蓝图**。它提供了一种抽象方法来管理系统复杂性，并在组件之间建立通信和协调机制。

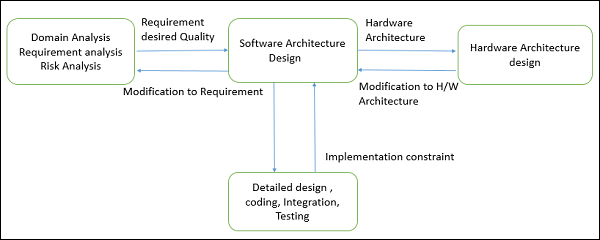
* 它定义了一种**结构化解决方案**，可以满足所有技术和运营要求，同时优化性能和安全性等常见质量属性。
* 此外，它涉及与软件开发有关的一组重要的组织决策，并且这些决策中的每一个都会对最终产品的质量，可维护性，性能和整体成功产生重大影响。这些决定包括-
  + 选择组成系统的结构元素及其界面。
  + 在这些元素之间的协作中指定的行为。
  + 将这些结构和行为元素组成大型子系统。
  + 体系结构决策与业务目标保持一致。
  + 体系结构风格指导组织。

二. 软件设计

软件设计提供了一个**设计计划**，该**计划**描述了系统的元素，它们如何适合并一起工作以满足系统的需求。制定设计计划的目标如下-

* 与系统需求进行协商，并与客户，市场营销和管理人员建立期望。
* 在开发过程中充当蓝图。
* 指导实施任务，包括详细的设计，编码，集成和测试。

它在详细的设计，编码，集成和测试之前，以及在域分析，需求分析和风险分析之后。



## 三. 体系结构的目标

该体系结构的主要目标是确定影响应用程序结构的需求。布局合理的体系结构可减少与构建技术解决方案相关的业务风险，并在业务和技术要求之间架起一座桥梁。

其他一些目标如下-

* 公开系统的结构，但隐藏其实现细节。
* 实现所有用例和方案。
* 尝试解决各种利益相关者的要求。
* 处理功能和质量要求。
* 降低所有权目标并改善组织的市场地位。
* 改善系统提供的质量和功能。
* 提高外部对组织或系统的信心。

但是软件体系结构仍然是软件工程中的新兴学科。它具有以下限制-

* 缺乏用于表示架构的工具和标准化方法。
* 缺乏分析方法来预测体系结构是否会导致实现符合要求的分析方法。
* 缺乏对体系结构设计对软件开发的重要性的认识。
* 缺乏对软件架构师角色的了解，以及利益相关者之间的沟通不畅。
* 缺乏对设计过程的了解，设计经验和设计评估。

## 五. 软件架构师的角色

软件架构师提供了一个解决方案，技术团队可以为整个应用程序创建和设计该解决方案。软件架构师应在以下领域具有专业知识-

### 设计专长

* 软件设计专家，包括多种方法和方法，例如面向对象的设计，事件驱动的设计等。
* 领导开发团队并协调开发工作，以确保设计的完整性。
* 应该能够审查设计建议并在它们之间进行权衡。

### 领域专长

* 正在开发的系统专家以及软件开发计划。
* 协助需求调查过程，确保完整性和一致性。
* 协调要开发的系统的域模型的定义。

### 技术专长

* 有助于系统实施的可用技术专家。
* 协调编程语言，框架，平台，数据库等的选择。

### 方法学专长

* SDLC（软件开发生命周期）中可能采用的软件开发方法专家。
* 选择对整个团队有帮助的合适的开发方法。

### 软件架构师的隐藏角色

* 促进团队成员之间的技术工作，并增强团队中的信任关系。
* 共享知识并拥有丰富经验的信息专家。
* 保护团队成员免受外力的干扰，这些外力会分散他们的注意力，并给项目带来较少的价值。

### 体系结构工程师的任务

* 清晰，完整，一致且可实现的功能目标集
* 系统的功能描述，至少有两层分解
* 系统的概念
* 系统形式的设计，至少具有两层分解
* 时间，操作员属性以及实施和操作计划的概念
* 确保遵循功能分解并控制接口形式的文档或过程

## 六. 质量属性

质量是衡量卓越水平或无缺陷或无缺陷状态的一种度量。质量属性是与系统功能分开的系统属性。

实施质量属性使区分好系统和坏系统变得更加容易。属性是影响运行时行为，系统设计和用户体验的总体因素。

他们可以分类为-

### 6.1 静态质量属性

反映与体系结构，设计和源代码直接相关的系统和组织的结构。它们对于最终用户是不可见的，但是会影响开发和维护成本，例如：模块化，可测试性，可维护性等。

### 6.2 动态质量属性

反映系统在执行过程中的行为。它们与系统的体系结构，设计，源代码，配置，部署参数，环境和平台直接相关。它们对于最终用户是可见的，并且在运行时存在，例如吞吐量，健壮性，可伸缩性等。

## 七. 质量方案

质量方案指定了如何防止故障成为故障。根据其属性规范，它们可以分为六个部分-

* **来源**-产生刺激的内部或外部实体，例如人员，硬件，软件或物理基础结构。
* **刺激**-到达系统时需要考虑的条件。
* **环境**-刺激在一定条件下发生。
* **工件**-整个系统或其中的一部分，例如处理器，通信通道，持久性存储，进程等。
* **响应**-刺激到达后进行的活动，例如检测故障，从故障中恢复，禁用事件源等。
* **响应措施**-应该测量已发生的响应，以便可以测试需求。

### 通用质量属性

下表列出了软件体系结构必须具有的常见质量属性：

|  |  |  |
| --- | --- | --- |
| 类别 | 质量属性 | 描述 |
| 设计品质 | 概念完整性 | 定义整体设计的一致性和连贯性。这包括组件或模块的设计方式。 |
| 可维护性 | 系统轻松进行更改的能力。 |
| 可重用性 | 定义组件和子系统的功能，使其适合在其他应用程序中使用。 |
| 运行时质量 | 互通性 | 一个系统或不同系统通过与外部各方编写和运行的其他外部系统进行通信和交换信息来成功运行的能力。 |
| 可管理性 | 定义系统管理员管理应用程序的难易程度。 |
| 可靠性 | 系统随时间推移保持运行的能力。 |
| 可扩展性 | 系统在不影响系统性能的情况下处理负载增加的能力或易于扩展的能力。 |
| 安全 | 系统具有防止超出设计用途的恶意或意外行为的功能。 |
| 表现 | 指示系统在给定时间间隔内执行任何操作的响应能力。 |
| 可用性 | 定义系统正常运行和工作的时间比例。可以将其衡量为预定时间段内系统总停机时间的百分比。 |
| 系统品质 | 可支持性 | 系统提供信息的能力，有助于在系统无法正常工作时识别和解决问题。 |
| 可测性 | 衡量为系统及其组件创建测试标准的难易程度。 |
| 用户素质 | 易用性 | 通过直观定义应用程序满足用户和消费者要求的程度。 |
| 体系结构质量 | 正确性 | 满足系统所有要求的责任。 |
| 非运行时质量 | 可移植性 | 系统在不同计算环境下运行的能力。 |
| 整体性 | 使系统的单独开发的组件能够正确一起工作的能力。 |
| 可修改性 | 每个软件系统都可以轻松地适应其软件的更改。 |
| 业务质量属性 | 费用和时间表 | 与上市时间，预期项目寿命和遗留资源利用有关的系统成本。 |
| 适销性 | 关于市场竞争的系统使用。 |

## 八. 体系结构的设计风格

软件体系结构的设计风格，也称为**架构模式**，是一组其形状的应用原理。它根据结构组织的模式定义了一个系统族的抽象框架。

体系结构的设计风格是指：

* 为组件和连接器的词典提供如何组合的规则。
* 通过为经常发生的问题提供解决方案，改进分区并允许设计的重用。
* 描述一种特定的方式来配置组件（具有良好定义的接口，可重用和可替换的模块）和连接器（模块之间的通信链接）的集合。

为基于计算机的系统而构建的软件表现出许多体系结构样式之一。每种样式都描述一个系统类别，其中包括：

* 一组由系统执行所需功能的组件类型。
* 一组连接器（子例程调用，远程过程调用，数据流和套接字），使不同组件之间能够进行通信，协调和协作。
* 语义约束，定义了如何集成组件以形成系统。
* 组件的拓扑布局，指示它们的运行时相互关系。

## 九. 普通结构设计

下表列出了可以按其主要关注领域进行组织的体系风格-

|  |  |  |
| --- | --- | --- |
| 类别 | 体系设计 | 描述 |
| 沟通 | 消息总线 | 规定使用可以使用一个或多个通信通道接收和发送消息的软件系统。 |
| 面向服务的架构（SOA） | 定义使用合同和消息将功能公开和使用为服务的应用程序。 |
| 部署方式 | 客户端服务器 | 将系统分为两个应用程序，客户端在其中向服务器发出请求。 |
| 3层或N层 | 将功能分成单独的部分，每个部分是位于物理上分开的计算机上的一层。 |
| 领域 | 域驱动设计 | 专注于对业务域进行建模，并基于业务域内的实体定义业务对象。 |
| 结构 | 基于组件 | 将应用程序设计分解为可重用的功能或逻辑组件，以暴露定义明确的通信接口。 |
| 分层的 | 将应用程序的关注点分为堆叠的组（层）。 |
| 面向对象 | 基于将应用程序或系统的职责划分为对象，每个对象包含与对象相关的数据和行为。 |

## 十. 体系结构类型

从企业的角度来看，有四种类型的体系结构，这些体系结构统称为**企业体系结构**。

* **业务架构**-定义企业内业务，治理，组织和关键业务流程的策略，并专注于业务流程的分析和设计。
* **应用程序（软件）架构**-用作各个应用程序系统，它们之间的交互以及与组织业务流程的关系的蓝图。
* **信息架构**-定义逻辑和物理数据资产以及数据管理资源。
* **信息技术（IT）架构**-定义构成组织整体信息系统的硬件和软件构建块。

## 十一. 体系结构设计过程

架构设计过程着重于将系统分解为不同的组件及其交互，以满足功能和非功能需求。软件架构设计的关键输入是-

* 由分析任务产生的需求。
* 硬件架构（软件架构师又向配置硬件架构的系统架构师提供要求）。

体系结构设计过程的结果或输出是**体系结构描述**。基本架构设计过程包括以下步骤-

### 了解问题

* 这是最关键的步骤，因为它会影响后续设计的质量。
* 如果没有清楚地了解问题，就不可能创建有效的解决方案。
* 许多软件项目和产品被认为是失败的，因为它们实际上并未解决有效的业务问题或具有可识别的投资回报率（ROI）。

### 识别设计元素及其关系

* 在此阶段，建立基线以定义系统的边界和上下文。
* 根据功能需求将系统分解为主要组件。可以使用设计结构矩阵（DSM）对分解建模，该结构结构矩阵显示设计元素之间的依赖性，而无需指定元素的粒度。
* 在此步骤中，通过描述许多系统实例来完成对体系结构的第一次验证，并且此步骤称为基于功能的体系结构设计。

### 评估架构设计

* 每个质量属性都有一个估计值，因此为了收集定性度量或定量数据，需要对设计进行评估。
* 它涉及评估体系结构是否符合体系结构质量属性要求。
* 如果所有估计的质量属性均符合要求的标准，则架构设计过程已完成。
* 如果不是，则进入软件体系结构设计的第三阶段：体系结构转换。如果观察到的质量属性不符合其要求，则必须创建一个新设计。

## 十二. 关键架构原则

以下是设计架构时要考虑的关键原则-

### 为改变而建，而不是为之持久

考虑一下应用程序可能需要随着时间的变化而变化，以解决新的要求和挑战，并建立支持它的灵活性。

### 降低风险并进行模型分析

使用设计工具，可视化效果，建模系统（例如**UML**）来捕获需求和设计决策。影响也可以分析。不要对模型进行形式化，以免抑制其轻易迭代和修改设计的能力。

### 将模型和可视化用作沟通和协作工具

设计，决策和设计的有效更改之间的有效沟通对于良好的体系结构至关重要。使用模型，视图和体系结构的其他可视化效果与所有涉众有效地交流和共享设计。这样可以快速传达设计变更。

识别并了解关键的工程决策和最容易出错的领域。在第一时间投资于正确的关键决策，以使设计更加灵活，并且不会因更改而被破坏。

### 使用增量迭代方法

从基线体系结构开始，然后通过迭代测试改进候选体系结构以改进体系结构。通过多次遍历将迭代细节添加到设计中，以获取大图或正确的图景，然后将注意力集中在细节上。

## 十二. 关键设计原则

以下是在最大程度地降低成本，维护要求并最大程度地提高体系结构的可扩展性和可用性时应考虑的设计原则-

### 关注点分离

将系统的组件划分为特定的功能，以使组件功能之间没有重叠。这将提供高内聚力和低耦合度。这种方法避免了系统组件之间的相互依赖性，这有助于简化系统。

### 单一责任原则

系统的每个模块都应负一个特定的责任，这有助于用户清楚地了解系统。它还应有助于组件与其他组件的集成。

### 最少知识原理

任何组件或对象都不应该了解其他组件的内部细节。这种方法避免了相互依赖，并有助于可维护性。

### 最大限度地减少前期的大型设计

如果应用程序的需求不清楚，则最大程度地减少前期的大型设计。如果有可能修改需求，则应避免对整个系统进行大型设计。

### 不要重复功能

“不重复功能”指定不应重复组件的功能，因此仅应在一个组件中实现一段代码，即组件可重用。应用程序中功能的重复会使其难以实施更改，降低清晰度并引入潜在的不一致之处。

### 重用功能时要优先考虑组合（而不是继承）

继承会在子类和父类之间建立依赖关系，因此会阻止子类的自由使用。相反，该组合提供了很大的自由度并减少了继承层次结构。

### 识别组件并将它们分组在逻辑层中

系统中满足要求所需的标识组件和关注区域。然后将这些相关组件分组在一个逻辑层中，这将帮助用户从较高的层次上理解系统的结构。避免在同一层中混合不同类型关注点的组件。

### 定义层之间的通信协议

了解组件之间如何通信，这需要对部署方案和生产环境有完整的了解。

### 定义图层的数据格式

各种组件将通过数据格式相互交互。不要混用数据格式，以使应用程序易于实现，扩展和维护。尝试使层的数据格式相同，以便各个组件在相互通信时无需对数据进行编码/解码。它减少了处理开销。

### 系统服务组件应该是抽象的

与安全性，通信或系统服务（如日志记录，概要文件和配置）相关的代码应在单独的组件中抽象出来。不能将此代码与业务逻辑混合使用，因为扩展设计和维护很容易。

### 设计异常和异常处理机制

预先定义异常，有助于组件以优雅的方式管理错误或意外情况。整个系统中的异常管理都是相同的。

### 命名约定

命名约定应事先定义。它们提供了一个一致的模型，可以帮助用户轻松理解系统。团队成员更容易验证其他人编写的代码，因此会增加可维护性。