# Design Strategy （理解）

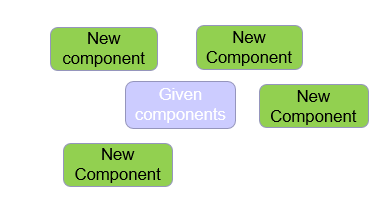
* 分解
* 设计符合建筑要求
* 生成和测试

## Decomposition

* 架构决定了质量属性
* 重要的质量属性是整个系统的特征。
* 因此，设计始于整个系统
  + 整个系统被分解成部分
  + 每个部分可以从整体继承全部或部分质量属性要求

**设计并不意味着绿色领域**

* 如果给出要在最终设计中使用的组件，则分解必须适应这些组件。



## Designing to Architecturally Significant Requirements

* 还记得架构上的重要要求（ASR）吗？
* 这些是您必须满足设计要求的要求
  + 这些中有少数
  + 它们是最重要的（根据定义）

**How Many ASRs Simultaneously（同时进行）?**

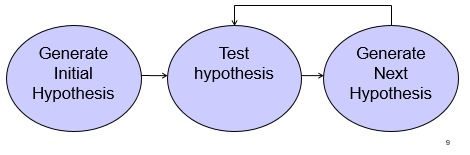
* 如果您缺乏设计经验，那么从最重要的一开始就为ASR设计一个。
* 随着您获得经验，您将能够同时设计多个ASR。

**What About Other Quality Requirements?**

* 如果您的设计不满足特定的非ASR质量要求那么
  + 调整您的设计以满足它
  + 削弱它以满足
  + 使它成为ASR之一
  + 声明它不可满足

## Generate and Test

* 将当前设计视为假设。
* 询问当前设计是否满足要求（测试）
* 如果没有，则生成新假设



**Raises the Following Questions**

* 初始假设来自哪里？
* 我如何测试假设？
* 我什么时候做的？
* 如何生成下一个假设？

**Where Does the Initial Hypothesis（假设） Come From?**

* **理想的来源**
  + 现有系统
  + 构架
* **不太理想的来源**
  + 模式和战术
  + 域分解
  + 设计清单
* **为什么“不太理想”？**
  + 不太理想的不满足所有要求。 它们通常会省略许多质量属性要求。

**How Do I Test a Hypothesis?**

* 使用已涵盖的分析技术
* 从质量属性讨论中设计策略。
* 架构上的重要要求
* 测试的输出是什么？
  + 当前设计未达到的要求清单。

**How Do I Generate the Next Hypothesis?**

* 添加缺少的职责。
* 使用策略来调整假设的质量属性行为。
  + 策略的选择取决于未满足哪些质量属性要求。
  + 注意战术的副作用。

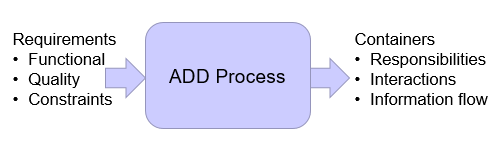
**When Am I Done?**

* **所有ASR都满意和/或......**
* **当活动预算不足时**
  + **使用到目前为止最好的假设**
  + **开始实施**
  + **继续设计工作，尽管现在将受到实施选择的限制。**

# The Attribute-Driven Design Method （掌握）

* 包装已经讨论过的许多技术。
* 迭代方法。 在每次迭代中你
  + 选择要设计的系统的一部分。
  + 决定该部分的所有架构重要要求。
  + 生成并测试该部件的设计。
* ADD不会产生完整的设计
  + 有责任的容器集
  + 容器之间的交互和信息流动
* 不为容器生成API或签名。

**ADD Inputs and Outputs**

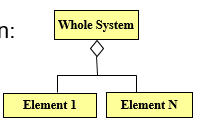


**The Steps of ADD**

1. 选择要设计的系统元素。
2. 确定所选元素的ASR。
3. 为所选元素生成设计解决方案。
4. 库存剩余需求并选择下一次迭代的输入。
5. 重复步骤1-4，直到满足所有ASR。

### Step 1: Choose an Element of the System to Design

* 对于green field designs，选择的元素通常是整个系统。
* 对于legacy designs，元素是要添加的部分。
* 第一次迭代后：



**Which Element Comes Next?**

* 两个基本的改进策略：
  + 广度优先
  + 深度第一
* 哪一个选择？
  + 这取决于
* 如果首先使用新技术=>深度优先：探索使用该技术的含义。
* 如果团队需要工作=>深度优先：为该团队生成需求。
* 否则=>广度优先。

### Step 2: Identify the ASRs for the Chosen Element

* 如果所选元素是整个系统，则使用实用程序树。
* 如果所选元素位于分解树的下方，则根据该元素的要求生成实用程序树。

### Step 3: Generate a Design Solution for the Chosen Element

* 使用ASR对所选元素应用generate和test （Apply generate and test to the chosen element with its ASRs）

### Step 4: Select the Input for the Next Iteration

* 对于每个功能要求
  + 确保满足要求。
  + 如果没有，则添加职责以满足要求。
    - 将它们添加到具有类似要求的容器中（一致性）
    - 如果没有这样的容器可能需要创建新的
    - 如果容器对团队的要求太多，请将其拆分为两部分。 分裂时尽量实现松散耦合。

**Quality Attribute Requirements**

如果满足质量属性要求，则不需要进一步考虑。

如果质量属性要求尚未满足，则

* + 将其委托给其中一个子元素
  + 将它拆分为子元素

如果无法满足质量属性，请查看它是否可以被削弱。 如果不能满足或削弱那么就无法满足。

**Constraints**

* 约束被视为已处理的质量属性要求。
  + Satisfied（满意）
  + Delegated（委托）
  + Split
  + Unsatisfiable（不可满足的）

### Repeat Steps 1–4 Until All ASRs are Satisfied

* 在步骤3结束时，每个子元素将与它相关联一组：
  + 功能要求，
  + 质量属性要求
  + 限制。
* 这将为方法的下一次迭代设置子元素。

# Garage Door Opener Example 设计一个车库门开关器产品线构架

* 为具有更大家庭信息系统的车库门开启器设计产品线架构
* Input to ADD: a set of requirements
  + Functional requirements as use cases（功能需求作为用例）
  + Constraints
  + 质量要求表示为系统特定的质量方案

## Scenarios（场景） for Garage(车库) Door System

* 设备和控件对于产品系列中的各种产品是不同的
* 处理器会有所不同
* 如果在下降过程中感觉到障碍物，则必须在0.1秒内停止
* 需要从家庭信息系统访问开门器系统以进行诊断和控制

## 1 Choose the Module to Decompose

* 系统🡪子系统🡪子模块
  + 对于车库门开启器，是系统
* 该级别的一个约束是开启者必须与家庭信息系统互操作。

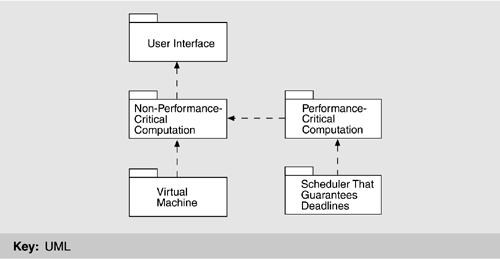
## 2 Identify the Architectural Drivers

* 在车库系统中，4个场景是架构驱动程序

## 3 Generate a Design

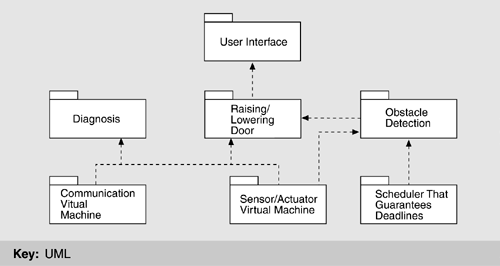
* 该模式需要满足架构驱动因素，并通过“组合”选择的策略来满足驾驶员的需求
* 选择战术涉及两个因素：
  + 当然，架构司机本身
  + 实施战术的模式对其他要求的副作用
* 性能（performance）和可修改性（modifiability）是最关键的质量属性。
* 对于可修改性，选择“语义连贯**（semantic coherence）**”和“信息隐藏**（information hiding）**”作为我们的策略，并将它们组合起来为受影响区域定义虚拟机。
* 为了提高性能，请从“资源需求”中选择“提高计算效率”，从“资源仲裁（resource arbitration）”中选择调度策略。
* **Semantic coherence and information hiding.** Separate responsibilities dealing with
  + User interface（用户界面）
  + Communication（通讯）
  + Sensors（传感器）
  + 调用这些虚拟机中的每一个并期望产品线内的变化（Call each of these virtual machines and expect variation within product line）
* 提高计算效率。 性能关键的计算应该尽可能高效。
* 明智地安排。 应安排对性能至关重要的计算，以确保实现时间期限。

**Architectural Pattern that Utilizes Tactics**



* 升高/降低门（正常模式）和诊断（diagnosis） - 非关键
* 管理障碍物检测 - 关键
* 虚拟机传感器/执行器

虚拟机通信



## Allocate Functionality Using Multiple Views

* 应用与父模块相关的用例有助于架构师更详细地了解功能的分布。 这也可能导致添加或删除子模块以满足所需的所有功能。
* 在分解中为子项分配职责也会导致发现必要的信息交换。 （生产者/消费者，发布 - 订阅......）

## Use Cases Illustrating（说明） Concurrency（并发）

* 两个用户同时做同样的事情。 这有助于识别资源争用或数据完整性问题。 在我们的车库门示例中，一个用户可能正在远程关闭门而另一个用户正在从开关打开门。
* 一个用户执行多项活动。 这有助于发现数据交换和活动控制问题。 在我们的示例中，用户可能在同时打开门的同时执行诊断。
* 启动系统，这有助于决定初始化策略。 在我们的例子中，车库门开启系统的启动是否取决于家庭信息系统的可用性？ 车库门开启系统是否始终工作，等待信号，还是每个门打开和关闭时启动和停止？
* 关闭系统。 这有助于发现清理问题，例如实现和保存一致的系统状态。

## Verify（验证） and Refine（优化） Use Cases and Quality Scenarios

* 我们现在有一个分析模块的“建议书”。
* 下一步是分析它，看看它适合的程度。 这涉及到
  + 通过“运行”父级用例来验证分解
  + 通过从父级继承用例/约束来为子级做好准备。
* 我们将通过观察来研究这一点
  + Functional requirements
  + Constraints
  + Quality Scenarios（质量方案）

**Functional Requirements**

* 使用功能需求进行验证和优化
  + 分解功能需求将责任分配给子模块
  + 我们可以使用这些职责为子模块生成用例
* 例如：车库门系统的职责
  + 在开关，遥控器或房屋处打开/关闭门
  + 当感知到障碍物时停止关闭
  + 与家庭信息系统互动
* 可以将这些职责分配给子模块

**Functional Groups**

* **User Interface:** 
  + 处理用户请求，（handle user requests,）
  + 翻译升降模块（translate for raising/lowering module）
  + 显示回复
* **升高/降低门模块（Raising/Lowering Door Module）**
  + 控制（actuators）升高/降低门
  + 完成开启或关闭时停止
* **障碍物检测：**
  + 识别何时检测到对象
  + 停止或反转门的关闭
* **Communication Virtual Machine**
  + 管理与房屋信息系统（HIS）的沟通
* **Sensor/Actuator virtual machine**
  + 管理与传感器/执行器的交互
* **Scheduler（调度程序）:** 确保在检测到障碍物时满足最后期限
* **Diagnosis（诊断）:** 管理与HIS的诊断交互

## Quality Scenarios Assigned in the Example

* 用于打开和关闭的装置和控制装置在产品系列的产品中是不同的。 方案委托给用户界面模块。
* 不同产品中使用的处理器会有所不同。 委托所有模块 - “不使用特定于处理器的代码”
* 如果检测到障碍物，则必须在.1秒内停止。 委托调度程序和障碍物检测模块。
* 车库门系统必须与HIS互动。 委托通讯和诊断模块。

# Summary

* Designing the architecture is a matter of
  + Determining the ASRs
  + Performing generate and test one an element to decompose it to satisfy the ASRs
  + Iterating until requirements are satisfied.
* 设计架构是一个问题：
  + 确定ASR
  + 执行生成并测试一个元素以对其进行分解以满足ASR
  + 迭代直到满足要求。