中国科学技术大学
University of Science and Technology of China

Software Architecture

SSE USTC Qing Ding
dingqing@ustc.edu.cn
http://staff.ustc.edu.cn/~dingqing

Designing the Architecture

Outline

· 生命周期中的体系结构
· 架构设计
· 形成团队结构及其与架构的关系
· 创建骨骼系统

- Architecture in the life cycle
- Designing the architecture
- Forming the team structure and its relationship to the architecture
- Creating a skeletal system

Architecture in the Life Cycle

- The evolutionary delivery life cycle
  - Get customer feedback
  - Iterate through several releases, each with new or modified functionality
  - Deliver limited versions if necessary
- When does one begin designing?
  - Some idea of system requirements is needed – the shaping requirements are called *architectural drivers*

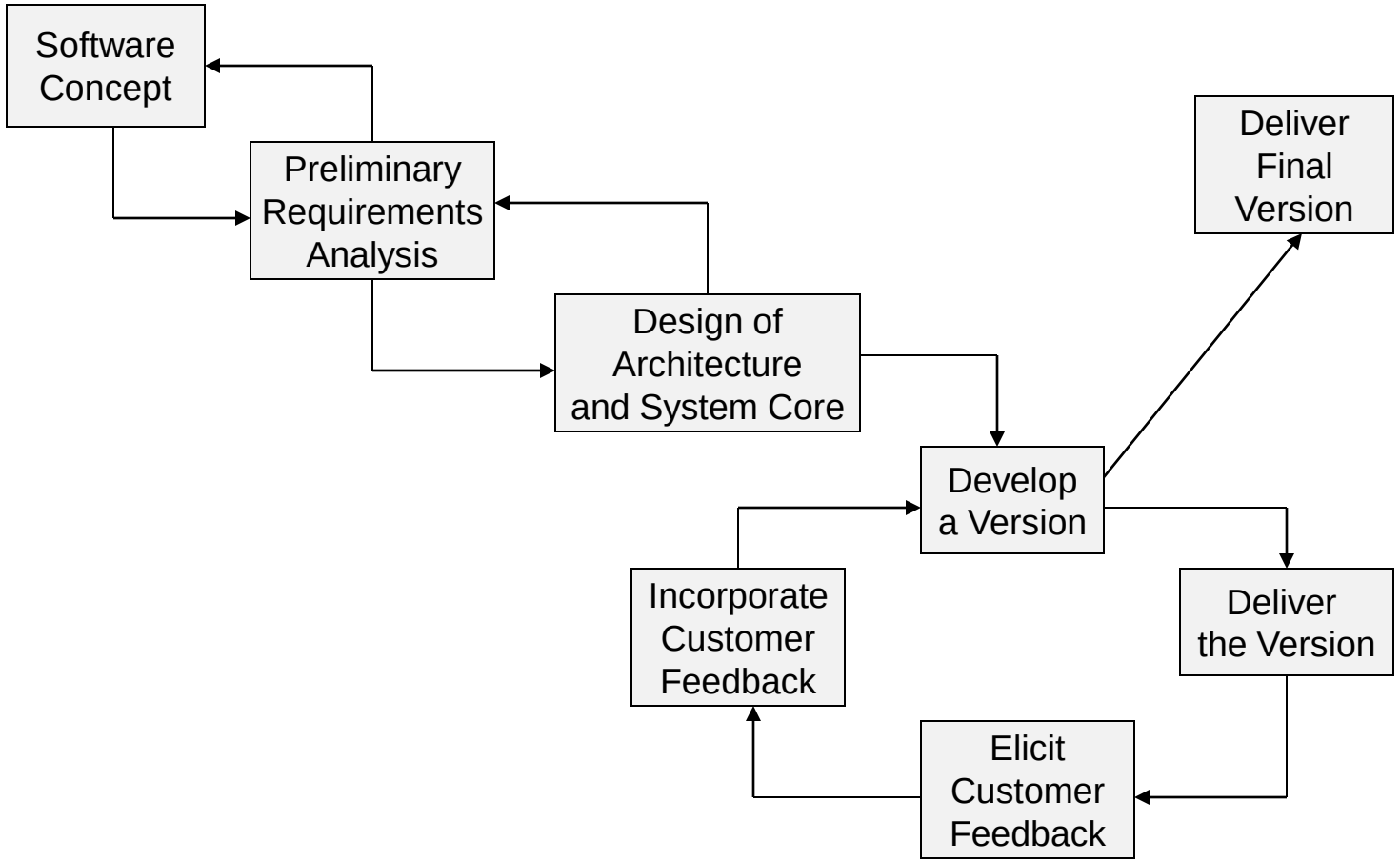
演进的交付生命周期
-客户反馈
-迭代多个版本，每个版本都有新的或修改过的功能
-如有必要，提供有限版本

什么时候开始设计?
-需要一些系统需求的概念-形成需求称为架构驱动

Architecture in the Life Cycle (Cont'd)

- When does one begin designing? (cont'd)
  - 确定最高优先级的业务目标，并将其转化为质量场景或用例
  - Identify the highest priority business goals and turn them into quality scenarios or use cases
  - 对架构影响最大的是架构驱动因素(应该少于10个)
  - The ones that have the most impact on the architecture are the architectural drivers (there should be fewer than 10)
  - 一旦了解了架构驱动，就可以开始架构设计了
  - Once the architectural drivers are known, the architectural design can begin
  - 需求分析过程将受到架构设计过程中产生的问题的影响
  - The requirements analysis process will then be influenced by questions generated during the architectural design,

Evolutionary Delivery Life Cycle



Designing the Architecture

一种称为属性驱动设计(ADD)的方法可以用来设计同时满足质量和功能需求的体系结构

- A method called Attribute Driven Design (ADD) can be used to design an architecture to satisfy both quality and functional requirements.

ADD可以被看作是其他开发方法的扩展，例如Rational统一过程(RUP)

- ADD can be viewed as an extension to other developments methods, such as the Rational Unified Process (RUP).

Attribute-Driven Design

ADD基于软件必须实现的质量属性的分解过程

- ADD bases the decomposition process on the quality attributes the software has to fulfill.

它是一个迭代的分解过程，在每个阶段，策略和架构模式被选择来满足一组高质量的场景，然后功能被分配来实例化模式提供的模块类型

- It is a recursive decomposition process, where, at each stage, tactics and architectural patterns are chosen to satisfy a set of quality scenarios and then functionality is allocated to instantiate the module types provided by the pattern.

Attribute-Driven Design (Cont'd)

系统被描述为一组用于存放功能和它们之间交互的容器

- The system is described as a set of containers for functionality and the interactions among them.

这是实现功能的粗粒度框架

- This is a coarse grained framework for achieving the functionality.

Example Application of ADD

示例问题:家庭信息系统中的车库开门器

- Sample problem: A garage door opener within a home information system.

该系统通过开关、远程控制或家庭信息系统负责门的升降。还可以从家庭信息系统内部诊断开门人的问题

- The system is responsible for raising and lowering the door via a switch, remote control, or the home information system. It is also possible to diagnose problems with the opener from within the home information system.

Input to ADD

一组需求(通常表示为用例)和约束

- A set of requirements (typically expressed as use cases) and constraints

一组表示为特定于系统的质量场景的质量需求，在本例中包括：

- A set of quality requirements expressed as system-specific quality scenarios including, in this case:

该系列产品的开门、关门装置和控制方式各不相同。它们可能包括家庭信息系统内部的控制

  - The device and controls for opening and closing the door are different for the various products in the product line. They may include controls from within a home information system

Input to ADD (Cont'd)

- A set of quality requirements (cont'd):
  - The processor used in different products will differ.
  - If an obstacle (person or object) is detected by the garage door during descent, it must halt (alternately re-open) within 0.1 second.
  - The garage door opener should be accessible for diagnosis and administration from within the home information system using a product-specific diagnosis protocol.

不同产品使用的处理器是不同的 兼容性

下降过程中，如果车库门发现障碍物(人或物体)，它必须在0.1秒内停止(或者重新打开)

车库开门器应该可以通过使用特定产品的诊断协议从家庭信息系统内部进行诊断和管理

ADD Steps

选择要分解的模块——要开始分解的模块通常是整个系统

1. Choose the module to decompose – the module to start with is usually the whole system.

按照以下步骤细化模块

2. Refine the module according to the following steps:

从一组具体的质量场景和功能需求中选择架构驱动

   a. Choose the architectural drivers from the set of concrete quality scenarios and functional requirements.

根据可用于实现架构驱动的策略，选择满足架构驱动的架构模式

   b. Choose an architectural pattern that satisfies the architectural drivers based on the tactics that can be used to achieve the drivers.

ADD Steps (Cont'd)

按照以下步骤优化模块(续)

2. Refine the module according to the following steps (cont'd):

实例化模块并从用例中分配功能，并使用多个视图表示

c. Instantiate modules and allocate functionality from the use cases and represent using multiple views.

定义子模块的接口。分解提供了模块和模块交互类型的约束。在每个模块的接口文档中记录这些信息

d. Define interfaces of the child modules. The decomposition provides modules and constraints on the types of module interactions. Document this information in the interface document for each module.

ADD Steps (Cont'd)

2. Refine the module according to the following steps (cont'd):
   e. Verify and refine use cases and quality scenarios and make them constraints for child modules.

验证和细化用例和质量场景，并使它们成为子模块的约束

3. Repeat the steps above for every module that needs further decomposition.

迭代：对每个需要进一步分解的模块重复上述步骤

1. Choose the Modules to Decompose

在我们的示例中，我们从整个系统开始，车库开门器系统

- In our example, we start with the whole system, the garage door opener system.

这个级别的一个限制是，开门器必须与家庭信息系统互操作

- One constraint at this level is that the opener must interoperate with the home information system.

2a. Choose the Architectural Drivers

前面给出的四个场景表示以下需求

- The four scenarios previously given indicate requirements for:
  - 实时性能
  - real-time performance
  - 支持产品线的可修改性
  - modifiability to support product lines
  - 网上诊断
  - online diagnosis
- 一般来说，可能需要进行详细的调查，以确定给定的需求是否真的是驱动因素
- In general, a detailed investigation may be required to determine whether given requirements are really drivers.

2a. Choose the Architectural Drivers (Cont'd)

- We do not treat all requirements as equal.
- The less important requirements are satisfied within the constraints of the most important.
- This is a significant difference between ADD and other architecture design methods.

· 我们不平等地对待所有的要求。
· 不太重要的要求在最重要要求的约束条件下得到满足。
· 这是ADD与其他建筑设计方法的一个显著区别

2b. Choose an Architectural Pattern

使用解释器是实现运行时可修改性的一种优秀技术，但它对性能有负面影响

- The use of an interpreter is an excellent technique for achieving modifiability at runtime, but it has a negative influence on performance.

使用解释器的决定取决于可修改性比性能要相对重要

- The decision to use an interpreter depends on the relative importance of modifiability versus performance.

一种可能的决定是只对模式的一部分使用解释器

- A possible decision is to use an interpreter for only a portion of the pattern.

2b. Choose an Architectural Pattern (Cont'd)

可修改策略包括“本地化更改”、“防止连锁反应”和“延迟绑定时间”。

- The modifiability tactics are “localize changes,” “prevent the ripple effect,” and “defer binding time.”

在这种用例，我们主要关注系统设计期间将发生的变化，主要策略是“本地化变化”

- In this case, where we are concerned primarily with changes that will occur during system design, the primary tactic is “localize changes.”

我们选择语义一致性和信息隐藏作为策略，并结合它们来定义受影响区域的虚拟机

- We choose semantic coherence and information hiding as our tactics and combine them to define virtual machines for the affected areas.

2b. Choose an Architectural Pattern (Cont'd)

绩效策略是“资源需求”和“资源仲裁”。

- The performance tactics are “resource demand” and “resource arbitration.”

我们分别选择一个例子:“提高计算效率”和“选择调度策略”

- We choose one example of each: “increase computational efficiency” and “choose scheduling policy.”

因此，最后一套策略是

- The final set of tactics is therefore:

语义一致性和信息隐藏——在各自的模块(虚拟机)中分别处理用户界面、通信和传感器。

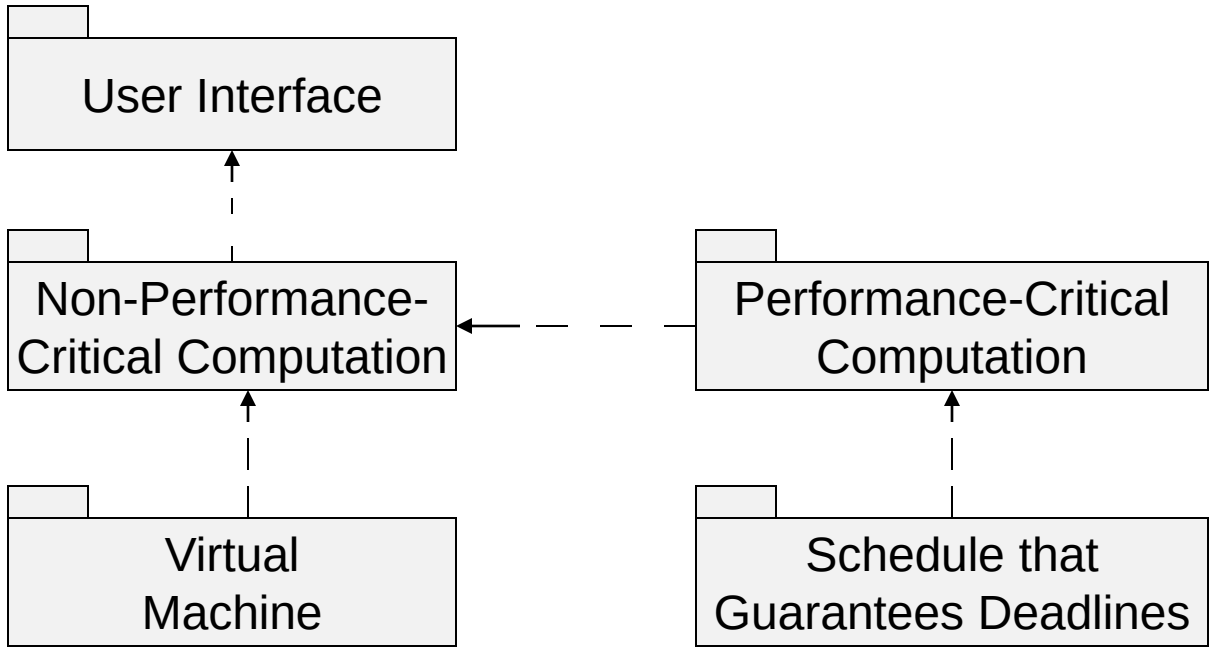
  - *Semantic coherence* and *information hiding* – Separate responsibilities dealing with the user interface, communication, and sensors into their own modules (virtual machines).

2b. Choose an Architectural Pattern (Cont'd)

- The final set of tactics (cont'd):
  - *Increase computational efficiency* – The performance-critical computations should be made as efficient as possible.
  - *Schedule wisely* – The performance-critical computations should be scheduled to ensure the achievement of the timing deadline.

· 提高计算效率--性能关键的计算应该尽可能高效。
· 智能调度--性能关键的计算应该被调度，以确保完成时间期限

Architectural Pattern that Utilizes Tactics to Achieve Garage Door Drivers



2c. Instantiate Modules and Allocate Functionality Using Multiple Views

使用多个视图实例化模块和分配功能

- We allocate the responsibility for managing obstacle detection and halting the garage door to the performance-critical section since the functionality has a deadline.
- The management of the normal raising and lowering of the door has no timing deadline so we can treat it as non-performance-critical

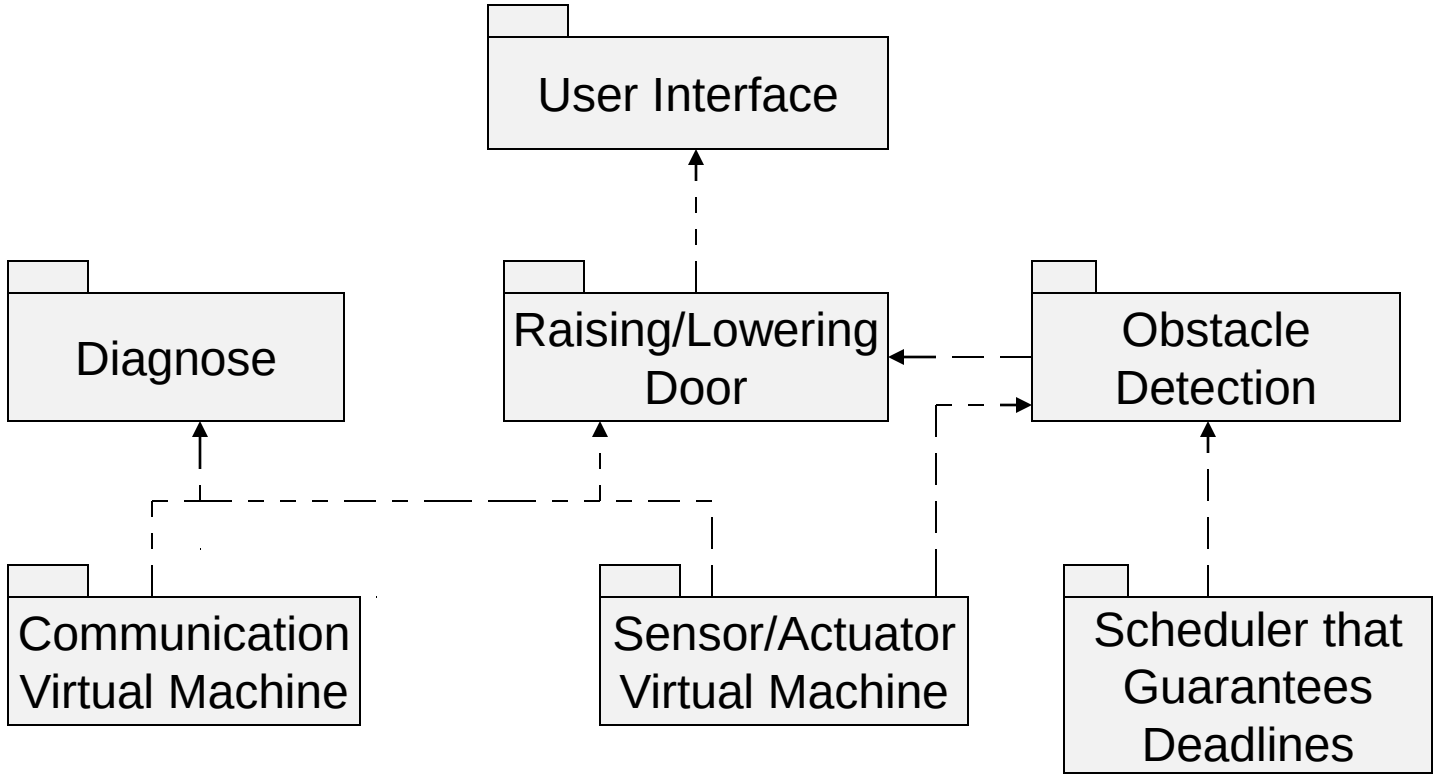
· 我们将管理障碍检测和停止车库门的责任分配到性能关键部分，因为功能有一个期限。
· 正常的开门和关门的管理没有时间期限，所以我们可以将其视为非性能关键型的

2c. Instantiate Modules and Allocate Functionality Using Multiple Views (Cont'd)

- The diagnosis capabilities are also non-performance-critical.
- Thus, the non-performance-critical module becomes instantiated as diagnosis and raising/lowering door modules.
- We also identify several responsibilities of the virtual machine: communication and sensor reading and actuator control.

· 诊断能力也是非性能关键型的。
· 因此，非性能关键模块被实例化为诊断和提升/降低门模块。
· 我们还确定了虚拟机的一些职责:通信和传感器读取以及执行器控制

First-Level Decomposition of Garage Door Opener



2c. Instantiate Modules and Allocate Functionality Using Multiple Views (Cont'd)

应用属于父模块的用例可以帮助架构师更好地理解功能的分布

- Applying use cases that pertain to the parent module helps the architect gain a better understanding of the distribution of functionality.

最终，父模块的每个用例都必须通过子模块中的职责序列来表示

- Ultimately, every use case of the parent module must be representable by a sequence of responsibilities within the child modules.

必须做足够的工作来获得系统能够交付所需功能的信心

- Enough must be done to gain confidence that the system can deliver the desired functionality.

体系结构应该由三个主要组中的每个组的一个视图表示

- The architecture should be represented by one view from each of the three major groups.

2c. Instantiate Modules and Allocate Functionality Using Multiple Views (Cont'd)

- The three common views
  - *Module decomposition view* – Containers for holding responsibilities and the major flow relationships among the modules
  - *Concurrency view* – Dynamic aspects of a system such as parallel activities and synchronization can be modeled

三种常见观点
-模块分解视图-保存职责和模块之间的主要流关系的容器
-并发视图-系统的动态方面，如并行活动和同步可以建模

2c. Instantiate Modules and Allocate Functionality Using Multiple Views (Cont'd)

- The three common views (cont'd)
  - *Deployment view* – The virtual threads of the concurrency view are decomposed into virtual threads within a particular processor and messages that travel between processors to initiate the next entry in the sequence of actions (thus is a basis for analyzing network traffic). Additionally, this view helps to decide if multiple instances of some modules are needed and supports reasoning about special purpose hardware.

部署视图——并发视图中的虚拟线程被分解为特定处理器中的虚拟线程和在处理器之间传递的消息，以启动操作序列中的下一个条目(因此是分析网络流量的基础)。此外，这个视图有助于决定是否需要某些模块的多个实例，并支持关于特殊用途硬件的推理

Understanding Concurrency in a System – Helpful Use Cases

理解系统中的并发性-有用的用例

- Two users doing similar things at the same time
- One user performing multiple activities simultaneously
- Starting up the system
- Shutting down the system

· 两个用户同时做类似的事情
· 一个用户同时执行多个活动
· 启动系统
· 关闭系统

2d. Define Interfaces of the Child Modules

- An interface of a module shows the services and properties provided and required.
- It documents what others can use and on what they can depend.
- Analyzing and documenting the decomposition in terms of structure (module decomposition view), dynamism (concurrency view) and runtime (deployment view) uncovers the interaction assumptions for the child modules.

· 模块的接口显示提供和需要的服务和属性。
· 它记录了其他人可以使用和依赖的内容。
· 根据结构(模块分解视图)、动态性(并发视图)和运行时(部署视图)对分解进行分析和记录，揭示了子模块的交互假设

2d. Define Interfaces of the Child Modules (Cont'd)

· 模块视图文档:
  -生产者/消费者的信息
  -需要模块提供服务并使用它们的交互模式
·并发视图文档:
  -线程之间的交互，引导到提供或使用服务的模块接口
  -组件是活动的信息
  -组件同步、排序和可能阻塞调用的信息

- The module view documents:
  - Producers/consumers of information
  - Patterns of interaction that require modules to provide services and to use them
- The concurrency view documents:
  - Interactions among threads that lead to the interface of a module providing or using a service
  - The information that a component is active
  - The information that a component synchronizes, sequentializes, and perhaps blocks calls

2d. Define Interfaces of the Child Modules (Cont'd)

· 部署视图文档:
–硬件要求，如专用硬件
–一些时间要求，例如处理器的计算速度
–通讯要求，例如，信息每秒钟更新不超过一次
· 所有这些信息应该在模块的接口文档中可用。

- The deployment view documents:
  - The hardware requirements, such as special-purpose hardware
  - Some timing requirements, e.g., the computational speed of a processor
  - Communication requirements, e.g., the information should not be updated more than once a second
- All this information should be available in the modules' interface documentation.

2e. Verify and Refine Use Cases and Quality Scenarios as Constraints for the Child Modules

验证并细化作为子模块约束的用例和质量场景

- Each child module has responsibilities that need to be translated into use cases for the module. Use case can also be defined by splitting and refining the parent use cases.
- For the garage door opener system, the responsibilities are decomposed into the following functional groups
  - User interface – recognize user requests and translate them into the form expected by the raising/lowering door module

· 每个子模块都有需要转换为该模块用例的职责。用例也可以通过分割和细化父用例来定义。
· 对于车库开门系统，职责分解为以下几个功能组
  -用户界面-识别用户请求，并将其转换为升降门模块所期望的形式

2e. Verify and Refine Use Cases and Quality Scenarios as Constraints for the Child Modules

- For the garage door opener system, the responsibilities are decomposed into the following functional groups (cont'd):
  - *Raising/lowering door module* – Control actuators to raise or lower the door. Stop the door when it reaches either fully open or fully closed.
  - *Obstacle detection* – recognize when an obstacle is detected and either stop the descent of the door or reverse it.

· 车库开门系统将职责分解为以下几个功能组(续):
-升降门模块-控制升降门的执行机构。当门完全打开或完全关闭时，停止。
-障碍物检测-检测到障碍物时立即识别，停止下降门或将其翻转

2e. Verify and Refine Use Cases and Quality Scenarios as Constraints for the Child Modules

- For the garage door opener system, the responsibilities are decomposed into the following functional groups (cont'd):
  - *Communication virtual machine* – Manage all communication with the home information system.
  - *Sensor/actuator virtual machine* – Manage all interactions with the sensors and actuators.
  - *Scheduler* – Guarantee that the obstacle detector will meet its deadlines.
  - *Diagnosis* – Manage the interactions with the home information system devoted to diagnosis.

．车库开门系统将职责分解为以下几个功能组(续)：
-通信虚拟机-管理与家庭信息系统的所有通信。
-传感器/致动器虚拟机-管理与传感器和致动器的所有交互
-调度器-保证障碍探测器将满足其最后期限。
-诊断-管理与诊断专用的家庭信息系统的交互。

2e. Verify and Refine Use Cases and Quality Scenarios as Constraints for the Child Modules

- The constraints of the parent module can be satisfied in one of the following ways:
  - The decomposition satisfies the constraint, e.g., if the constraint is to use a certain operating system, by defining the operating system as a child the constraint is satisfied.
  - The constraint is satisfied by a single child module, e.g., if the constraint is to use a special protocol, it can be satisfied by defining an encapsulation child module for the protocol.

可以通过以下方式满足父模块的约束：
-分解满足约束，例如，如果约束是使用某个操作系统，通过将操作系统定义为子系统，约束就满足了。
-约束由单个子模块满足，例如，如果约束使用一个特殊的协议，它可以通过为协议定义一个封装子模块来满足

2e. Verify and Refine Use Cases and Quality Scenarios as Constraints for the Child Modules

- The constraints of the parent module can be satisfied in one of the following ways (cont'd):
  - The constraint is satisfied by multiple child modules, e.g., using the Web requires two modules (client and server) to implement the necessary protocols.

约束由多个子模块来满足，例如，使用Web需要两个模块(客户端和服务器)来实现必要的协议

2e. Verify and Refine Use Cases and Quality Scenarios as Constraints for the Child Modules

- In the garage door opener system, one constraint is that the communication with the home information system is maintained.
- The communication virtual machine will recognize if this communication is unavailable, so the constraint is satisfied by a single child.

· 在车库开门系统中，一个约束是与家庭信息系统保持通信。
· 通信虚拟机将识别该通信是否不可用，因此由一个子节点满足约束

2e. Verify and Refine Use Cases and Quality Scenarios as Constraints for the Child Modules

- Quality scenarios also need to be refined and assigned to child modules:
  - A quality scenario may be completely satisfied by the decomposition without any additional impact and thereby marked as satisfied.
  - A quality scenario may be satisfied by the current decomposition with constraints on child modules.

质量场景也需要细化并分配给子模块:
-质量场景可以通过分解完全满足，没有任何额外的影响，因此被标记为满足。
-通过对子模块进行约束的当前分解可以满足质量场景

2e. Verify and Refine Use Cases and Quality Scenarios as Constraints for the Child Modules

- Quality scenarios also need to be refined and assigned to child modules (cont'd):
  - The decomposition may be neutral with respect to a quality scenario, e.g., a usability scenario, in which case it should be assigned to one of the child modules.
  - A quality scenario may not be satisfiable with the current decomposition. Either the decomposition should be reconsidered or rationale justifying its omission must be provided.

-对于质量场景，分解可能是中立的，例如，可用性场景，在这种情况下，它应该被分配给一个子模块。
-当前分解可能无法满足质量场景。要么应该重新考虑分解，要么必须提供不作为的理由

Refined Quality Scenarios for the Garage Door Opener Example

- The devices and controls for opening and closing the door are different for different products in the product line. They may include controls from within a home information system. This scenario is delegated to the user interface module.
- The processor used in different products will differ. This scenario is delegated to all of the modules. Each module becomes responsible for not using processor-specific features not supported by standard compilers.

· 不同产品的开门和关门装置和控制方式不同。它们可能包括家庭信息系统内部的控制。此场景被委托给用户界面模块。
· 不同产品使用的处理器会有所不同。此场景被委托给所有模块。每个模块都要负责不使用标准编译器不支持的处理器特定特性

Refined Quality Scenarios for the Garage Door Opener Example (Cont'd)

- If an obstacle is detected by the garage door during descent, the door must halt (or re-open) within 0.1 second. This scenario is delegated to the scheduler and the obstacle detection module.
- The garage door opener should be accessible for diagnosis and administration from within the home information system using a product-specific diagnosis protocol. This scenario is split between the diagnosis and communication modules. The communication module is responsible for the protocol to communicate with the home information system, and the diagnosis module is responsible for other diagnosis interactions.

· 如果在下降过程中车库门发现障碍物，门必须在0.1秒内停止(或重新打开)。此场景委托给调度程序和障碍检测模块。
· 车库开门器应该可以通过使用产品特定的诊断协议从家庭信息系统内部进行诊断和管理。此场景分为诊断模块和通信模块。通信模块负责与家庭信息系统通信的协议，诊断模块负责其他诊断交互

Status at the End of the Iteration

- We now have a decomposition of a module into its children.
- Each child has a collection of responsibilities:
  - A set of use cases
  - An interface
  - Quality scenarios
  - A collection of constraints
- This is sufficient to start the next iteration of the decomposition

· 我们现在已经将一个模块分解为它的子模块。
· 每个子模块都有一系列的责任:
  -一组用例
  -接口
  -质量场景
  -约束的集合
· 这足以开始分解的下一个迭代

Things We Still Do Not Know Regarding Our Sample Problem

- The language for communication between the user interface module and the raising/lowering modules
- The algorithm for performing obstacle detection
- How the performance-critical section communicates with the non-performance critical section

· 用于用户界面模块和升降模块之间通信的语言
· 执行障碍检测的算法
· 性能关键部门如何与非性能关键部门通信

Forming the Team Structure

形成团队结构

- Once the first few levels of the architecture's module decomposition structure are fairly stable, those modules can be allocated to development teams.
- Within teams there needs to be high-bandwidth communications.
- Between teams, low-bandwidth communications are sufficient (and in fact crucial).

· 一旦架构的模块分解结构的前几个级别相当稳定，这些模块就可以分配给开发团队。
· 团队内部需要高带宽的通信。
· 在团队之间，低带宽的通信就足够了(事实上也是至关重要的)

Forming the Team Structure (Cont'd)

- If interactions between the teams is complex, either:
  - The interactions among the elements they are creating are needlessly complex
  - Or, the requirements for those elements were not sufficiently "hardened" before development commenced
- Like software systems, teams should strive for high cohesion and low coupling.

· 如果团队之间的交互是复杂的，那么：
-它们所创造的元素之间的相互作用不必要复杂
-或者，在开发开始之前，对这些元素的需求没有充分“硬化”
· 与软件系统一样，团队应该追求高内聚、低耦合。

Forming the Team Structure (Cont'd)

- Each module of the system constitutes its own small domain (area of specialized knowledge or expertise).
- This makes for a natural fit between teams and modules of the decomposition structure.
- The effective use of staff, therefore, is to assign members to a team based on their expertise.

· 系统的每个模块都有自己的小领域(专业知识或专长领域)。
· 这使得团队和分解结构的模块之间有一个自然的契合。
· 有效利用员工，因此，是根据他们的专业知识来分配成员到一个团队中

Forming the Team Structure (Cont'd)

- The organizational structure also affects the architecture.
- Organizational entities formed for one project are motivated to preserve their existence and will want to maximize their importance in new projects.

· 组织结构也会影响架构。
· 为一个项目而形成的组织实体被激励以维持其存在，并将希望最大化其在新项目中的重要性

Creating a Skeletal System

- Once an architecture is sufficiently designed and teams are in place to build it, a skeletal system can be constructed.
- The architecture provides a guide as to the order in which portions of the system should be implemented.
  - First implement the software that deals with the execution and interaction of architectural components.
  - Add some simple functions.
- The result is a running system onto which useful functionality can be added

· 一旦架构被充分设计并且团队已经准备好构建它，一个骨架系统就可以构建了。
· 架构提供了一个指南，关于系统的各个部分应该被实现的顺序。
-首先实现处理架构组件的执行和交互的软件。
-添加一些简单的功能。
· 结果是一个可以添加有用功能的运行系统

Creating a Skeletal System (Cont'd)

- To lower risk the most problematic functions should be added first.
- Then the functionality needed to support those problematic functions is added.
- This process is continued, growing larger and larger increments of the system until it is complete.

· 为了降低风险，应该首先添加最有问题的功能。
· 然后添加支持这些有问题的功能所需的功能。
· 这个过程是持续的，系统的增量越来越大，直到完成