Software Architecture

- principle and practice

第七讲　结构设计空间及其量化(Architecture Design Guidance)

对于一个给定的需求，人们如何选择一个合适的软件结构？具有多年建立系统经验的优秀系统设计员（good system designer)心中想到的只是当前可用结构的专门技术。然而，作为一门成熟的领域，我们希望从这些专门技术之中找到一些方法，帮助我们在建立软件结构时能在一个结构设计空间中依据原理进行选择。在这一讲中，我们将重点介绍两种结构设计的原型方法（prototype tool)，一种是用户接口结构设计（user-interface architecture)，另一个是量化设计空间(Quantified Design Space)。

- 一.设计空间和规则（design space and rule)
  - 1.设计空间（design space)
    - 把能用来描述、分类，并可供系统设计员选择的有效的结构集合称为设计空间。
    - 一个指定的应用系统的设计空间(desigm space)是指在建立该系统设计时所有可供选择的功能维(functional dimensions)和结构维(structure dimensions)的集合。
  - 2.维(Deimension)
    - 设计空间是多维的，每个维描述系统的一个特征（characteristic)或需求(requirement)的变化(variation)，维的值表示维所对应的特征或需求所有可选择范围。
    - 功能维(functional dimensions)表示问题空间（ploblem space)即功能设计或性能设计的可选择的各个方面。
    - 结构维(structure dimensions)表示方法空间(solution space)即实现一个特征的所有可选择的方法。
    - 设计空间的维通常是不连续的，并且没有度量单位。表示也可能是。表示结构的维通常是一组离散的(discrete)值的集合，它们可能是没有任何意义的序列，如状态转换图(state transition diagram)、菜单(menu)、表单(form)等。有一些维在理论上是连续的(如表示性能的维)，我们通常把它分成几个不连续的部分，如low、medium、high等。
  - 3.设计规则（design rule)
    - 设计空间中的维数，可能是相关的，也可能是独立的，设计规则(design rule，简称规则)是指维之间的所有关系，主要两种类型：有功能维与结构维的关系规则、结构维内部之间的关系规则。
    - 规则（rule)可用正（positive)、负(negative)权值（weight)表示,这样规则体现了维之间关系的紧密程度或选定一种组合的好、坏。
    - 一旦从设计空间中选定一个设计，那么就可以通过权值来计算这个设计的得分(score)，如果能将一个应用的设计空间的所有设计的得分计算出来，就可以比较各种设计的得分，以找出一个最合适的设计。
    - 对于一个经验丰富的高级设计员，可以建立一个足够完备(complete)和可靠(reliable)的规则，作为自动系统设计(automated system design)的基础。但是并不是说有了规则，就可以设计出一个完善的或一个最有可能的系统。规则可以帮助一个初级设计员（journeyman designer)像高级设计员（master designer)一样去选择一个合适的、没有大错误的设计。一个自动系统设计(automated system design)的实现将是一个漫长的过程。
  - 4.设计知识库（design vocabulary)
    - 设计知识库（design vocabulary)
      - 表示软件设计的知识（knowledge）,一种好的方法就是建立一个设计知识库（design vocabulary)，一组容易理解（well-understood)、可重用(reusable）的设计概念(concept)和模式(pattern)。
    - 使用设计知识库的好处（major benefit)
      - 便于建立新系统（mental building blocks）；
      - 帮助理解和预见（predicting)一个设计的属性(property)；
      - 减少了学习新的概念的数量，因而减少用于理解他人的设计的努力（effort)。
      - 比如，程序结构由控制流程的少数几个标准概念【条件（conditional)、循环(iteration)、子过程调用(subroutine call)等】来描述，程序避免了早期所使用的复杂的底层测试和分枝等操作，而直接使用这些大家普遍理解和接受的控制流程模型，并把精力集中于这些模型的属性（如循环语句中的条件变量、结束条件等），这样不仅容易编写、容易阅读而且更加可靠。
      - 随着软件工程的成熟和研究关注转向大的系统，我们期待类似于程序结构的标准也会在大型软件系统中出现，现存中型(medium-size)软件的结构模型的正在不断走向成熟，人们已经注意到（anticipate)对于一个大型系统更高层的抽象是必要的。
  - 5.进一步讨论
    - 一个简单的设计空间的例子
      - 上面，我们主要介绍由不同的系统结构构成的多维设计空间(multidimensional design space)的概念，设计空间中的每个维描述了系统的一个特征及其可选择的范围，维上的值(即规则)则表示了需求与选定设计(design choices)的关系程度。比如，在一个并发进程的同步机制(interprocess synchronization mechanism)中，响应时间(response time)作为一个维，需求为Fast,Med和Slow,同步机制作为另一个维,结构有Message,Semaphore,Monitor,Rendezvous,Other,None等。

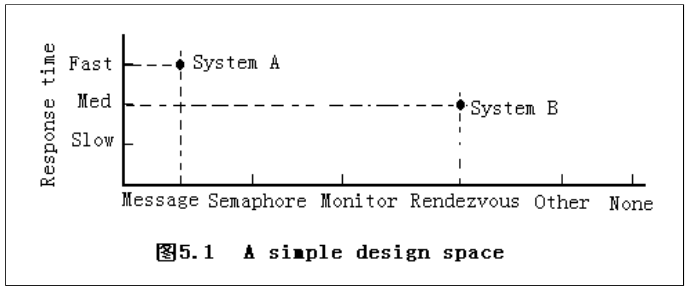


图5.1 A simple design space

- 一个设计空间中不同维之间不一定是独立的(independent)，事实上，为了建立设计规则，发现维之间的关系是非常重要的。
- 构造设计空间的关键是选择维，一些维反映需求或评估标准(evaluation criteria)，如功能、性能等，另一些维则反映结构及其可用的设计。在维之间找出关系可以提供直接的设计指导，因为这些关系表明了设计选择是否最符合新系统的功能需求。在图5.1的例子，假设message机制能提供Fast的响应时间，而Rendezvous机制只能提供Med的响应时间。

- 设计空间的组成
  - 功能设计空间(functional design space)：描述设计空间中功能(function)和性能(performence)方面的需求。
  - 结构设计空间(structure design space)：表示设计空间可用的结构。
  - 比如，在瀑布模型(watefall model)中功能设计空间表示需求分析的结果,而结构设计空间则是指设计阶段初始系统的分解(decomposition)

二、用户接口结构（user-interface architecture)设计

1.用户接口结构的设计空间

- 下面，我们介绍一个设计用户交互接口的软件的设计空间的建立。
- 基本结构模型(a basic structure model)
  - 为了描述可用的结构，我们需要定义几个术语(terminology)用于表示系统的组件，这些术语要求是通用的，主要有：
    - 特定应用组件(aplication-specific component)：组件应用于一个特定的应用程序，不期望在其他程序中重用，这些组件主要是应用程序的功能核心(functional core)。
    - 可共享用户接口(shared user interface)组件:这些组件是供多个应用程序使用的用户接口。如果一个软件要求能适应(accommodate)各种不同的I/O设计，那么只有能应用于所有这些设备类型的组件才作为可共享用户接口组件。
    - 依赖设备(device-dependent)组件:这些组件依赖于特定的设备类型且不是特定应用组件。
  - 在一些简单的系统中，可以没有后两种组件，比如，系统可能没有设备驱动程序的代码，或者系统不要求提供支持多类型设备的应用。
  - 在一些中等规模的设计空间中，接口分成两种设备接口(device interface)和应用接口(application interface)，图5.2表示了用户接口的一种基本结构模型。

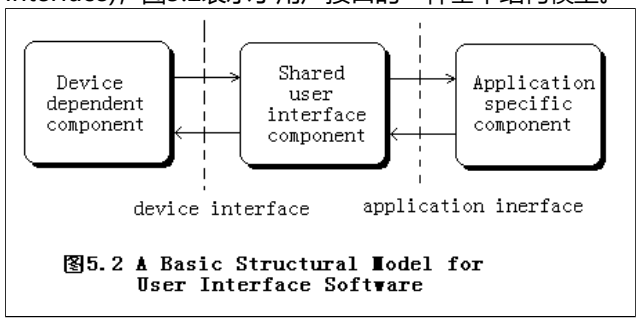


图5.2 A Basic Structural Model for User Interface Software

- 功能维例子(sample functional dimensions)
  - 有研究表明一个完整的用户接口系统的设计空间包括25个功能维和19个结构维，这里我们只能选择一些有代表性的6个功能维和5个结构维。
  - 用户接口系统的功能维

Functional Dimensions	
1.External event handling	
	No external events
	Process events while waiting for input
	External events preempt user commands
2.	User customizability
	High
	Medium
	Low
3.	User Interface adaptability across devives

	None
	Local behavior change
	Global behavior change
	Application semantics change
4.	Computer system organization
	Uniprocessing
	Multiprocessing
	Distributed processing
5.	Basic interface class
	Menu selection
	Form filling
	Command language
	Natural language
	Direct manipulation
6.	Application portability across user interface styles
	High
	Medium
	Low

- 以上6个功能维可分为3组
  - External requirements(外部需求）：这组维数主要包括特定应用、用户、I/O设备的支持需求，以及周围计算机系统的约束。
  - Basic interactive behavior（基本交互行为）:这组维数主要包括影响内部结构的用户接口行为。
  - Practical considerations（考虑实际应用）:主要指系统的适应性。
- External requirements(外部需求)
  - External event handling（外部事件处理)
    - 这是一个反映赋于应用程序影响(impose)外部需求的维，指出应用程序是否需要响应外部事件、何时响应等这方面的选择。维的取值如下：
    - No external events（没有外部事件）：
      - 应用程序不受外部事件（external events）的影响，或者只是在执行到指定的用户命令时自动检查外部事件。例如，一个电子邮件程序可能检查是否有新的邮件，这种检查仅仅是在执行指定的命令时才进行。用户接口（user-interface)如果选择这种类型将为支持外部事件。
    - Process events while waiting for input（等待输入时处理事件)
      - 应用程序必需处理外部事件（external events），但响应时间的要求不是很严格(stringent)，只是在执行用户输入命令时响应中断。
    - External events preempt user commands(外部事件剥夺用户命令)
      - 要求外部事件具有较高的优先级， 随时中断用户命令的执行，这种需求在实时控制系统(real-time control)中是很普遍的。
  - User customizability(用户定制)
    - 这是一个反映赋于用户影响外部需求的维，设计空间可取三种值：
    - High(高级)
      - 用户可以添加新的命令或重定义原命令的执行，同时也修改用户接口的细节。
    - Medium(中级)
      - 用户可以修改不影响命令语义(semantics)的用户接口细节，如，菜单项中的命令名、窗口的大小、颜色等允许用户自己定制。
    - Low(低级)
      - 很少或没有用户自定义的接口需求。
  - User Interface adaptability across devives(适应跨设备的用户接口)
    - 该维依赖于应用程序中用户接口支持的I/O设备。它反映当改变不同的I/O设备类型时用户接口行为的改变程度(extent)，设计空间取值有：
    - None(无)
      - 对于系统所支持的所有设备，所有的接口行为都是一样的，即无需任何改变。
    - Local behavior change(局部行为改变)
      - 当设备改变时，只允许改变个别的接口细节，如修改菜单的外观(appearance)。
    - Global behavior change(全局行为改变)
      - 跨越不同设备时，用户接口行为在表面上有较大的改变，比如需要改变用户接口类型。
    - Application semantics change(应用语义改变)
      - 需要根本上修改命令的语义。
  - Computer system organization(计算机系统组织)
    - 该维反映应用程序执行的周围环境。主要有
    - Uniprocessing(单一处理)
      - 一个时间内只有一个应用程序

        - Multiprocessing(多重处理)
            - 多个应用程序并发执行(concurrent)
        - Distributed processing(分布式处理)
            - 计算环境是网络的、多CPU的和不可忽略的(non-negligible) 通信开销等。
    - Basic interactive behavior（基本交互行为）
        - Basic interface class(基本接口类型)
            - 该维表示用户接口所支持的人-机交互能力
            - Menu selection(菜单选择)
                - 可以从事先定义的菜单中选择执行，每次选择后对应命令的执行也显示出来，允许多次选择。
            - Form filling(表单填写)
                - 通过填写表单中指定的参数，决定程序的运行。
            - Command language(命令语言)
                - 提供一种人工的符号语言，扩充过程定义，类似于一种程序语言。
            - Natural language(自然语言)
                - 接口操作基于一种自然语言(如英语)的子集，关键问题是解释那些不明确的(ambiguous)输入。
            - Direct manipulation(直接操作)
                - 接口操作直接基于图形表示
            - 需要提出，上述中只有菜单选择和表单填写可以支持同一种系统结构，其他类型的需求是单一的。
    - Practical considerations（考虑实际应用）
        - Application portability across user interface styles(应用程序对用户接口风格的适应)
            - 该维定义了用户接口系统的适应性级别，表明用户接口对特定应用(aplication-specific)组件的隔离程度。
            - High(高级)
                - 应用程序可以交叉使用不同的用户接口风格, 比如命令语言和菜单。
            - Medium(中级)
                - 应用程序独立于用户接口的少量风格变化，如菜单的外观。
            - Low(低级)
                - 没有考虑用户接口的改变，或者当用户接口以改变时应用程序的改变是可以接受的。
- 结构维例子(sample structure dimensions)
    - 用户接口的结构维

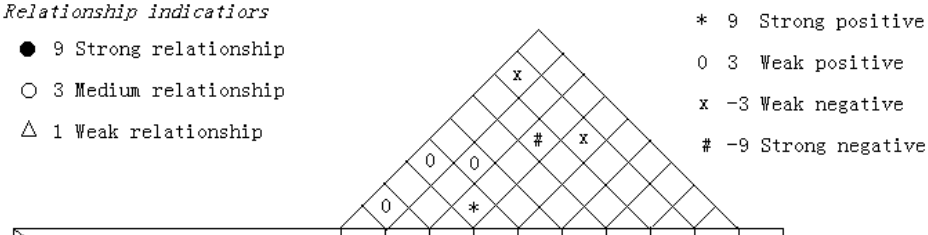
Structural Dimensions	
1	Application interface abstraction level
	Monolithic program
	Abstract device
	Toolkit
	Interaction manager with fixed data types
	Interaction manager with extensible data types
	Extensible interaction manager
2	Abstract device variability
	Ideal device
	Parameterized device
	Device with variable operations
	Ad-hoc device
3	Notation for user interface definition
	Implicit in shared user interface code
	Implicit in application code
	External declarative notation
	External procedural notation
	Internal declarative notation
	Internal procedural notation
4	Basic of communication
	Event
	Pure state
	State with hints
	State plus events
5	Control thread mechanism
	None
	Standard processes

	Lightweight processes
	Non-preemptive processes
	Event handlers
	Interrupt service routines

- 以上5种结构维可分为3组
  - Division of functions and knowledge among modules(模块中功能和信息的划分)：这组维主要说明模块中功能的确定、模块间的连接、以及每个模块所包含的信息。
  - Representation issues（表示问题）:这组主要考虑系统中使用的数据表示方法，也就是说要既要考虑在用户接口中使用的实际数据，又要考虑用于指定用户接口行为和外观的元数据(metadata)，系统中元数据可以是显式的(explicitly)，比如，用于描述对话框外观的数据结构，也可以是隐含的(implicitly)。
  - Control flow、communication、synchronization issues（控制流程、通信、同步问题）：这组主要考虑用户接口的动态行为(dynamic behavior)。
- Division of functions and knowledge among modules(模块中功能和信息的划分)
  - Application interface abstraction level （应用接口抽象层）
    - 在一定程度上，该维是一个关键的结构维，将其设计空间分为6个应用接口通过类型，它们由通信的抽象层来区分。
    - Monolithic program （单一程序）
      - 将所有接口功能集中在一个单一的程序中，没有区分特定应用程序和共享程序，只有一个模块也就不存在模块间的接口。主要适合于小的系统或是一些对用户接口的细节考虑全面的特殊系统。
    - Abstract device（抽象设备）
      - 共享组件是一些简单的设备驱动程序，为应用程序提供抽象设备的操作，接口行为的多数是由应用程序控制，少数也可由共享组件完成。在这类中应用程序接口就设备接口。
    - Toolkit （工具包）
      - 共享组件提供一个交互技术库（工具箱），如菜单、滚动条，应用程序则负责选择工具箱元素，并将它们组合成一个完整的接口。这样共享组件只能控制局部的用户接口风格部分，接口的全局行为由应用程序控制。共享组件和应用程序组件之间的交互是按照指定的交互技术（如得到菜单的选择）。对于工具箱中没有提供的交互技术，应用程序可以直接访问抽象设备层，此时应用程序负责实现特定应用的数据类型和面向设备的表示间的转换（conversion）。
    - Interaction manager with fixed data types （固定数据类型的交互管理）
      - 共享组件控制局部和全局的交互顺序(sequence)t 和格式（stylistic)的决定，它与应用程序的交互被表示为抽象信息的传递，这些抽象传递使用一组标准的数据（如整数、字符串），应用程序必须以标准的类型解释它的输入和输出。
    - Interaction manager with extensible data types（可扩展数据类型的交互管理）
      - 在上一种功能的基础上，可能扩展数据类型，包括输入和输出的数据转换也要求是新的数据类型，要完整地使用这种方式，一般要求接口的外部表示与应用主程序分开。
    - Extensible interaction manager可扩展交互管理）
      - 应用程序与共享组件间的通信也是通过抽象信息传递，应用程序可以定制扩展交互管理，这要求有而面向对象语言和继承机制。
  - Abstract device variability （抽象设备可变性）
    - 这是描述设备接口(device interface)的关健维，为了设备独立性的操作我们把设备接口视为抽象设备（abstract device)。
    - Ideal device（观念设备）
      - 在观念设备中，操作及其结果经过详细说明(specify),比如，广泛使用的postScript的图像模型，可以不考虑实际的打印机或显示器的分辨率。在这种方法中，设备可变性都由设备驱动层隐藏，因此应用程序的可移植性很强，当观念设备与实际设备偏离很小时，这种方法非常有用。
    - Parameterized device（参数化设备）
      - 设备按一些指定的参数分类，如参数可以是屏幕大小、颜色数、鼠标的键数等。设备独立性组件能根据参数值调整其行为，操作和结果也是经过详细说明的，但要依赖于对应的参数，这种方式的好处是可以设备以最适合的参数操作，但它使得设备独立性代码要分析设备的参数所有范围，在应用程序中实现开销很大。
    - Device with variable operations（可变操作的设备）
      - 拥有一组良定义的设备操作，但设备独立性代码在选择操作实现时可能有很大地偏离(leeway),操作的结果也就可能不明确。这种方法在设备操作的选择在在抽象层而驱动程序可以在很大范围内选择时可以发挥作用。
    - Ad-hoc device（专用设备）
      - 在很多实际系统中，抽象设备定义局限于一种特别的方式，设备的不同其操作行为也改变，这样应用程序可使用的设备种类的范围很小，也就是说应用只能使用专门的设备。
- Representation issues（表示问题）
  - Notation for user interface definition （用户接口定义符号）
    - 这是一个表示维，它说明用于定义用户接口外观和行为的技术。
    - Implicit in shared user interface code（隐含在可共享用户接口代码）

        - 这是一种完全可接受的方式，比如，菜单的可视外观是隐含中工具箱提供的菜单过程中。
      - Implicit in application code（隐含在应用代码）
        - 用户接口外观和行为隐含在应用中,可用于当应用程序中紧含着用户接口的情况。
      - External declarative notation（外部说明符号）
        - 用户接口外观和行为在非过程部分定义，与应用程序分开。这种方式支持用户自定义。
      - External procedural notation （外部过程符号）
        - 用户接口外观和行为在一个过程中定义并与应用程序分开，这种方式较灵活但使用困难。
      - Internal declarative notation （内部说明符号）
      - Internal procedural notation（内部过程符号）
    - Control flow、communication、synchronization issues（控制流程、通信、同步问题）
      - Basic of communication （基本通信）
        - 这是一个通信维，该维按模块间的可共享状态(shared state)、事件(event)通信方法区分。事件(event)是指发生在一个离散时间的信息传递。
        - Event（事件）
        - Pure state （纯状态）
        - State with hints （状态提示）
        - State plus events（状态＋事件）
      - Control thread mechanism （控制线程机制）
        - 描述是否支持逻辑多线程，在用户接口系统中支持多线程是很重要的。
        - None
          - 只使用一个单独的线程（只有一个线程的进程）
        - Standard processes
          -
        - Lightweight processes
        - Non-preemptive processes
        - Event handlers
        - Interrupt service routines
  - 2.用户接口结构的设计规则
    - 规则指示了维之间的关系程度，主要有两种类型
      - 功能维与结构维的关系规则
        - 允许系统需求驱动结构设计
      - 结构维内部之间的关系规则
        - 描述系统内部的一致性(consistency),这种规则使要发现一个具有更高得分的设计变得复杂。
    - 规则例子(sample rules)
      - if exteral event handling requires preemption of user commands, then a preemptive control-thread mechanism(standard process, lightweight processes,or interrupt service routines)is strongly favored. Without such a mechanism, very severe constraints must be placed on all user-interface and application processing in order to guarantee adequate response time.
      - High user customizability requirements favor external notation for user-interface behavior. Implicit and internal notations are usually more difficult to access and more closly coupled to application logic than external notations.
      - A distributed system organization favors event-based communication. State-based communication requires shared memory or some equivalent, which is offen expensive to access in such an environment.
- 三、量化设计空间(Quantified Design Space，QDS)
  - 1.简介
    - 人们对软件产品(artifact)的分析已经有多年的研究和实践。在理论和经验上，系统的性能(performance)、复杂性(complexity)、可靠性(reliable)、可测试性(testability)以及其他所期望的质量的度量已经成为软件工程的主要组成(integral part)。然而软件产品作为软件开发过程的最后一个阶段，对其分析的是重要的。现在已经达到共识：软件产品可以与开发过程所依赖的设计一样好。
    - 在过去，软件分析工具的开发并没伴随(accompany)人们对软件设计的逐渐关注而发展。原型技术（ prototyping techniques)能证明（demonstrate）设计阶段的可行性（feasibility）或研究（investigate）设计中的特殊问题。形式软件检查(inspection)可以在后期开发阶段展开之前尽早地发现设计上的错误。形式描述语言(formal specification language）可以用于描述一个设计。但现在不没有一个能从理论和质量上对设计进行分析的工具。这样的工具应能提供一些度量设计符合产品所拥有的需求的程度，而无需构造一个原型，并且能使一个设计者将自己的知识应用于一个应用领域或致力于软件设计和构造。
    - 量化设计空间(Quantified Design Space,QDS)是美国卡内基.梅隆（Carnegic Mellon)大学（CMU）提出的。**QDS**的目标是分析和比较一个指定应用的各种软件设计，**QDS**的思想是基于设计空间(design space)和质量配置函数（quality function deployment，**QFD**)，它是一种将系统需求转换为可选择的功能和结构设计，量化（quantified)分析各种的可能选择及其组合。**QDS**可以为一个期望系统(desired system)制作(produce)一个设计模型，也可以用来分析和对比已有的设计，或对现有的设计提出改进建议。
    - 质量配置函数（QFD)

- QFD是一个质量保证（assurance）技术，其作用是将客户的需求转换为产品从需求分析到设计开发各个阶段的技术需求，或转换为制造(manufacturing)和维护(support)。1997年QFD应用于日本的汽车制造并获得显著的效果，现在已经在日本和美国得到普遍应用。QFD的基本目标（fundamental）是“客户的声音”(voice of customer)出现在（manifested）产品开发过程的各个阶段。QFD的观点（philosophy）是让所有的投资者对产品开发达成一个共识（common understanding）。
- QFD技术带来的好处
  - 问题(issues)发现在开发阶段，并且是问题形成之前。
  - 决定(decision)记录在一个有结构的框架中，必要时可以跟踪(trace)。
  - 在开发过程中过程的各个阶段需求(requrement)不容易被误解（misinterpreted)。
  - 重复工作(rework）减少，需求改变最小化，减少产品进入市场所花的时间。
  - 由定义的框架表示产品知识(knowledge)，不仅可供现有产品开发使用，还可用于将来的工作。
  - 最终产品更接近的客户需求。
- QFD过程
  - QFD的基本原理（philosophy）是通过一组详细说明的图形符号和用于开发过程各个阶段的良好定义的转换机制（将需求转换为实现）。QFD实现协作工程的多用户设计系统的产品分析（同时有多用户从从的单个设计侧面协同工作）。QFD过程的步骤如下
  - 步骤1：建立产品或服务开发的范围，招集投资者代表参与QFD工作，收集各阶段的用户需求（这些需求用户可按它们自己的语言表示），由交叉功能小组分析这些需求并达成共识，将这些需求加入到QFD表（如图5.4）的左边列中。



Realization mechanisms / Customer requirements	Distributed system	Multiple applications	Database manager	Client-server architecture	Customizable user interface	Rule-based reasoning	Store data in ASCII files	Real-time conference mode	etc.	Customer importance
Support cooperative engineering	○	●	●	△	○			●		10
Work with large designs	△		●	●						5
Support full building life cycle	△	●	●		○		●	△		5
Apply expert knowledge		○	△			●				1
etc.										
Organizational difficulty	1	10	1	2	1	10	1	10		
Objective target values	LAN or WAN	One per type of user	etc							
Technical importance — Absolute	40	138	181	55	45	9	45	95		
Technical importance — Relative	7	2	1	4	5	8	5	3		

图5.4 QFD框架实例

  - 步骤2:QFD小组选择满足用户需求的实现机制(realization mechanism),这些机制组织在QFD图表的上端，建立用户需求对实现机制的关系矩阵。
  - 步骤3:为每个实现机制建立目标值（target value),并填入QFD图表的关系矩阵的下方，这些目标可以是量化数值或量化目标，但这些目标都是可以主观验证的（verifiable）。
  - 步骤4:建立机制与需求的关系（relationship），以确定不实现用户需求哪一种机制是最重要的每一种关系可用图形符号表示其程度（high、medium、Weak），如果某个机制不满足一个需求，则对应的关系矩阵的值为空。例如，图中，database manager机制与需求Work with large designs有强（strong)的系统性能关系，但与需求Apply expert knogledge系统能力上关系较弱(Weak)。这些关系赋于一个权值(weight)以便计算每个实现机制的重要性。分析这些关系是QFD的主要能力(strength)之一,比如，图5.4中，起始可能认为database manager机制主要满足Support cooperative design, 但更进一步分析表明，有效的数据库database manager机制对需求work with large designs也有很好的性能，这种“副效益”(side-benefit)在QFD框架中可以很清楚地表示出来。
  - 步骤5:计算机制间的相关性（correlations）即各种实现机制间的正、负关系，正数关系表示两种机制彼此都很好，负数关系表示一种机制的实现会干扰(interfere)另一个机制的实现。这些关系在QFD图表的

上方，用图形符号表示关系程度，空格表示两个机制没有联系。比如，图5.4中表明ASCII文件的使用与数据库管理有很严重的负关系。

- 步骤6:估计(assess)每一种机制的实现难度也根据组织的强弱，困难等级(rating）记录在QFD图表中target value的上方。
- 步骤7:计算每种机制的重要等级：一种机制的各关系值乘以QFD表右边列(customer importance)对应客户值的并相加得到这种机制的技术绝对值。为了明确起见，可以再将这些绝对值转换为相对值，它们分别记录在QFD图表的底部。
- 步骤8: 建立了关系（relationship）和相关性(correlation)后，FQD小组就可以分析该框架，选择在产品中使用的机制。
  - a.确定每种机制的目标值；
  - b.检查（inspect）每种相关性,找出可能冲突的机制，这突出某种机制允许为其相关性赋于一个权值，组织困难等级用数据表示出来。
  - c.排除具有负相关性、高困难等级、风险性和低技术重要性的机制
  - d.对于尚未充足表达的需求，可以再次检查。
- 过程的结果可以作为下一阶段的产品开发。
- QFD总结：为完成这个QFD例子，每个实现机制 （realization mechanism)要指出其目标值(target value),实现机制的各列要按相对技术重要性重排。这个框架还可以作为下一个QFD的输入，比如选定database manager机制，另个指定通信使用(client-server),作为下一阶段QFD的输入。

2.量化设计空间(QDS)

QDS原理

- QDS的内容是设计空间的概念和QFD技术。
- QDS从设计空间中提取概念、一个设计分解成维及其可选择的设计、设计维间的正负相关性，分析需求与实现机制间的关系，应用QFD过程。
- 这种基于QFD框架的设计空间的实现称为量化设计空间(QDS)

应用QFD框架实现一个设计空间

- QFD过程的共同目标是获取设计知识，即关于实现机制与客户需求间的关系和机制间相关性的知识。
- 设计空间的规则(rule)能从QFD框架中的关系和相关性获得。比如，一个规则表明一个“distributed system”不能在一个“monolithic program”可实现。
- 通用功能设计空间（Generalized Functional Design Space）
  - 客户需求列在表的左边，赋于这些客户需求的权置于表中另一个列，功能维及其可选择的设计空间位于表的上方，并与需求构成的关系矩阵，关系值越大表明功能越能满足需求，计算每个功能的“calculated weight”,及各需求的权，表的“design decision”中1表示可选用的功能。“effective weight”为“calculated weight”与“design decision”的积。

Concepts		Customer weight	Functional #1			Functional #2			Functional #3			
			Alt#1	Alt#2	Alt#3	Alt#1	Alt#2	Alt#3	Alt#1	Alt#2	Alt#3	Alt#4
Requierment#1		10	1	9	0	9	3	1	0	0	0	0
Requierment#2		9	3	1	0	3	0	3	3	3	9	0
Requierment#3		2	1	3	0	3	0	0	1	1	3	0
Requierment#4		8	3	3	1	9	1	3	1	1	3	0
Calculated weight			63	129	8	195	38	61	37	37	111	0
	Requierment#1		10	90	0	90	30	10	0	0	0	0
	Requierment#2		27	9	0	27	0	27	27	27	81	0
	Requierment#3		2	6	0	6	0	0	2	2	6	0
	Requierment#4		24	24	8	72	8	24	8	8	24	0
Design decision				1		1					1	0
Effective weight			0	129	0	195	0	0	0	0	111	0

图5.5 Generalized function design space

- 通用结构设计空间（Generalized Structure Design Space）

Concepts		Effective	Structural#1					Structural#2		Structural#3		
Dimensions	Alternatives	weight	Alt#1	Alt#2	Alt#3	Alt#4	Alt#5	Alt#1	Alt#2	Alt#1	Alt#2	Alt#3
Function#1	Alt#1	0	1	1	1	3	1	0	9	3	0	3
	Alt#2	129	3	1	1	2	1	0	3	3	0	9
	Alt#3	0	3	1	1	2	1	0	3	9	0	9
Function#2	Alt#1	195	3	1	0	0	1	1	9	0	0	9
	Alt#2	0	3	1	0	0	1	1	9	9	0	3
	Alt#3	0	1	1	0	0	1	1	9	0	0	3
Function#3	Alt#1	0	0	0	1	0	0	0	3	3	1	3
	Alt#2	0	0	0	1	0	0	0	3	3	1	3
	Alt#3	111	0	1	0	0	1	0	3	3	1	3
	Alt#4	0	9	0	0	0	0	1	3	0	0	0
Calculated weight			972	435	129	258	435	195	2475	720	111	3249
	Requirement#1		540	180	90	180	180	90	1080	270	0	1620
	Requirement#2		108	117	9	18	117	27	513	270	81	567
	Requirement#3		36	18	6	12	18	6	90	36	6	126
	Requirement#4		288	120	24	48	120	72	792	144	24	936
Design decision			1						1			1

图5.6 Generalized structural design space

- 结构设计可选择相关性(Corrlation of Structural Deisgn Alternative)

Concepts		Structural#1					Structural#2		Structural#3		
Dimensions	Alternatives	Alt#1	Alt#2	Alt#3	Alt#4	Alt#5	Alt#1	Alt#2	Alt#1	Alt#2	Alt#3
Structural#1	Alt#1		0	0.5	0.5	1	0.5	0.5	1	0.5	1
	Alt#2			0.5	1	0.5	1	1	1	0.5	0.5
	Alt#3				0	0.5	1	0	0.5	0	0
	Alt#4					0.5	0.5	0.5	0.5	0	0.5
	Alt#5						0.5	1	1	0.5	1
Structural#2	Alt#1							0	1	0.5	0.5
	Alt#2								0.5	0	0
Structural#3	Alt#1									0	0
	Alt#2										0
	Alt#3										

图5.7 Correlation of Structural Design Alternative

- 量化设计空间值(score)

Score		5944.5
	Requirement#1	2970.0
	Requirement#2	985.5
	Requirement#3	225.0
	Requirement#4	1764.0

图5.8 Sample Score of a Quantified Design space

- Let E1、E2、...、En be the effective weight of each selected structural design alternative
- Let Cmn be the correlation factor for alternatives m and n
- for each selected structural design alternative, add the effective weight of the design alternative to that of each other selected alternative, and multipy each term of the sum by the correlation factor from the matrix. The product sums are then added together, giving the final score for the design:Sroce=∑Cij(Ei+Ej)

- 3.结论

返回

==========