Styles and Greenfield Design
样式与绿地设计

- Heterogeneous Styles
  异构样式

- C2 Style
  C2 样式
  - Components
    组件
      - Independent, potentially concurrent message generators and/or consumers
        独立的、可能是并发的消息生成器和/或消费者
  - Connectors
    连接器
      - Message routers that may filter, translate, and broadcast messages
        可以过滤、翻译和广播消息的消息路由器
  - Data Elements
    数据元素
      - Messages – data sent as first-class entities over the connectors
        作为第一类实体通过连接器发送的数据
      - Notification messages announce changes of state
        通知消息宣布状态变化
      - Request messages request performance of an action
        请求消息请求执行操作
  - Topology
    拓扑结构
      - Layers of components and connectors, with a defined “top” and “bottom”
        组件和连接器的层，有定义的“顶部”和“底部”
      - Notifications flow downwards and requests flow upwards
        通知向下流动，请求向上流动

- Distributed Objects: CORBA
  分布式对象：CORBA
  - Components
    组件
      - Objects (software components exposing services through well-defined provided interfaces)
        对象（通过定义良好的提供的接口公开服务的软件组件）
  - Connector
    连接器
      - (Remote) Method invocation
        （远程）方法调用
  - Data Elements
    数据元素

- Arguments to methods, return values, and exceptions
  方法的参数、返回值和异常
- Topology
  拓扑结构
  - General graph of objects from callers to callees
    从调用者到被调用者的对象的一般图
  - Additional constraints imposed: Data passed in remote procedure calls must be serializable.
    附加强制约束：在远程过程调用中传递的数据必须是可序列化的
  - Callers must deal with exceptions that can arise due to network or process faults.
    调用者必须处理由于网络或进程故障而可能引起的异常
  - Location, platform, and language “transparency”
    位置、平台和语言“透明性”

Styles and Greenfield Design
样式和绿地设计
Heterogeneous Styles
异构样式
More complex styles created through composition of simpler styles
通过组合更简单的样式创建更复杂的样式

C2
C2
Implicit invocation + Layering + other constraints
隐式调用 + 分层 + 其他约束
Distributed objects
分布式对象
OO + client-server network style
面向对象 + 客户端-服务器网络样式
CORBA

C2 Style
C2 样式
An indirect invocation style in which independent components communicate exclusively through message routing connectors. Strict rules on connections between components and connectors induce layering.
一种间接调用的样式，其中独立的组件仅通过消息路由连接器进行通信。对组件和连接器之间连接的严格规则引发分层。
Components: Independent, potentially concurrent message generators and/or consumers
组件：独立的、可能是并发的消息生成器和/或消费者

Connectors: Message routers that may filter, translate, and broadcast messages of two kinds: notifications and requests.
连接器：消息路由器，可以过滤、翻译和广播两种类型的消息：通知和请求
Data Elements: Messages – data sent as first-class entities over the connectors. Notification messages announce changes of state. Request messages request performance of an action.
数据元素：消息 - 作为第一类实体通过连接器发送的数据。通知消息宣布状态变化。请求消息请求执行操作。
Topology: Layers of components and connectors, with a defined “top” and “bottom”, wherein notifications flow downwards and requests upwards.
拓扑结构：组件和连接器的层，有定义的“顶部”和“底部”，其中通知向下流动，请求向上流动。

Distributed Objects: CORBA
分布式对象：CORBA
“Objects” (coarse- or fine-grained) run on heterogeneous hosts, written in heterogeneous languages. Objects provide services through well-defined interfaces. Objects invoke methods across host, process, and language boundaries via remote procedure calls (RPCs).
“对象”（粗粒度或细粒度）在异构主机上运行，使用异构语言编写。对象通过定义良好的接口提供服务。对象通过远程过程调用（RPC）跨越主机、进程和语言边界调用方法。
Components: Objects (software components exposing services through well-defined provided interfaces)
组件：对象（通过定义良好的提供的接口公开服务的软件组件）
Connector: (Remote) Method invocation
连接器：（远程）方法调用
Data Elements: Arguments to methods, return values, and exceptions
数据元素：方法的参数、返回值和异常
Topology: General graph of objects from callers to callees.
拓扑结构：从调用者到被调用者的对象的一般图。
Additional constraints imposed: Data passed in remote procedure calls must be serializable. Callers must deal with exceptions that can arise due to network or process faults.
附加强制约束：在远程过程调用中传递的数据必须是可序列化的。调用者必须处理由于网络或进程故障而可能引起的异常。
Location, platform, and language “transparency”.
位置、平台和语言“透明性”。
CAUTION!
注意！
Observations
观察

Different styles result in
不同的样式导致

Different architectures
不同的架构

Architectures with greatly differing properties
具有极大差异性质的架构

A style does not fully determine resulting architecture
样式并不能完全确定最终的架构

A single style can result in different architectures
单一样式可能导致不同的架构

Considerable room for
有相当大的空间

Individual judgment
个人判断

Variations among architects
架构师之间的差异

A style defines domain of discourse
样式定义了讨论的领域

About problem (domain)
关于问题（领域）

About resulting system
关于结果系统

Design Recovery
设计恢复

What happens if a system is already implemented but has no recorded architecture?
如果系统已经实施但没有记录的架构，会发生什么？

The task of design recovery is
设计恢复的任务是

examining the existing code base
审查现有的代码库

determining the system’s components, connectors, and overall topology
确定系统的组件、连接器和整体拓扑结构

A common approach to architectural recovery is clustering of the implementation-level entities

into architectural elements.
对架构恢复的常见方法是将实现级别的实体聚类成架构元素。

Syntactic clustering
语法聚类

Focuses exclusively on the static relationships among code-level entities
专注于代码级实体之间的静态关系

Can be performed without executing the system
可以在不执行系统的情况下执行

Embodies inter-component (a.k.a. coupling) and intra-component (a.k.a. cohesion) connectivity
包含组件间（又名耦合）和组件内部（又名内聚）的连接性

May ignore or misinterpret many subtle relationships, because dynamic information is missing
可能会忽略或误解许多微妙的关系，因为缺少动态信息

Semantic Clustering
语义聚类

Includes all aspects of a system’s domain knowledge and information about the behavioral similarity of its entities.
包括系统领域知识的所有方面以及有关实体行为相似性的信息。

Requires interpreting the system entities’ meaning, and possibly executing the system on a representative set of inputs.
需要解释系统实体的含义，并可能在代表性输入集上执行系统。

Difficult to automate
难以自动化

May also be difficult to avail oneself of it
可能也难以利用它

When There’s No Experience to Go On…
当经验不足时...

Examine other fields and disciplines unrelated to the target problem for approaches and ideas that are analogous to the problem.
研究与目标问题无关的其他领域和学科，寻找与问题类似的方法和思路。

Formulate a solution strategy based upon that analogy.
基于类似性制定解决方案策略。

A common “unrelated domain” that has yielded a variety of solutions is nature, especially the biological sciences.
一个常见的“无关领域”是自然，尤其是生物科学，已经产生了各种解决方案。

E.g., Neural Networks
例如，神经网络

Brainstorming
头脑风暴
Technique of rapidly generating a wide set of ideas and thoughts pertaining to a design problem without (initially) devoting effort to assessing the feasibility.
一种快速生成与设计问题相关的广泛想法和思考的技术，最初不投入评估可行性的努力。
Brainstorming can be done by an individual or, more commonly, by a group.
头脑风暴可以由个人或更常见的是由一个团队进行。
Problem: A brainstorming session can generate a large number of ideas… all of which might be low-quality.
问题：头脑风暴会产生大量的想法...其中所有可能都是低质量的。
The chief value of brainstorming is in identifying categories of possible designs, not any specific design solution suggested during a session.
头脑风暴的主要价值在于识别可能设计的类别，而不是在一次会议期间提出的任何具体设计解决方案。
After brainstorm the design process may proceed to the Transformation and Convergence steps.
头脑风暴后，设计过程可以继续进行转化和融合步骤。

“Literature” Searching
文献搜索
Examining published information to identify material that can be used to guide or inspire designers.
检查已发布的信息，以确定可以用来指导或激发设计师的材料。
Many historically useful ways of searching “literature” are available.
有许多在历史上有用的搜索“文献”的方法。
Digital library collections make searching extraordinarily faster and more effective.
数字图书馆收藏使搜索变得异常快速和更有效。
IEEE Xplore
ACM Digital Library
Google Scholar
免费和开源软件的可用性为这项技术增添了特殊的价值。

Morphological Charts
形态学图表
The essential idea:
基本思想：

identify all the primary functions to be performed by the desired system
确定所需系统执行的所有主要功能

for each function identify a means of performing that function
对于每个功能，确定执行该功能的手段

attempt to choose one means for each function such that the collection of means performs all the required functions in a compatible manner.
尝试为每个功能选择一种手段，使这些手段的集合以兼容的方式执行所有所需的功能。
The technique does not demand that the functions be shown to be independent when starting out.
该技术并不要求在开始时显示功能是独立的。
Sub-solutions to a given problem do not need to be compatible with all the sub-solutions to other functions in the beginning.
对于给定问题的子解决方案在开始时不需要与其他功能的所有子解决方案兼容。

Removing Mental Blocks
消除心理障碍
If you can’t solve the problem, change the problem to one you can solve.
如果你无法解决问题，就把问题改成你能解决的问题。
If the new problem is “close enough” to what is needed, then closure is reached.
如果新问题“足够接近”所需，那么就达到了封闭状态。
If it is not close enough, the solution to the revised problem may suggest new venues for attacking the original.
如果不够接近，对修订问题的解决方案可能会提示攻击原始问题的新途径。

Controlling the Design Strategy
控制设计策略
The potentially chaotic nature of exploring diverse approaches to the problem demands that some care be used in managing the activity
探索问题的多种方法可能具有潜在的混乱性，因此在管理活动时需要一些注意。
Identify and review critical decisions
确定和审查关键决策
Relate the costs of research and design to the penalty for taking wrong decisions
将研究和设计的成本与错误决策的惩罚联系起来
Insulate uncertain decisions
隔离不确定的决策
Continually re-evaluate system “requirements” in light of what the design exploration yields
在设计探索产生的信息的光线下不断重新评估系统的“要求”

Insights from Requirements
来自需求的洞察

In many cases new architectures can be created based upon experience with an improvement to

pre-existing architectures.
在许多情况下，可以基于对现有架构的改进经验创建新的架构。

Requirements can use a vocabulary of known architectural choices and therefore reflect experience.
需求可以使用已知架构选择的词汇，因此能够反映经验。

The interaction between past design and new requirements means that many critical decisions for a new design can be identified or made as a requirement.
过去设计与新需求之间的交互意味着新设计的许多关键决策可以被识别或作为一个需求制定。

Insights from Implementation
来自实施的洞察

Constraints on the implementation activity may help shape the design.
对实施活动的限制可能有助于塑造设计。

Externally motivated constraints might dictate
外部驱动的约束可能会决定

Use of a middleware
使用中间件

Use of a particular programming language
使用特定的编程语言

Software reuse
软件重用

Design and implementation may proceed cooperatively and contemporaneously.
设计和实施可以协同进行并行。

Initial partial implementation activities may yield critical performance or feasibility information.
最初的部分实施活动可能产生关键的性能或可行性信息。