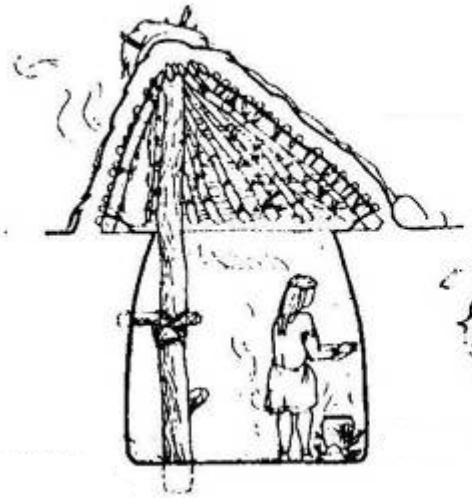
第1章 软件体系结构的基本概念（1）

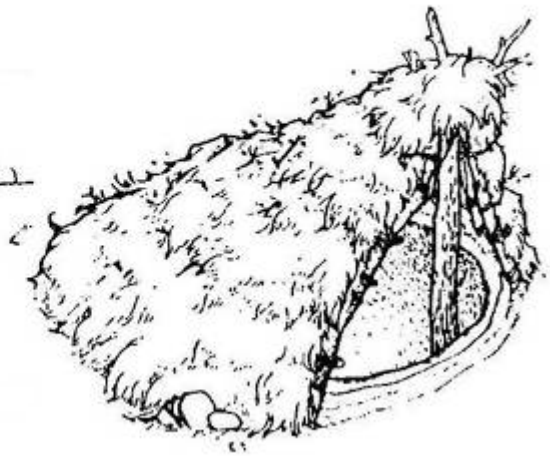
内容

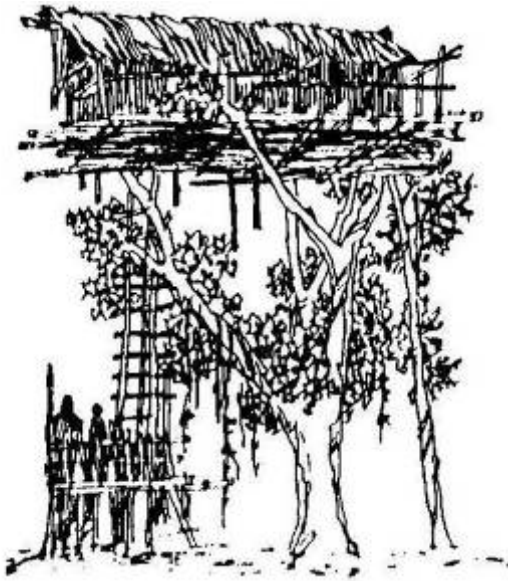
- **1.1 什么是软件体系结构**
- 1.2 软件架构结构和视图
- 1.3 软件架构视图模型
- 1.4 软件体系结构核心元模型
- 1.5 软件架构风格
- 1.6 其他相关概念

人类最早的居住方式：巢居和穴居

- 炎热或高海拔地区的穴居方式，可获得相对稳定的室内热环境，顶部的天窗既可采光又可排烟。

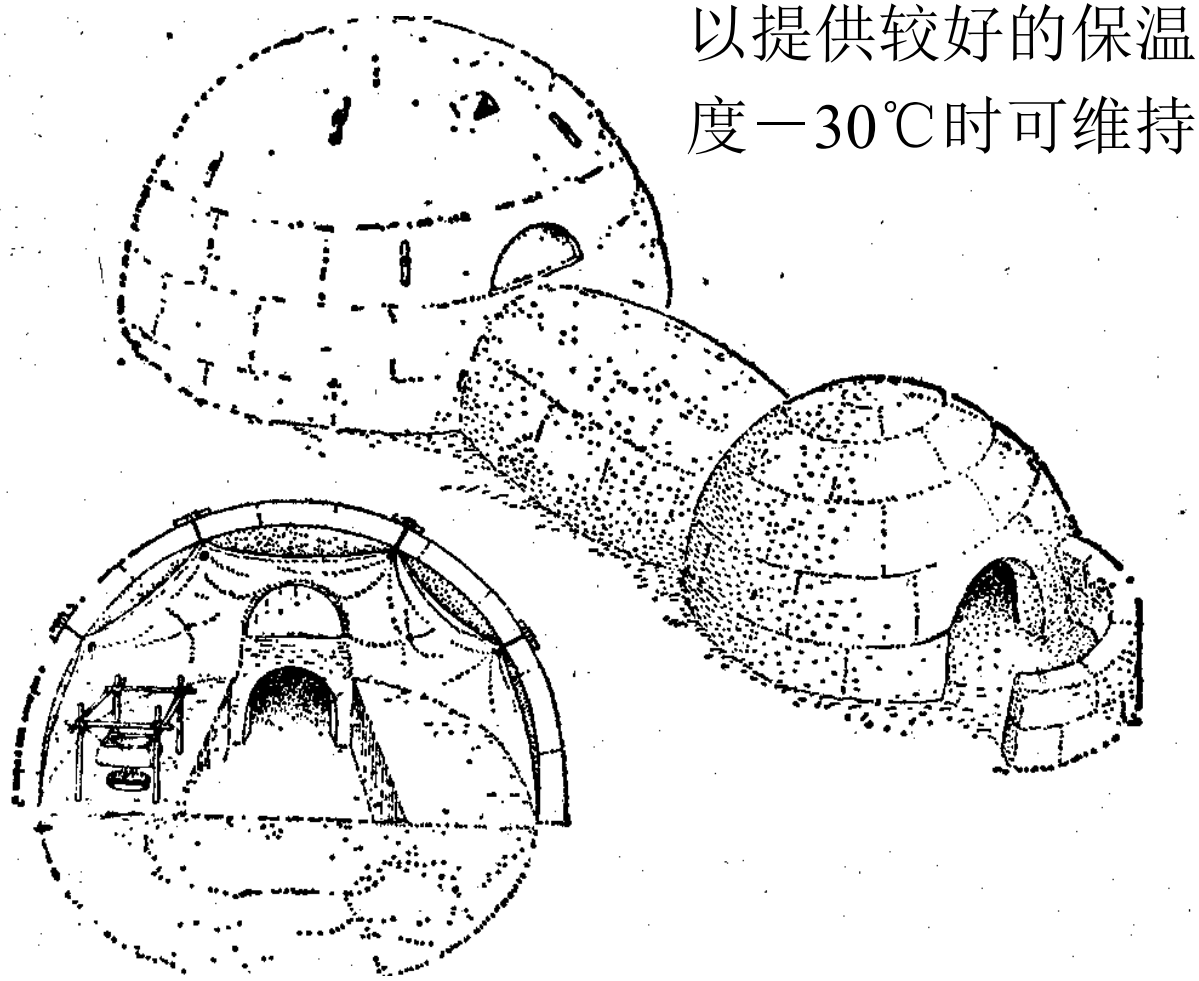






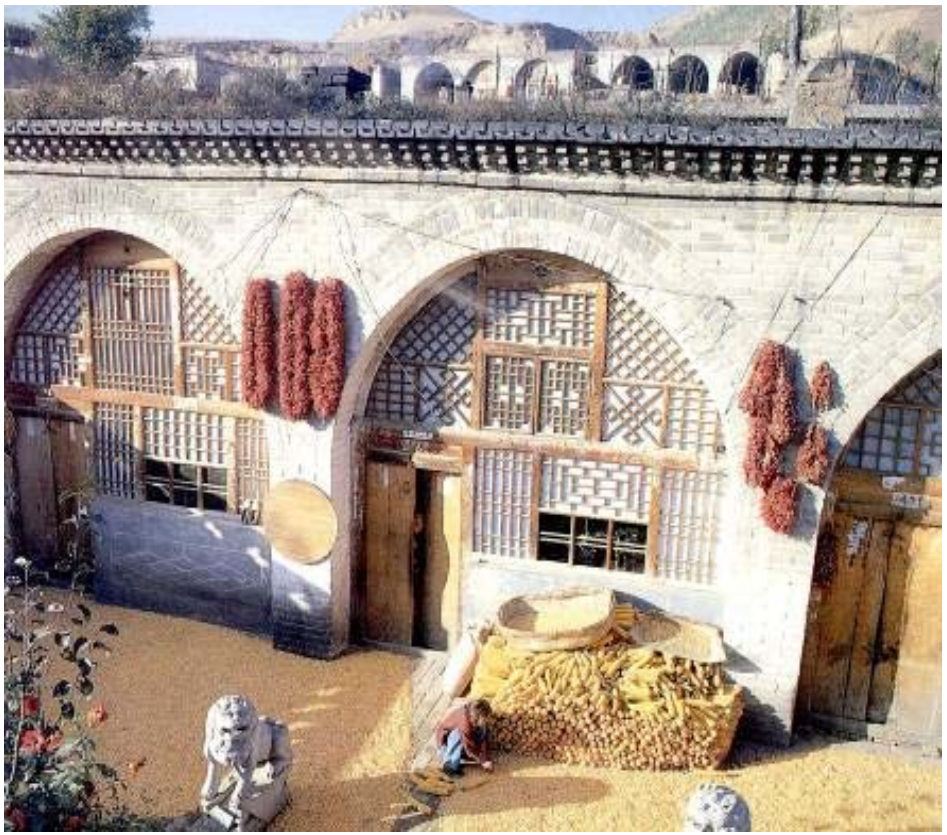
爱斯基摩雪屋的外观和室内布置

用干雪沏成，厚度500ｍｍ的墙体可以提供较好的保温性能。当室外平均温度－30℃时可维持室内温度－5℃以上。



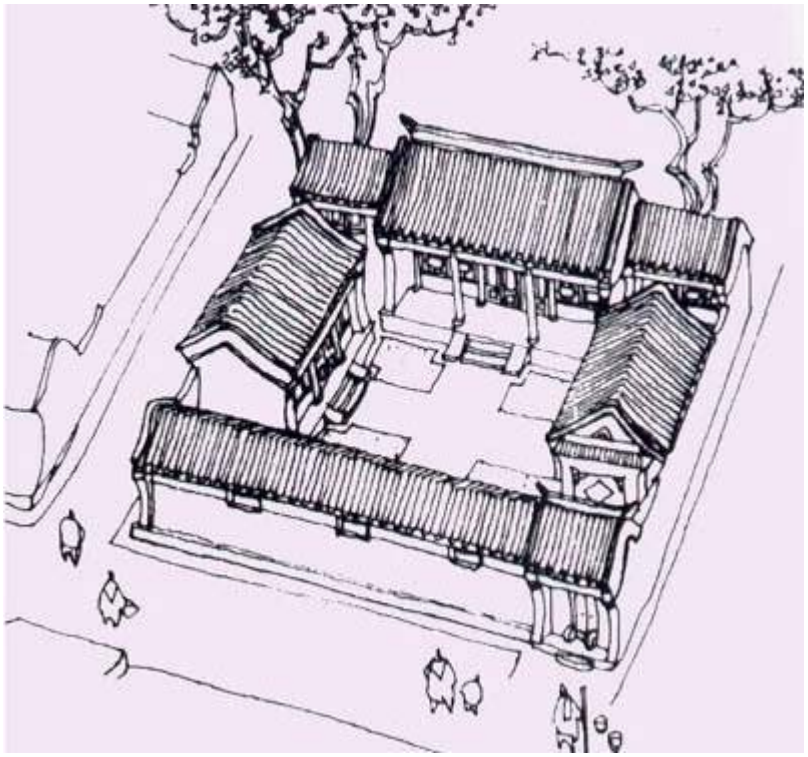
大陆气候的中国民居

土窑洞借助土壤大热惯性，达到冬暖夏凉的目的。



中国四合院：座北朝南的典范

- 利用太阳高度角的特点，仅在北方出现。
- 四合院建筑冬季有效地利用了太阳能采暖和抵御北风侵袭，屋顶设计避免了夏季室内过热。

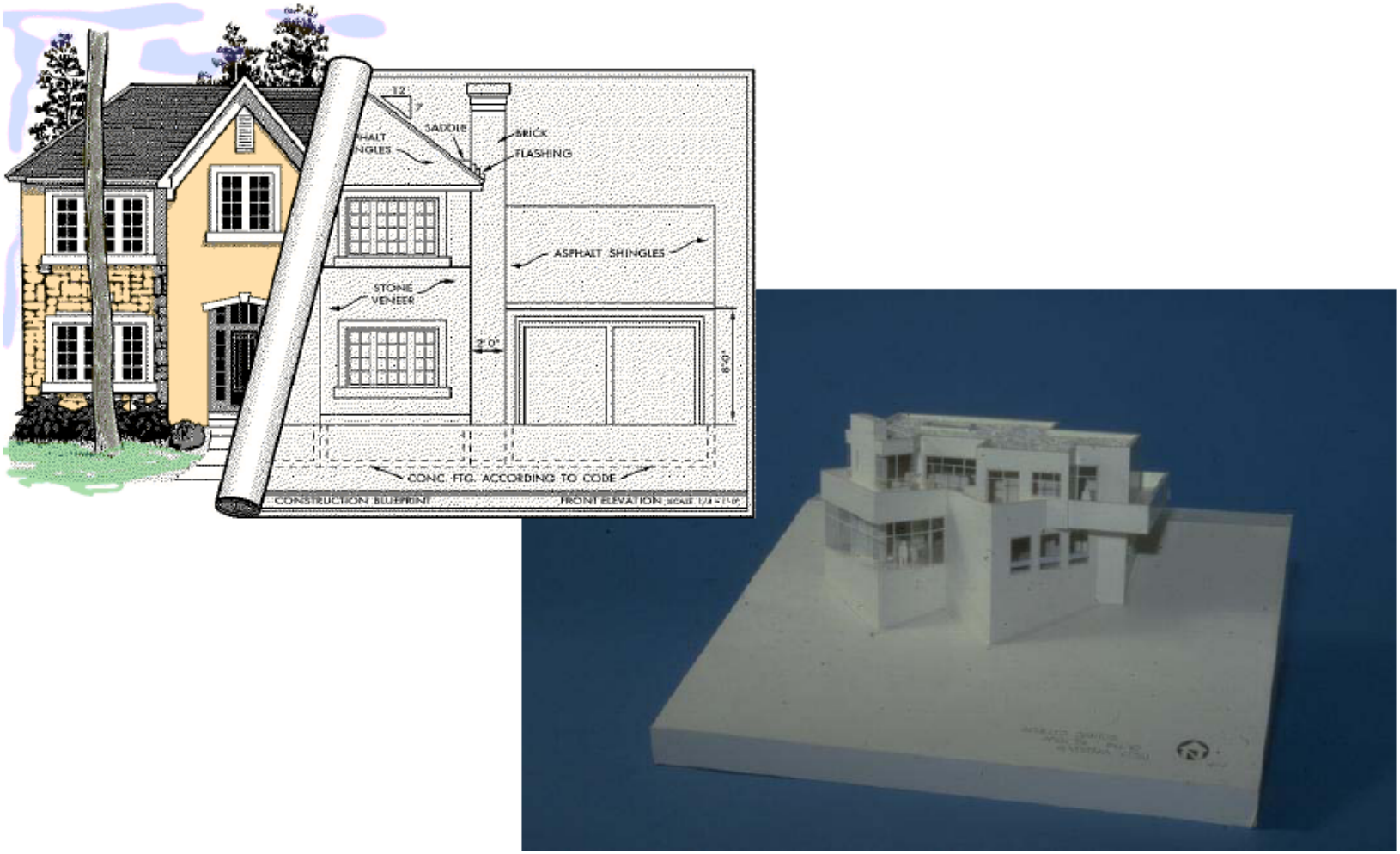


从建筑业谈起

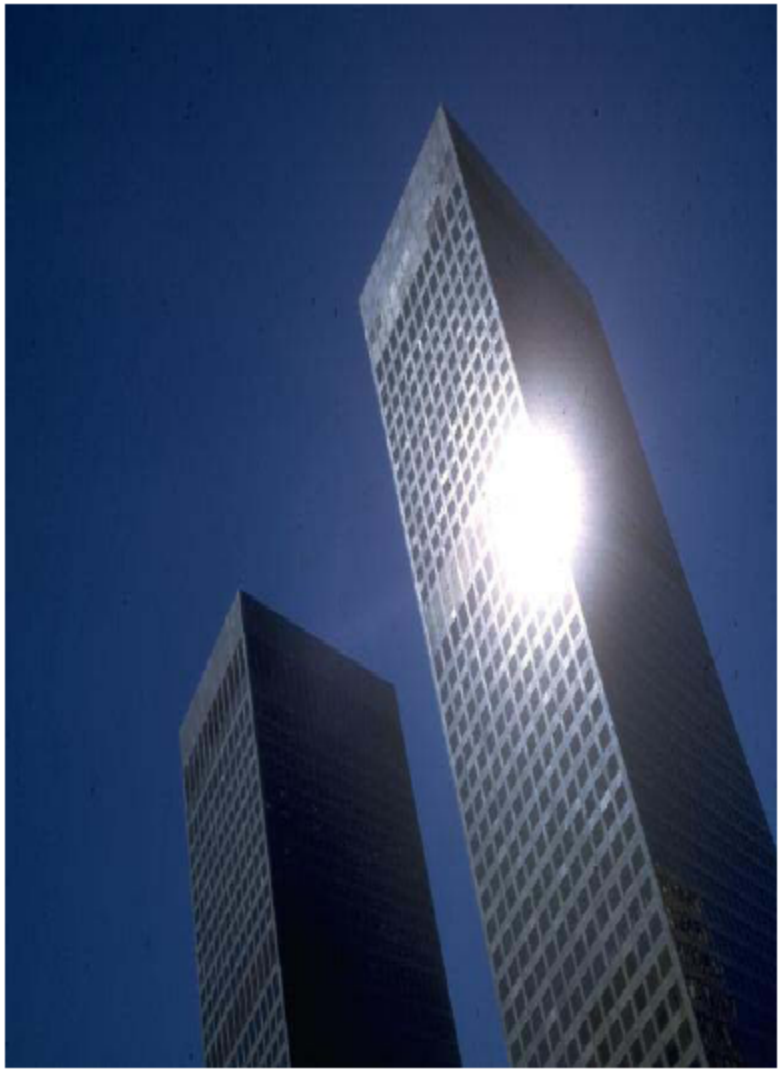
一个人搭建
需要
- 简单建模
- 简单的过程
- 简单工具

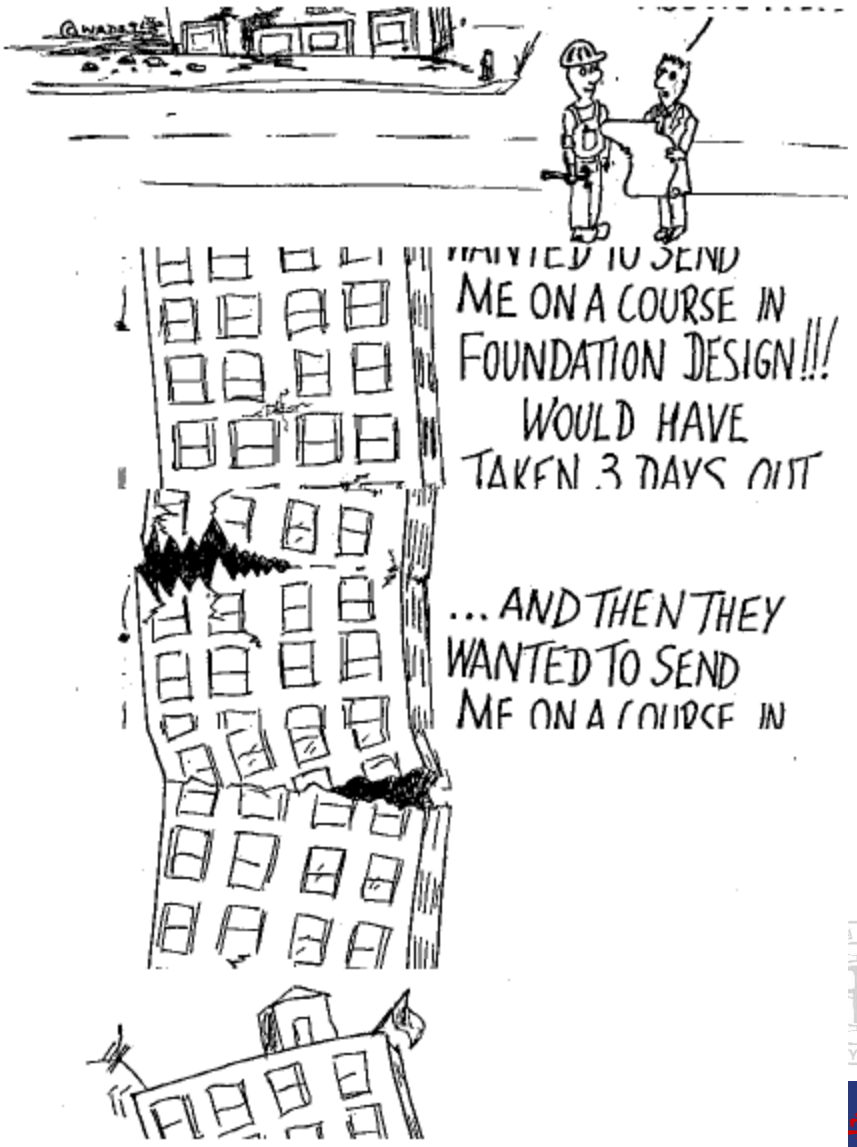
一个团队高效适时地建造，
需要
- 建模
- 良好的过程定义
- 良好的工具

对房子进行建模


12
SADDLE
BRICK
FLASHING
ASPHALT SHINGLES
STONE VENEER
2'-0"
8'-0"
CONC. FTG. ACCORDING TO CODE
CONSTRUCTION BLUEPRINT
FRONT ELEVATION

建模的重要性




ME ON A COURSE IN
FOUNDATION DESIGN!!!
WOULD HAVE
...AND THEN THEY
WANTED TO SEND

建模的重要性

上海环球金融中心大厦（Shanghai World Financial Center，SWFC）总高492米。在100层、距地面472米处设计了长度约为55米的观光天阁（倒梯形风洞）

理解软件体系结构

“软件体系结构”就是关于软件工程的“科学”

- 软件体系结构之所以出现，就是为了弥补软件开发领域在工程上有余而在理论上不足的缺点；
- 借助于计算机科学中其他领域的理论研究方法，试图来用模型分析与理论推理的方法解决软件研发过程中涉及到的各类功能与非功能性问题（尤其是非功能）；
- 将软件工程中总结出来的各类方法论提升为模型与理论；
- 进而用这些理论来指导软件的开发。

可看作是对软件的建模

理解软件体系结构

从字面上理解，

软件体系结构= 软件的 体系结构

Software Architecture (SA) = Software's Architecture (S'A)

学习软件体系结构，应首先弄清楚两个问题：

1. 什么是“软件”(Software)?
2. 什么是“体系结构”(Architecture)?

然后方可回答：

什么是“软件的体系结构”？

Software Engineering Difficulties

- Software engineers deal with unique set of problems
  - Young field with tremendous expectations
  - Building of vastly complex, but **intangible systems**
  - Software is not useful on its own e.g., unlike a car, thus
  - It must **conform to changes** in other engineering areas
- Some problems can be eliminated(屡禁不绝)
  - These are Brooks' "**accidental difficulties**"
- Other problems can be lessened, but not eliminated
  - These are Brooks' "**essential difficulties**"

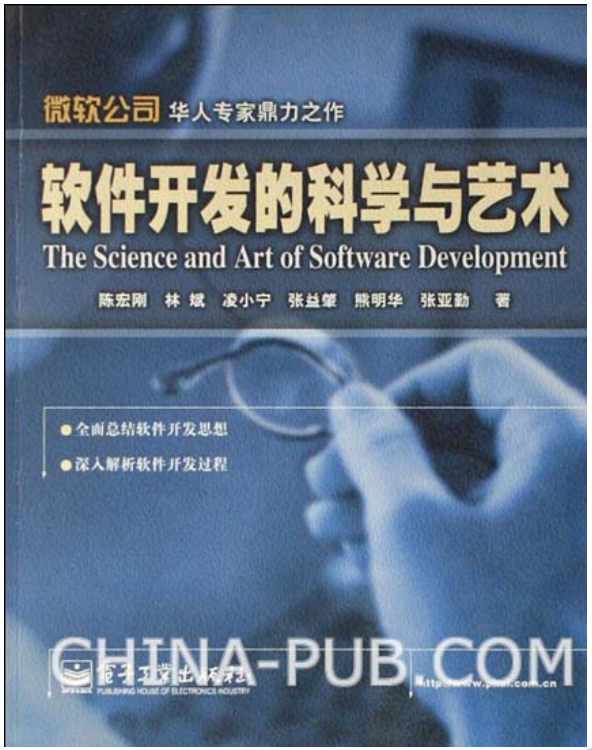
什么是“体系结构”（Architecture）？

- 词典的定义：
  - The art and science of designing and erecting buildings (建筑学：设计和建造建筑物的艺术与科学);
  - A style and method of design and construction (设计及构造的方式和方法);
  - Orderly arrangement of parts; structure (部件的有序安排；结构);
  - The overall design or structure of a computer system, including the hardware and the software required to run it, especially the internal structure of the microprocessor (计算机系统的总体设计或结构，包括其硬件和支持硬件运行的软件，尤其是微处理器内部的结构)。

软件—科学与艺术的结合

- 建筑学：艺术+科学

- 软件：《软件开发的科学与艺术》



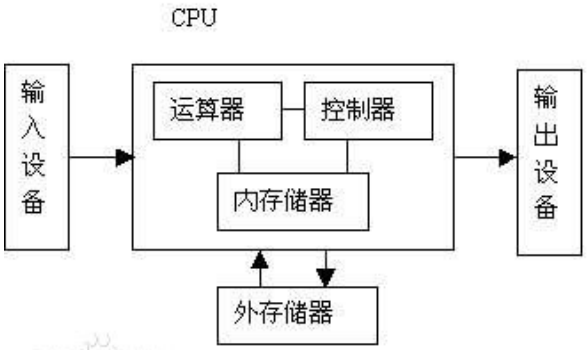
起源于建筑学的“体系结构”

- “体系结构(Architecture)”一词起源于建筑学
  - 如何使用基本的建筑模块构造一座完整的建筑？
- 包含两个因素：
  - 基本的建筑模块：砖、瓦、灰、沙、石、预制梁、柱、屋面板…
  - 建筑模块之间的粘接关系：如何把这些“砖、瓦、灰、沙、石、预制梁、柱、屋面板”有机的组合起来形成整体建筑？

建筑体系结构，从人类建造第一座遮风挡雨的住所开始，就已经成为一门科学

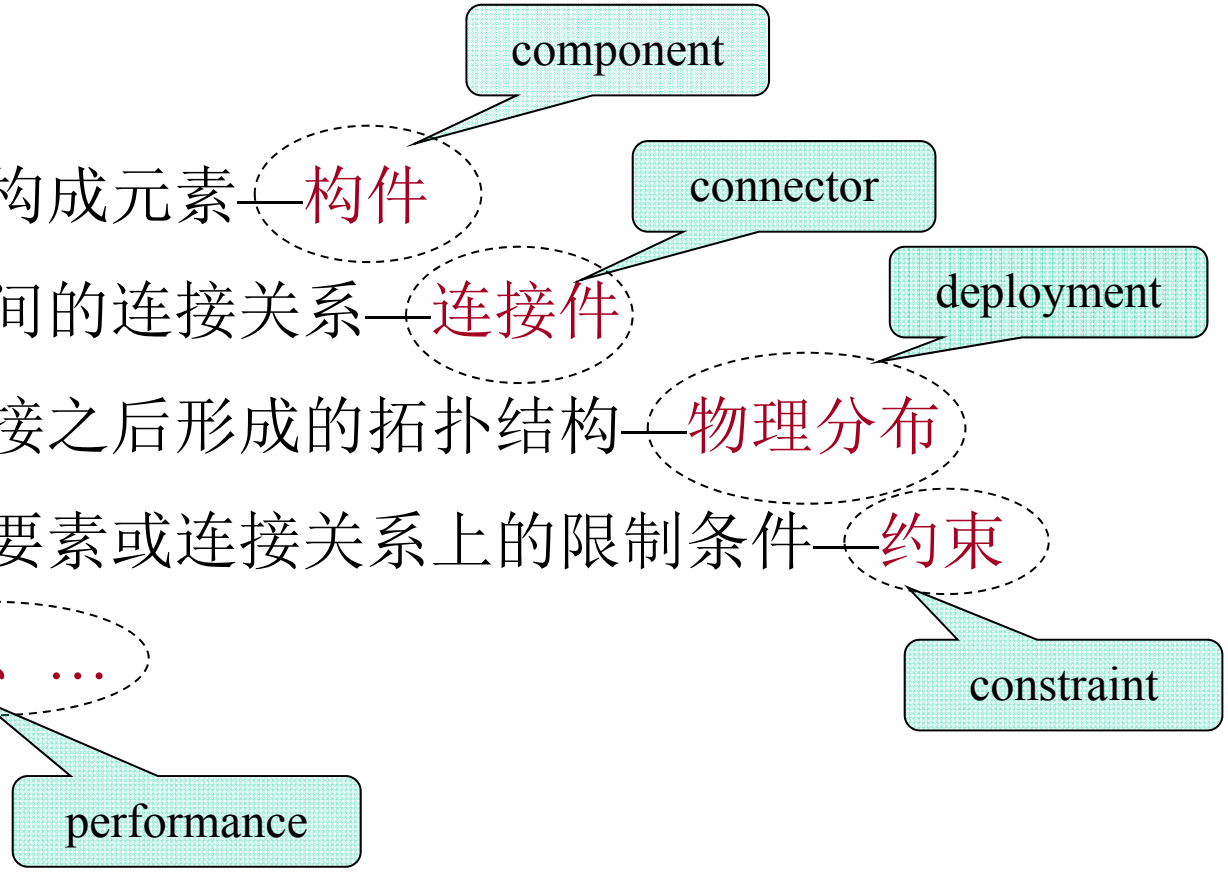
计算机硬件系统的“体系结构”

- 如何将设备组装起来形成完整的计算机硬件系统？
- 包含两个因素：
  - 基本的硬件模块：控制器、运算器、内存储器、外存储器、输入设备、输出设备…
  - 硬件模块之间的连接关系：总线
- 计算机体系结构的风格：
  - 以存储程序原理为基础的冯·诺依曼结构
  - 存储系统的层次结构
  - 并行处理机结构
  - ……



“体系结构”的共性

- 共性：
  - 一组基本的构成元素——构件
  - 这些要素之间的连接关系——连接件
  - 这些要素连接之后形成的拓扑结构——物理分布
  - 作用于这些要素或连接关系上的限制条件——约束
  - 质量——性能、…



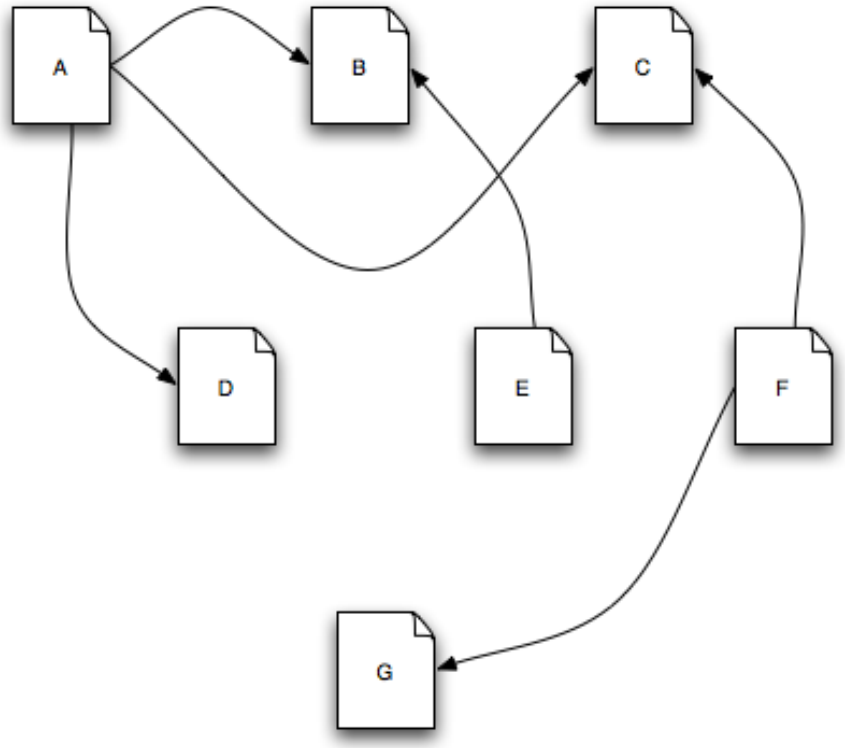
类推“软件体系结构”

- 软件体系结构(Software Architecture, SA)：
  - 构件：各种基本的软件构造模块(函数、对象、模式等);
  - 连接件：将它们组合起来形成完整的软件系统;
  - 物理分布
  - 约束
  - 性能

Architecture in Action: WWW

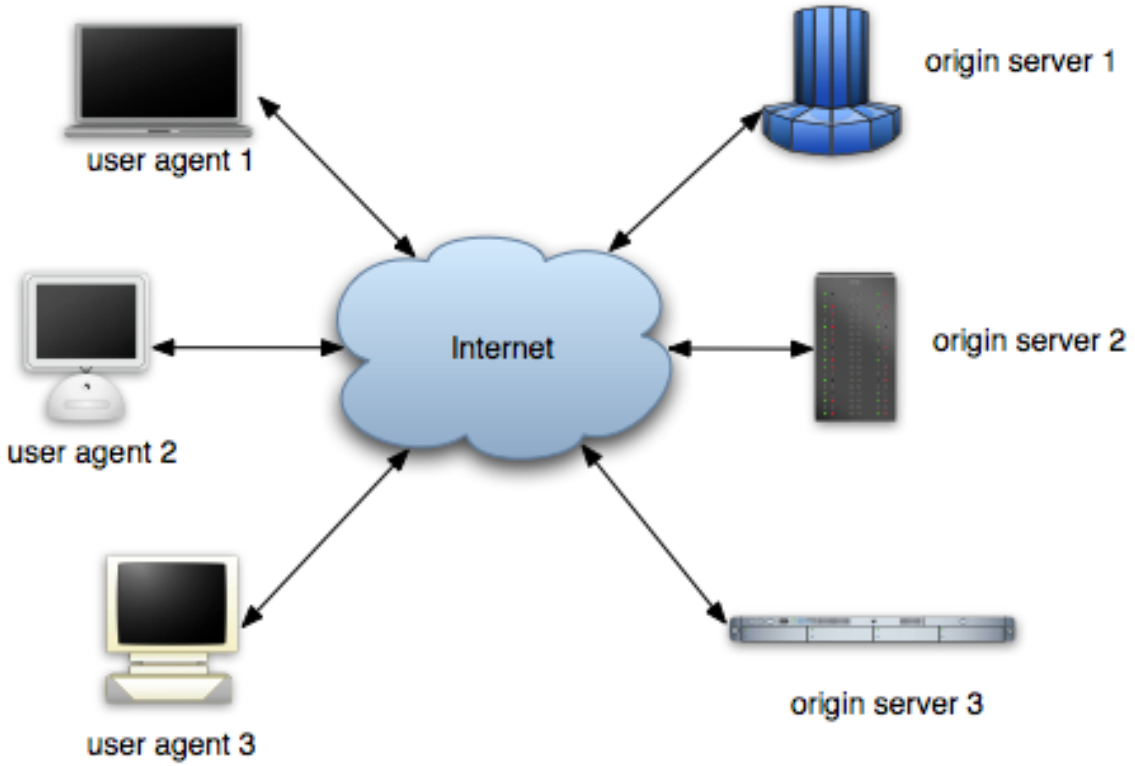
Software Architecture: Foundations, Theory, and Practice; Richard N. Taylor, Nenad Medvidovic, and Eric M. Dashofy;

- This is the Web



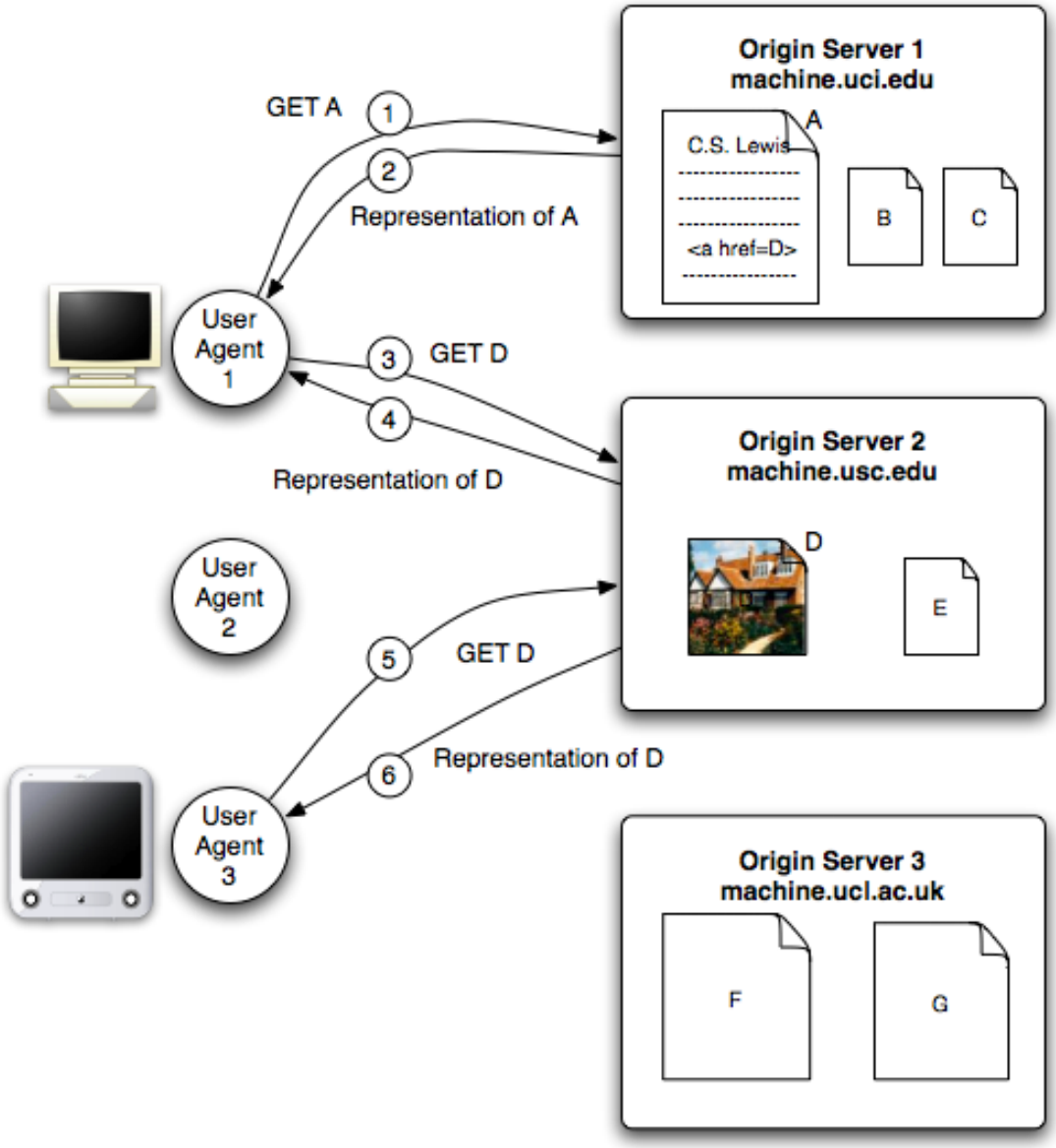
Architecture in Action: WWW

- So is this



Architecture in Action: WWW

- And this



“SA”的定义（1）

1994年(D. Garlan and M. Shaw)

- 软件体系结构是设计过程的一个层次，它处理那些超越算法和数据结构的设计，研究整体结构设计和描述方法。
  - SA={components, connectors, constrains}
    - 构件(component)可以是一组代码，如程序的模块，也可以是一个独立的程序，如数据库的SQL 服务器。
    - 连接件(connector)表示构件之间的相互作用，它可以是过程调用、管道、远程过程调用等。
    - 一个软件体系结构还包括某些限制(constrain)。
- 该模型视角是程序设计语言，构件主要是代码模块。

“SA”的定义（2）

1998年(IEEE 610.12-1990)

- Architecture={component, connector, environment, principle}.
- 体系结构是以构件、构件之间的关系、构件与环境之间的关系为内容的某一系统的基本组织结构，以及指导上述内容设计与演化的原理。

“SA”的定义（3）

SEI

(http://www.sei.cmu.edu/architecture/)

- The software architecture of a program or computing system is a **depiction of the system that aids in the understanding of how the system will behave**.
- Software architecture serves as the **blueprint** for both the system and the project developing it, defining the **work assignments** that must be carried out by design and implementation teams.
- The architecture is the **primary carrier of system qualities** such as performance, modifiability, and security, none of which can be achieved without a unifying architectural vision.
- Architecture is **an artifact** for early analysis to make sure that a design approach will yield an acceptable system.
- By building effective architecture, you can **identify design risks** and mitigate them early in the development process.

归纳：SA定义

- 提供了一个结构、行为和属性的高级抽象
- 从一个较高的层次来考虑组成系统的构件、构件之间的连接，以及由构件与构件交互形成的拓扑结构
- 这些要素应该满足一定的限制，遵循一定的设计规则，能够在一定的环境下进行演化
- 反映系统开发中具有重要影响的设计决策，便于各种人员的交流，反映多种关注，据此开发的系统能完成系统既定的功能和质量需求。

体系结构 = 构件 + 连接件 + 约束

Architecture = Components + Connectors + Constraints

“SA”的相关工作

- 设计：怎样创建一个架构
- 分析：最终产品质量属性与架构之间的关系

第2、3、5章

- 实现：怎样基于架构描述建立一个实际的系统

第3章

- 表达：创建怎样的架构制品来解决人与人、人与机器之间的交流问题

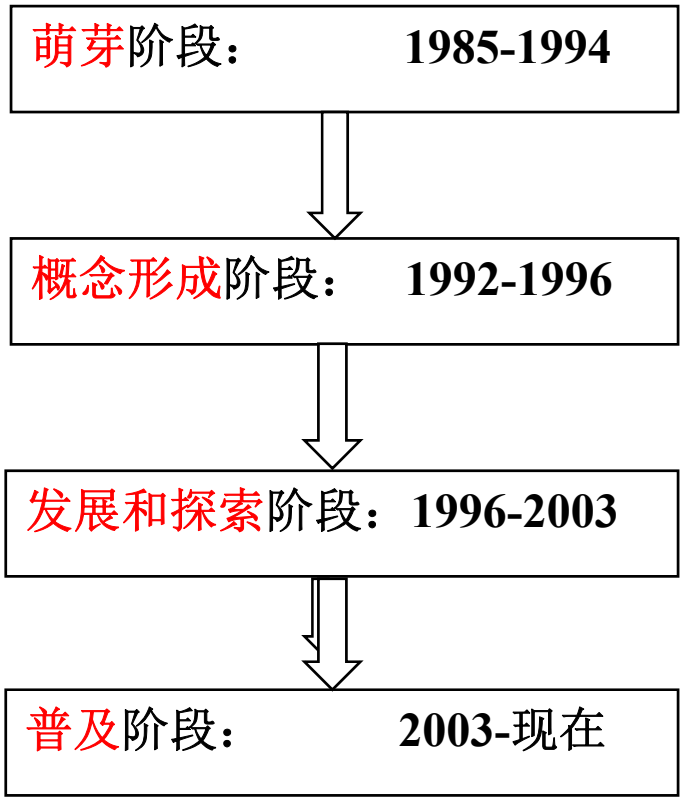
第4、6章

- 经济：架构问题和商业决策怎样关联

第7章

——国际信息处理联盟工作组IFIP WG 2.10（International Federation of Information Processing Working Group 2.10）。

SA的发展阶段



软件危机、软件质量属性的重视
、特定领域软件

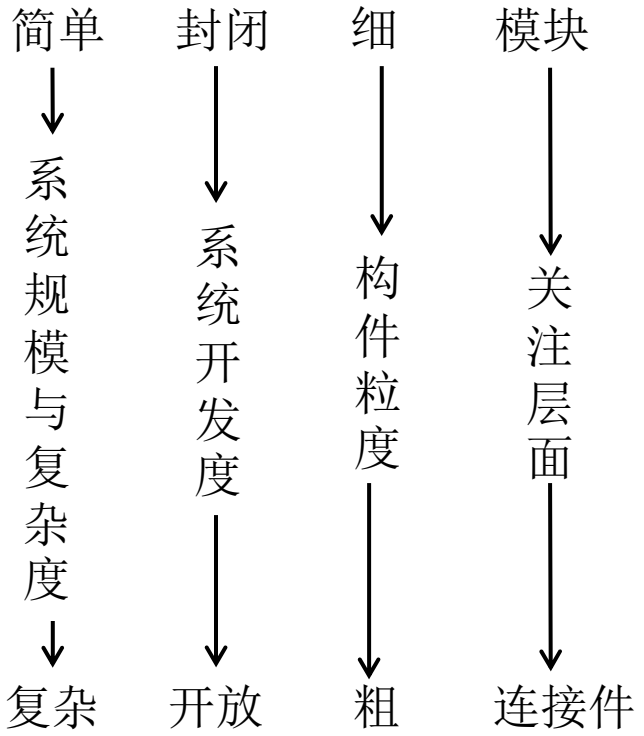
ADL出现、《面向模式的软件体系结构：模式系统》、Kruchten“4+1视图模型”

架构分析和评估方法ATAM、
1997 UML1.1

IBM并购Rational、UML2.0及RSA出现

SA的演化

- 系统＝算法＋数据结构（1960’s）
- 系统＝子程序＋子程序（1970’s）
- 系统＝对象＋对象关联机制（1980’s）
- 系统＝构件＋连接件（1990’s）
- 系统＝服务＋服务总线（2000’s）



SA与软件生命周期

SA在软件生命周其中的位置

项目规划

考虑项目的规模、复杂度、可行性等；

需求分析

利用SA，支持用户、项目负责人、系统架构师、程序员、测试人员之间**进行交流和协商**；

从不同视角审查备选的SA，对得出的意见进行综合，找出合理的平衡方案；

从用户角度考虑未来的需求会发生什么变化，并使SA能够提前支持这些变化；

项目设计

参考经典SA风格，设计系统体系结构模型，推敲其存在的缺陷和替代方案，并进行评估；

进而逐步细化SA，并对定型后的SA作文档化工作；

项目实施

各开发团队**按照SA规定的“构建及其之间的相互关系”进行开发**，保证最终得到的系统与最初的SA一致；

测试与评估

根据SA的约束条件，对软件的质量属性进行测试。

维护与升级

把SA文档作为维护和升级的重要依据

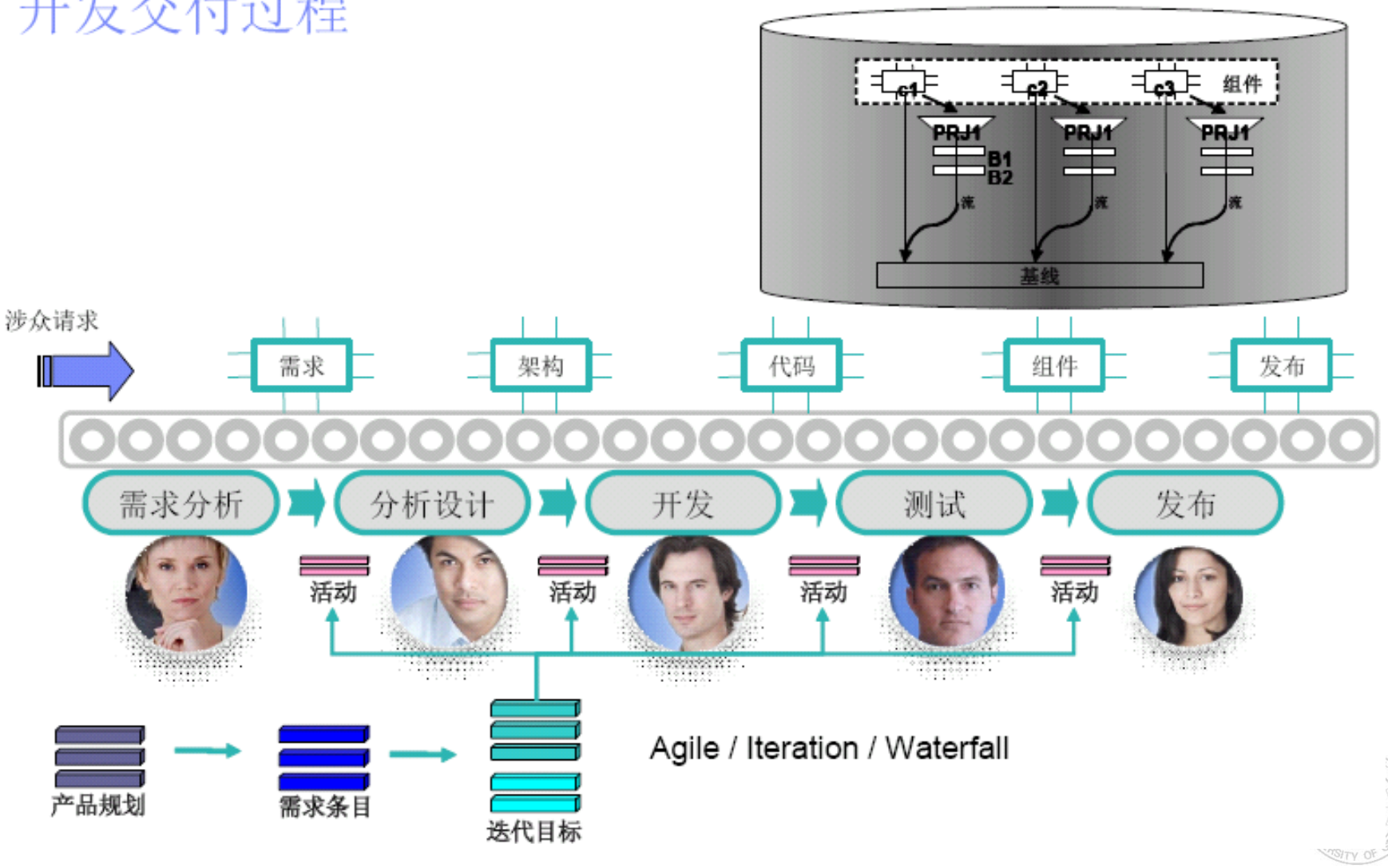
与其他软件设计活动的比较

- 起点模糊
  - 在用户需求尚没有明确定义、用户对其所期望的系统功能、行为和质量尚未确定的情况下就要进行SA设计；——故而需要交流与反复；
- 高抽象层次
  - SA分析处理的是高层次的系统构件或子系统之间的关系，而非变量、函数等低层次的概念；
- 分布决策
  - 来自用户、架构师、测试人员等多方面；
- 贯穿全局
  - SA设计活动在项目开始时进行，但其作用则贯穿整个项目周期，越往后越显出重要性。

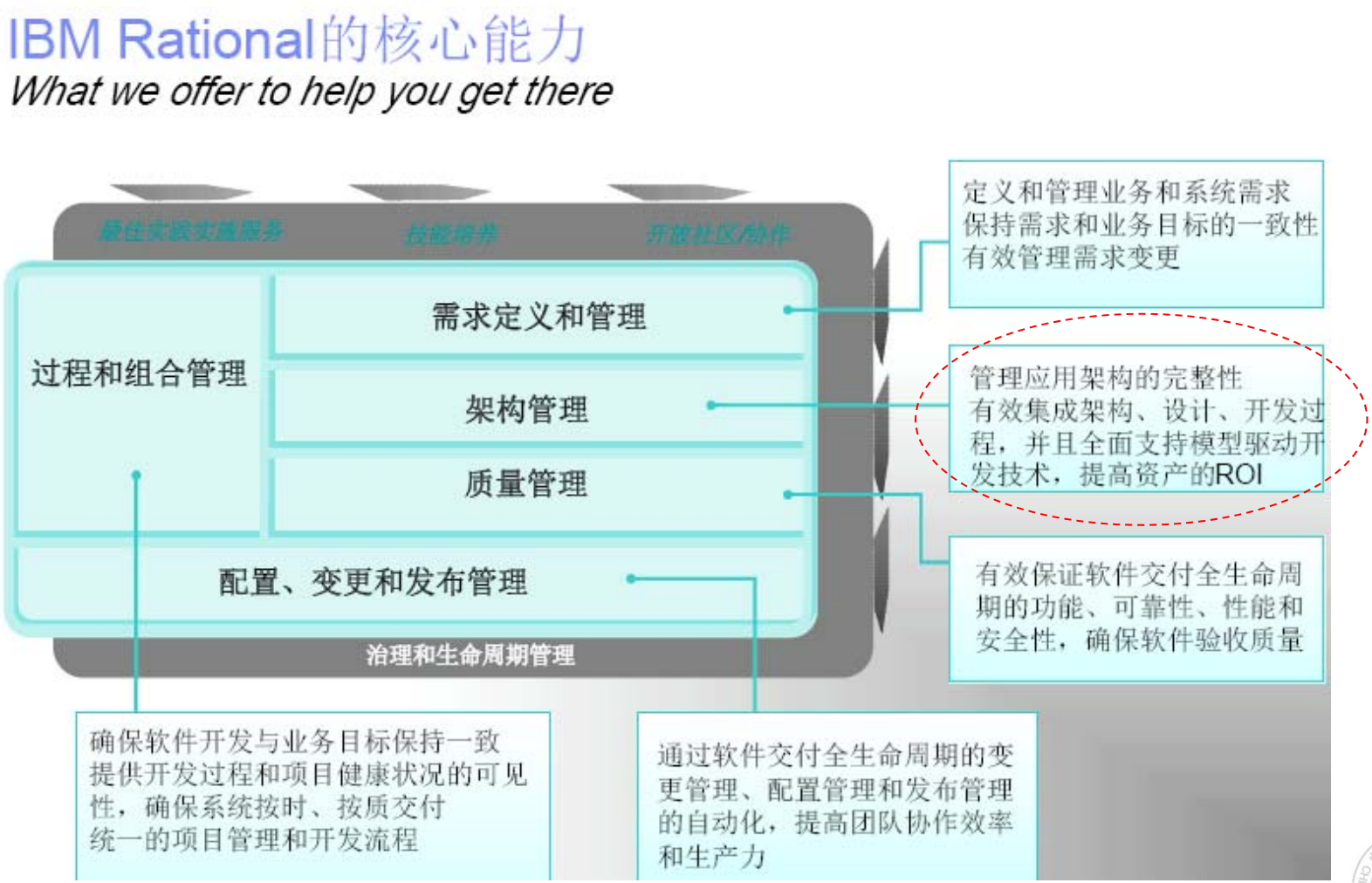
案例一、SA与RUP

IBM Rational软件开发交付过程

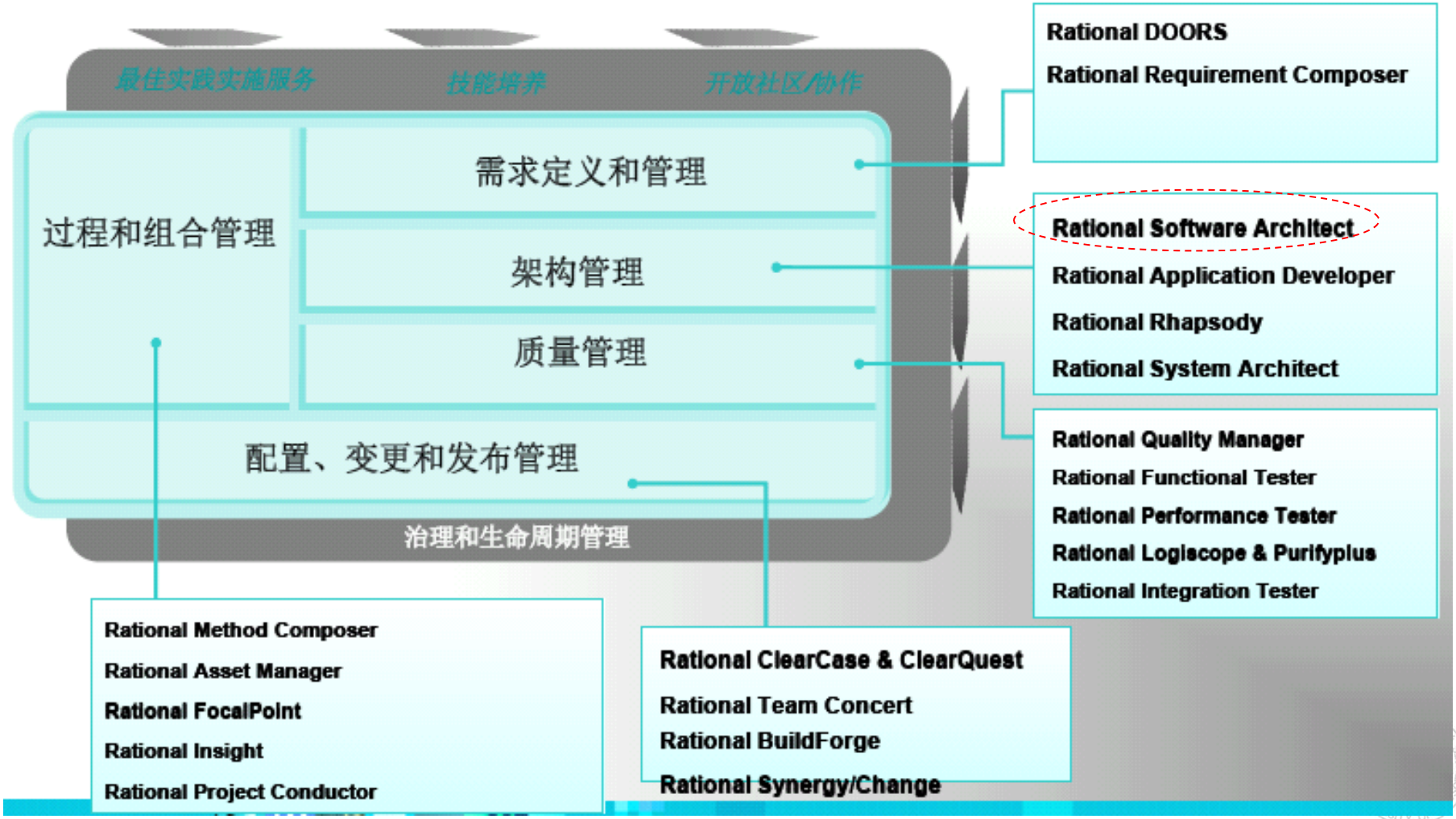
开发交付过程



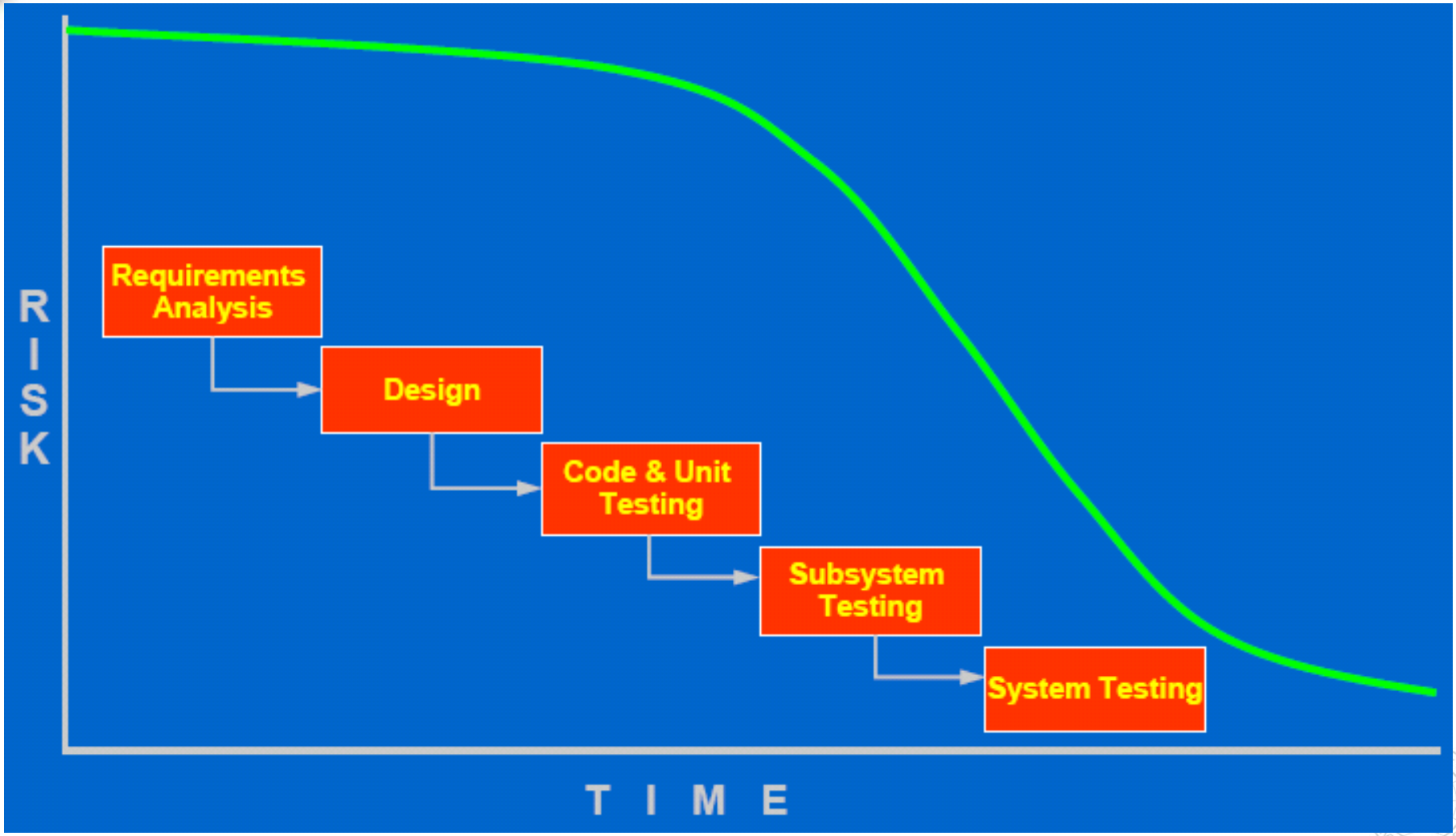
IBM Rational的核心能力



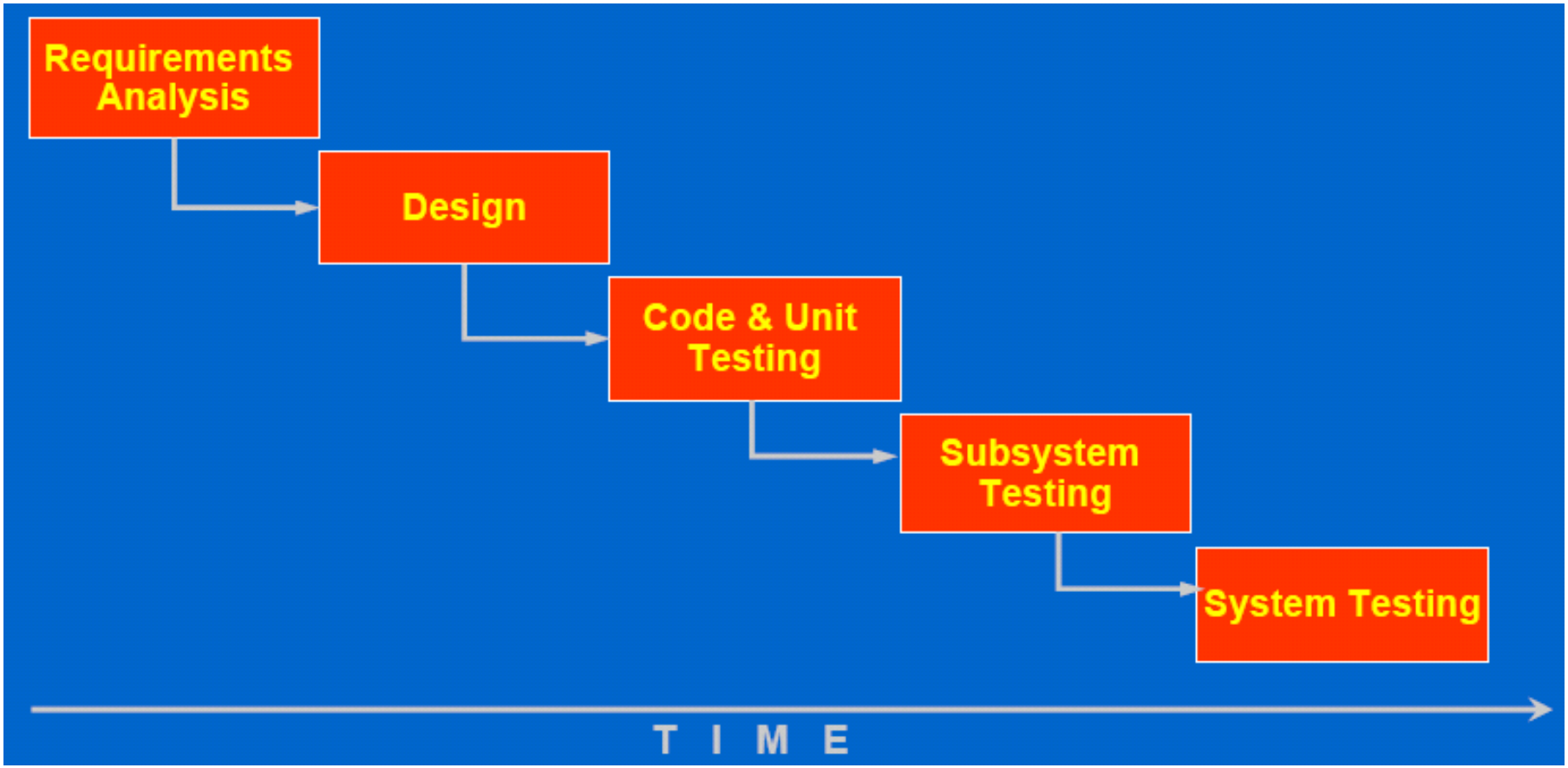
IBM Rational的核心能力
——配套产品



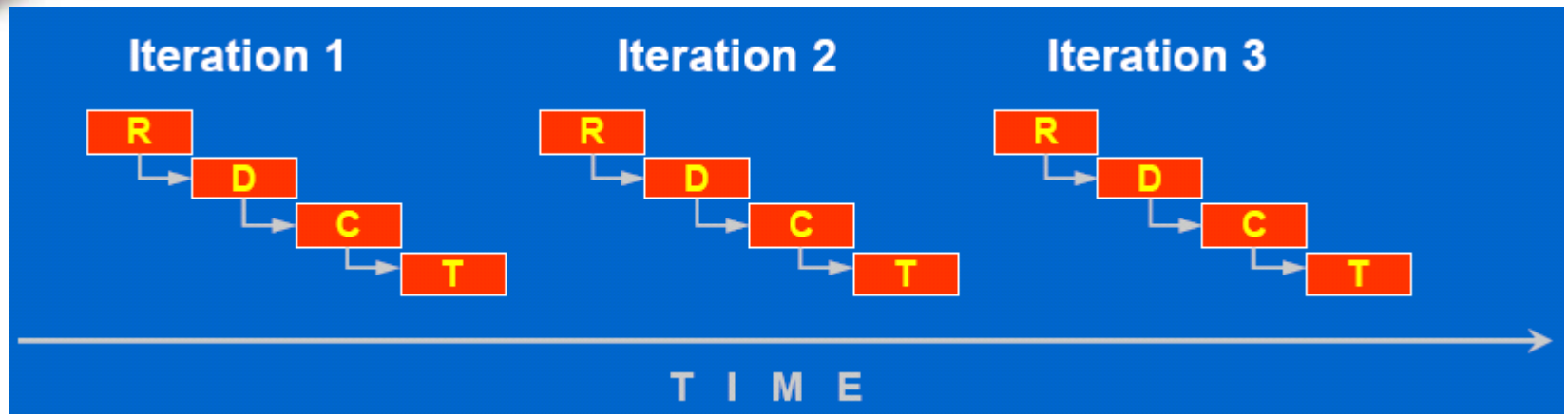
瀑布式开发推迟了风险的规避



传统的瀑布式开发


Requirements Analysis
Design
Code & Unit Testing
Subsystem Testing
System Testing
T I M E

将瀑布开发迭代地应用于系统的增量

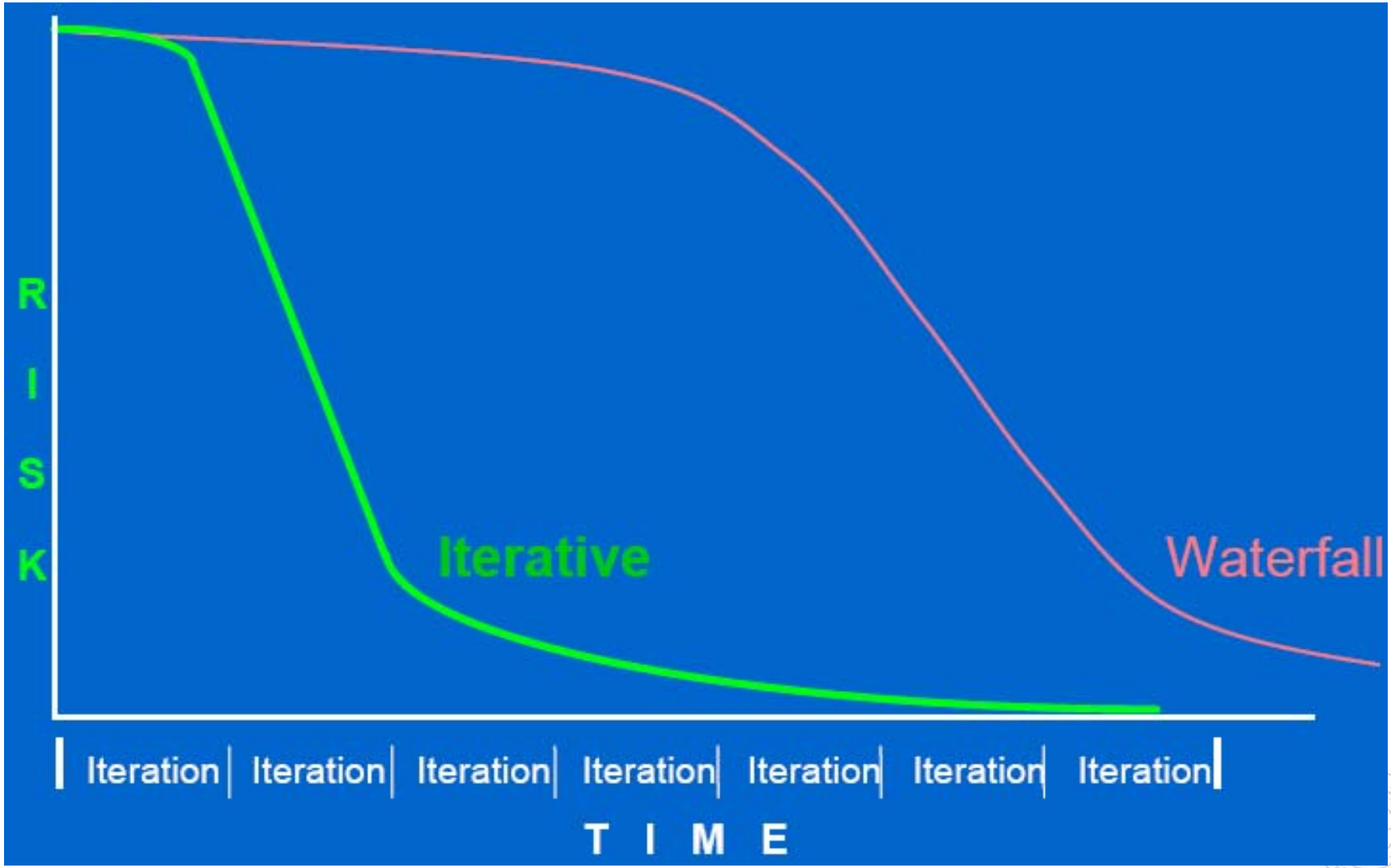


- 最早的迭代触及最重大的风险（例如需求或项目可行性风险）
- 每次迭代产出一个可执行（可以通过测试等较客观的途径加以验证）的交付，是系统的一个增量
- 每次迭代都包含集成与测试

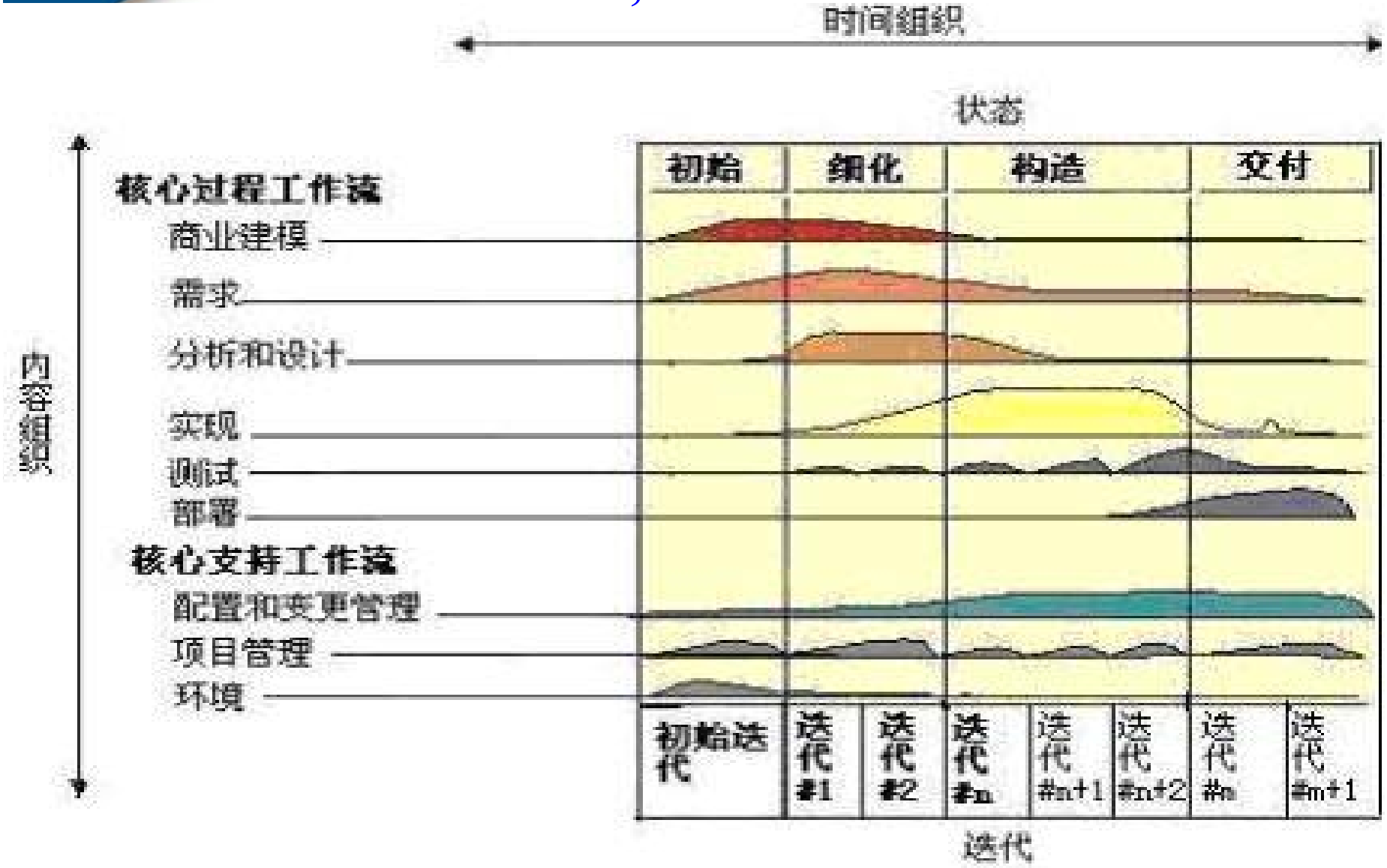
迭代式开发的特点

- 严重的风险在（项目）大规模投入之前被解决
- 初期的迭代能获取更早的用户反馈
- 测试与集成是连续的（增量式）
- 客观（可验证）的里程碑提供了短期的焦点
- 进度的度量直接依靠对实施成果的评估（而非主观的估计）
- 部分的实现可以被先行部署

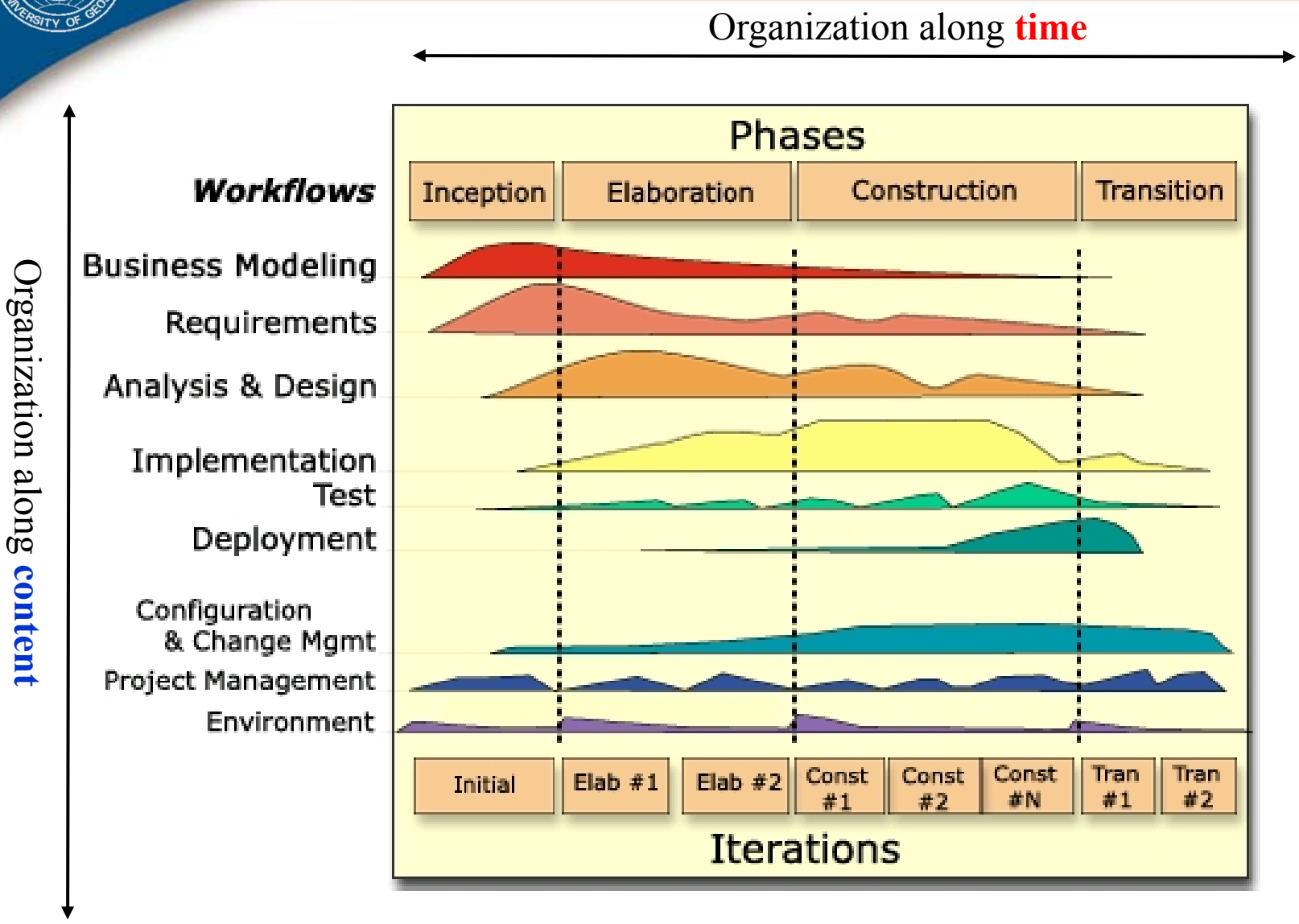
迭代式开发促进风险规避



Rational 统一过程
（RUP, Rational Unified Process）



Rational 统一过程(英)



基于RUP的增量迭代开发模式

◇ **“三个以”：**

以用例驱动、以体系结构为核心、以增量式迭代为开发模式

◇ **“四个阶段”：**

初始、细化、构造、交付

◇ **“九个工作流”：**

商业建模、需求、分析和设计、实现、测试、部署、配置和变更管理、项目管理、环境

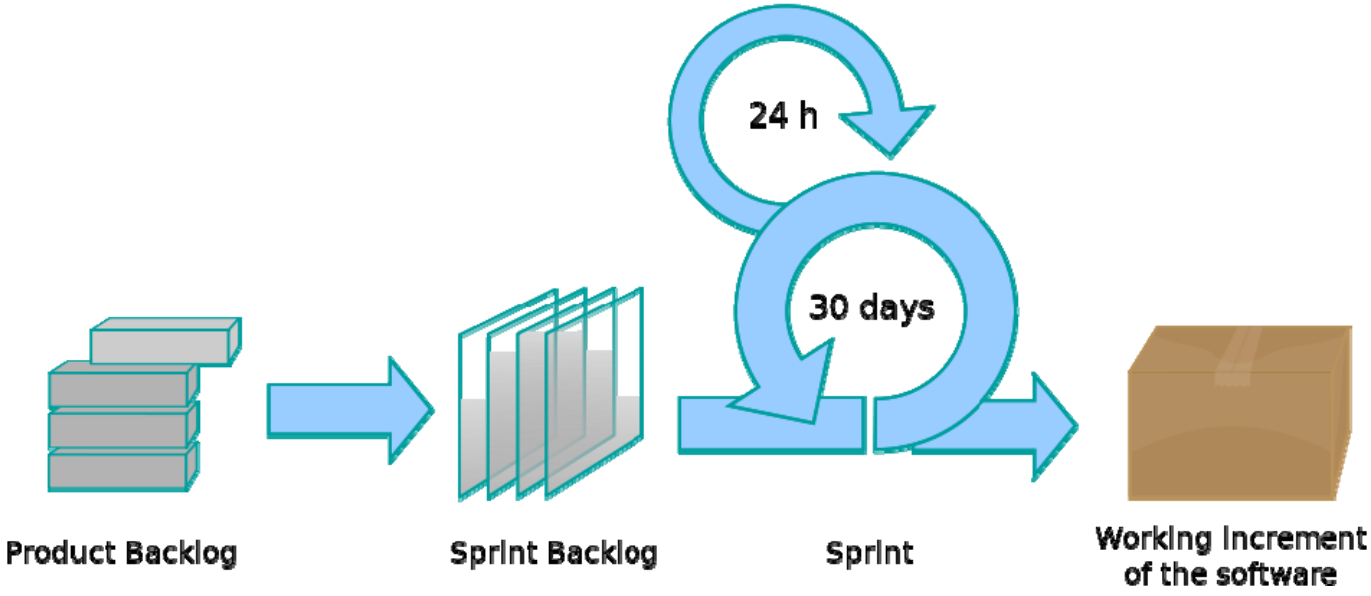
风险驱动的开发模型-RUP

- 在RUP初始阶段的迭代中，项目组必须解决开发目标与范围，以及技术和商业可行性的根本风险---值不值得做，能不能做
- 而到了细化阶段的迭代，项目组关注的焦点则转到构架风险上——可否大量投入去做
- 进入项目中成本最高的构造阶段后，控制成本、进度和开发质量的风险将成为所有成员的责任——准备好交付给用户了？
- 最后到了交付阶段，项目组将面临从客户和用户方面引入的各种风险（日程安排、需求变更等）——客户是否满意

案例二、SA与敏捷开发

敏捷开发Scrum

- 敏捷开发以用户的需求进化为核心，采用迭代、循序渐进的方法进行软件开发。
- 逐渐应用模式。高效的建模者会学习通用的架构模式、设计模式和分析模式，并适当的把它们应用在模型敏捷开发之中。



归纳：SA的作用(1)

- 顺利的话，占据整个设计过程中部分工作量的构架设计活动，能够为其它开发任务（绝大部分工作量）奠定一个坚实基础，使得开发组不需要再进行太多的开创性（高风险）工作——RUP中构架基线里程碑的提出，其理论基础便源于此
- 在项目的较早时期（细化阶段），便解决大部分的难题和风险，是开发组永远的追求目标

但是，只要是缺少整体的结构规划，或是通用问题还没有得到充分的解决，都无法做到让开发组在一定时间过后，就不再需要进行太多的开创性（高风险）工作。

归纳：SA的作用(2)

- 软件构架的静态方面，其着眼点在于——保持目标系统的最终交付在结构上的一致性；为分工协作提供划分依据，并避免结构上的重叠和冗余
- 软件构架的动态方面，其着眼点在于——保持目标系统在关键行为实现上的一致性，突出系统的既有风格；同时通过为各类关键重复问题提供通用解决方案来提高复用度，避免实施代码的冗余

上述两个方面，共同提供了构造目标系统过程中的健壮性与可扩展性——大量的功能实现将在这个构架基础上被不断添加，而同时系统整体上仍然保持既有的一致和完整。