第一章

概述

为什么学习软件工程

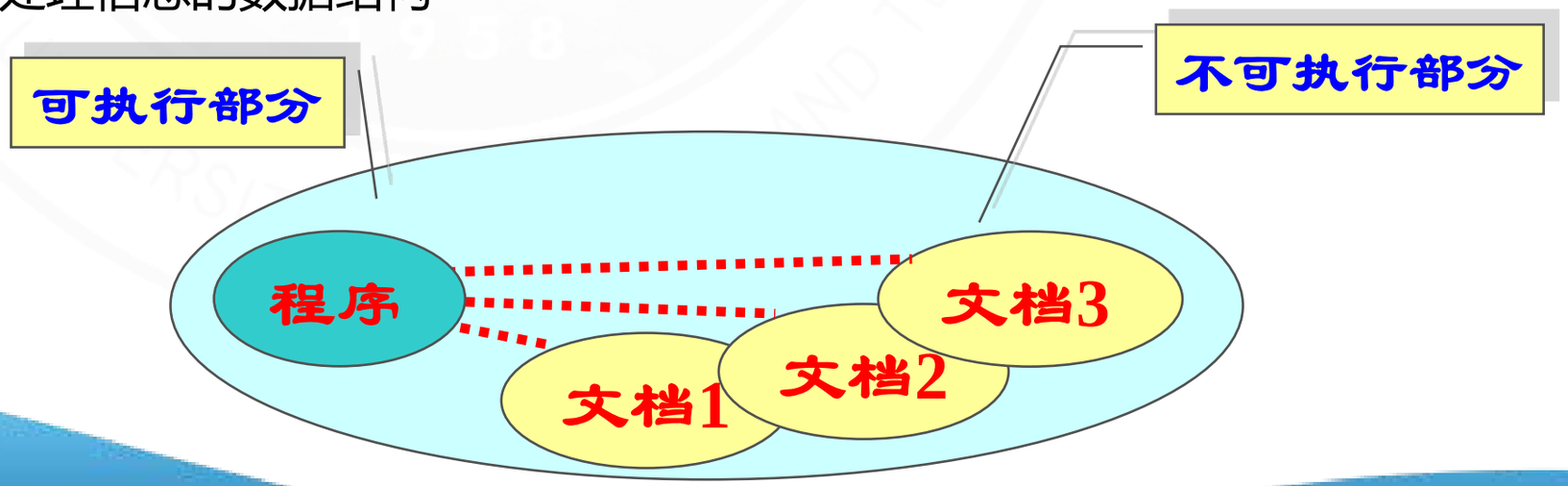
- 课程目标
  - 学习和应用软件工程理论与技术
  - 指导软件工程实践
- 学习意义
  - 用先进理念(理论)
  - 和体现先进理念的软件产品规划与管理技术(手段)
  - 提高软件产品的质量及个人的职业竞争力(目标)

如何学习软件工程

- 理解软件工程的原理
  - 理念、思想、方法与技术
- 熟悉计算机辅助软件工程技术
  - 熟悉工作机理或运行模式
  - 熟悉软件配置系统与配置过程
  - 熟悉软件工作流程与企业业务流程
- 运用软件工程理论开发软件系统
  - 软件设计/抽象➥软件编码➥软件部署➥软件测试➥软件服务
  - 软件建模、软件工程、协同软件工程与大型软件工程
- 运用软件工程理论实施软件系统
  - 需求分析与业务建模➥数据规范化与标准化的实施➥流程优化的实施➥软件配置与部署➥软件培训
- 形成能力的渐进增长
  - 由基本原理到系统抽象
  - 由系统抽象到系统设计
  - 由系统设计到系统实现：程序编码与测试
  - 由一个应用对象到多个应用对象：扩展与深化

什么是软件

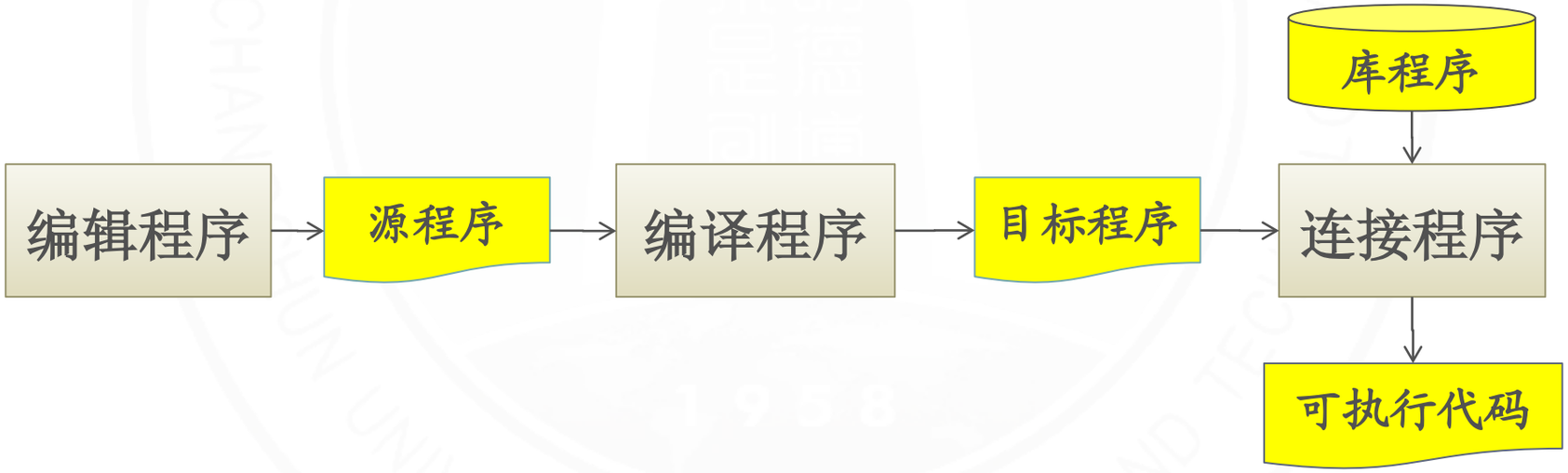
- 计算机软件
  - 与计算机系统操作有关的程序、规程、规则及任何与之有关的文档和数据
- 软件 = 程序 + 数据 + 文档
  - 程序(program)
    - 是计算机为完成特定任务而执行的指令集合
      - 即，用程序设计语言描述的，适合于计算机处理的语句序列
  - 文档(document)
    - 一种数据媒体和其上所记录的数据
    - 文档记录软件开发活动和阶段成果
      - 具有永久性，可供人或机器阅读
    - 文档可用于
      - 专业人员和用户之间的通信和交流、软件开发过程的管理、运行阶段的维护
- 数据(data)
  - 使程序能够适当地处理信息的数据结构



软件=程序+文档

什么是程序

- 程序是为了解决某个特定问题而用程序设计语言描述的适合计算机处理的语句序列。
- 程序需要经过编辑、编译、链接等过程才能成为在计算机上执行的机器语言序列。程序执行一般需要一定的输入数据，同时也会输出运行结果。



什么是文档

- 文档：软件开发活动的记录，主要供人们阅读，既可用于专业人员和用户之间的通信和交流，也可以用于软件开发过程的管理和运行阶段的维护。
- 文档类型：需求分析文档、软件设计文档、软件测试文档等。
- 编写文档的目的
  - 促进对软件的开发、管理和维护；
  - 便于各种人员(用户、开发人员)的交流。

软件的特征

- 软件开发不同于硬件设计
  - 软件开发更依赖人员素质、智力、组织、合作和管理
  - 开发成本、进度很难估算
- 软件生产不同于硬件制造
  - 软件生产只需复制
- 软件维护不同于硬件维修
  - 不会老化
  - 维护困难和复杂

传统的软件类型

- 系统软件：
  - 计算机系统软件是计算机管理自身资源(如CPU、内存、外存、外设等)，提高计算机使用效率并为计算机用户提供各种服务的基础软件。例如，操作系统、数据库管理系统等。
- 实时软件：
  - 监测、分析和控制现实世界发生的事件，能以足够快的速度对输入信息进行处理，并在规定的时间内作出反应的软件。例如，各种设备运行监控软件等。
- 嵌入式软件：
  - 嵌入式计算机系统将计算机嵌入在某一系统之中，使之成为该系统的重要组成部分，控制该系统的运行，进而实现某一特定的物理过程。用于嵌入计算机系统的软件称为嵌入式软件。例如，航空航天系统、指挥系统、汽车控制系统等。
- 科学和工程计算机软件：
  - 它们以数值算法为基础，对数值量进行处理和计算，主要用于科学和工程计算。例如，数值天气预报、导弹计算、石油勘探、计算辅助设计(CAD)等。
- 事务处理软件：
  - 用于处理事务信息，特别是商务信息的计算机软件。事务信息处理是软件最大的应用领域。例如，企业资源计划系统(ERP)、物流管理系统等。

主流的软件类型

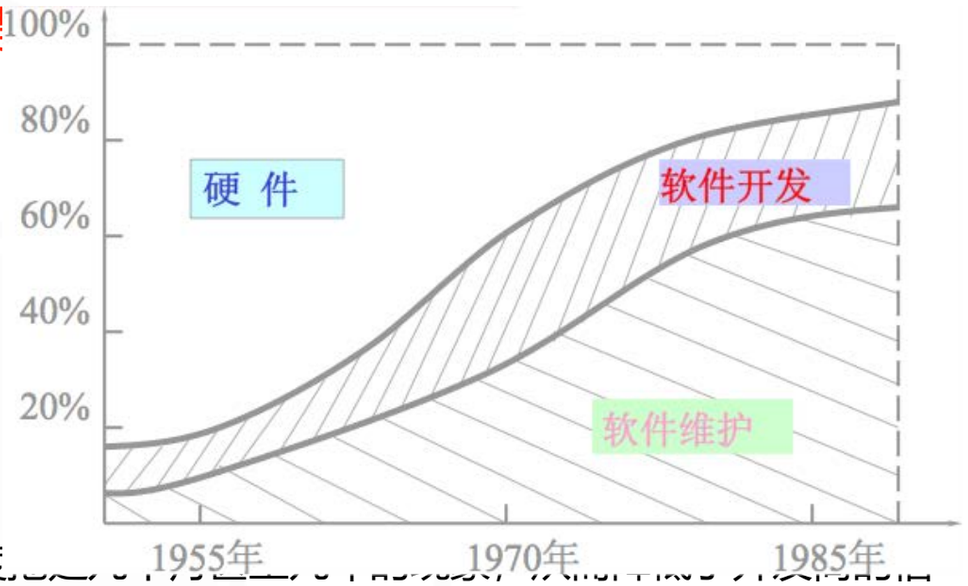
- 人工智能软件：
  - 支持计算机系统产生人类某些智能的软件。它们求解复杂问题不是用传统的计算或分析方法，而是采用诸如基于规则的演绎推理技术和算法。应用领域有专家系统、模式识别、自然语言理解、人工神经网络、程序验证、自动程序设计、机器人学等。
- CASE工具软件：
  - CASE工具软件一般为支撑软件生存周期中不同活动而研制，包括项目管理工具、需求分析工具、编程环境(编辑器、编译器、链接器和测试器于一体)、软件测试工具等。
- 其它软件

软件 / 软件系统的发展历程

- 20世纪60年代中期以前
  - 软件 ➥ 规模较小的程序
    - 编写容易
  - 大多数人认为软件开发是无需预先计划的事情，只是在人们头脑中隐含进行的一个模糊过程
  - 除了程序清单，根本没有其他文档资料保存
- 60年代中期～70年代中期
  - 出现了“软件作坊”
    - 沿用早期形成的个体化软件开发方法
    - 产品软件被广泛使用
      - 维护工作的耗费严重
  - “软件危机”成为业界共识
    - 1968年，软件工程学科诞生
- 70年代中期～80年代中后期
  - 出现了微处理器并获得了广泛应用，个人计算机已经成为大众化商品
  - 软件开发进入产业化生产，出现了众多大型的“软件公司”
  - 软件开发开始采用“工程”的方法，软件产品急剧增加，质量大有提高
- 80年代末期～
  - 硬件和软件耦合增加
  - 由复杂操作系统控制各类终端（桌面、移动终端）组成局域网和广域网
  - 软件开发开始采用面向对象的技术和可视化的集成开发环境

软件危机

- 在计算机软件的开发和维护过程中所遇到的一系列严重问题
  - 几乎所有软件都不同程度地存在问题
    - 问题不仅仅是软件“不能正常运行”
  - 危机涉及的主要方面，即核心问题
    - 如何开发软件，以满足对软件日益增长的需求
    - 如何维护软件，以保证数量不断膨胀的已有软件的正常运行



- 具体表现
  - 对软件开发成本和进度的估计常常很不准确
    - 常常出现实际成本比估算成本高出一个数量级、实际进度比计划进度[illegible]，引起用户不满
    - 可行性分析以及开发进度表没有做好
  - 用户对已完成的软件不满意的现象时有发生
    - 需求分析没有做好
  - 软件产品的质量往往是靠不住的
    - 分析设计没有做好，软件测试工作没有做好；程序调试、测试只能证明程序有错，不能证明无错
  - 软件常常是不可维护的
    - 没有用科学的软件分析、设计方法开发软件；软件通常没有适当的文档资料
  - 软件通常没有适当的文档资料
    - 文档资料不全或不合格，给软件开发和维护工作带来许多难以想象的困难和难以解决的问题
  - 软件成本在计算机系统总成本中所占比例逐年上升
    - 特别是软件维护成本迅速增加，已经占据软硬件总成本的40%~75%
  - 开发生产率提高的速度远跟不上软件需求

软件危机的表现

- 软件成本高
  - IBM 360 OS, 5000多人年，耗时4年(1963-1966)，花费2亿多美元。
    - 共约100万条指令，花费了5000个人年；经费达数亿美圆，但错误多达2000个以上，系统根本无法正常运行
    - OS/360系统负责人Brooks对开发过程的困难和混乱描述为：“...像 巨兽在泥潭中作垂死挣扎，挣扎得越猛，泥浆就沾得越多，最后没有一个野兽能够逃脱淹没在泥潭中的命运...”
  - 美国空军：1955年软件占总费用(计算机系统)的18%，70年60%，85年达到85％。
  - 美国全球军事指挥控制系统，硬件1亿美元，软件高达7.2亿美元。
- 软件质量得不到保证
  - 软件应用面的扩大：科学计算、军事、航空航天、工业控制、企业管理、办公、家庭。
  - 软件越来越多的应用于安全犹关(safety critical)的系统，对软件质量提出更高的要求。
  - 1963年美国飞往火星的火箭爆炸，造成1000万美元的损失
    - 原因是FORTRAN程序：
    - DO 5 I=1,3
    - 误写为:DO 5 I=1.3
  - 英国1986年开发的办公室信息系统Folios经4年，因性能达不到要求，1989年取消。
  - 日本第5代机因为软件问题在投入50亿美元后于1993年下马。
  - 1967年苏联“联盟一号”载人宇宙飞船在返航时，由于软件忽略一个小数点，在进入大气层时因打不开降落伞而烧毁

软件危机的表现

- 软件进度难以控制
  - 项目延期比比皆是
  - 由于进度问题而取消的软件项目较常见
  - 只有一小部分的项目能够按期完成

- 软件维护非常困难
  - 软件维护的多样性
  - 软件维护的复杂性
  - 软件维护的副作用。

软件危机的产生原因

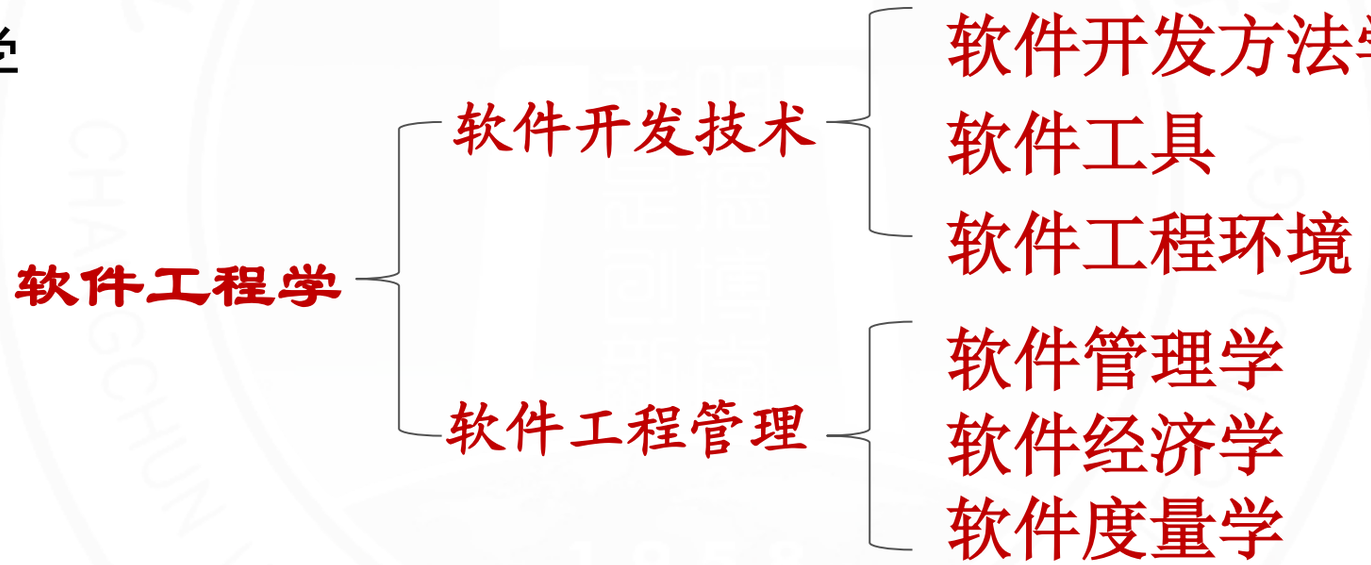
- 用户对软件需求的描述不精确
  - 可能有遗漏、二义性，甚至在软件开发过程中用户还提出修改软件功能、界面、支撑环境等方面的要求
- 过早、过于乐观、过于匆忙地开始程序编写
  - 轻视需求分析、可行性分析的作用
- 软件开发人员对用户需求的理解有偏差
  - 导致软件产品与用户的需求不一致
- 缺乏处理大型软件项目的经验
  - 开发大型软件项目需要组织众多人员共同完成
    - 一般来说，多数管理人员缺乏大型软件的开发经验，而多数软件开发人员又缺乏大型软件项目的管理经验，致使各类人员的信息交流不及时、不准确、容易产生误解
- 开发大型软件易产生疏漏和错误
- 缺乏有力的方法学的指导和有效的开发工具的支持
  - 软件开发过多地依靠程序员的“技巧”，加剧了软件产品的个性化
  - 重开发、轻测试、轻维护
    - 测试应占软件开发全部工作量的40%-50%，维护费用应占全部费用的55%-70%，编程只占软件开发全部工作量的10%-20%
- 日益增长的软件需求
  - 供求矛盾将是一个永恒的主题

缓解软件危机的途径

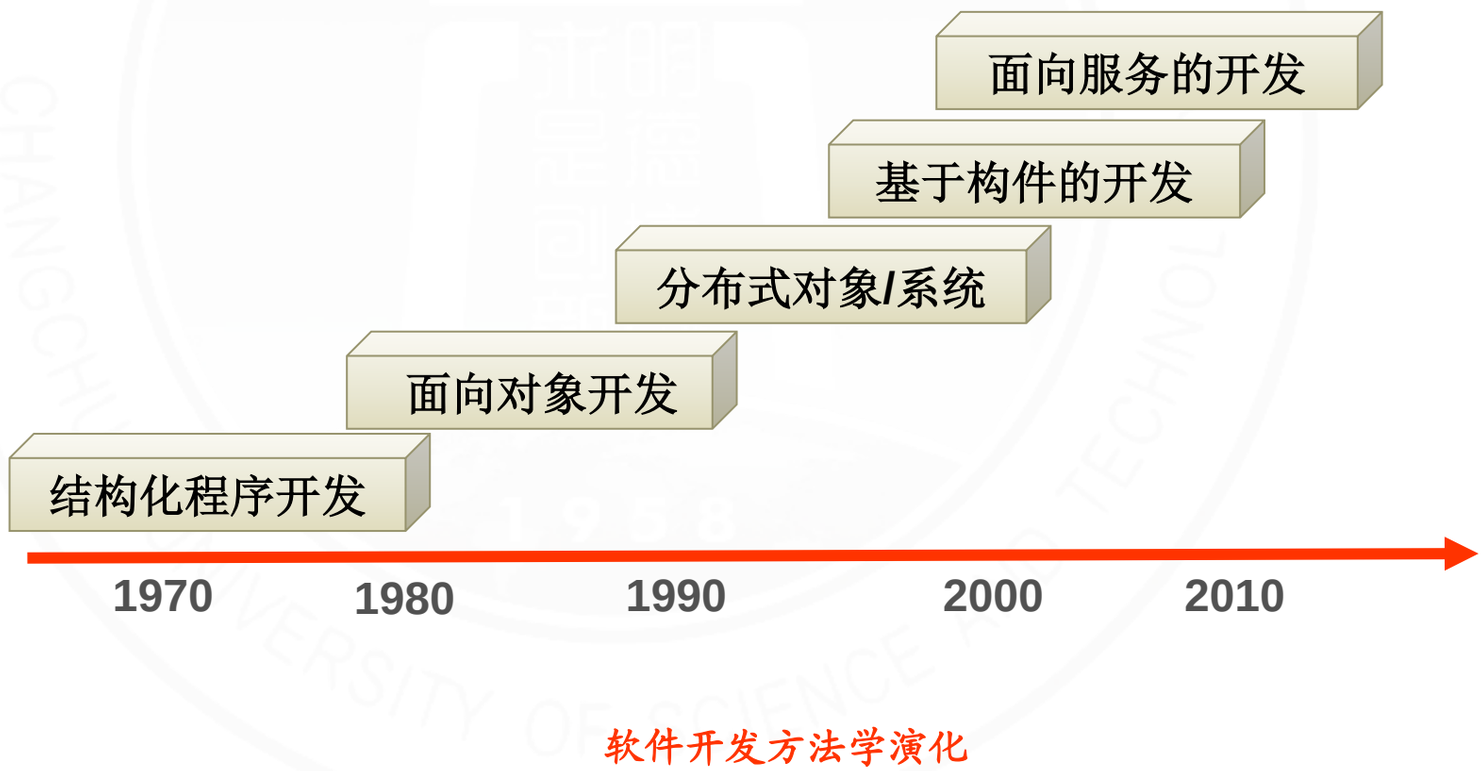
- 用现代工程的原理、技术和方法进行软件的开发、管理、维护和更新
  - 充分认识到软件开发不是某种个体劳动的神秘技巧，而应该是一种组织良好、管理严密、各类人员协同配合、共同完成的工程项目
  - 应该推广使用在实践中总结出来的开发软件的成功的技术和方法，并且研究探索更好更有效的技术和方法，尽快消除在计算机系统早期发展阶段形成的一些错误概念和做法
  - 应该开发和使用更好的软件工具
- 1968年，北大西洋公约组织在原西德召开计算机科学会议，首次提出了“软件工程”的概念
  - 用工程、科学和数学的原则与方法开发、维护计算机软件的有关技术和管理方法
- IEEE定义软件工程为
  - (1)The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software, that is, the application of engineering to software. (2)The study of the approaches as in (1)

软件工程学范畴

- “软件工程”首次提出： 1968年，北大西洋公约组织在原西德召开计算机科学会议，首次提出了“软件工程”的概念。
- 提出背景：解决软件危机。
- 软件工程学



软件开发方法学



软件开发方法学演化

软件工程管理

- 概念：
  - 软件工程管理的目的是为了按照进度及预算完成软件计划，实现预期的经济和社会效益。

- 软件工程管理包括：成本估算、进度安排、人员组织、质量保证等内容，另外还涉及到管理学、度量学和经济学等多个学科的知识。

软件工程三要素

- 由方法、工具和过程三部分组成
  - 软件工程中的各种方法是完成软件工程项目的技术手段，它们支持软件工程的各个阶段
    - 回答“如何做”的问题
  - 软件工程使用的软件工具能够自动或半自动地支持软件的开发、管理和文档的生成
  - 软件工程中的过程贯穿于整个工程的各个环节
    - 过程中，管理人员应对软件开发的质量、进度、成本等进行评估、管理和控制
      - 包括计划跟踪与控制、成本估算、人员的组织、质量保证、配置管理等

软件工程的基本原理

- 著名的软件工程专家B. W. Boehm于1983年综合了软件工程专家学者们的意见并总结了开发软件的经验，提出了软件工程的7条基本原理
  - 用分阶段的生存周期计划严格管理软件开发;
  - 坚持进行阶段评审;
  - 实行严格的产品版本控制;
  - 采用现代程序设计技术;
  - 结果应能清楚地审查;
  - 开发小组的人员应少而精;
  - 要不断改进软件工程实践的经验和技术，要与时俱进。
- 上述7条原理被认为是确保软件产品质量和开发效率的原理的最小集合，又是相互独立、 缺一不可、相当完备的最小集合
- 经验
  - 一般人认为已经完成了80%, 但实际上只完成了20%;
  - 对程序中存在问题的估计80%的问题存在于20%的程序中;
  - 对程序模块的估计20%的模块实现了80%的功能;
  - 对人力资源的估计20%的人解决了软件中80%的问题;
  - 对资金的估计信息系统中80%的问题可以用20%的资金解决;
  - etc.

软件工程的基本原理

- 用分阶段的生存周期计划严格管理软件开发
  - 应该把软件生存周期划分成若干个阶段，并相应地制定出切实可行的计划，然后严格按照计划对软件开发与维护工作进行管理
  - 应该制定的计划有项目概要计划、里程碑计划、项目控制计划、产品控制计划、验证计划和运行维护计划等
  - 各级管理人员都必须严格按照计划对软件开发和维护工作进行管理
  - 据统计，不成功的软件项目，有一半左右是由于计划不周造成的
- 坚持进行阶段评审
  - 据统计，在软件生存周期各阶段中，编码阶段之前的错误约占63%，而编码错误仅占37%
  - 错误发现并改正得越晚，所花费的代价越高
  - 坚持在每个阶段结束前进行严格的评审，就可以尽早发现错误，从而以最小的代价改正错误

软件工程的基本原理

- 实行严格的产品版本控制
  - 决不随意改变需求，仅依靠科学的产品控制技术来顺应用户提出的改变需求的要求
  - 为了保持软件各个配置成分的一致性，必须实行严格的产品控制
    - 其中主要是实行基准配置管理(又称为变动控制)
      - 即凡是修改软件的建议，尤其是涉及基本配置的修改建议，都必须按规程进行严格的评审，评审通过后才能实施
      - “基准配置”是指经过阶段评审后的软件配置成分，即各阶段产生 的文档或程序代码等
- 采用现代程序设计技术
  - 实践表明，采用先进的程序设计技术既可以提高软件开发与维护的效率，又可以提高软件的质量
    - 20世纪60年代末提出的结构程序设计技术
    - 后又发展出各种结构分析(SA)和结构设计 (SD)技术
    - 之后又出现了面向对象分析(OOA)和面向对象设计(OOD)技术等

软件工程的基本原理

- 结果应能清楚地审查
  - 软件产品是一种看不见、摸不着的逻辑产品。因此，工作进情况可见性差，难于评价和管理
  - 应根据软件开发的总目标和完成期限，尽量明确地规定软件开发小组的责任和产品标准，从而使所得到的结果能清楚地审查
- 开发小组的人员应少而精
  - 软件开发团队人员素质和数量是影响软件质量和开发效率的重要因素
  - 实践表明,素质高的人员与素质低的人员相比，开发效率可能高几倍至几十倍、而且所开发的软件中的错误也要少得多
  - 开发团队的人数不宜过多，随着人数的增加，人员之间交流情况、讨论问题的通信开销将急剧增加，反而由于误解等原因可能增加出错的概率
- 承认不断改进软件工程实践的必要性

软件工程的发展

- 结果应能清楚地审查
  - 软件产品是一种看不见、摸不着的逻辑产品。因此，工作进情况可见性差，难于评价和管理
  - 应根据软件开发的总目标和完成期限，尽量明确地规定软件开发小组的责任和产品标准，从而使所得到的结果能清楚地审查
- 开发小组的人员应少而精
  - 软件开发团队人员素质和数量是影响软件质量和开发效率的重要因素
  - 实践表明,素质高的人员与素质低的人员相比，开发效率可能高几倍至几十倍、而且所开发的软件中的错误也要少得多
  - 开发团队的人数不宜过多，随着人数的增加，人员之间交流情况、讨论问题的通信开销将急剧增加，反而由于误解等原因可能增加出错的概率
- 承认不断改进软件工程实践的必要性

软件工程的发展

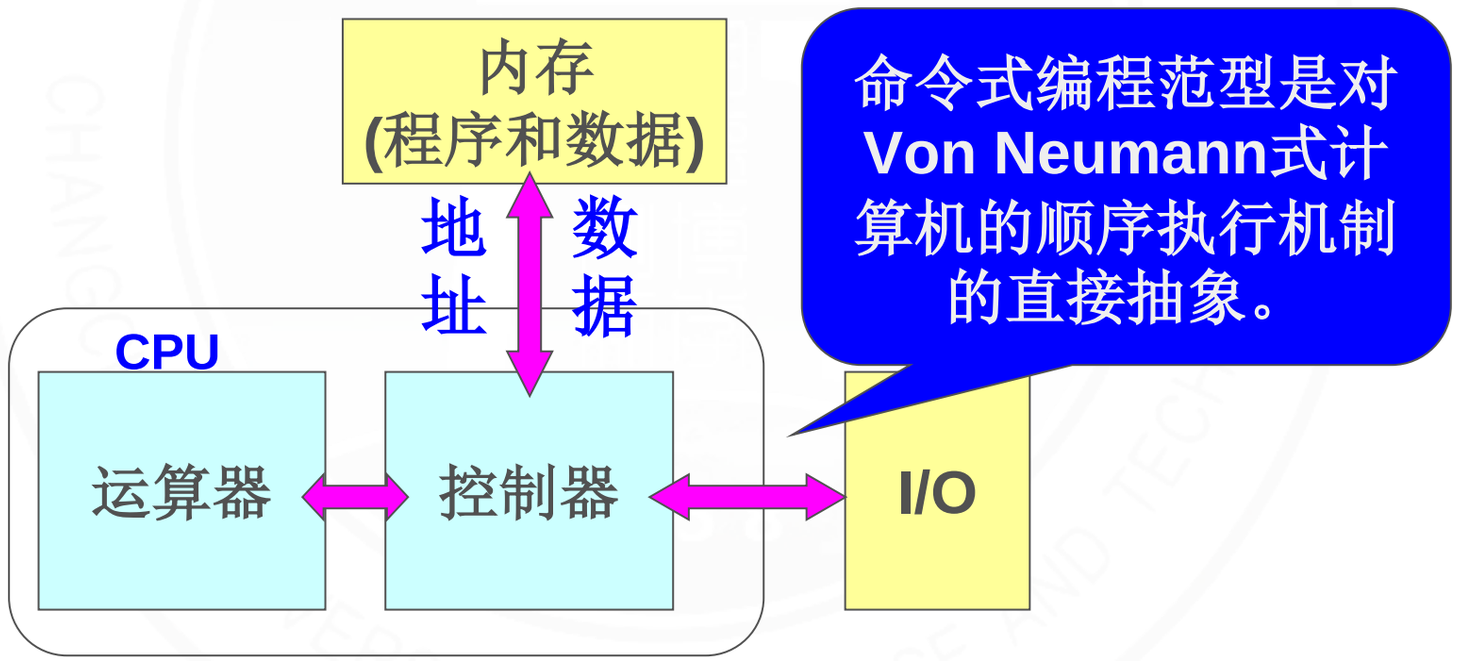
- 软件编程范型
  - 范型：又称为科学基质，代表科学共同体成员所共有的信念、价值、技术手段的总称。简单地说，范型是某一学科在一定时期内展开研究活动共有的基础和准则。
  - 编程范型：是指导和制约编程活动的范型，通常在编程语言中体现。

目前主要的编程范型有：

- 过程式编程范型
- 面向对象编程范型
- 基于构件的编程范型
- 面向服务的编程范型

软件工程的发展

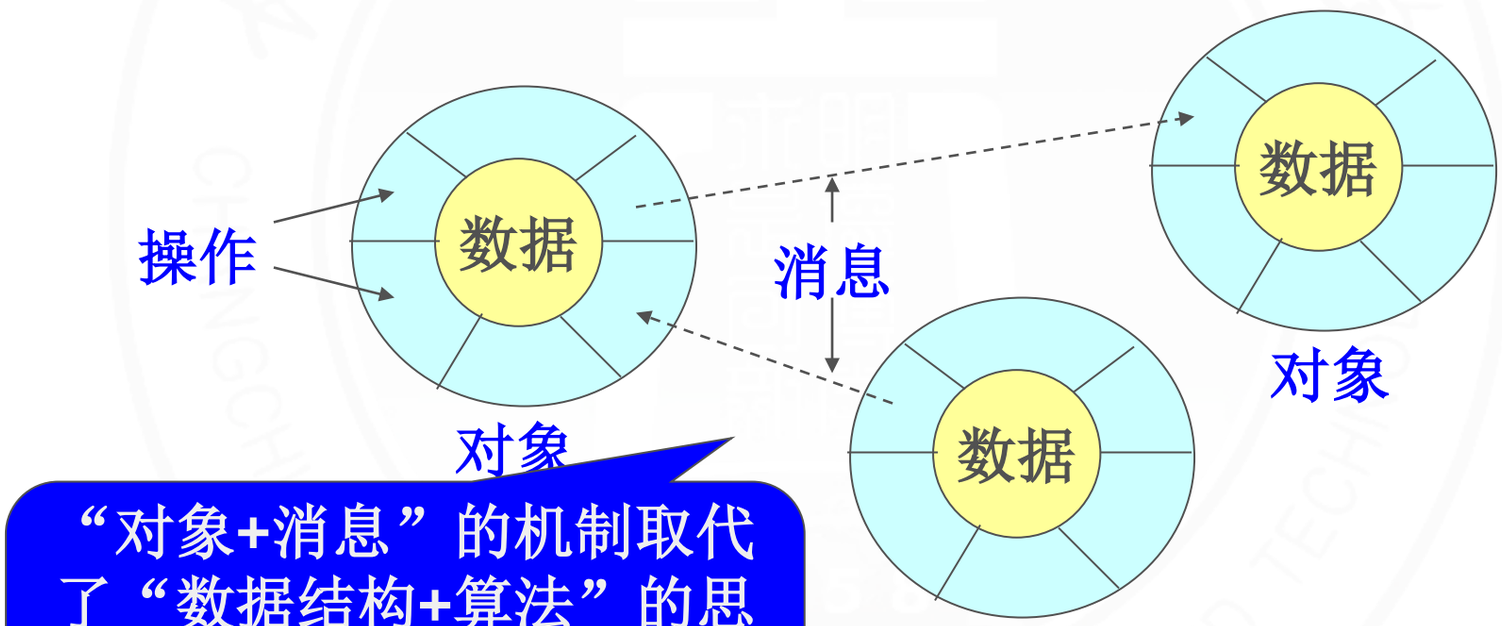
- 过程式编程范型：
  - 过程式编程范型遵循“程序=数据结构+算法”的思路，把程序理解成一组被动的数据和一组能动的过程所构成。典型过程式编程语言包括COBOL、Fortran、 Pascal和C等。



Von Neumann式计算机体系结构

软件工程的发展

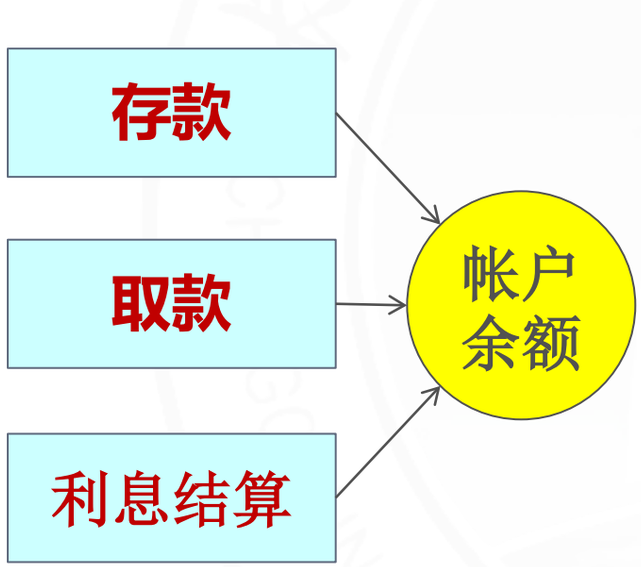
- 面向对象编程范型：
  - 指用封装了数据和对数据操作的对象以及对象之间的消息传递描述计算的编程范型。Java、C#等是典型的面向对象编程范型语言。



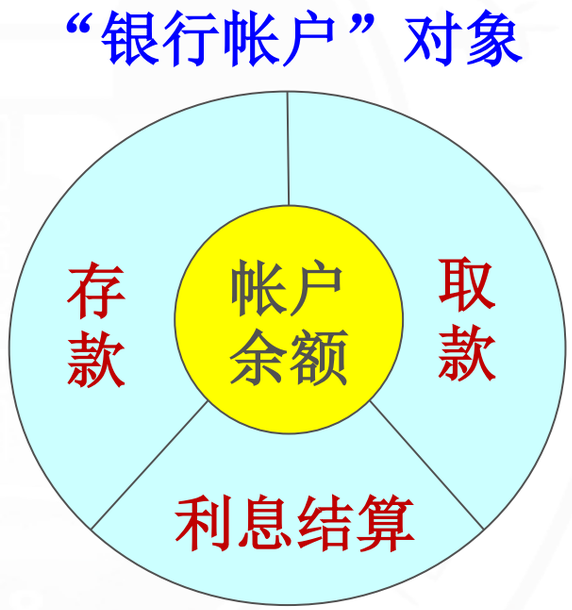
面向对象编程范型的示意图

软件工程的发展

- 例如，银行储蓄处理事务。
  - 数据：帐户余额；
  - 操作：存款、取款、利息结算。



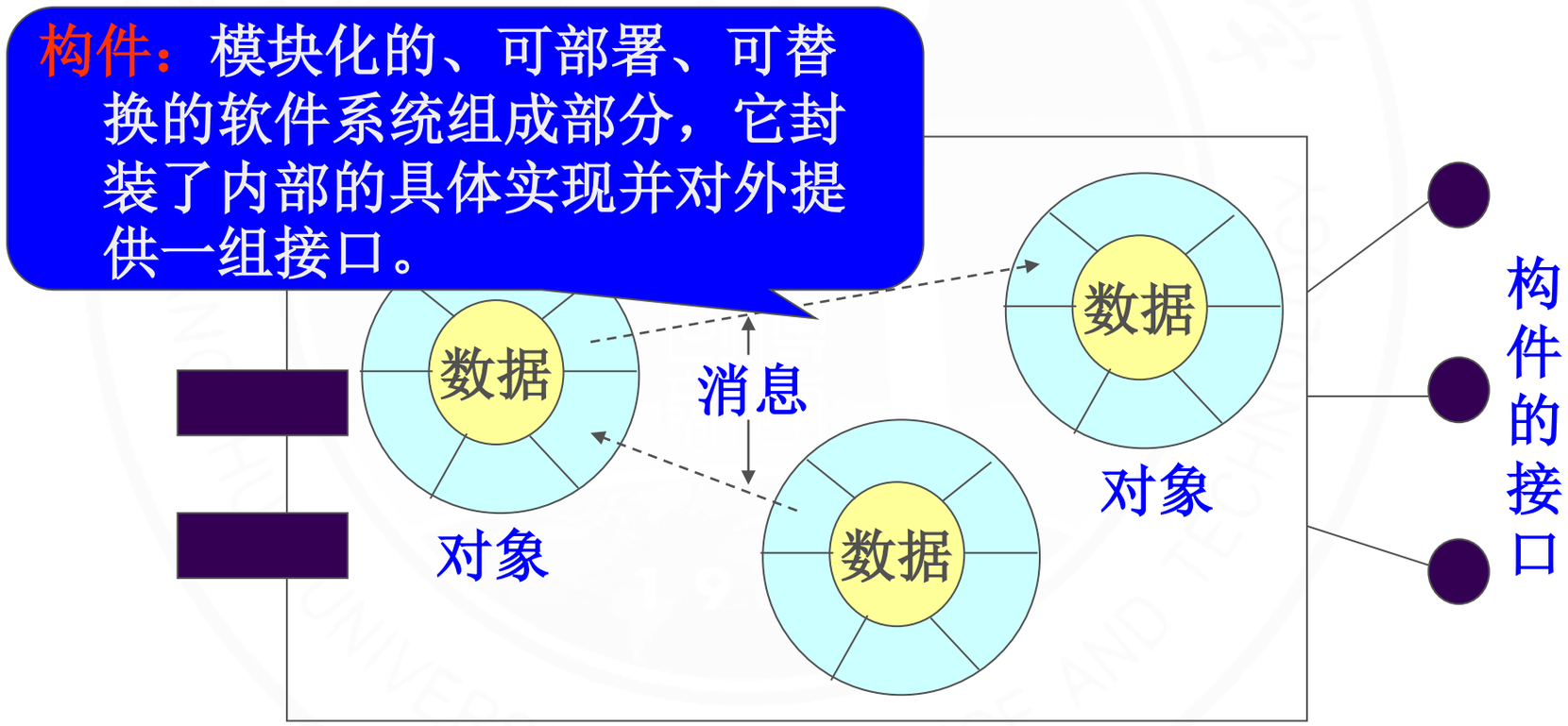
过程式编程范型



面向对象编程范型

软件工程的发展

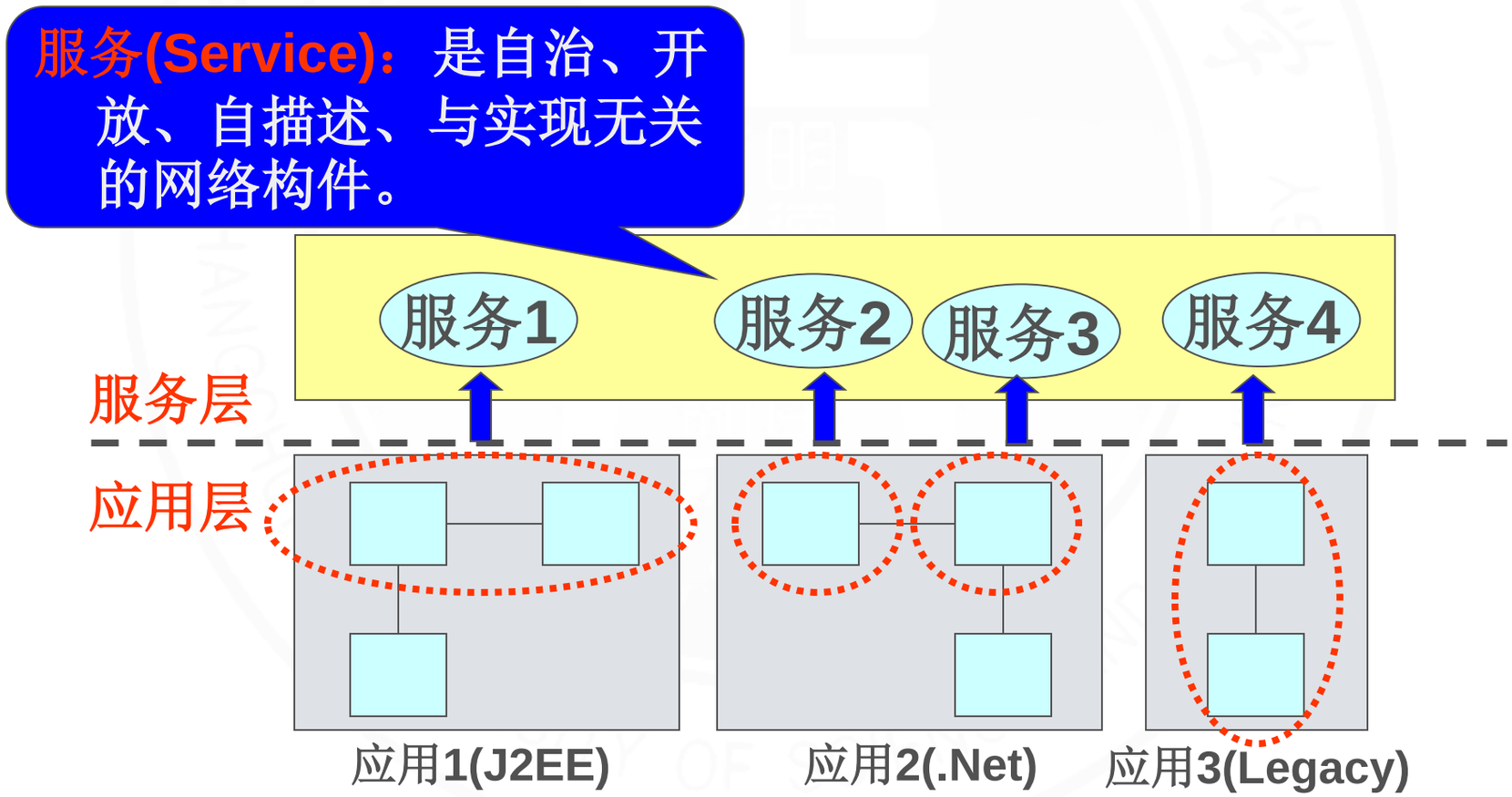
- 基于构件的编程范型：
  - 指以构件的创建，构件的管理以及复用已有的构件组装形成应用为基本活动的编程范型。目前主要的构件开发技术有：COM/DOCM、CORBA、EJB、Spring、OSGi等。



一个封装了多个对象的构件

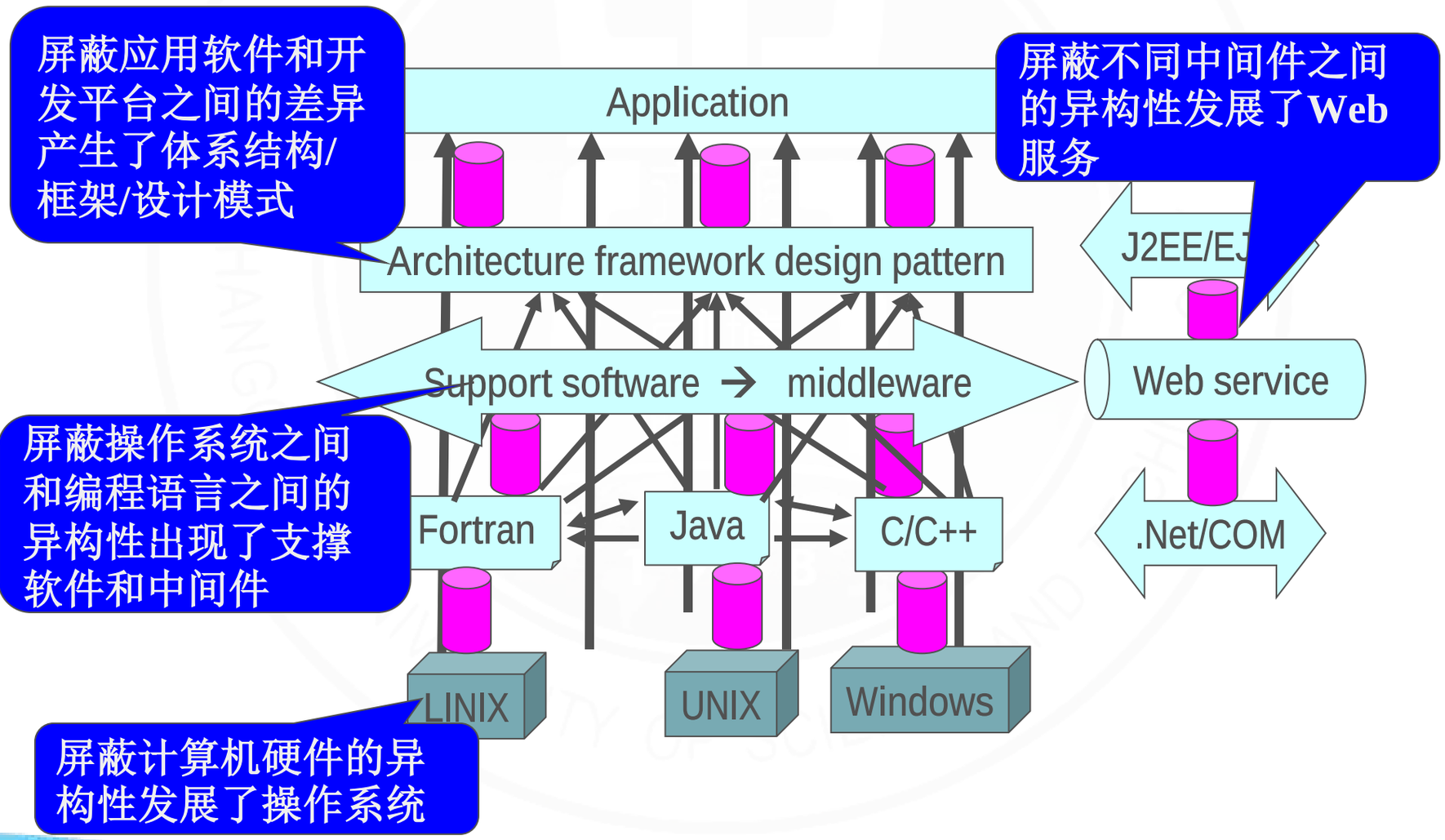
软件工程的发展

- 面向服务的编程范型：
  - 指以服务的创建、服务的管理以及复用已有的服务组装形成应用为基本活动的编程范型。 目前面向服务开发的技术有Web Service、 SCA等。



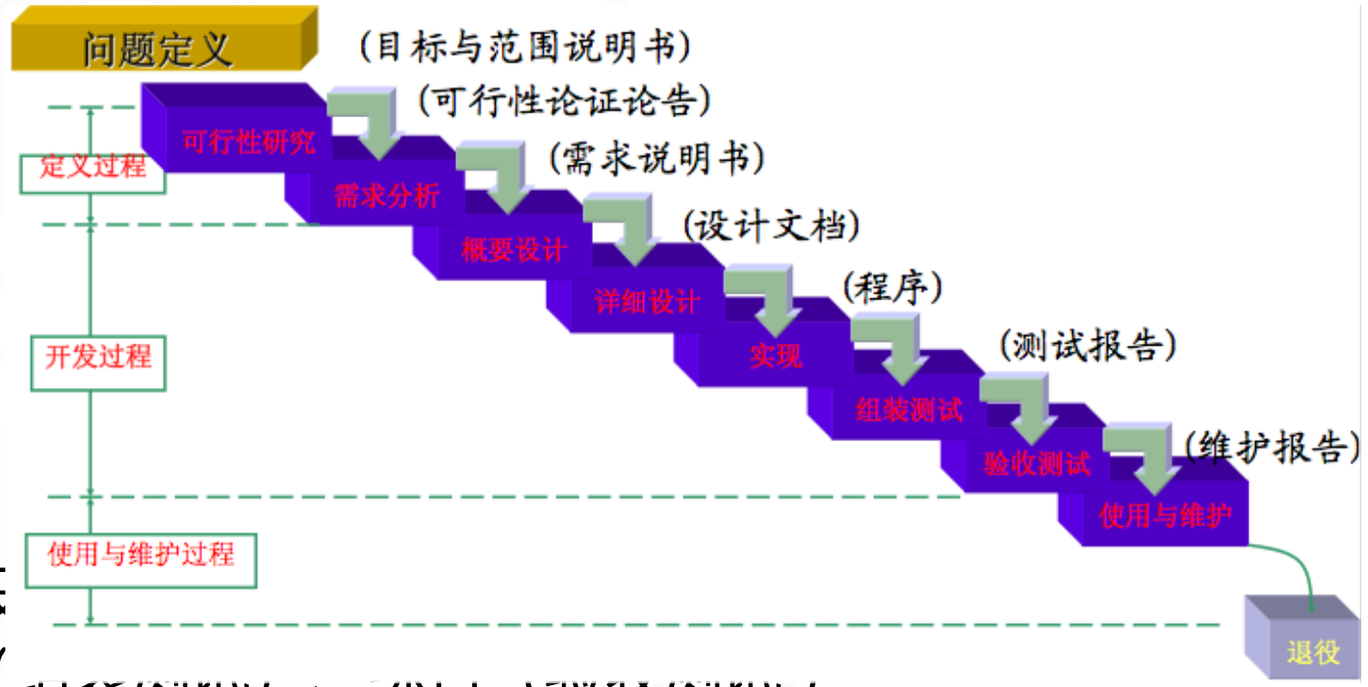
异构系统的功能被封装为服务以方便复用

软件工程技术发展途径


屏蔽应用软件和开发平台之间的差异产生了体系结构/框架/设计模式
Application
屏蔽不同中间件之间的异构性发展了Web服务
Architecture framework design pattern
J2EE/EJ
Support software → middleware
Web service
屏蔽操作系统之间和编程语言之间的异构性出现了支撑软件和中间件
Fortran
Java
C/C++
.Net/COM
LINIX
UNIX
Windows
屏蔽计算机硬件的异构性发展了操作系统

软件的生命周期

- 软件从定义开始，经过开发、使用和维护，直到最终退役废弃的全过程
- 由三个时期/过程组成
  - 软件定义过程（系统分析）
    - 问题定义
    - 可行性研究
    - 需求分析
  - 软件开发过程
    - 系统设计：概要设计、详细设
    - 系统实现：编码、单元测试（
  - 软件运行与维护过程
    - 使用与维护
    - 退役废弃



软件生命周期的基本任务

- 问题定义
  - 关键问题：“要解决的问题是什么”
  - 通过调研，系统分析员应该提出关于问题性质、工程目标、工程规模的书面报告，并且需要得到客户对这份报告的确认
  - 这一部分的工作量很少，因此一些软件开发规范并没有将其作为一个独立的阶段
- 可行性研究
  - 关键问题：“上一个阶段所确定的问题是否有行得通的解决办法”
    - 可行性研究的目的不是解决问题，而是确定问题是否值得去解
    - 目的：用最小的代价在尽可能短的时间内确定问题是否能够解决
  - 任务：根据用户提出的工程项目的性质、目标和规模，进一步了解 用户的要求及现有的环境及条件，从技术、经济和社会等多方面研究并论证可行性。即，该项目是否值得去解决，是否存在可行的解决办法
    - 系统分析员必须进一步概括地了解用户的需求，并在此基础上提出若干种可能的系统实现方案，对每种方案都从技术、经济、社会因素（法律）等方面进行分析，从而最终确定这项工程的可行性

软件生命周期的基本任务

- 需求分析
  - 关键问题：“目标系统必须做什么”
  - 目标：确定系统必须完成哪些工作
    - 即对目标系统提出完整、准确、 清晰、具体的要求
    - 不是确定系统怎样完成它的工作
  - 任务
    - 确定软件系统的功能需求、性能需求和运行环境约束
    - 编制软件需求规格说明书（SRS）、软件系统的验收测试准则和初步的用户手册
      - SRS是一个关键性的文档
      - 多数场合，面向开发者的软件需求用需求规格说明语言来描述，它是软件开发人员进行软件设计的依据；同时，SRS又起到与用户签定合同的合同书的作用
      - SRS中应包括软件系统的全部功能需求、性能需求、 接口需求、设计需求、基本结构、开发标准和验收准则等
  - 开发人员必须和用户密切配合、充分交流信息，以得出经过用户确认的系统需求
    - 用户：知道做什么，不知道怎么做
    - 开发人员：不知道做什么，知道怎么做
  - 对大型、复杂的软件系统的主要功能、接口、人机界面等还要进行模拟或建造原型，以便向用户和开发方展示待开发软件系统的主要特征
    - 确定软件需求的过程有时需要反复多次，最终得到用户和开发者的确认

软件生命周期的基本任务

- 概要设计
  - 关键问题：“怎样实现目标系统?”
  - 任务
    - 设计出实现目标系统的几种方案，交用户确认
    - 设计程序的体系结构，即确定程序由哪些模块组成以及模块间的关系
      - 建立软件系统的总体结构和各子系统之间、各模块之间的关系
      - 定义各子系统接口界 面和各功能模块的接口
      - 设计全局数据库或数据结构
      - 规定设计约束,制定组装测试计划
      - 进而给出每个功能模块的功能描述、全局数据定义和外部文件定义等
- 详细设计
  - 任务
    - 将概要设计产生的功能模块进一步细化，形成可编程的程序模块
    - 设计程序模块的内部细节,包括算法、数据结构以及各程序模块间的接口信息，
    - 设计模块的单元测试计划
  - 途径
    - 可以采用结构化的设计方法，采用结构化的程序流程图等工具进行描述，也可以采用面向对象的设计方法等

软件生命周期的基本任务

- 编码和单元测试
  - 任务：根据详细设计规格说明，用某种选定的程序设计语言把详细设计的结果转化为机器可运行的源程序模块
    - 即，编程和调试程序的过程
    - 编码阶段应注意遵循编程标准、养成良好的编程风格,以便编写出正确的便于理解、调试和维护的程序模块
  - 一般来说，对软件系统所采用的分析方法、设计方法、编程方法以及所选用的程序设计语言应尽可能保持一致

- 单元测试
  - 每编写出一个程序模块的源程序,调试通过后,即对该模块进行测试

软件生命周期的基本任务

- 综合测试
  - 根据概要设计提供的软件结构、各功能模块的说明和组装测试计划，把经过单元测试检验的模块按照某种选定的策略逐步进行组装和测试
  - 任务
    - 测试系统各模块间的连接是否正确，系统或子系统的正确处理能力、容错能力、输入/输出处理是否达到要求
- 集成测试
  - 根据设计的软件结构，把经过单元测试检验的模块按某种选定的策略起来，在装配过程中对程序进行必要的测试
- 验收测试（试用）
  - 按照规格说明书的规定，由用户对目标系统进行验收。通常需要对用户进行培训

软件生命周期的基本任务

- 软件使用与维护
  - 任务
    - 通过各种维护活动使软件系统持久地满足用户的需求
      - 改正性维护，也就是诊断和改正在使用过程中发现的软件错误
      - 适应性维护，即修改软件以适应环境的变化
      - 完善性维护，即根据用户的要求改进或扩充软件使它更完善
      - 预防性维护，即修改软件为将来的维护活动预先做准备
  - 每项维护活动实质上都是一次压缩和简化了的软件定义和软件开发过程
    - 要经历提出维护要求、分析维护要求、提出维护 方案、审批维护方案、确定维护计划、修改软件设计、修改程序、测试程序、评审、验收等步骤
    - 应及时收集被发现的软件错误，并定期撰写“软件问题报告”；每一项维护活动都应准确地记录，作为正式的文档资料保存
    - 软件维护人员为了分析和理解原软件系统所花费的工作量约占整个维护工作量的60%以上。应重视对软件可维护性的支持
- 退役废弃

软件工程的目标

- 在给定成本、进度的前提下，开发出具有(1)可修改性、(2)有效性、(3)可靠性、(4)可理解性、(5)可维护性、(6)可重用性、(7)可适应性、(8)可移植性、(9)可追踪性和(10)可互操作性并满足用户需求的软件产品

- 可修改性（modifiability）
  - 允许对软件系统进行修改而不增加其复杂性，以支持软件调试与维护
- 有效性（efficiency）
  - 是指软件系统的时间和空间效率，是一个开发过程的重要目标
- 可靠性（reliability）
  - 是指在给定的时间间隔内，程序成功运行的概率。可靠性是衡量软件质量的一个重要目标
- 可理解性（understandability）
  - 是指系统具有清晰的结构，能直接反映问题的需求。可理解性有助于控制软件系统的复杂性，并支持软件的维护、移植和重用
- 可维护性（maintainability）
  - 是指软件产品交付使用后，在实现改正潜在的错误、改进性能、适应环境变化等方面工作的难易程度。可维护性是软件工程中的重要的目标。软件的可理解性和可修改性支持软件的可维护性
- 可重用性（reusability）
  - 是指软部件可在多种场合使用的程度。重用的层次越高，带来的效益越大。可重用性有助于提高软件产品的质量和开发效率、降低软件开发和维护费用
- 可适应性（adaptability）
  - 是指软件在不同的系统约束条件下，满足用户需求的难易程度。选择流行的软硬件支持环境、采用流行的程序设计语言编码、采用标准的术语和格式书写文档可增强软件产品的可适应性
- 可移植性（portability）
  - 是指软件从一个计算机系统或环境移植到另一个上去的难易程度。采用通用的运行支持环境和通用的程序设计语言的标准部分可提高可移植性。可移植性支持软件的可重用性和可适应性
- 可追踪性（traceability）
  - 是指根据软件需求对软件设计、程序进行正向追踪,或根据程序、软件设计对软件需求进行逆向追踪的能力。软件开发各阶段的文档和程序的完整性、一致性、可理解性支持软件的可追踪性
- 可互操作性（interoperability）
  - 是指多个软件元素相互通信并协同完成任务的能力

软件工程的原则

- 抽象（abstraction）
  - 抽取各个事物中共同的最基本的特征和行为，暂时忽略之间的差异
    - 一般采用分层次抽象的方法来控制软件开发过程的复杂性
    - 抽象使软件的可理解性增强并有利于开发过程的管理
- 信息隐蔽（information hiding）
  - 将模块内部的信息（数据和过程）封装起来。其他模块只能通过简单的模块接口来调用该模块，而不能直接访问，即将模块设计成“黑箱”
    - 信息隐藏的原则可使开发人员把注意力集中于更高层次的抽象上
- 局部化（localization）
  - 即在一个物理模块内集中逻辑上相互关联的计算资源
  - 局部化支持信息隐藏，从而保证模块之间具有松散的耦合、模块内部有较强的内聚，有助于控制每一个解的复杂性
- 一致性（consistency）
  - 整个软件系统（包括程序、数据和文档）的各个模块应使用一致的概念、符号和术语
  - 程序内部接口应保持一致；软件与环境的接口应保持一致；系统规格说明应与系统行为保持一致；用于形式化规格说明的公理系统应保持一致
- 完全性（completeness）
  - 软件系统不丢失任何重要成分，完全实现所需的系统功能的程度
  - 为了保证系统的完全性，在软件的开发和维护过程中需要严格的技术评审
- 可验证性（verifiability）
  - 开发大型软件系统需要对系统逐层分解。系统分解应遵循易于检查、测试、评审的原则，以使系统可验证

软件工程的发展阶段

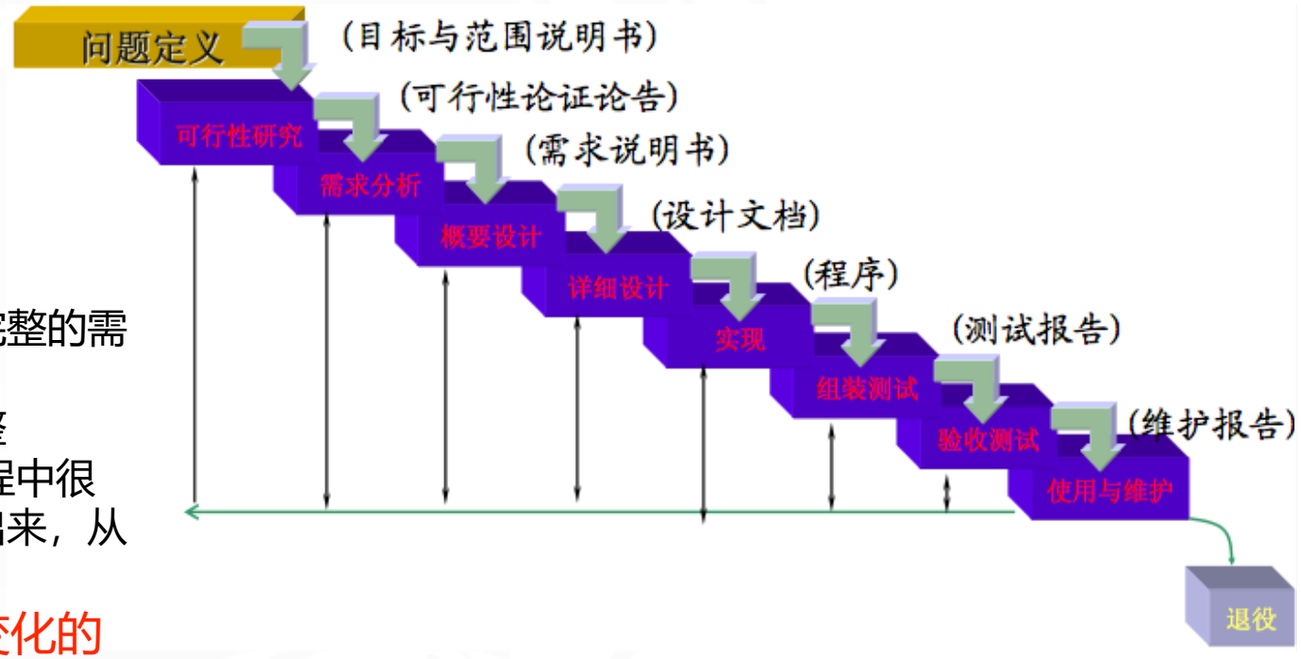
- 传统软件工程：
  - 以结构化程序设计为基础，其开发过程一般包括：结构化分析→结构化设计→面向过程的编码→软件测试。
- 面向对象软件工程：
  - 以面向对象程序设计为基础，其开发过程一般包括:面向需求建模→面向对象分析→面向对象设计→面向对象的编码与测试。
- 基于构件的软件工程：
  - 以软件复用为目标，其开发过程一般包括两个阶段：领域工程和应用工程，连接两个阶段的纽带是可复用构件库。
- 面向服务的软件工程：
  - 目前软件工程领域研究的热点。

软件开发模型

- 软件项目开发和维护的总体过程思路的框架
  - 指出了软件开发过程各阶段之间的关系和顺序，是软件开发过程的概括
  - 为软件开发过程提供原则和方法
  - 为软件工程管理提供里程碑和进度表
- 基本类型
  - 以软件需求完全确定为基础
    - 瀑布模型
  - 在开发初期仅给出基本需求的渐进式模型
    - 原型模型
    - 螺旋模型
    - 喷泉模型等
  - 以形式化开发方法为基础的变换模型、基于四代技术的模型
  - 基于知识的智能模型
- 在实际开发时
  - 应根据项目的特点和现有的条件选取合适的模型，也可以把几种模型组合起来使用以便充分利用各模型的优点

瀑布模型

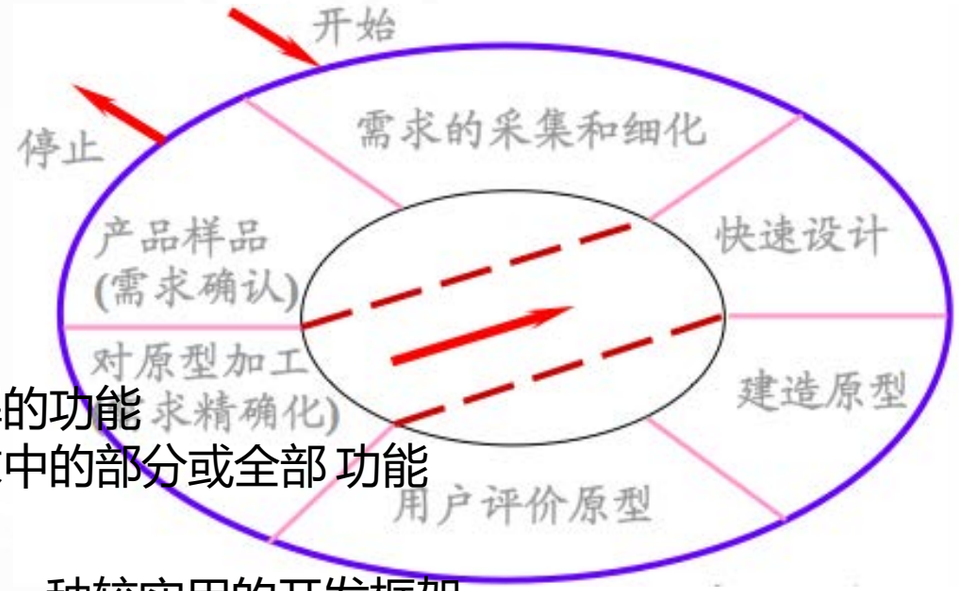
- 由W. Royce于1970年提出，又称为线性顺序模型或软件生存周期模型
  - 瀑布模型遵循软件生存期的划分，明确规定各个阶段的任务，各个阶段的工作自上而下、顺序展开，如同瀑布流水，逐级下落
  - 严格按照软件生存周期各个阶段来进行开发
    - 上一阶段的输出即是下一阶段的输入，并强调每一阶段的严格性
  - 规定了各阶段的任务和应提交的成果及文档，每一阶段的任务完成后，都必须对其阶段性产品(主要是文档)进行评审
    - 通过后才能开始下一阶段的工作
    - 是一种以文档作为驱动的模型



- 优点
  - 提供了软件开发的基本框架
    - 有利于大型软件开发过程中人员的组织、管理
    - 有利于软件开发方法和工具的研究与使用
- 缺点
  - 在软件开发的初期阶段就要求做出正确、全面、完整的需求分析不太现实
  - 当需求确定后，无法及时验证需求是否正确、完整
  - 由于不支持产品的演化，缺乏灵活性，对开发过 程中很难发现的错误，只有在最终产品运行时才能暴露出来，从而使软件产品难以维护
- 一般适用于功能、性能明确、完整、无重大变化的软件系统的开发
  - 例如操作系统、编译系统、数据库管理系统等系统

原型模型

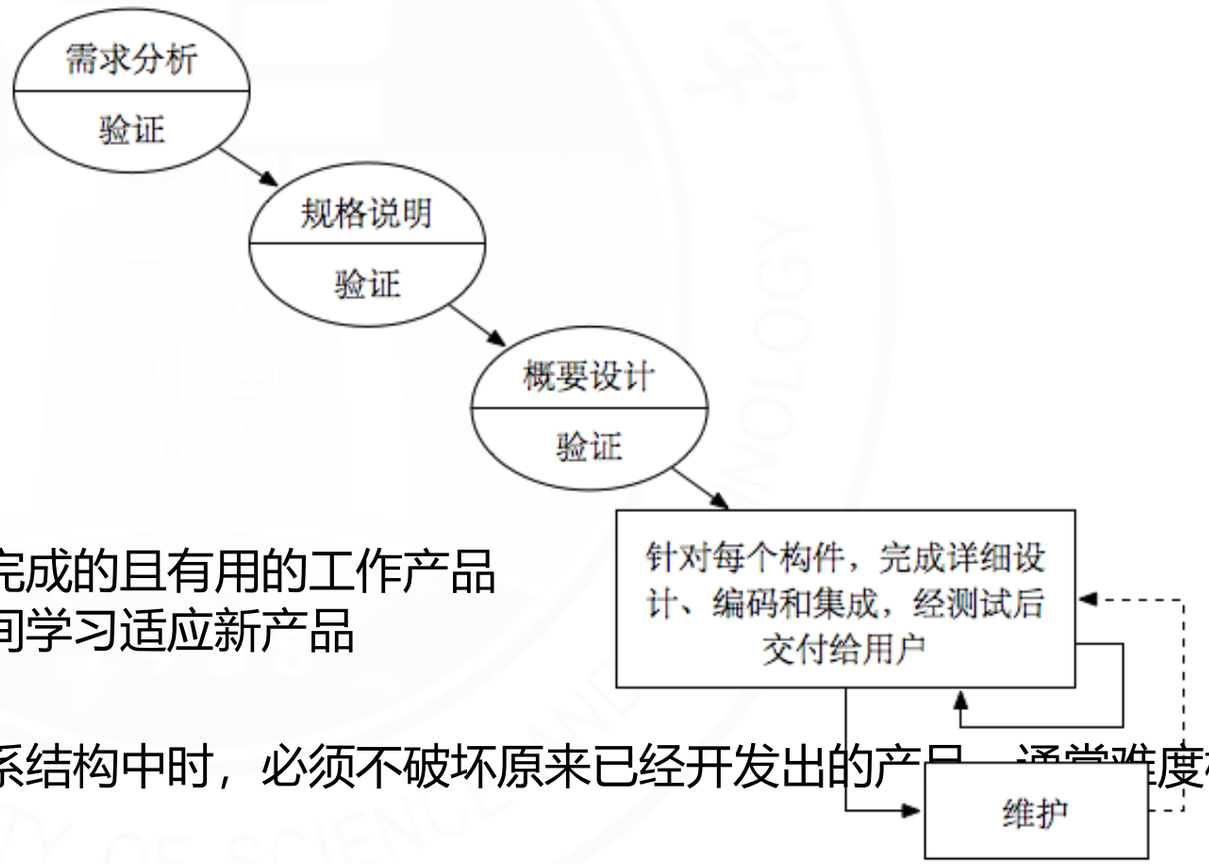
- 软件开发人员根据用户提出的软件基本需求快速开发一个原型，以便向用户展示软件系统应有的部分或全部功能和性能
- 在征求用户对原型的评价意见后，进一步使需求精确化、完全化，并据此改进、完善原型
- 如此迭代，直到软件开发人员和用户都确认软件系统的需求并达成一致的理解为止
- 软件需求确定后，便可进行设计，编码、测试等以后的各个开发步骤



- 开发途径
  - 仅模拟软件系统的人机界面和人机交互方式
  - 开发一个工作模型，实现软件系统中重要的或容易产生误解的功能
  - 利用一个或几个类似的正在运行的软件向用户展示软件需求中的部分或全部 功能
- 适用场合
  - 原型模型比瀑布模型更符合人们认识事物的过程和规律，是一种较实用的开发框架
  - □ 适合于不能预先确切定义需求的软件系统开发
  - 适合于项目组成员(包括分析员、 设计员、程序员和用户) 不能很好交流或通信有困难的情况

增量模型（渐增模型）

- 把软件产品作为一系列的增量构件来设计、编码、集成和测试
- 每个构件由多个相互作用的模块构成，并且能够完成特定的功能
- 使用增量模型时，第一个增量构件往往实现软件的基本需求，提供最核心的功能
- 增量模型分批地向用户提交产品，每次提交一个满足用户需求的可运行的产品



- 优点
  - 能在较短的时间内向用户提供一些已完成的且有用的工作产品
  - 逐步增加的产品功能使用户有充足时间学习适应新产品
- 缺点
  - 在把每个新的构件集成到现有软件体系结构中时，必须不破坏原来已经开发出的产品。通常难度极大

螺旋模型

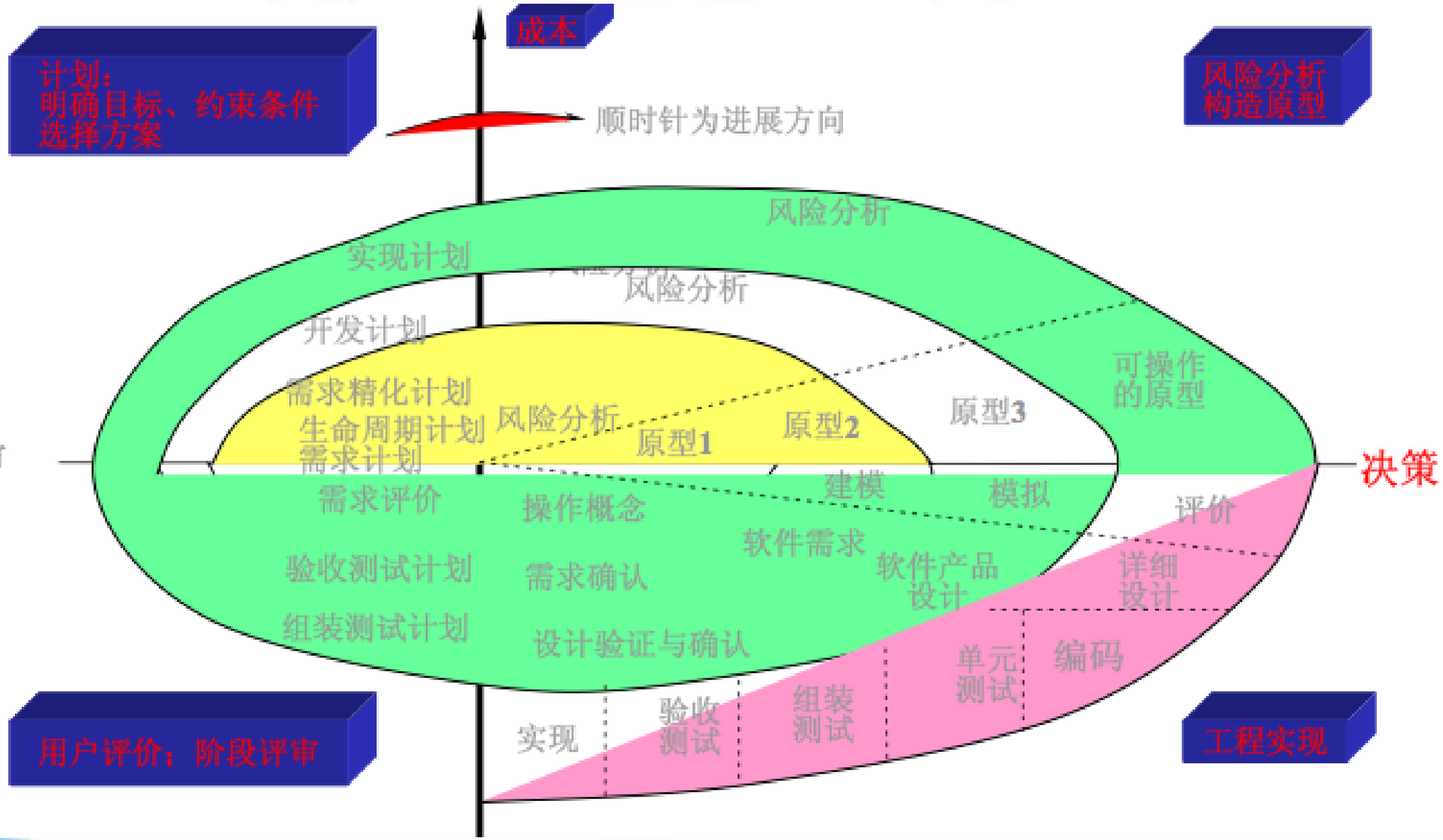
- 是B. Boehm于1988年提出的。它综合了瀑布模型和原型模型的优点，即将两者结合，并加入了风险分析机制
- 螺旋模型的每一个周期都包括
  - 计划（需求定义）、风险分析、工程实现和评审4个阶段
- 优点
  - 对软件开发风险有充分认识，因此适用于内部开发的大规模软件项目
  - 支持软件系统的可维护性，每次维护过程只是沿螺旋模型继续多走一两个周期
- 缺点
  - 风险评估需要开发人员具有丰富的开发经验和各方面的专业知识
  - 如果每次迭代的效率不高，致使迭代次数过多，将会增加成本并推迟提交时间
- 适用场景
  - 支持需求不明确、特别是大型软件系统的开发

螺旋模型

- 计划（需求定义）
  - 第一周期开始利用需求分析技术理解应用领域，获取初步用户需求，制定项目开发计划（即整个软件生命周期计划）和需求分析计划
  - 经过一个周期后，根据用户和开发人员对上一周期工作成果评价和评审，修改、完善需求，明确下一周期软件开发的目标、约束条件，并据此制定新一轮的软件开发计划
- 风险分析
  - 根据本轮制定的开发计划，进行风险分析，评估可选方案，并构造原型进一步分析风险，给出消除或减少风险的途径
  - 根据风险分析的结果决策项目是否继续
    - 螺旋模型是一个风险驱动的模型
- 工程实现
  - 利用构造的原型进行需求建模或进行系统模拟，直至实现软件系统
- 用户评价与阶段评审
  - 将原型提交用户使用并征求改进意见
  - 开发人员应在用户的密切配合下进一步完善用户需求，直到用户认为原型可满足需求，或对软件产品设计进行评价或确认等

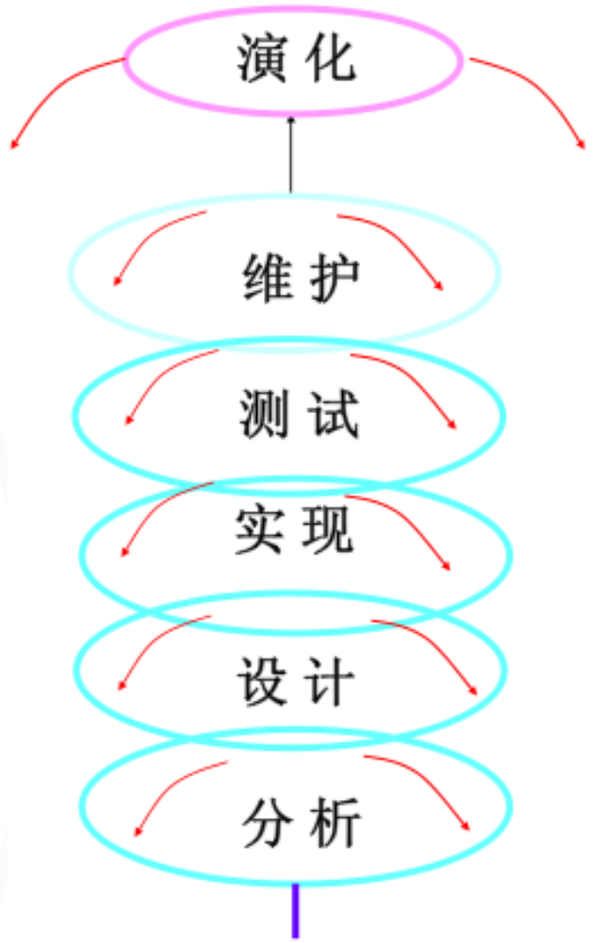
- 螺旋模型从第一个周期的计划开始，一个周期、一个周期地不断迭代，直到整个软件系统开发完成

螺旋模型


计划：
明确目标、约束条件
选择方案
成本
顺时针为进展方向
风险分析
构造原型
风险分析
实现计划
风险分析
开发计划
可操作
的原型
需求精化计划
风险分析
原型3
生命周期计划
原型1
原型2
评审
需求计划
决策
需求评价
操作概念
建模
模拟
评价
软件需求
验收测试计划
需求确认
软件产品
设计
详细
设计
组装测试计划
设计验证与确认
单元
测试
编码
实现
验收
测试
组装
测试
用户评价：阶段评审
工程实现

喷泉模型

- 由B.H.Sollers和 J.M.Edwards于1990年提出的一种开发模型
- 以面向对象的软件开发方法为基础，以用户需求为动力，以对象来驱动的模型
- 克服了瀑布模型不支持软件重用和多项开发活动集成的局限性
- 使开发过程具有迭代性和无间隙性
- 特点
  - 软件系统可维护性较好
  - 各阶段相互重叠，表明了面向对象开发方法各阶段间的交叉和无缝过渡
  - 整个模型是一个迭代的过程，包括一个阶段内部的迭代和跨阶段的迭代
  - 模型具有增量开发特性
    - 即能做到分析一点、设计一点、实现一点、测试一点，使相关功能逐步加入到演化的系统中
  - 模型是对象驱动的
    - 对象是各阶段活动的主体，也是项目管理的基本内容
  - 模型支持部件的重用



软件工程的应用

软件规模分类表

分类	程序规模	子程序数	开发时间	开发人数
极小	500行以下	10-20	1-4周	1人
小	1K-2K行	25-50	1-6月	1人
中	5K-100K行	250-1000	1-2年	2-5人
大	50K-100K行		2-3年	5-20人
甚大	1M行		4-5年	100-1000人
极大	1M-10M行		5-10年	2000-5000人

软件规模

- 极小程序
  - 个人软件，由个人开发和使用。
  - 一般不需要正式的分析和详细的设计文档，也不必制定完整的测试计划。
- 小程序
  - 与其它外部程序没有什么联系，开发者通常一人，无需或很少需要和用户或其他开发人员打交道。例如，求解数值问题的科学计算程序，生成报表或完成数据操作所用的小型商业应用程序，大学生的课程设计程序等。
  - 贯彻软件工程中的技术标准和表示方法，按标准编写文档，并系统地进行复审，但并不是非常严格。

软件规模

- 中规模程序
  - 与其它程序有少量联系，开发人员与用户间或开发人员之间存在一定联系。例如，汇编程序、小型MIS系统、仓库系统以及用于过程控制的应用程序等。
  - 在开发过程中如果能够系统地应用软件工程的原理，对改进软件质量、提高开发人员效率和满足用户需求有很大帮助。
- 大型程序
  - 与其它程序或软件系统有种种联系，开发人员一般由几个开发小组组成，在组与组间、组内不同成员间、开发人员同管理人员及用户之间，都存在着大量通信，在长期的开发过程中，存在人员的变动问题。
  - 严格按照软件工程的方法开发。

软件规模

- 甚大型程序
  - 大型数据库系统
  - 军事部门指挥与控制系统
  - 操作系统
  - ...
- 极大型程序
  - 空中交通管制系统
  - 洲际导弹防御系统
  - 大型军事指挥控制系统
  - ...