

物流信息系统开发方法

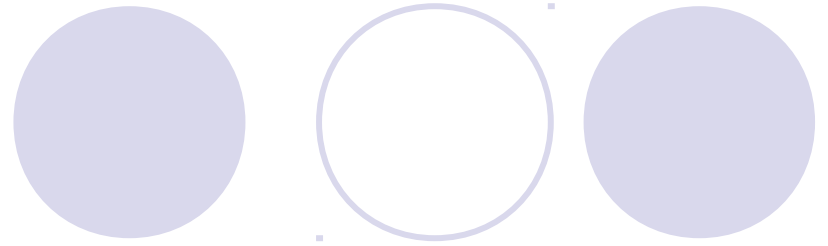
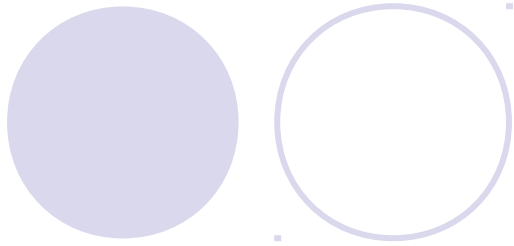
上海海事大学物流研究中心

软件的概念、特点

- 软件是计算机系统中与硬件相互依存的另一部分，它是包括程序，数据及其相关文档的完整集合。
- 其中，程序是按事先设计的功能和性能要求执行的指令序列；数据是使程序能正常操纵信息的数据结构；文档是与程序开发，维护和使用有关的图文材料。

软件的特点

- (1) 软件是一种逻辑实体，而不是具体的物理实体。因而它具有抽象性。
- (2) 软件的生产与硬件不同，它没有明显的制造过程。
- (3) 在软件的运行和使用期间，没有硬件那样的机械磨损，老化问题。
- (4) 软件的开发和运行常常受到计算机系统的限制，对计算机系统有着不同程度的依赖性。



- (5) 软件的开发至今尚未完全摆脱手工艺的开发方式。
- (6) 软件本身是复杂的。
- (7) 软件成本相当昂贵。
- (8) 相当多的软件工作涉及到社会因素。

软件的分

如果按软件规模进行划分：

类别	参加人员数	研制期限	产品规模（源程序行数）
微型	1	1 ~ 4 周	0.5k
小型	1	1 ~ 6 月	1k ~ 2k
中型	2 ~ 5	1 ~ 2 年	5k ~ 50k
大型	5 ~ 20	2 ~ 3 年	50k ~ 100k
甚大型	100 ~ 1000	4 ~ 5 年	1M(=1000k)
极大型	2000 ~ 5000	5 ~ 10 年	1M ~ 10M

软件的发展阶段



- 用户自己开发、自己使用、自己维护
- 多用户、多信道、人机交互
- 分布式、计算机网络、嵌入式； **PC** 时代
- 现在：面向对象、专家系统、人工智能；
Internet 时代
- 未来： 网格计算 （ **Grid Computation** ）

计算机软件发展的三个时期及其特点

特点\时期	程序设计	程序系统	软件工程
软件所指	程序	程序及说明书	程序，文档，数据
主要程序设计语言	汇编及机器语言	高级语言	软件语言
软件工作范围	程序编写	包括设计和测试	软件生存期
软件使用者	程序设计者本人	少数用户	市场用户
软件开发组织	个人	开发小组	开发小组及大中型软件开发机构
软件规模	小型	中小型	大中小型
决定质量的因素	个人编程技术	小组技术水平	技术水平及管理水平
开发技术和手段	子程序和程序库	结构化程序设计	数据库，开发工具，开发环境，工程化开发方法，标准和规范，网络及分布式开发，面向对象技术及软件复用
维护责任者	程序设计者	开发小组	专职维护人员
硬件特征	价格高，存储容量小，工作可靠性差	降价，速度、容量及工作可靠性有明显提高	向超高速，大容量，微型化及网络化方向发展
软件特征	完全不受重视	软件技术的发展不能满足需要，出现软件危机	开发技术有进步，但未获突破性进展，价格高未完全摆脱软件危机

软件危机

- 定义

计算机软件的开发和维护过程所遇到的一系列严重问题。1968 年在西德 Garmish 召开的国际软件工程会议上正式提出。

- 表现

- 对软件开发成本和进度的估算很不准确
- 难以准确获取用户需求，用户不满意
- 质量很不可靠，没有适当的文档
- 缺乏方法指导和工具支持，大型软件系统经常失败
- 供不应求：软件开发生产率跟不上计算机应用的迅速发展

软件危机的典型表现

- (1) 对软件开发成本和进度的估计常常很不准确。实际成本比估计成本有可能高出一个数量级，实际进度比预期进度拖延几个月甚至几年的现象并不罕见。
- (2) 用户对“已完成的”软件系统不满意的现象经常发生。软件开发人员常常在对用户要求只有模糊的了解，甚至对所要解决的问题还没有确切认识的情况下，就匆忙着手编写程序。

- (3) 软件产品的质量往往靠不住。软件可靠性和质量保证的确切的定量概念刚刚出现不久，软件质量保证技术（审查、复审和测试）还没有坚持不懈地应用到软件开发的全过程中，这些都导致软件产品发生质量问题。

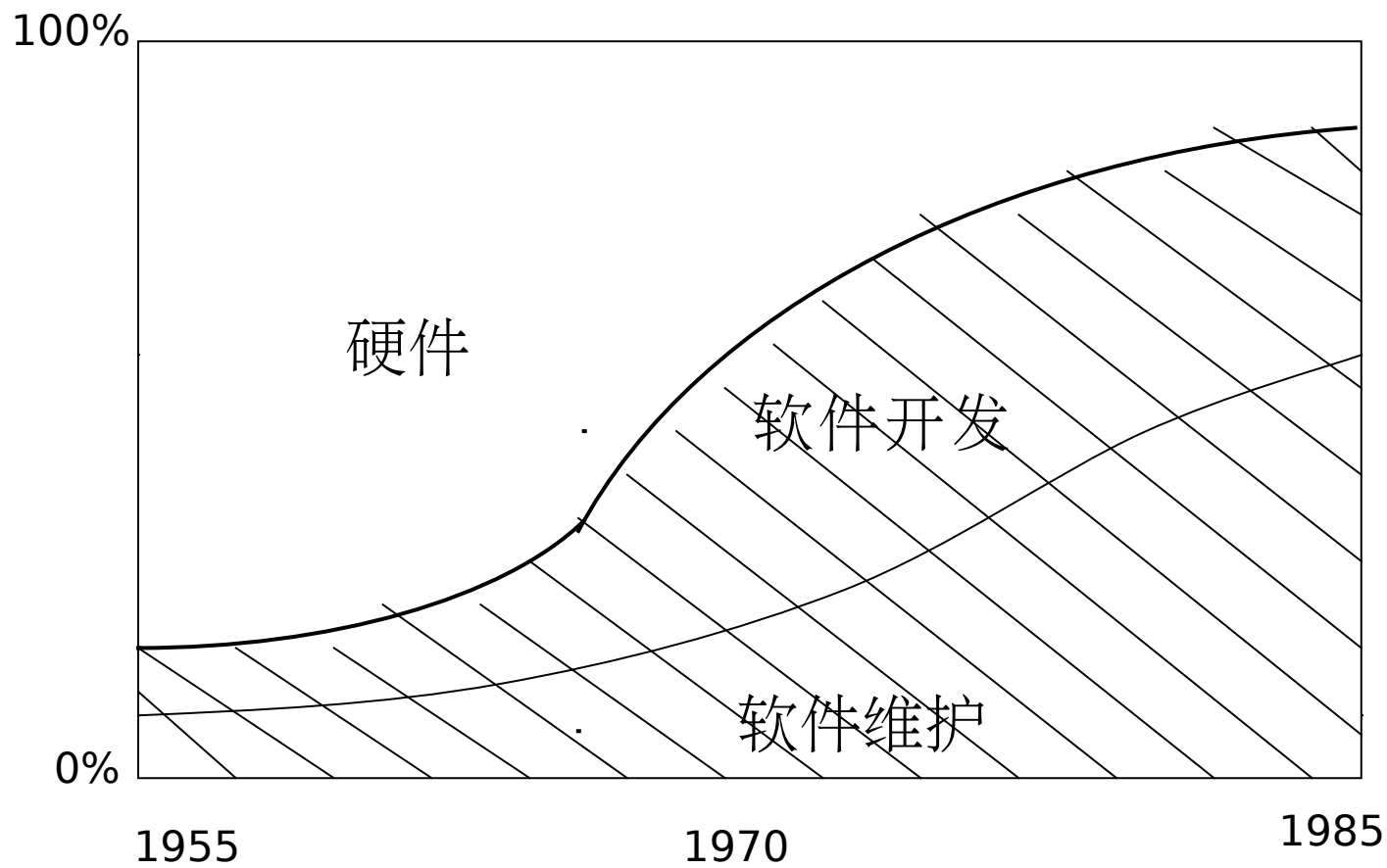
- (4) 软件常常是不可维护的。很多程序中的错误是非常难改正的，实际上不可能使这些程序适应新的硬件环境，也不能根据用户的需要在原有程序中增加一些新的功能。“可重用的软件”还是一个没有完全做到的、正在努力追求的目标，人们仍然在重复开发类似的或基本类似的软件。

- (5) 软件通常没有适当的文档资料。计算机软件不仅仅是程序，还应该有一整套文档资料。这些文档资料应该是在软件开发过程中产生出来的，而且应该是“最新式的”（即和程序代码完全一致的）。软件开发组织的管理人员可以使用这些文档资料作为“里程碑”，来管理和评价软件开发工程的进展状况；软件开发人员可以利用它们作为通信工具，在软件开发过程中准确地交流信息；对于软件维护人员而言，这些文档资料更是必不可少的。

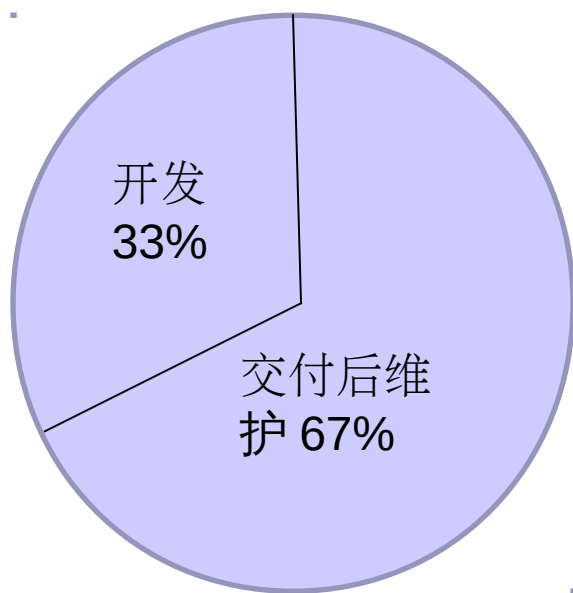
- (6) 软件成本在计算机系统总成本中所占的比例逐年上升。由于微电子学技术的进步和生产自动化程度不断提高，硬件成本逐年下降，然而软件开发需要大量人力，软件成本随着通货膨胀以及软件规模和数量的不断扩大而持续上升。美国在 1985 年软件成本大约已占计算机系统总成本的 90% 。

- (7) 软件开发生产率提高的速度，远远跟不上计算机应用迅速普及深入的趋势。软件产品“供不应求”的现象使人类不能充分利用现代计算机硬件提供的巨大潜力。

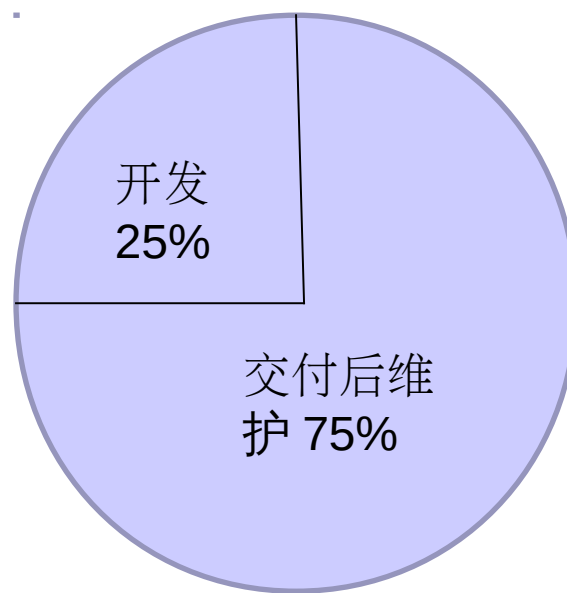
硬件 / 软件成本变化趋势



开发和交付后维护成本近似百分比

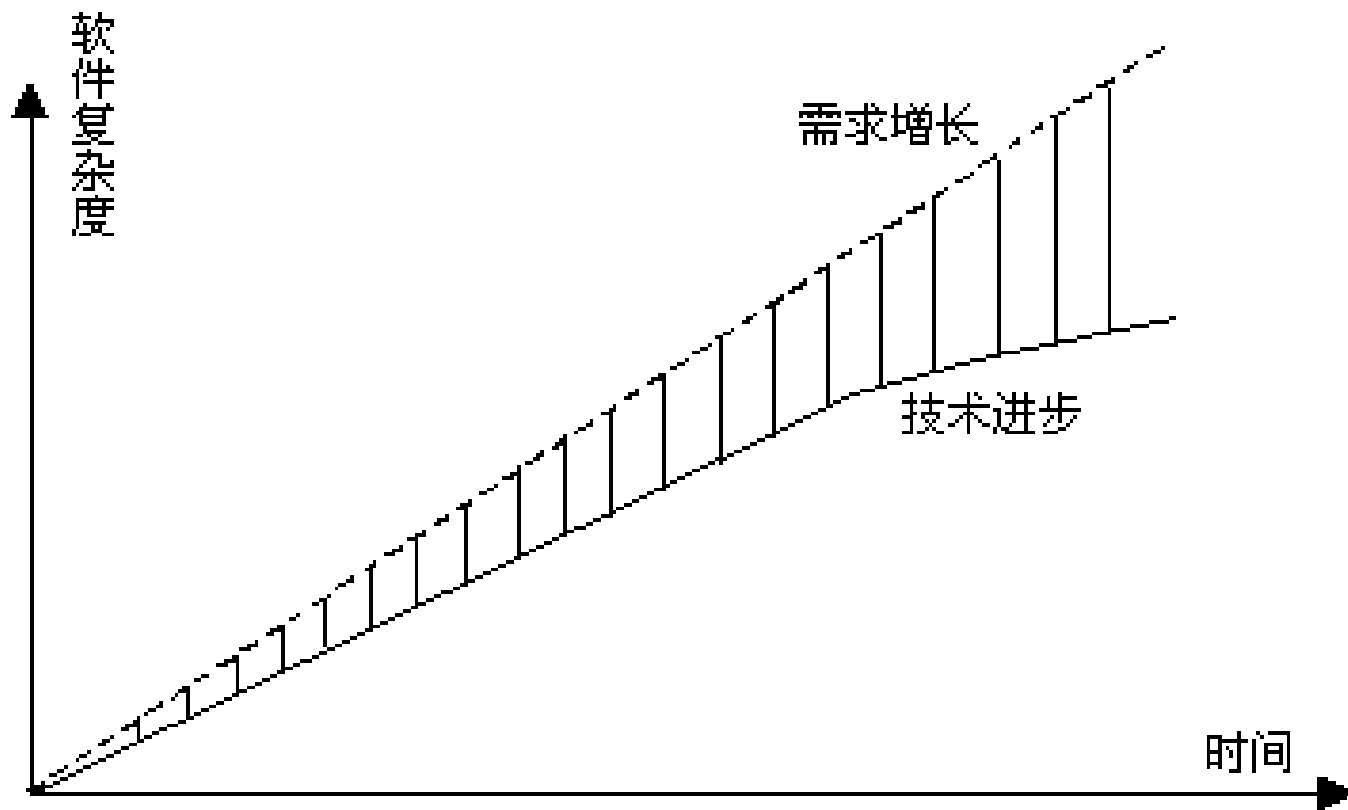


1976-1981



1992-1998

软件技术进步落后于需求增长



目前情況 (2004) CHAOS Summary Standish Group

- Statistics

- 18% of IT projects are cancelled before completion
- 53% of IT projects
 - are over-budget (+45% \$)
 - are over schedule (+63%)
 - Delivered with reduced functionality(-33%)
- 29% are deemed to be successful of IT projects

目前情況 (2009) CHAOS Summary Standish Group

- Statistics

- 24% of IT projects are cancelled before completion
- 44% of IT projects
 - are over-budget
 - are over schedule
 - Delivered with reduced functionality
- 32% are deemed to be successful of IT projects

软件危机

● 原因

○ 客观：软件本身特点

- 逻辑部件
- 规模庞大

○ 主观：不正确的开发方法

- 忽视需求分析
- 错误认为：软件开发 = 程序编写
- 轻视软件维护



Reasons

- Lack of support from management
- Lack of support from users
- Poor project management
- Poorly defined scope and objectives
- Lack of vision
- Miscommunications

产生软件危机的原因

- 在软件开发和维护的过程中存在这么多严重问题，一方面与软件本身的特点有关，另一方面也和软件开发与维护的方法不正确有关。

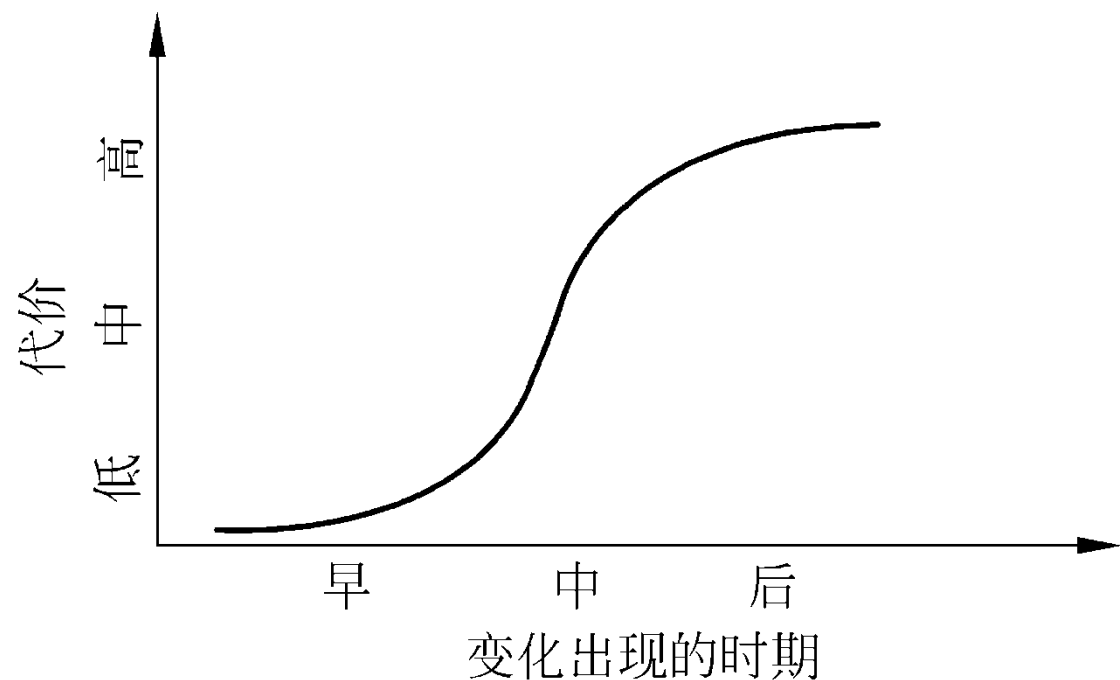
- 软件不同于硬件，它是计算机系统逻辑部件而不是物理部件。
- 此外，软件在运行过程中不会因为使用时间过长而被“用坏”，如果运行中发现了错误，很可能是遇到了一个在开发时期引入的在测试阶段没能检测出来的错误。
- 软件不同于一般程序，它的一个显著特点是规模庞大，而且程序复杂性将随着程序规模的增加而呈指数上升。

- 目前相当多的软件专业人员对软件开发和维护还有不少糊涂观念，在实践过程中或多或少地采用了错误的方法和技术，这可能是使软件问题发展成软件危机的主要原因。

- 错误的认识和作法主要表现为忽视软件需求分析的重要性，认为软件开发就是写程序并设法使之运行，轻视软件维护等。

- 对用户要求没有完整准确的认识就匆忙着手编写程序是许多软件开发工程失败的主要原因之一。
- 另一方面还必须认识到程序只是完整的软件产品的一个组成部分，一个软件产品必须由一个完整的配置组成，主要包括程序、文档和数据等成分。
- 作好软件定义时期的工作，是降低软件成本提高软件质量的关键。

- 严重的问题是，在软件开发的不同阶段进行修改需要付出的代价是很不相同的。



引入同一变动付出的代价随时间变化的趋势

- 轻视维护是一个最大的错误。
- 统计数据表明，实际上用于软件维护的费用占软件总费用的 **55%~70%**。软件工程学的一个重要目标就是提高软件的可维护性，减少软件维护的代价。

软件危机的解决途径

- 解决途径

- 组织管理

- 工程项目管理方法

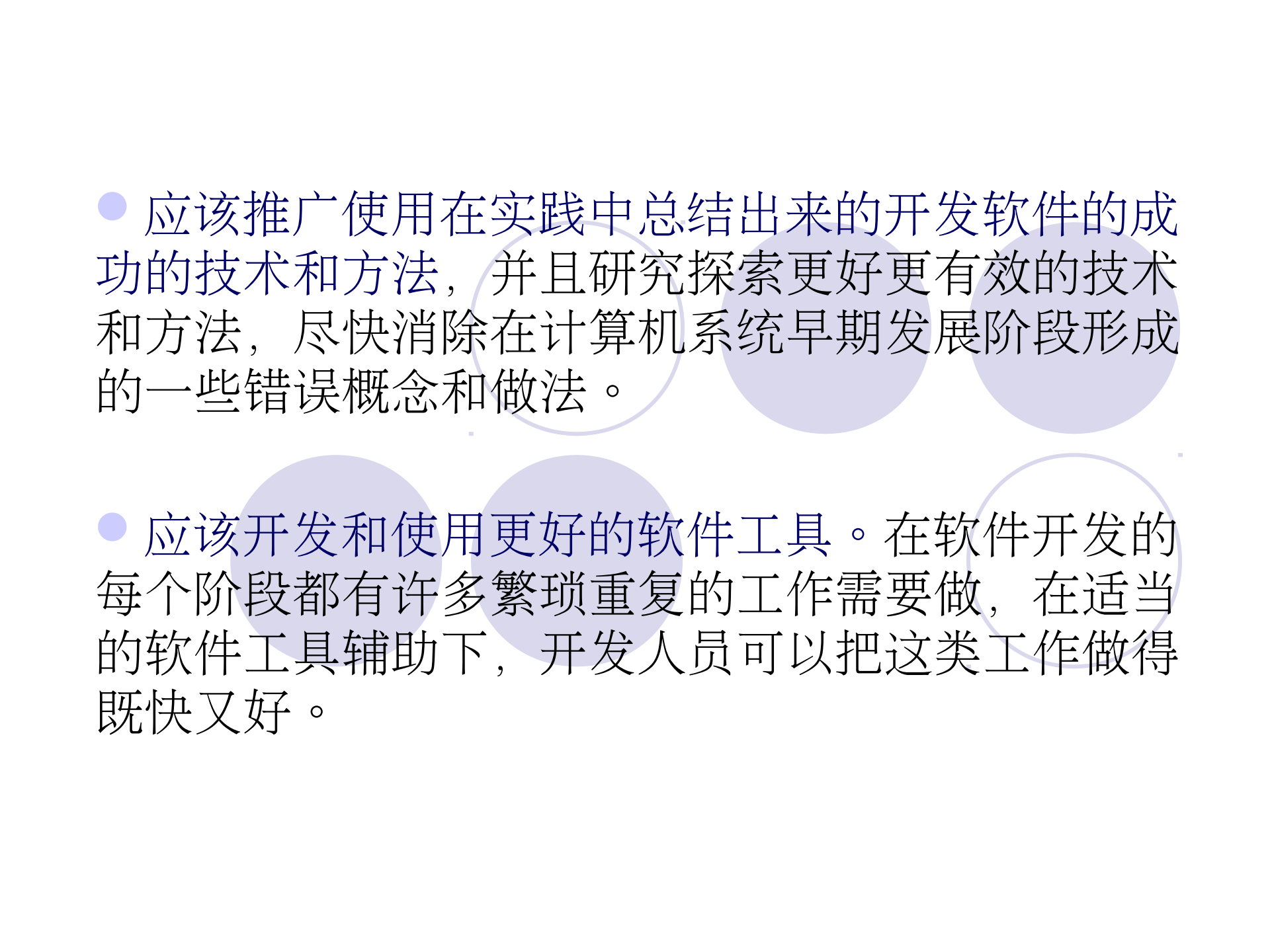
- 技术措施

- 软件开发技术与方法
 - 软件工具

- 为了消除软件危机，首先应该对计算机软件有一个正确的认识，彻底消除“软件就是程序”的错误观念。
- 一个软件必须由一个完整的配置组成，是程序、数据及相关文档的完整集合。

- **1983 年 IEEE** 为软件下的定义是：计算机程序、方法、规则、相关的文档资料以及在计算机上运行程序时所必需的数据。

- 更重要的是，必须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。在软件开发的每个阶段都有许多繁琐重复的工作需要做，在适当的软件工具辅助下，开发人员可以把这类工作做得既快又好。

总之，为了解决软件危机，既要有技术措施（方法和工具），又要有必要的组织管理措施。

软件工程正是从管理和技术两方面研究如何更好地开发和维护计算机软件的一门新兴学科。

软件工程

- 提出

1968 年，北约计算机科学会议， **Fritz Bauer**

- 定义

用工程、科学和数学的原则与方法研制、维护计算机软件的有关技术及管理方法。

- 三要素

- 方法

- 工具

- 过程

软件工程的定义

- **1968** 年在第一届 **NATO** 会议上曾经给出了软件工程的一个早期定义：“软件工程就是为了经济地获得可靠的且能在实际机器上有效地运行的软件，而建立和使用完善的工程原理。”
- **1993** 年 **IEEE** 进一步给出了一个更全面更具体的定义：“软件工程是：①把系统的、规范的、可度量的途径应用于软件开发、运行和维护过程，也就是把工程应用于软件；②研究①中提到的途径。”

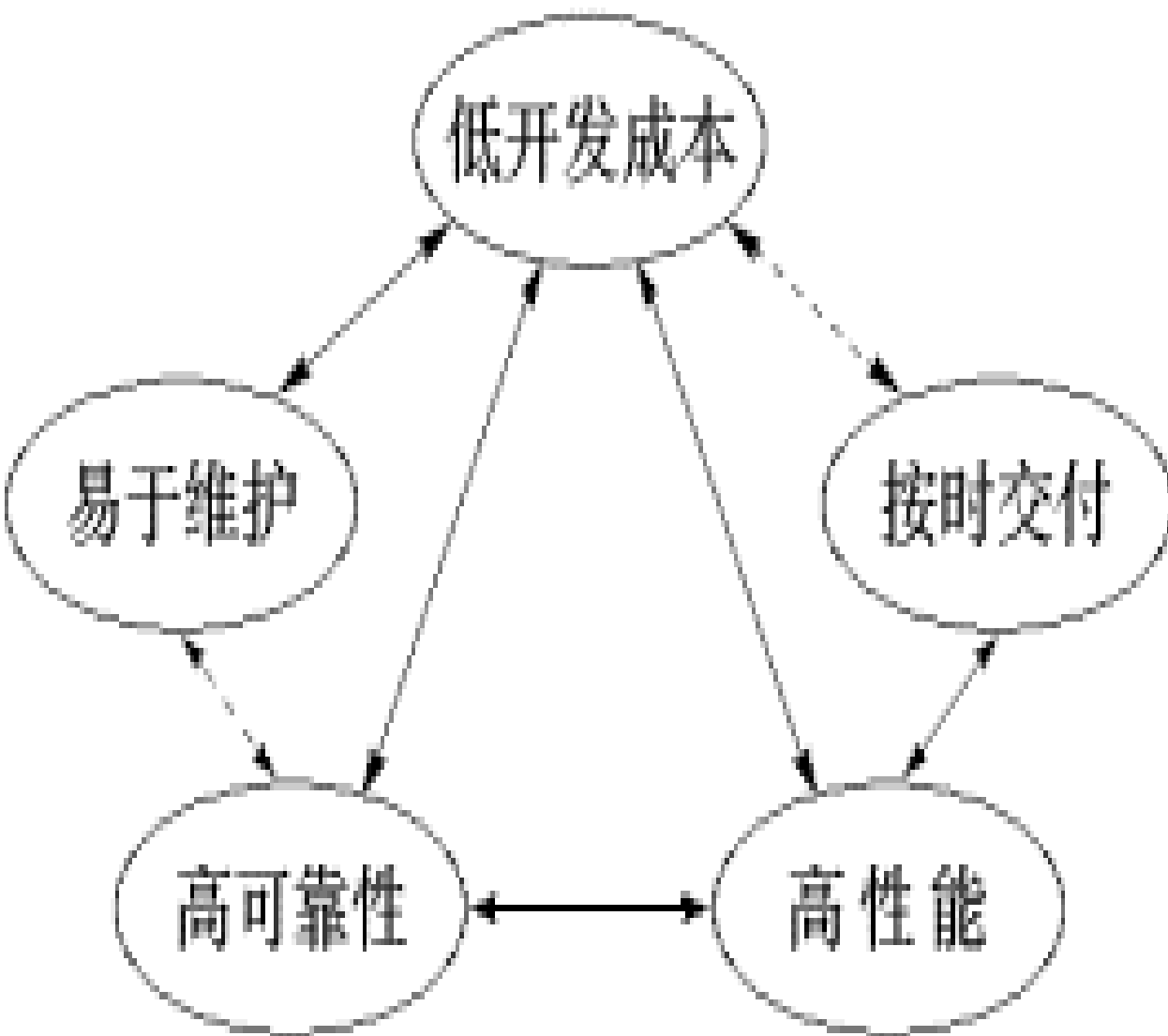
软件工程学的范畴

- 软件工程学
 - 指导计算机软件开发和维护的工程学科
 - 工程管理 + 开发技术
- 软件开发技术
 - 软件开发方法学
 - 软件工具
 - 软件工程环境
- 软件工程管理
 - 软件管理学
 - 软件经济学
 - 软件度量学

软件工程项目的基本目标

- 组织实施软件工程项目，从技术上和管理上采取了多项措施以后，最终希望得到项目的成功。所谓成功指的是达到以下几个主要的目标：
 - 付出较低的开发成本；
 - 达到要求的软件功能；
 - 取得较好的软件性能；
 - 开发的软件易于移植；
 - 需要较低的维护费用；
 - 能按时完成开发工作，及时交付使用。

软件工程目标之间的关系

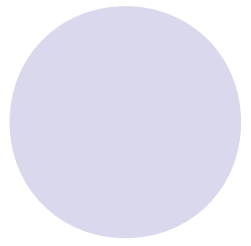
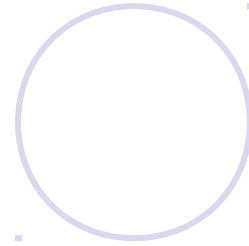
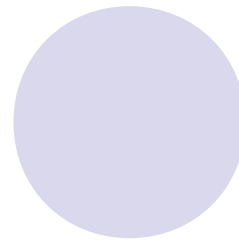
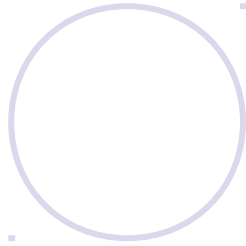
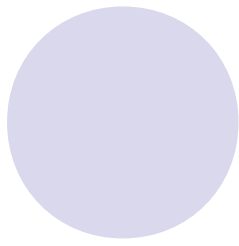


↔ 互斥关系

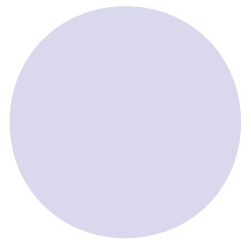
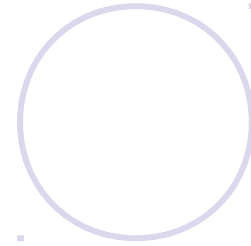
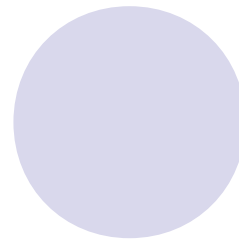
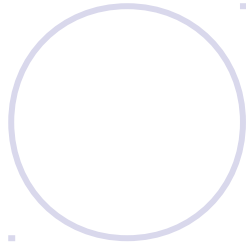
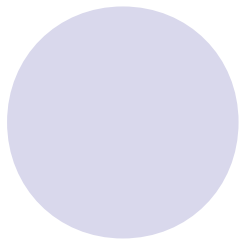
⋯↔ 互补关系

软件工程的原则

- 抽象：
抽取事物最基本的特性和行为，忽略非基本的细节。采用分层次抽象，自顶向下、逐层分解的办法控制软件开发过程的复杂性。例如，软件瀑布模型、结构化分析方法、结构化设计方法，以及面向对象建模技术等都体现了抽象的原则。
- 信息隐蔽：
将模块设计成“黑箱”，实现的细节隐藏在模块内部，不让模块的使用者直接访问。这就是信息封装，使用与实现分离的原则。使用者只能通过模块接口访问模块中封装的数据。
- 模块化：
模块是程序中逻辑上相对独立的成分，是独立的编程单位，应有良好的接口定义。如C语言程序中的函数过程，C++语言程序中的类。模块化有助于信息隐蔽和抽象，有助于表示复杂的系统。



- 局部化：
要求在一个物理模块内集中逻辑上相互关联的计算机资源，保证模块之间具有松散的耦合，模块内部具有较强的内聚。这有助于加强模块的独立性，控制解的复杂性。
- 确定性：
软件开发过程中所有概念的表达应是确定的、无歧义性的、规范的。这有助于人们之间在交流时不会产生误解、遗漏，保证整个开发工作协调一致。
- 一致性：
整个软件系统（包括程序、文档和数据）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。软件和硬件、操作系统的接口应保持一致。系统规格说明与系统行为应保持一致。用于形式化规格说明的公理系统应保持一致。



- 完备性:

软件系统不丢失任何重要成分，可以完全实现系统所要求功能的程度。为了保证系统的完备性，在软件开发和运行过程中需要严格的技术评审。

- 可验证性:

开发大型的软件系统需要对系统自顶向下、逐层分解。系统分解应遵循系统易于检查、测试、评审的原则，以确保系统的正确性。

软件工程的本质特征

- 1. 软件工程关注于大型程序的构造

- “大”与“小”的分界线并不十分清晰。通常把一个人在较短时间内写出的程序称为小型程序，而把多人合作用时半年以上才写出的程序称为大型程序。

- 2. 软件工程的中心课题是控制复杂性

- 软件所解决的问题十分复杂，通常不得不把问题分解，使得分解出的每个部分是可理解的，而且各部分之间保持简单的通信关系。用这种方法并不能降低问题的整体复杂性，但是却可使它变成可以管理的。

● 3. 软件经常变化

○ 绝大多数软件都模拟了现实世界的某一部分。现实世界在不断变化，软件为了不被很快淘汰，必须随着所模拟的现实世界一起变化。因此，在软件系统交付使用后仍然需要耗费成本，而且在开发过程中必须考虑软件将来可能的变化。

● 4. 开发软件的效率非常重要

○ 目前，社会对新应用系统的需求超过了人力资源所能提供的限度，软件供不应求的现象日益严重。因此，软件工程的一个重要课题就是，寻求开发与维护软件的更好更有效的方法和工具。

● 5. 和谐地合作是开发软件的关键

○ 软件处理的问题十分庞大，必须多人协同工作才能解决这类问题。为了有效地合作，必须明确地规定每个人的责任和相互通信的方法。纪律是成功地完成软件开发项目的一个关键。

● 6. 软件必须有效地支持它的用户

○ 开发软件的目的是支持用户的工作。软件提供的功能应该能有效地协助用户完成他们的工作。

○ 有效地支持用户意味着必须仔细地研究用户，以确定适当的功能需求、可用性要求及其他质量要求（例如，可靠性、响应时间等）。有效地支持用户还意味着，软件开发不仅应该提交软件产品，而且应该写出用户手册和培训材料。

● 7. 在软件工程领域中是由具有一种文化背景的人替具有另一种文化背景的人

○ 软件工程师是诸如 **Java** 程序设计、软件体系结构、测试或统一建模语言 (UML) 等方面的专家，他们通常并不是图书馆管理、航空控制或银行事务等领域的专家，但是他们却不得不为这些领域开发应用系统。缺乏应用领域的相关知识，是软件开发项目出现问题的常见原因。

○ 有时候，软件开发通过访谈、阅读书面文件等方法了解到用户组织的“正式”工作流程，然后用软件实现了这个工作流程。但是，决定软件系统成功与否的关键问题是，用户组织是否真正遵守这个工作流程。

软件工程的基本原理

- 著名的软件工程专家 **B.W.Boehm** 综合学者们的意见，于 **1983** 年在一篇论文中提出了软件工程的 **7** 条基本原理。这 **7** 条原理是确保软件产品质量和开发效率的原理的最小集合。
- 这 **7** 条原理是互相独立的，其中任意 **6** 条原理的组合都不能代替另一条原理；这 **7** 条原理又是相当完备的，虽然不能用数学方法严格证明它们是一个完备的集合，但是可以证明在此之前已经提出的 **100** 多条软件工程原理都可以由这 **7** 条原理的任意组合蕴含或派生。

软件工程的 7 条基本原理

● 1. 用分阶段的生命周期计划严格管理

- 经统计发现，在不成功的软件项目中有一半左右是由于计划不周造成的。
- 应该把软件生命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的开发与维护工作进行管理。
- 不同层次的管理人员都必须严格按照计划各尽其职地管理软件开发与维护工作，绝不能受客户或上级人员的影响而擅自背离预定计划。

● 2. 坚持进行阶段评审

- 软件的质量保证工作不能等到编码阶段结束之后再进行。这样说至少有两个理由：
- 第一，大部分错误是在编码之前造成的，根据统计，设计错误占软件错误的 **63%**，编码错误仅占 **37%**；
- 第二，错误发现与改正得越晚，所需付出的代价也越高。
- 因此，在每个阶段都进行严格的评审，以便尽早发现在软件开发过程中所犯的 errors，是一条必须遵循的重要原则。

● 3. 实行严格的产品控制

○ 在软件开发过程中改变需求是难免的，只能依靠科学的产品控制技术来顺应这种要求。也就是说，当改变需求时，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。

○ 所谓基准配置又称为基线配置，它们是经过阶段评审后的软件配置成分。基准配置管理也称为变动控制：一切有关修改软件的建议，特别是涉及到对基准配置的修改建议，都必须按照严格的规程进行评审，获得批准以后才能实施修改。绝对不能谁想修改软件，就随意进行修改。

● 4. 采用现代程序设计技术

- 从提出软件工程的概念开始，人们一直把主要精力用于研究各种新的程序设计技术，并进一步研究各种先进的软件开发与维护技术。
- 实践表明，采用先进的技术不仅可以提高软件开发和维护的效率，而且可以提高软件产品的质量。

● 5. 结果应能清楚地审查

- 软件产品不同于一般的物理产品，它是看不见摸不着的逻辑产品。软件开发人员（或开发小组）的工作进展情况可见性差，难以准确度量，从而使得软件产品的开发过程比一般产品的开发过程更难于评价和管理。
- 为了提高软件开发过程的可见性，更好地进行管理，应该根据软件开发项目的总目标及完成期限，规定开发组织的责任和产品标准，从而使得所得到的结果能够清楚地审查。

● 6. 开发小组的人员应该少而精

- 软件开发小组的组成人员的素质应该好，而人数则不宜过多。
- 素质高的人员的开发效率比素质低的人员的开发效率可能高几倍至几十倍，而且素质高的人员所开发的软件中的错误明显少于素质低的人员所开发的软件中的错误。
- 此外，随着开发小组人员数目的增加，因为交流情况讨论问题而造成的通信开销也急剧增加。当开发小组人员数为 N 时，可能的通信路径有 $N(N-1)/2$ 条，可见随着人数 N 的增大，通信开销将急剧增加。

● 7. 承认不断改进软件工程实践的必要性

○ 遵循上述 6 条基本原理，就能够按照当代软件工程基本原理实现软件的工程化生产，但是，仅有上述 6 条原理并不能保证软件开发与维护的过程能赶上时代前进的步伐，能跟上技术的不断进步。

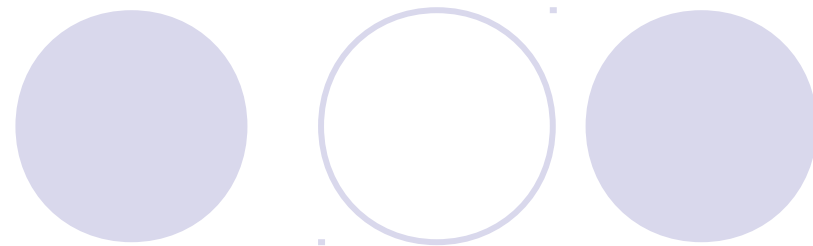
○ 因此，**Boehm** 提出应把承认不断改进软件工程实践的必要性作为软件工程的第 7 条基本原理。按照这条原理，不仅要积极主动地采纳新的软件技术，而且要注意不断总结经验。

软件工程方法学

- 软件工程包括技术和管理两方面的内容，是技术与管理紧密结合所形成的工程学科。
- 通常把在软件生命周期全过程中使用的一整套技术方法的集合称为方法学 (methodology) 也称为范型 (paradigm)。

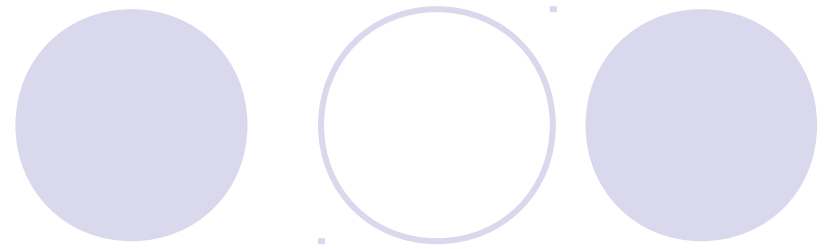
- 软件工程方法学包含 **3** 个要素：
 - 方法是完成软件开发的各项任务的技术方法，回答“怎样做”的问题；
 - 工具是为运用方法而提供的自动的或半自动的软件工程支撑环境；
 - 过程是为了获得高质量的软件所需要完成的一系列任务的框架，它规定了完成各项任务的工作步骤。
- 目前使用得最广泛的软件工程方法学：
 - 传统方法学
 - 面向对象方法学

两种程序设计方法



- 程序设计的两次飞跃
- 结构化程序设计
 - 程序 = 数据结构 + 算法
- 面向对象程序设计
 - 程序 = 对象 + 消息

两类软件工程方法



- 传统软件工程

- 软件分析 → 总体设计 → 详细设计 → 面向过程的编码 → 测试

- 面向对象软件工程

- 软件分析与对象抽取 → 对象详细设计 → 面向对象的编码 → 测试

软件生存周期（生命周期）

○ 计划时期

- 问题定义
- 可行性分析

○ 开发时期

- 需求分析
- 软件设计
- 编码
- 测试

○ 运行时期

- 软件维护

● 1. 问题定义

- 问题定义阶段必须回答的关键问题是：“要解决的问题是什么？”如果不知道问题是什么就试图解决这个问题，显然是盲目的。尽管确切地定义问题的必要性是十分明显的，但是在实践中它却可能是最容易被忽视的一个步骤。
- 通过对客户的访问调查，系统分析员扼要地写出关于问题性质、工程目标和工程规模的书面报告，经过讨论和必要的修改之后这份报告应该得到客户的确认。

● 2. 可行性研究

- 这个阶段要回答的关键问题是：“对于上一个阶段所确定的问题有行得通的解决办法吗？”
- **系统分析员**需要进行一次大大压缩和简化了的系统分析和设计过程，也就是在较抽象的高层次上进行的分析和设计过程。
- 可行性研究应该比较简短，这个阶段的任务不是具体解决问题，而是研究问题的范围，探索这个问题是否值得去解，是否有可行的解决办法。
- 可行性研究的结果是使用部门负责人作出是否继续进行这项工程的决定的重要依据。可行性研究以后的那些阶段将需要投入更多的人力物力。及时终止不值得投资的工程项目，可以避免更大的浪费。

● 3. 需求分析

- 这个阶段的任务仍然不是具体解决问题，而是确定“为了解决这个问题，目标系统必须做什么？”
，主要是确定目标系统必须具备哪些功能。
- 系统分析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逻辑模型。通常用数据流图、数据字典和简要的算法表示。
- 在需求分析阶段确定的系统逻辑模型是以后设计和实现系统的基础。这个阶段的一项重要任务，是用正式文档准确地记录对目标系统的需求，这份文档通常称为规格说明书 (specification) 。

● 4. 总体设计

- 这个阶段必须回答的关键问题是：“概括地说，应该怎样实现目标系统？”总体设计又称为概要设计。
- 首先，应该设计出实现目标系统的几种可能的方案。通常至少应该设计出低成本、中等成本和高成本等 3 种方案。软件工程师在充分权衡各种方案的利弊的基础上，推荐一个最佳方案。制定出实现最佳方案的详细计划。
- 一个程序应该由若干个规模适中的模块按合理的层次结构组织而成。总体设计的另一项主要任务就是设计程序的体系结构，也就是确定程序由哪些模块组成以及模块间的关系。

● 5. 详细设计

- 详细设计阶段的任务就是把解法具体化，也就是回答这个关键问题：“应该怎样具体地实现这个系统呢？”
- 这个阶段的任务还不是编写程序，而是设计出程序的详细规格说明。这种规格说明应该包含必要的细节，程序员可以根据它们写出实际的程序代码。
- 详细设计也称为模块设计，在这个阶段将详细地设计每个模块，确定实现模块功能所需要的算法和数据结构。

● 6. 编码和单元测试

- 这个阶段的关键任务是写出正确的、容易理解、容易维护的程序模块。
- 程序员应该根据目标系统的性质和实际环境，选取一种适当的高级程序设计语言（必要时用汇编语言），把详细设计的结果翻译成用选定的语言书写的程序，并且仔细测试编写出的每一个模块。

● 7. 综合测试

- 这个阶段的关键任务是通过各种类型的测试（及相应的调试）使软件达到预定的要求。
- 最基本的测试是集成测试和验收测试。
 - 所谓集成测试是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装配起来，在装配过程中对程序进行必要的测试。
 - 所谓验收测试则是按照规格说明书的规定，由用户对目标系统进行验收。
- 必要时还可以再通过现场测试或平行运行等方法对目标系统进一步测试检验。

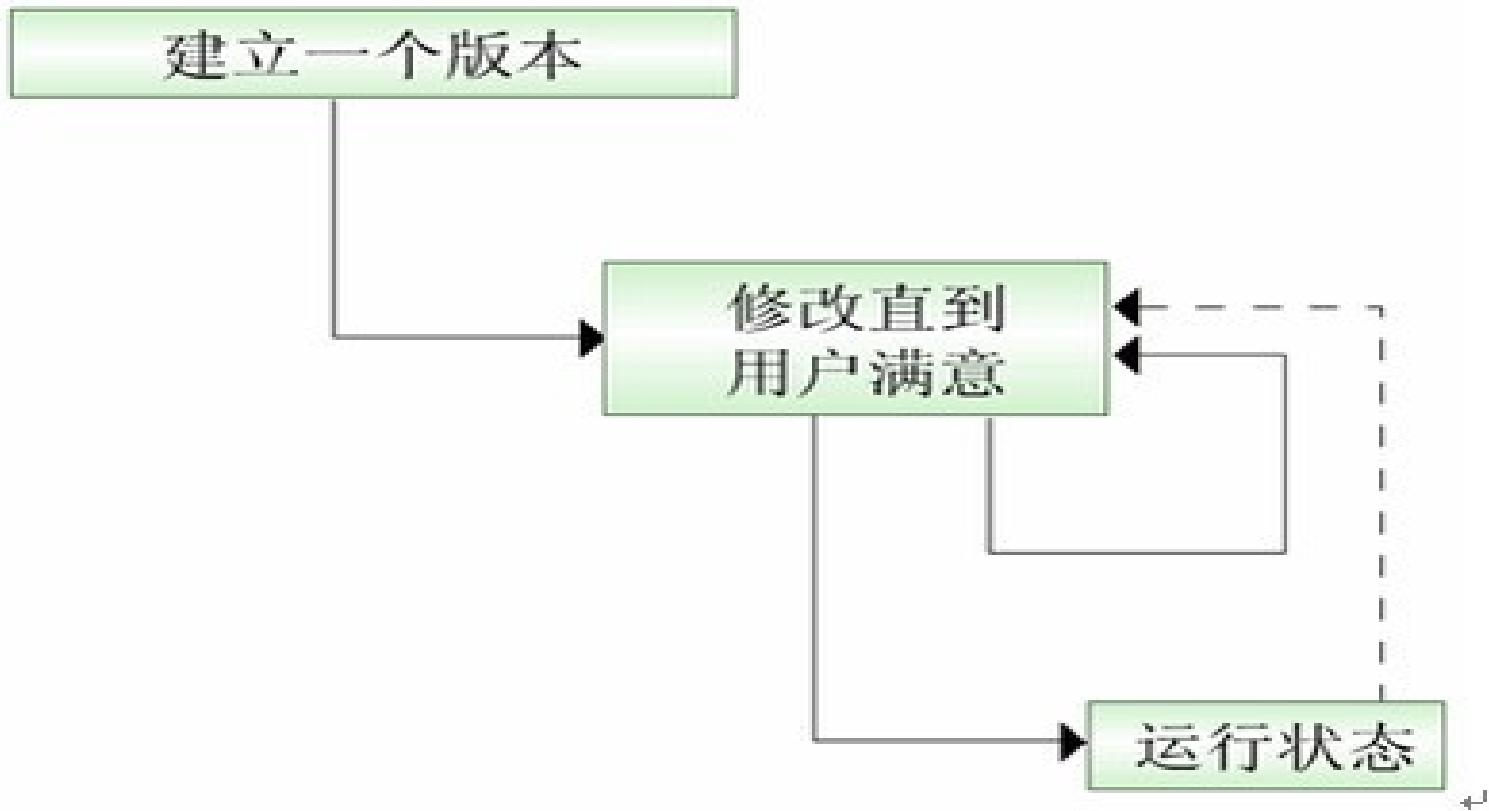
● 8. 软件维护

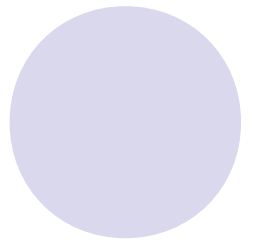
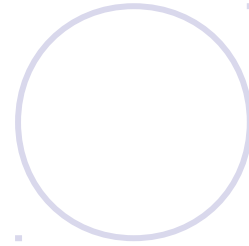
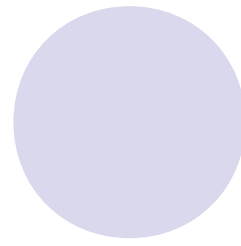
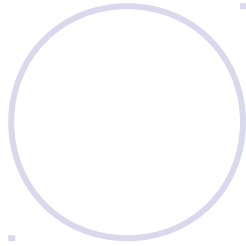
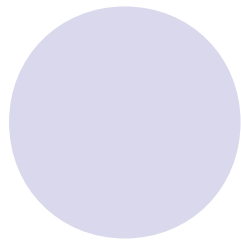
- 维护阶段的关键任务是，通过各种必要的维护活动使系统持久地满足用户的需要。
- 通常有 4 类维护活动：改正性维护，也就是诊断和改正在使用过程中发现的软件错误；适应性维护，即修改软件以适应环境的变化；完善性维护，即根据用户的要求改进或扩充软件使它更完善；预防性维护，即修改软件为将来的维护活动预先做准备。
- 每一项维护活动都实质上是经历了一次压缩和简化了的软件定义和开发的全过程。

软件开发模型

- 传统开发模型
 - 瀑布模型 (waterfall model)
 - 快速原型模型 (rapid prototype model)
- 演化开发模型
 - 增量模型 (incremental model)
 - 螺旋模型 (spiral model)
- 面向对象开发模型
 - 构件集成模型 (component integration model)
- 形式化开发模型
 - 转换模型 (transformational model)
 - 净室模型 (clean room model)

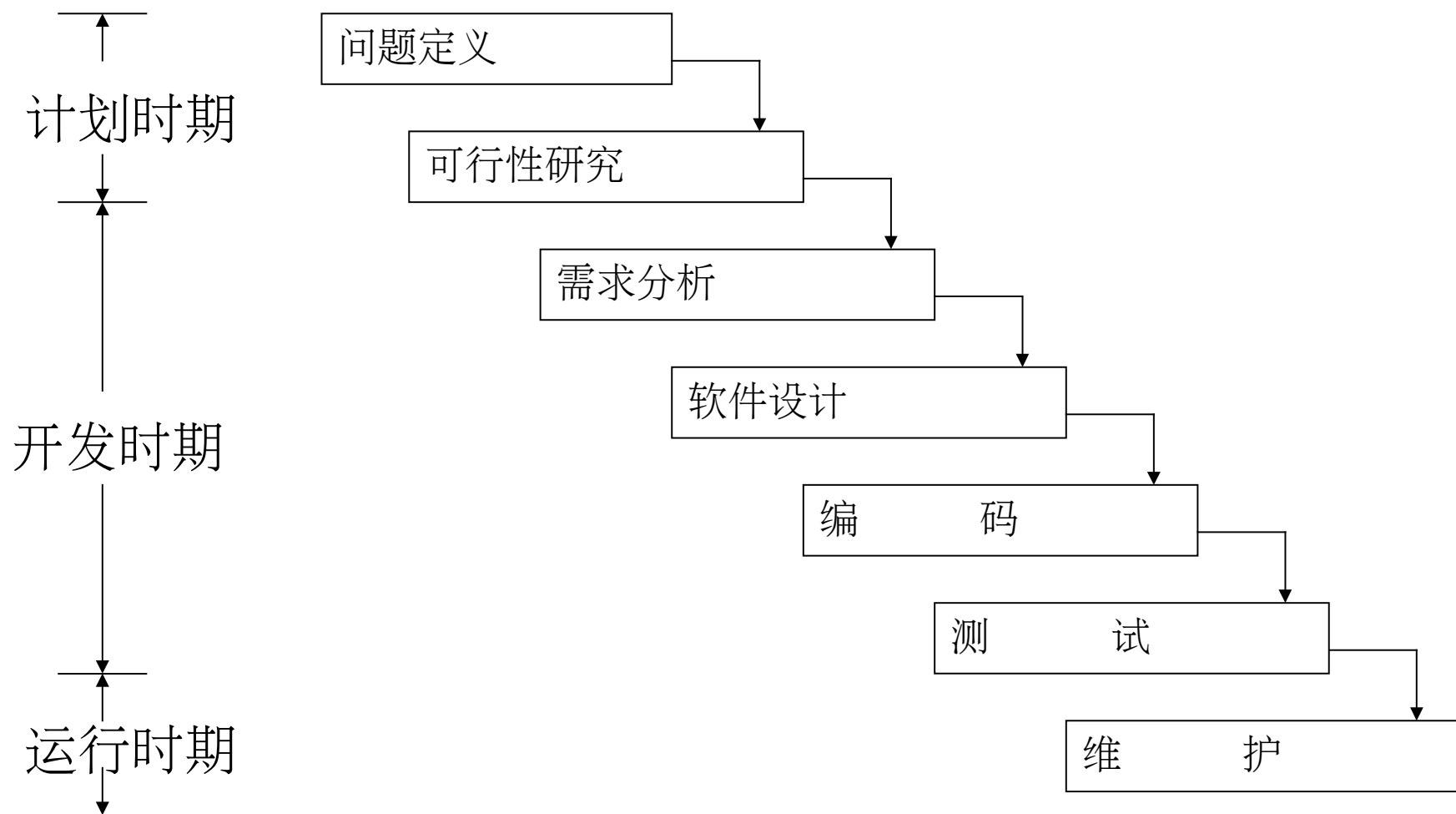
边做边改模型 (build-and-fix model)





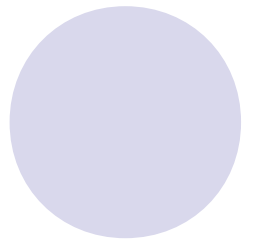
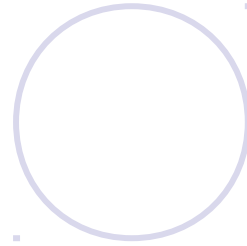
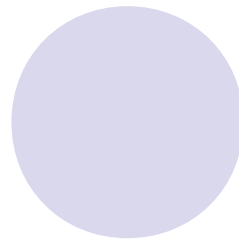
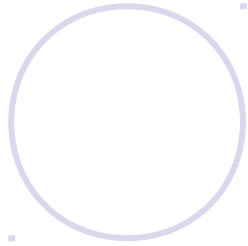
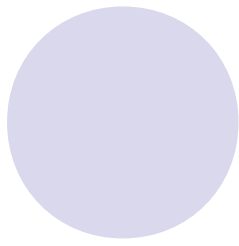
- 这种模型一般不会使用。 是不断的修正版本不断的供用户使用，如果出现错误或是新的需求又不断的修改代码 的过程。
- 缺点： 缺少规划和设计环节。忽略需求环节，风险大。周期长费用高。

瀑布模型



瀑布模型的特点

- 1. 阶段间具有顺序性和依赖性
 - ① 必须等前一阶段的工作完成之后，才能开始后一阶段的工作；
 - ② 前一阶段的输出文档就是后一阶段的输入文档，因此，只有前一阶段的输出文档正确，后一阶段的工作才能获得正确的结果。



● 2. 推迟实现的观点

- 对于规模较大的软件项目来说，往往编码开始得越早最终完成开发工作所需要的时间反而越长。这是因为，前面阶段的工作没做或做得不扎实，过早地考虑进行程序实现，往往带来灾难性后果。
- 瀑布模型在编码之前设置了系统分析与系统设计的各个阶段，清楚地区分逻辑设计与物理设计，尽可能推迟程序的物理实现，是按照瀑布模型开发软件的一条重要的指导思想。

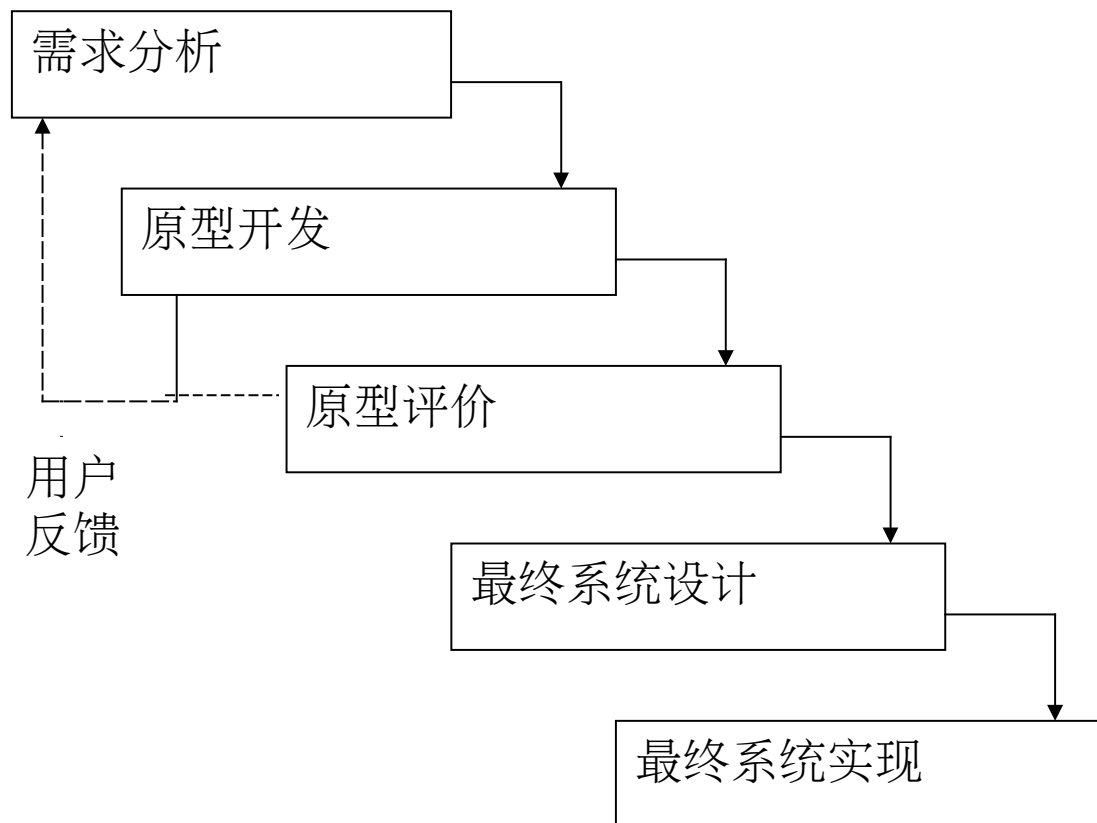
● 3. 质量保证的观点

○ 软件工程的基本目标是优质、高产。为了保证所开发的软件的质量，在瀑布模型的每个阶段都应坚持两个重要做法：

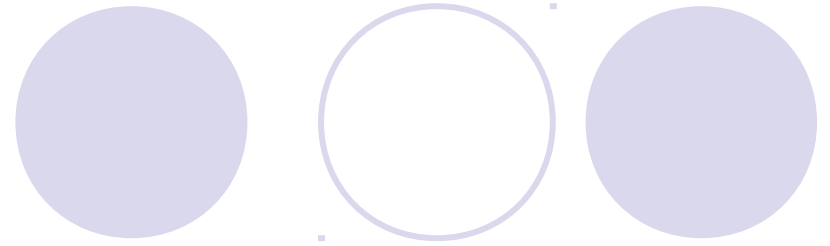
- (1) 每个阶段都必须完成规定的文档，没有交出合格的文档就是没有完成该阶段的任务。
- (2) 每个阶段结束前都要对所完成的文档进行评审，以便尽早发现问题，改正错误。

- 缺点：缺乏灵活性，太过于理想化。如果开发其中，客户难以明确需求，需求错误在后期就难以纠正。

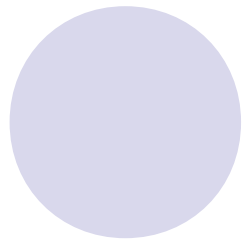
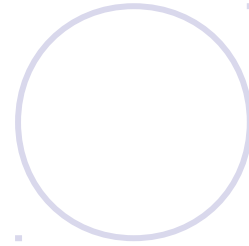
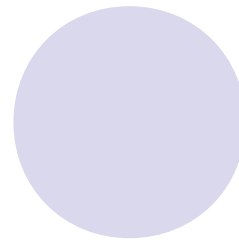
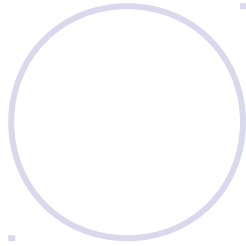
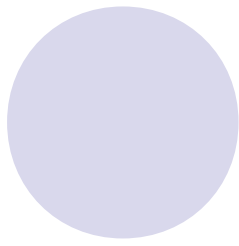
快速原型模型



快速原型模型



- 特点
 - 快速开发工具
 - 循环
 - 低成本
- 种类
 - 渐进型
 - 抛弃型

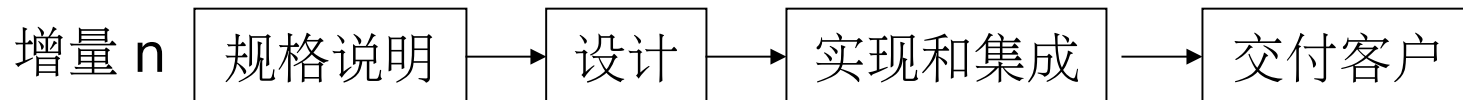
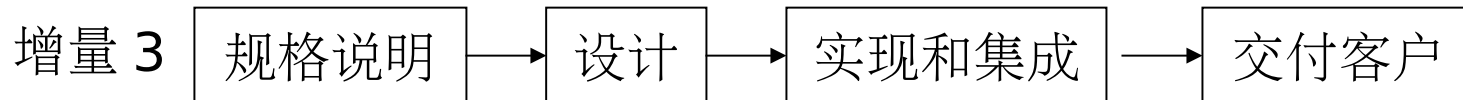
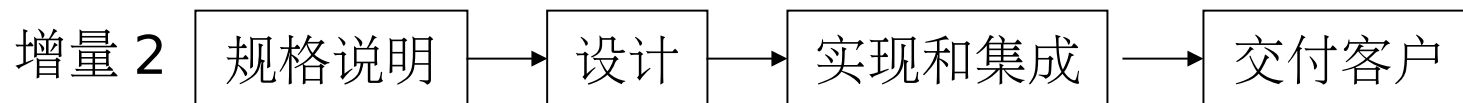
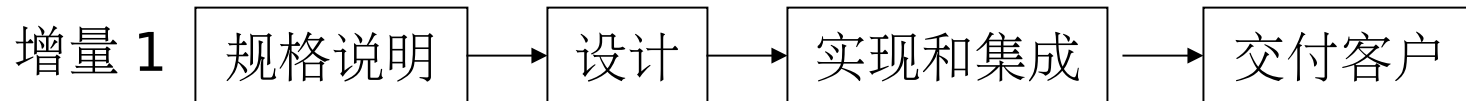


- 快速原型是运行的模型 在功能上等价产品的一个子集。适用于 用户需求不明确。一旦需求确定，原型将被抛弃。原型内部结构不重要，重要的是能迅速的构建原型，并迅速修改以满足客户的需求。
- 适用于：小型或是交互型式的系统。大型系统的某些部分，例如用户界面。 生命周期较短的。
- 特点：快速模型克服瀑布模型的特点，减少由于软件需求不明确带来的开发风险，具有显著的效果


增量模型

- 增量模型也称为渐增模型。
- 使用增量模型开发软件时，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。

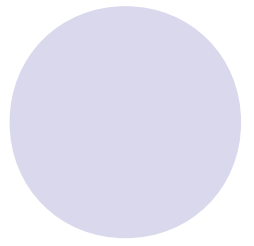
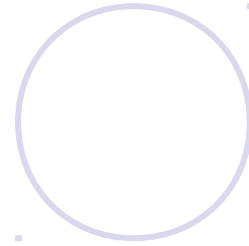
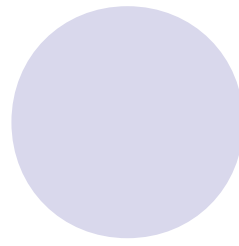
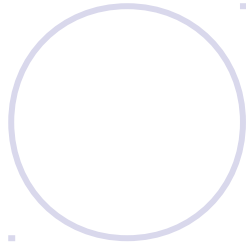
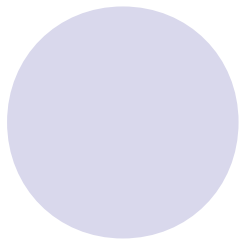
增量模型



增量模型



- 增量
 - 小而可用的软件
- 特点
 - 在前面增量的基础上开发后面的增量
 - 每个增量的开发可用瀑布或快速原型模型
 - 迭代的思路

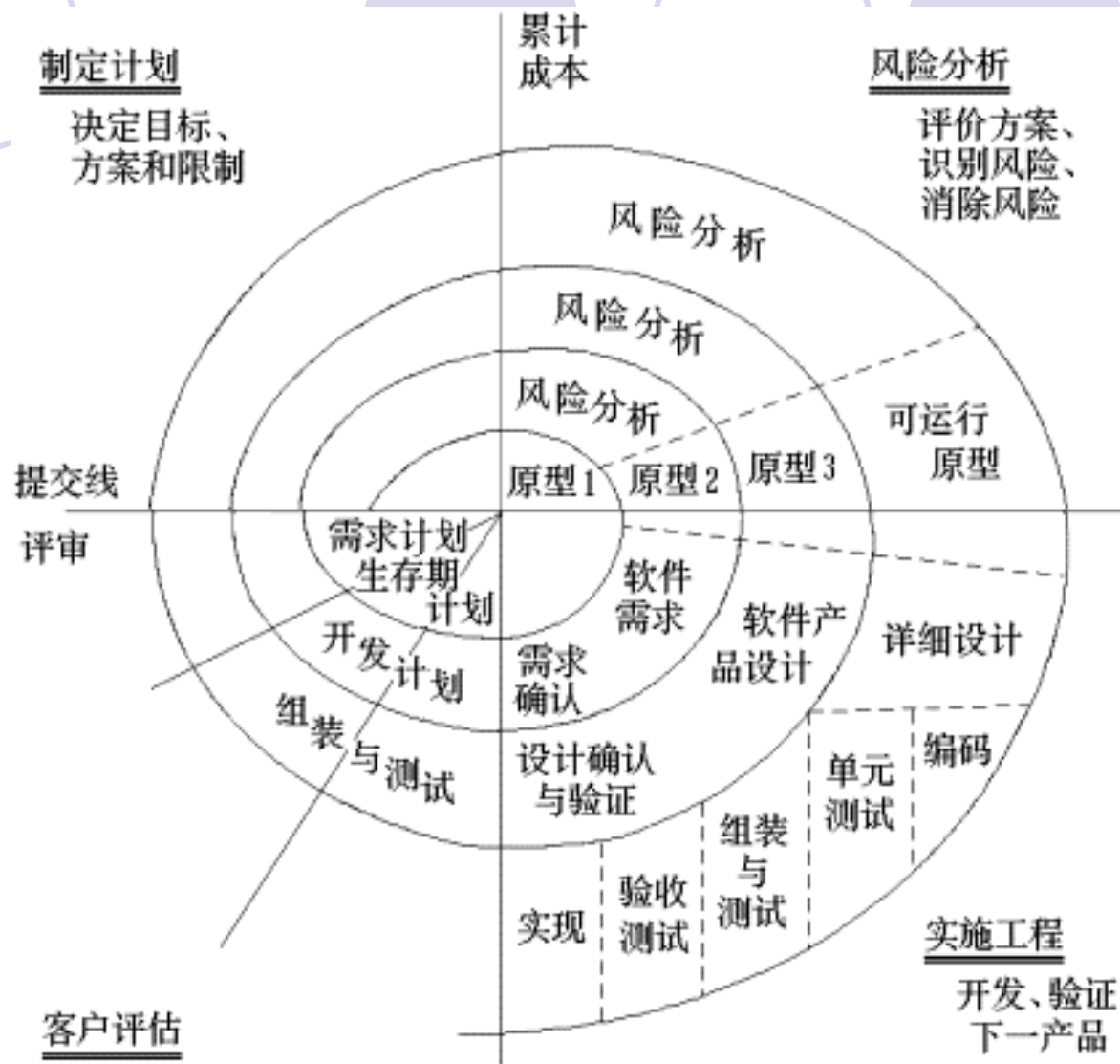


- **缩短时间**
- 开发人员与用户可以通过原型充分的交流；有利于用户的培训和开发的同步。加入构建必须不破坏已构造好的体系结构。模型的灵活性可以使其适应需求的变化，但也很容易退化为边做边改模型。

螺旋模型

- 螺旋模型的基本思想是，使用原型及其他方法来尽量降低风险。理解这种模型的一个简便方法，是把它看作在每个阶段之前都增加了风险分析过程的快速原型模型。
- 螺旋模型的主要优势在于，它是风险驱动的。

螺旋模型



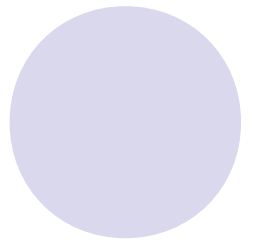
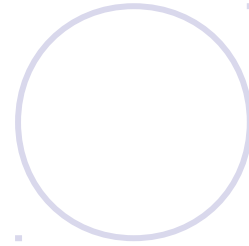
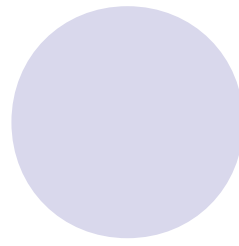
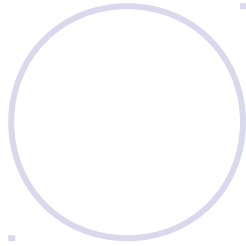
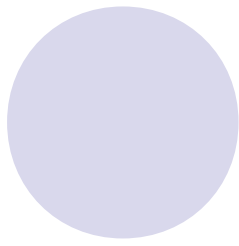
螺旋模型

- 特点

- 瀑布模型 + 快速原型 + 风险分析
- 迭代过程

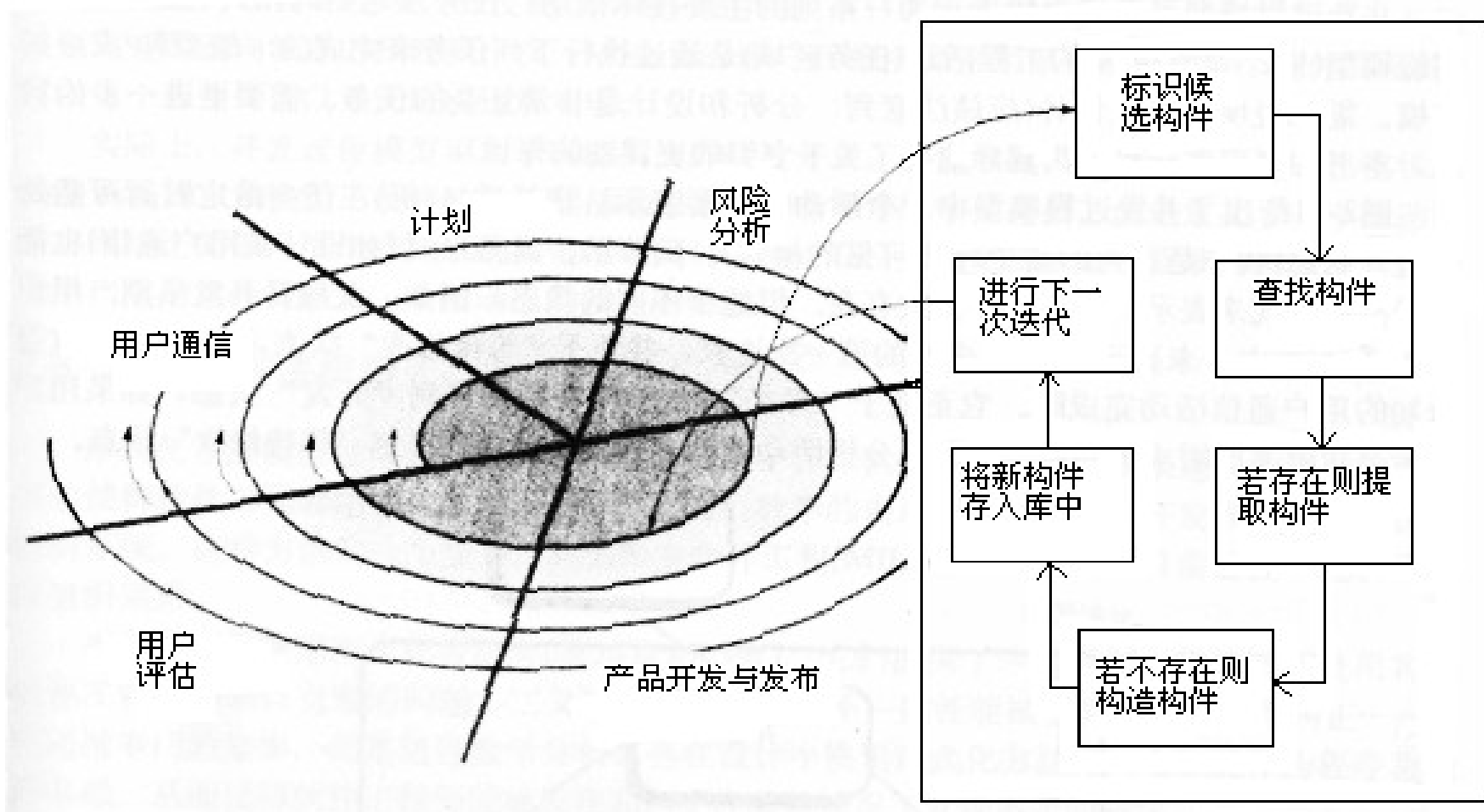
- 一个螺旋式周期

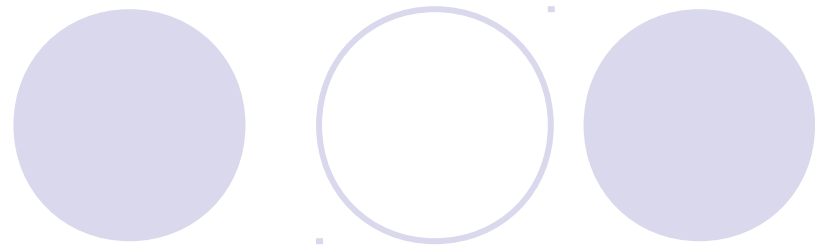
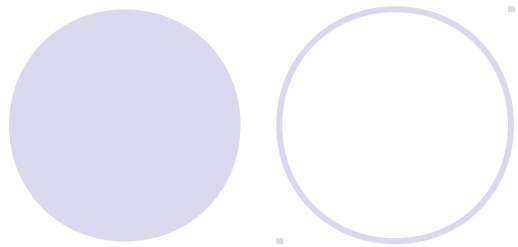
- 确定目标，选择方案，选定完成目标的策略
- 风险角度分析该策略
- 启动一个开发阶段
- 评价前一步的结果，计划下一轮的工作



- 主要是分为如下的活动：
- 制定计划：明确软件的目标，选定实施方案，弄清项目开发的限制条件。
- 风险分析：分析评估所选方案，考虑如何识别和消除风险。
- 实施工程：实施软件开发和验证
- 客户评估：评估开发工作，提出修正建议，指定下一步计划。

构件集成模型





- 软件构件模型是关于开发可重用软件构件和构件之间相互通信的一组标准的描述。通过重用已有的软构件，使用构件对象模型的开发软件者可以像搭积木一样快速构造应用程序。这样不仅可以节省时间和经费，提高工作效率，而且可以产生更加规范、更加可靠的应用软件。

CORBA

- ----CORBA 构件模型的底层结构为 ORB 。一个 CORBA 构件采用 IDL 进行描述。CORBA 提供了 IDL 到 C、C++、Java、COBOL 等语言的映射机制 --IDL 编译器。IDL 编译器可以生成 Server 方的 Skelton 和 Client 方的 Stub 代码，通过分别与客户端和服务端程序的联编，即可得到相应的 Server 和 Client 程序。
- ----CORBA 同时提供了一系列的公共服务规范 --COSS，其中包括名字服务、永久对象服务、生命周期服务、事务处理服务、对象事件服务和安全服务等，它们相当于一类用于企业级计算的公共构件。此外，CORBA 还针对电信、石油等典型的应用行业提供了一系列的公共设施。
- ----CORBA 是一种语言中性的软件构件模型，可以跨越不同的网络、不同的机器和不同的操作系统，实现分布对象之间的互操作。

DCOM

- ----DCOM 是 Microsoft 与其他业界厂商合作提出的一种分布构件对象模型（Distributed Component Object Model），其发展经历了一个相当曲折的过程。DCOM 起源于动态数据交换（DDE）技术，通过剪切 / 粘贴（Cut/Paste）实现两个应用程序之间共享数据的动态交换。对象连接与嵌入 OLE 就是从 DDE 引伸而来的。
- ---- 随后，Microsoft 引入了构件对象模型 COM，形成了 COM 对象之间实现互操作的二进制标准。COM 规定了对象模型和编程要求，使 COM 对象可以与其他对象相互操作。这些对象可以用不同的语言实现，其结构也可以不同。基于 COM，微软进一步将 OLE 技术发展到 OLE2。其中，COM 实现了 OLE 对象之间的底层通信工作，其作用类似于 CORBA/ORB。不过此时的 COM 只能作用在单机 Wintel 平台上。在 OLE2 中，也出现了我们今天熟知的拖 - 放技术以及 OLE 自动化。
- ---- 同时，微软在 VB 中引入了可以嵌入任何可视构件的通用模型 VBX。VBX 的主要局限在于它并不是一个开放的结构，也没有为第三方软件开发商提供 VBX 集成的标准。最后，微软将上述思想集中在一起，以 COM 作为构件通信框架。VBX 也发展为 OLE 控件 OCX 的形式。DCOM 是 COM 在分布计算方面的自然延续，它为分布在网络不同节点的两个 COM 构件提供了互操作的基础结构，而所有以 OLE 为标志的技术如今也已挂上了 ActiveX 标志。

JAVA

- ----Java 的软件构件称为 **JavaBean**，或者简称 **Bean**。按照 **JavaSoft** 给出的定义，**Bean** 是能够在构造工具中进行可视化操作的可重用软件。**JavaBean** 的组件模型包含组件和容器两个基本要素，这一思想在 **ActiveX/DCOM** 技术中同样存在。作为一种典型的组件模型，**JavaBean** 具有属性、方法、事件、自我检查、定制和永久性等 6 个方面的特征。其中前 3 种特征（属性、方法、事件）是面向对象的组件必须满足的基本要求，属性和方法保证 **Bean** 成为一个对象，而事件可以描述组件之间的相互作用以及组件与容器之间相互感兴趣的事情。通过事件的生成、传播和处理，构件相互之间关联在一起，共同完成复杂的任务。后三种特征（自我检查、定制和永久性）主要侧重于对 **JavaBeans** 组件性质的刻画

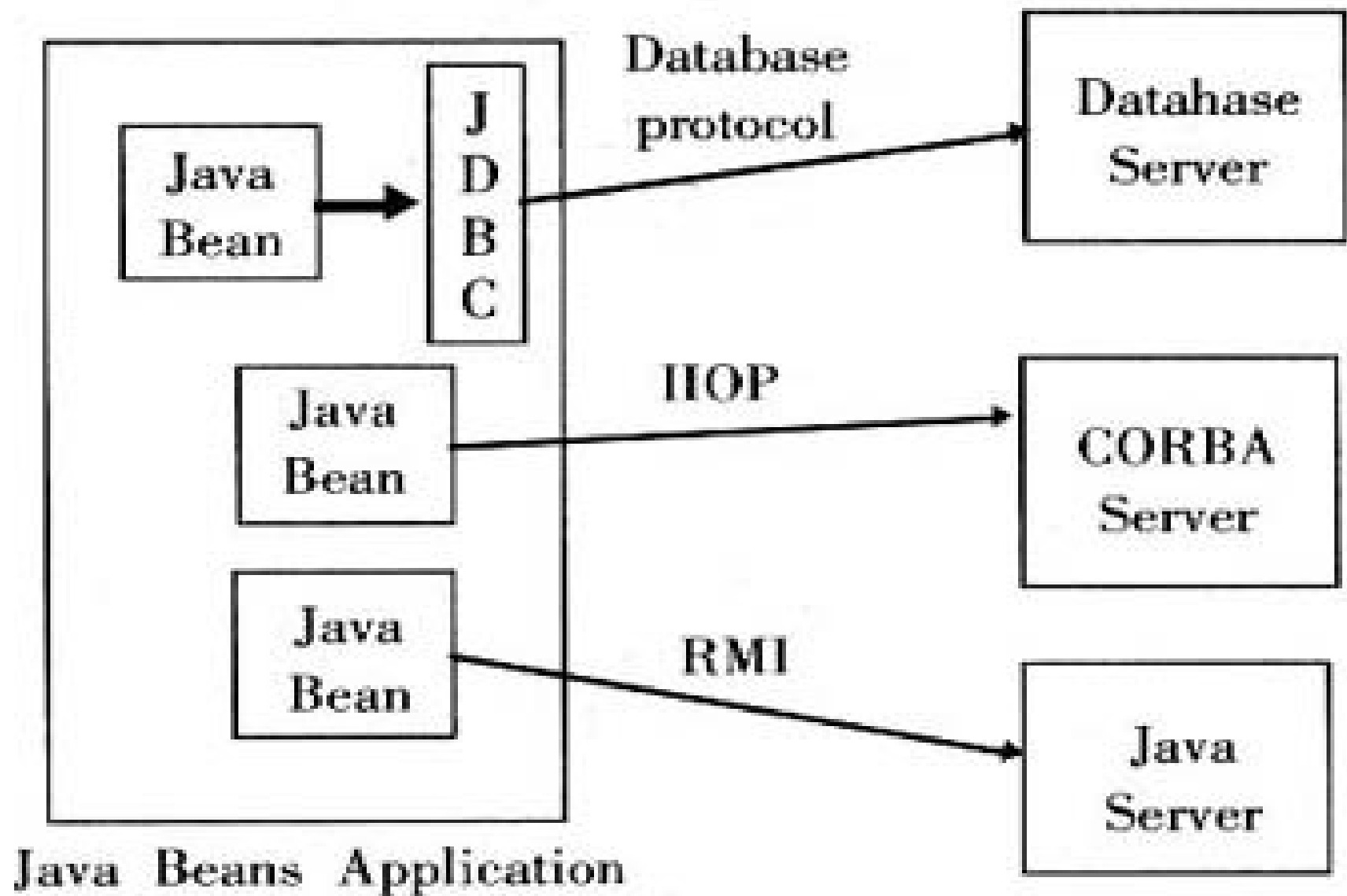


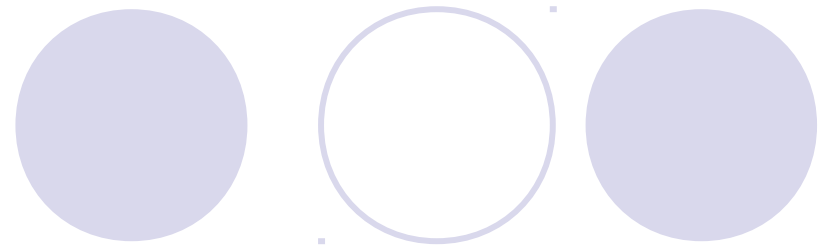
图 1 Java 三种网络访问机制

Java



- ----**JavaBean** 构件的本地活动是在与其容器相同的地址空间内进行的。在网络上，**JavaBean** 构件可以以三种方式进行活动：
- **JDBC** 使 **Bean** 构件能够访问 **SQL** 数据库。**Bean** 可以实现给定数据库中的表操作，完成相应的业务逻辑；
- **JavaRMI**（远程方法调用）使分布在网络不同地址上的两个构件之间实现互操作。构件之间的调用方式采用经典的 **Client/Server** 计算模型；
- **JavaIDL** 是一个 **Java** 版的 **CORBA/ORB**。通过 **JavaIDL** 可以实现一个 **JavaBean** 和一个 **CORBA** 服务之间的互操作。基于 **JavaIDL** 的 **Java** 构件互操作模型完全等同于 **CORBA** 的思想，只不过具体的编程语言采用 **Java**，而 **CORBA/ORB** 选择了 **JavaIDL**。

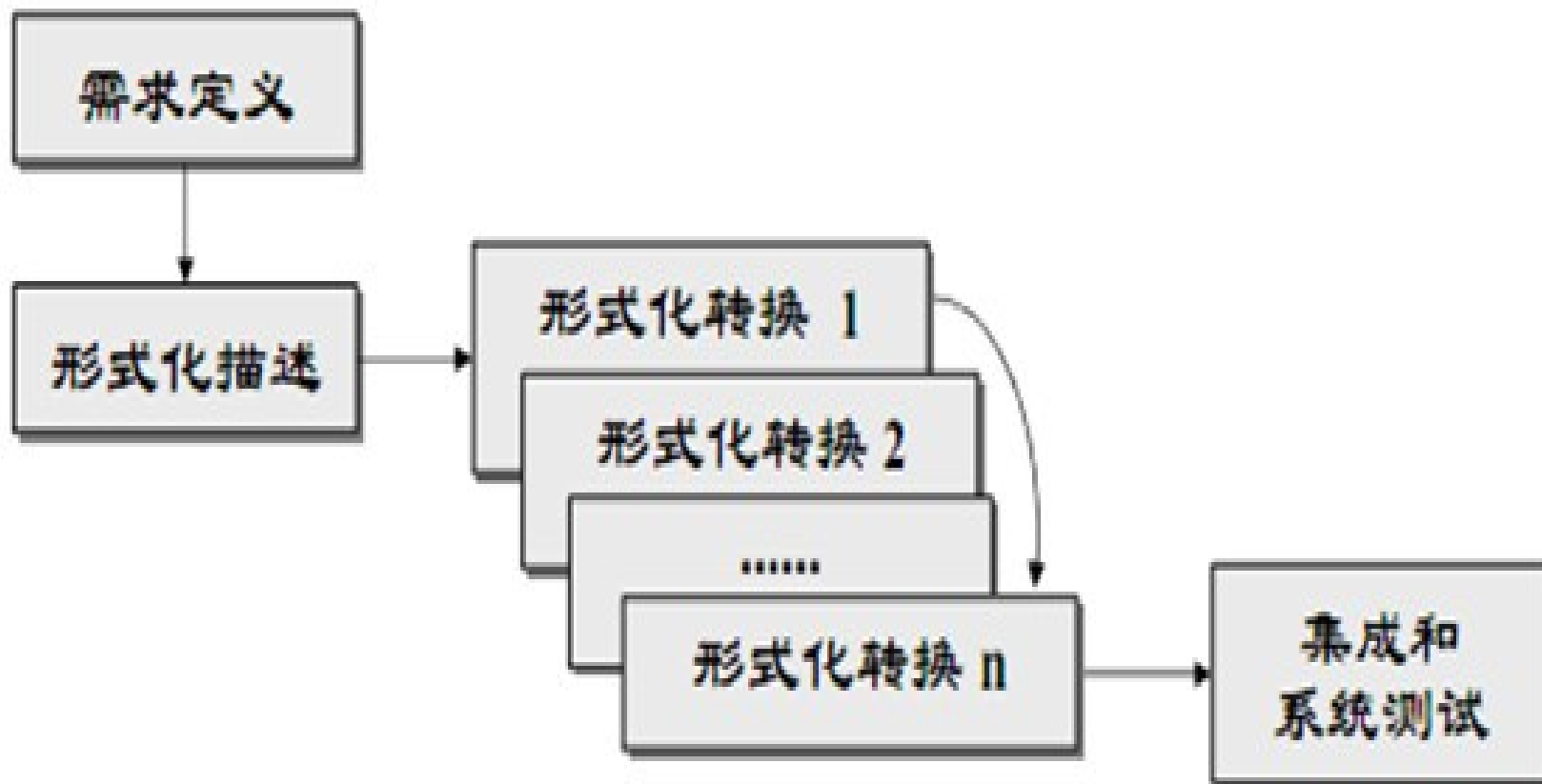
构件集成模型

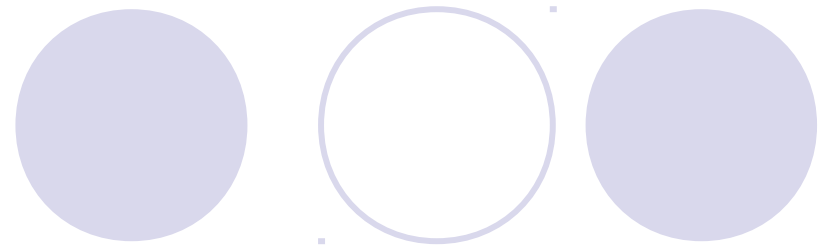
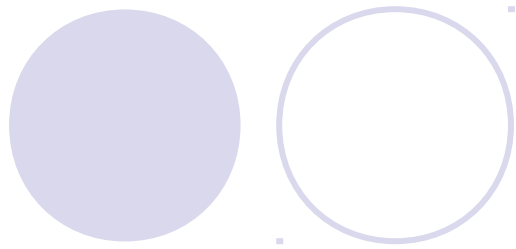


- 特点

- 面向对象
- 基于构件库
- 融合螺旋模型特征
- 支持软件开发的迭代方法
- 基于组建的开发技术是使用技术是使用可重用的组建或是商业组建建立复杂的软件系统。
- 组建开发急速的两个重要的因素。
- 基于组建的软件体系。 基于组建的开发过程。
-
- 优点：充分体现软件复用的思想，实现快速的交付。

形式化模型方法





- 形式化模型方法采用数学方法将系统描述转化为可执行的程序
- 适用：适用于对于那些安全性和保密性要求极高的软件系统，这些需要在投入运行前进行验证。
- 优点：犹如教学方法具有的严密和准确性，形式化方法开发过程中所交付的软件信息具有较少的缺陷和较高的安全性。
- 缺点：费时费力，开发人员需要经过特殊的训练。
。难以进行形式化描述。

几种模型的比较:

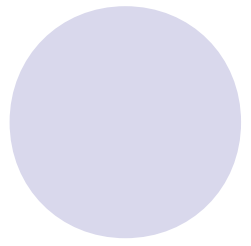
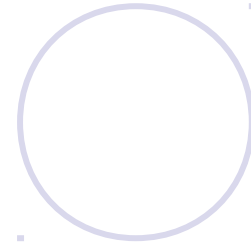
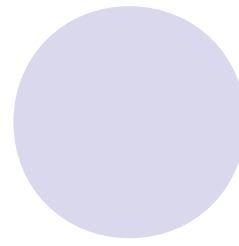
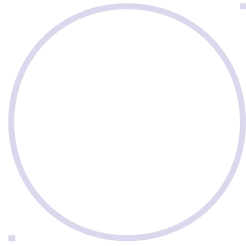
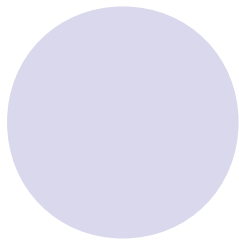
模型	优点	缺点
瀑布模型	文档驱动	系统可能不满足客户的需求
快速原型模型	关注满足客户需求	可能导致系统设计差、效率低，难遇维护
增量模型	开发早期反馈及时，易于维护	需要开放式体系结构，可能会设计差、效率低
螺旋模型	风险驱动	风险分析人员需要有经验且经过充分训练。

软件开发模型——小结

- 软件开发模型是不断发展的
- 各种软件开发模型各有优缺点
- 选用时不必拘泥与某种模型
- 可组合多种模型
- 也可根据实际创建新的模型

CASE 工具

- 计算机辅助软件工程 (Computer Aided Software Engineering)
 - 项目管理工具
 - 需求分析工具
 - 编程环境
 - 软件测试工具



- **CASE 技术**：一种软件技术。为软件的开发、维护和项目管理提供一种自动化工程原理，包括自动化结构化方法和自动化工具。
- **CASE 工具**：一种软件工具。对某个具体的软件生命周期的任务实现自动化（至少是某一部分的自动化）。
- **CASE 系统**：一种集成的 **CASE 具**。使用一个公共的用户接口，并在一个公共的计算机环境下运行。
- **CASE 具箱**：一组集成的 **CASE 具**。用来协同工作以实现某个软件生命周期的阶段或某类具体的软件作业的自动化（或部分地实现自动化）。
- **CASE 工作台**：一组集成的 **CASE 具**，被设计用来协同工作以实现整个软件生存期的自动化（或提供自动化的辅助手段），包括分析、设计、编码和测试。
- **CASE 方法**：一种“可自动化”的结构化方法。为软件的开发和维护的整个过程或某个方面定义了一个类似工程的方法。

- **CASE** 工具主要包括：事务系统规划工具、项目管理工具、支撑工具、分析和设计工具、程序设计工具、测试工具、原型建造工具、维护工具、框架工具。
- 目前 **CASE** 的标准是 **UML**，最常用的 **CASE** 工具是 **Rational Rose**、**Sybase PowerDesigner**、**Microsoft Visio**、**Microsoft Project**、**Enterprise Architect**、**MetaCase**、**ModelMaker**、**Visual Paradigm** 等。这些工具集成在统一的 **CASE** 环境中，就可以通过一个公共接口，实现工具之间数据的可传递性，连接系统开发和维护过程中各个步骤，最后，在统一的软、硬件平台上实现系统的全部开发工作。