

第1章 软件工程学概述

目录

- 1.1 软件与软件危机
- 1.2 软件工程
- 1.3 软件生命周期
- 1.4 软件过程

1.1 软件与软件危机

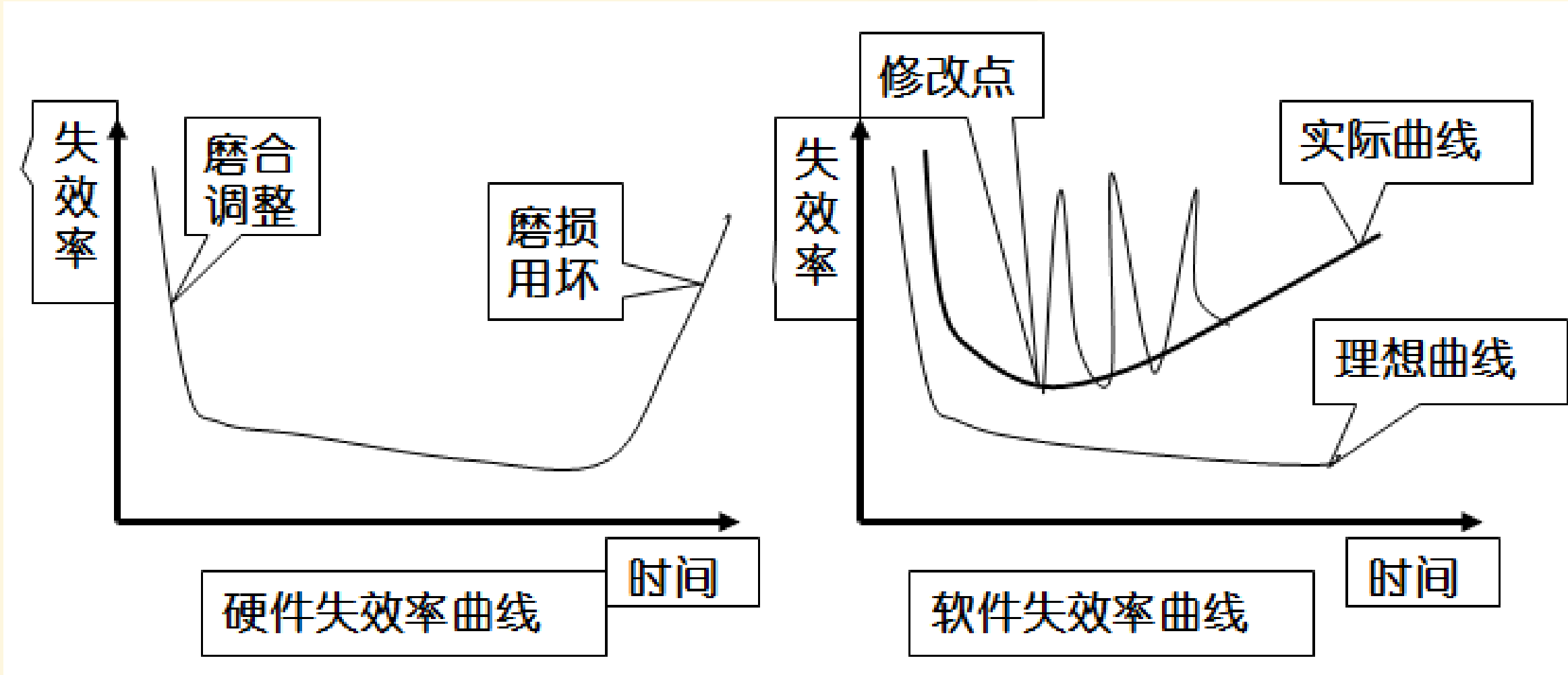
什么是软件

- 1983年IEEE关于“软件”的定义
 - 计算机程序、文档、以及运行程序必须的数据、方法、规则。
 - 方法和规则在文档中说明，在程序中实现。
- 简单地说：
 - 软件 = 程序 + 文档 + 数据

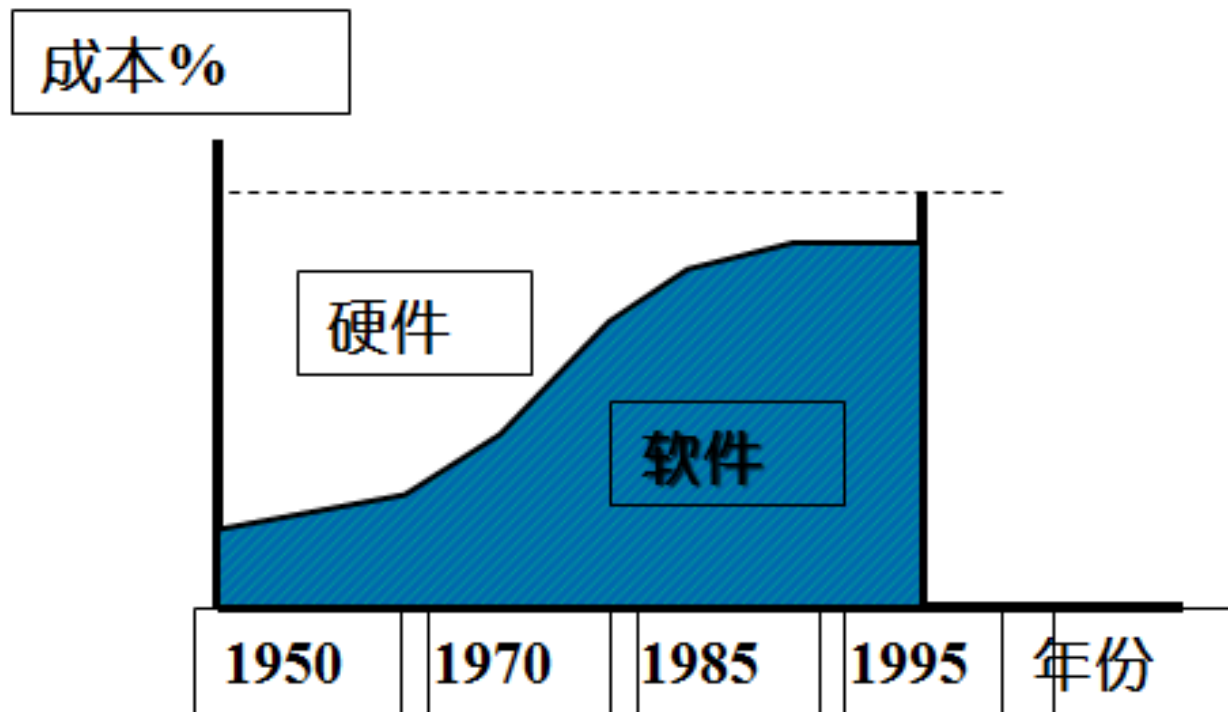
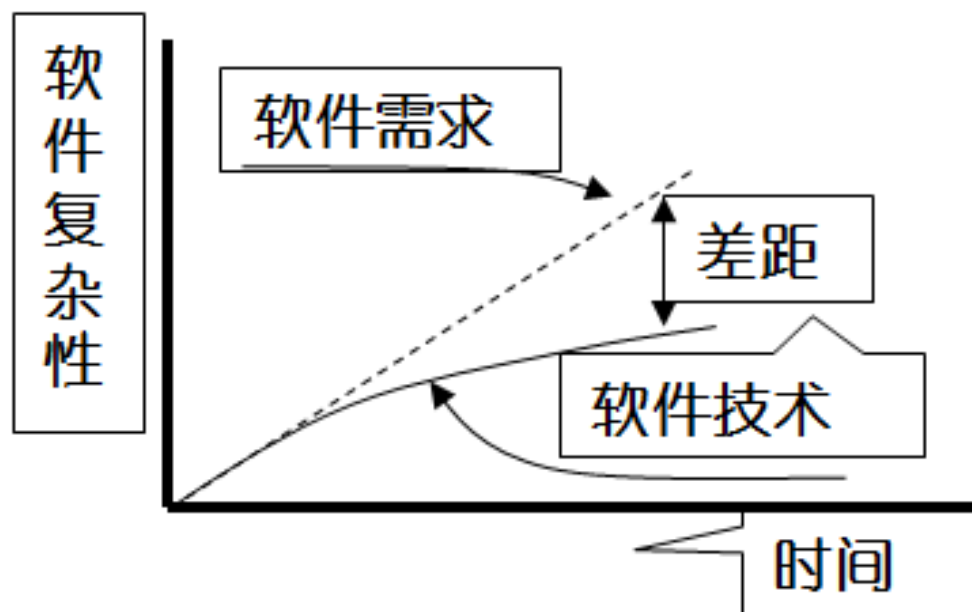
软件的特点

- 软件是一种逻辑实体，而不是具体的物理实体
- 软件的生产与硬件不同
- 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
- 在软件的运行和使用期间，经常被要求修改
- 软件的成本相当昂贵

硬件和软件的失效率（故障率）曲线



软件的成本



软件危机的产生

- 60年代中期以前：通用硬件相当普遍，软件却是为某个具体的应用而编写的
 - 个体化开发方法，缺乏文档资料
- 60年代中到70年代中：软件作坊
 - 沿用个体化开发方法，生产通用软件
 - 软件缺乏可维护性
- 软件危机：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。（正常、不正常运行软件都具有这种问题）

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；
2. 用户对完成的软件系统不满意的现象经常发生；
3. 软件产品的质量往往靠不住；
4. 软件常常是不可维护的；
5. 软件通常没有适当的文档资料；
6. 软件成本在计算机系统总成本中所占的比例逐年上升；
7. 软件开发生产率提高的速度跟不上计算机应用的发展趋势。

产生软件危机的原因

- 软件本身特点
- 软件开发与维护的方法不正确
 - 忽视软件需求分析
 - 认为软件开发就是写程序并使之运行
 - 轻视软件维护

解决软件危机的途径

1. 推广使用在实践中总结出来的开发软件的成功技术和方法，并研究探索更有效的技术和方法；
2. 开发和使用更好的软件工具；
3. 良好的组织管理措施。

软件工程学 and 程序设计方法学

为了解决软件危机产生的问题，软件工程与方法学逐渐形成，然后出现了两个相辅相成又各有侧重的学科：

- **软件工程学**：主要应用工程的方法和技术研究软件开发与维护的方法、工具和管理的一门交叉学科。（侧重实践）
- **程序设计方法学**：主要应用数学的方法研究程序的性质以及程序设计的理论和方法的学科。（侧重理论）

1.2 软件工程

软件工程的定义

- 1968年NATO会议：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：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；（2）研究（1）中提到的途径。

软件工程的本质特性

1. 软件工程关注于大型程序的构造；
2. 软件工程的中心课题是控制复杂性；
3. 软件经常变化；
4. 开发软件的效率非常重要；
5. 和谐地合作是软件开发的关键；
6. 软件必须有效地支持它的用户；
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创造产品。

软件工程的基本原理

1. 用分阶段的生命周期计划严格管理；
2. 坚持进行阶段评审；
3. 实行严格的产品控制；
4. 采用现代程序设计技术；
5. 结果能清楚地审查；
6. 开发小组的人员应该少而精；
7. 承认不断改进软件工程实践的必要性。

Adding manpower to a late software project makes it later. (Fred Brooks, "The Mythical Man-Month")

软件工程方法学

- 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 ）。
- 软件工程方法学的3要素：方法、工具和过程
- 传统方法学
 - 也称为生命周期方法学或结构化范型
 - 结构化分析方法，结构化设计方法等
- 面向对象方法学
 - 把数据和对数据的操作紧密结合起来的方法
 - 对象 + 类 + 继承 + 基于消息的通信

1.3 软件生命周期

- 软件生命周期：指软件从提出到最终被淘汰的这个存在期

软件生命周期的组成

软件定义	问题定义
	可行性研究
	需求分析
软件开发	总体设计（概要设计）
	详细设计
	编码和单元测试
	综合测试
运行维护	运行维护

1.4 软件过程

- 软件过程：为了获得高质量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。
- 软件过程模型，也叫软件生命周期模型

瀑布模型

1. 阶段间具有顺序性和依赖性
2. 推迟实现的观点
3. 质量保证的观点

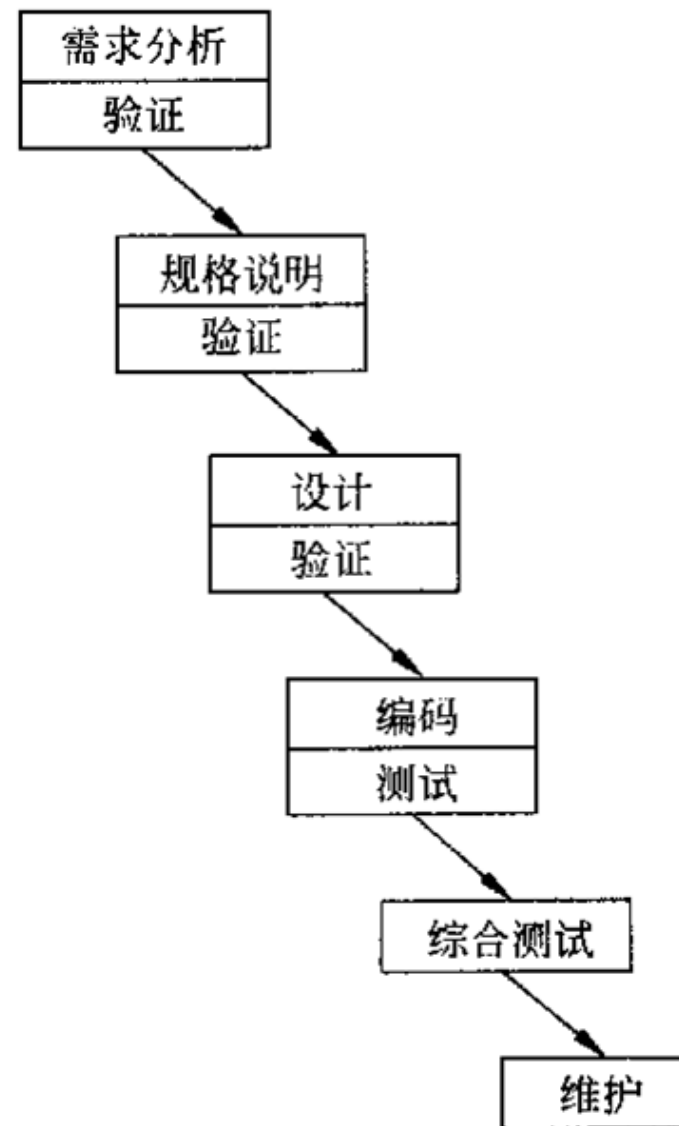


图 1.2 传统的瀑布模型

带反馈的瀑布模型

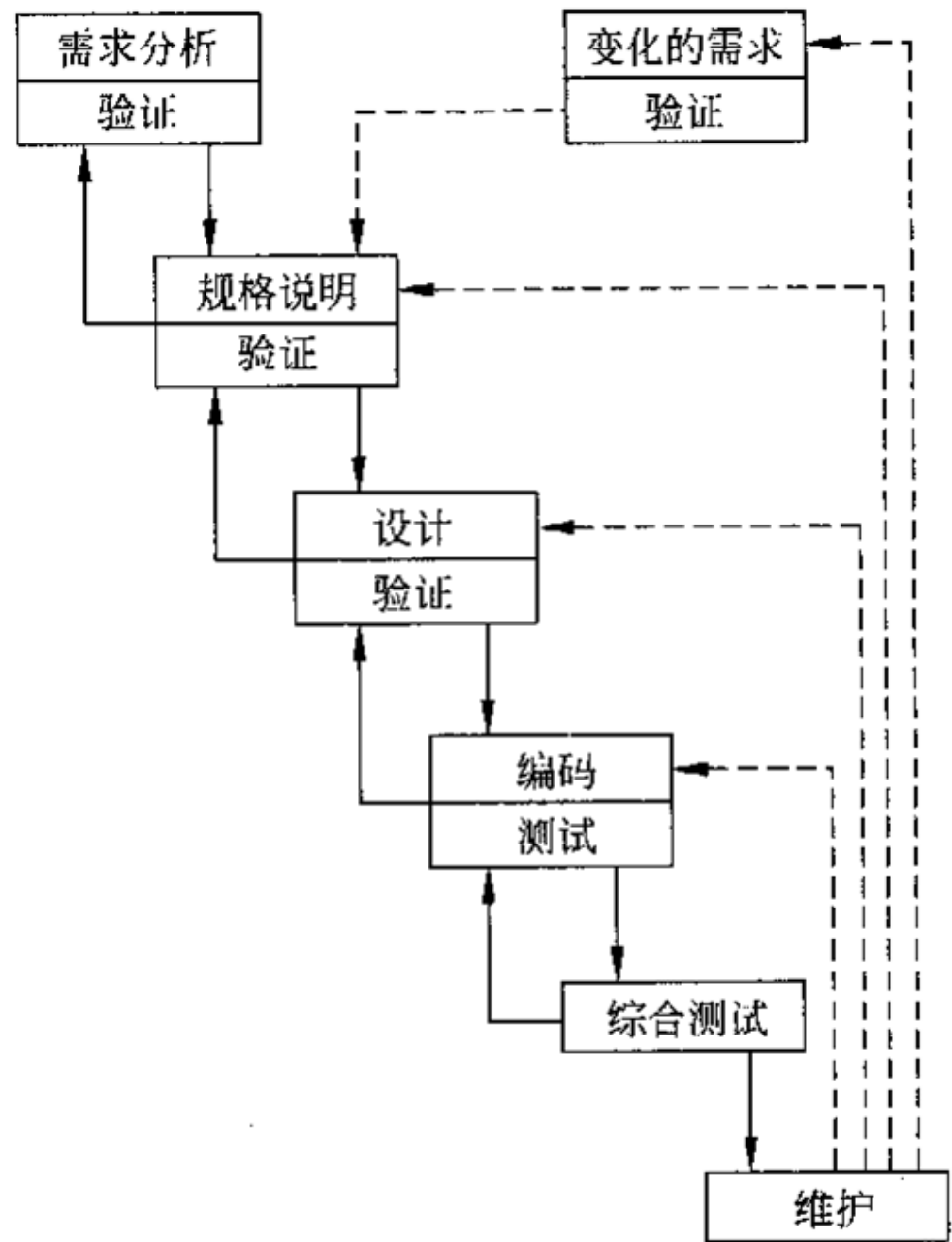


图 1.3 实际的瀑布模型

瀑布模型的优缺点

- 优点：采用规范的方法；严格规定每个阶段提交的文档；要求每个阶段交出的产品必须经过验证。
- 缺点：在可运行的软件交付之前，用户只能通过文档来了解产品；开发人员与用户缺乏有效的沟通。
- 适用场合：需求比较明确和稳定。

快速原型模型

prototype: 原型

- 优点：不带反馈环，基本上是线性顺序进行。
- 适用场合：（1）用户需求不确定；（2）开发人员对关键技术缺乏把握。
- 对原型的处理：（1）抛弃；（2）集成到正式系统中。

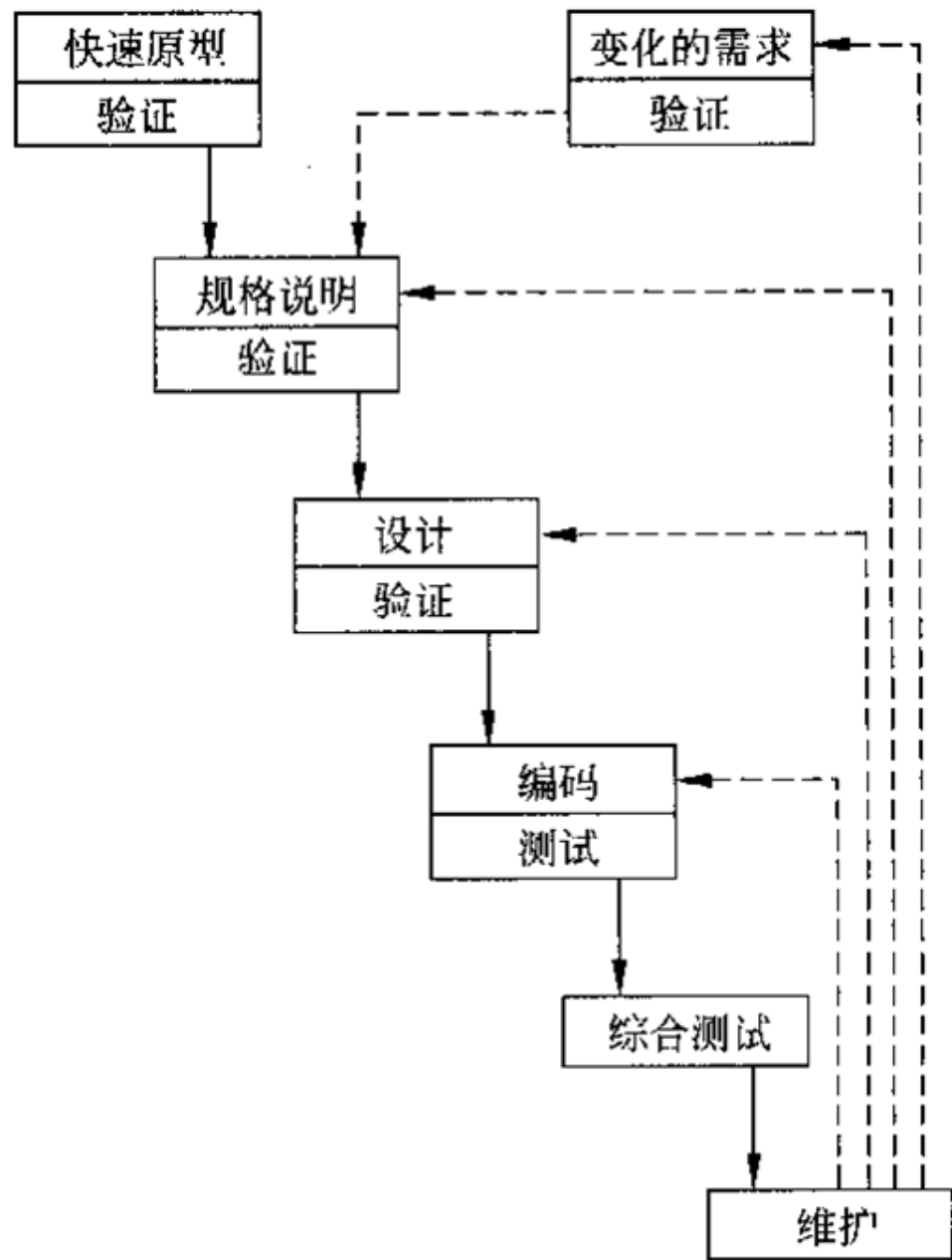


图 1.4 快速原型模型

增量模型

- 优点：能较短时间内提交可完成部分工作的产品；可以使用户有充裕的时间学习和适应新产品。
- 适用场合：用户需要尽早得到可运行的产品。

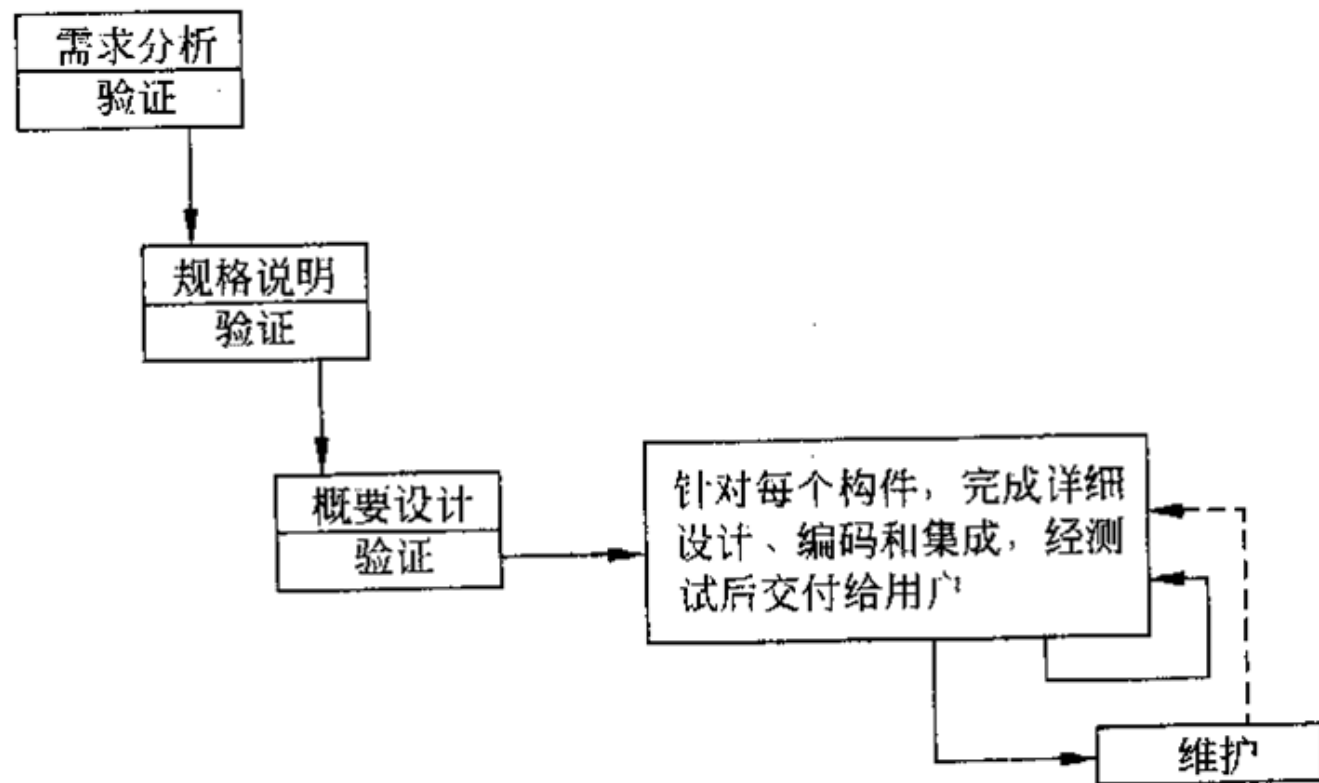


图 1.5 增量模型

螺旋模型

多次迭代，每次迭代产出不同阶段的产品（比如原型、正式软件的不同版本），在每次迭代之前都增加风险分析。

- 适用场合：（1）比较大型的软件；
（2）风险控制比较重要。

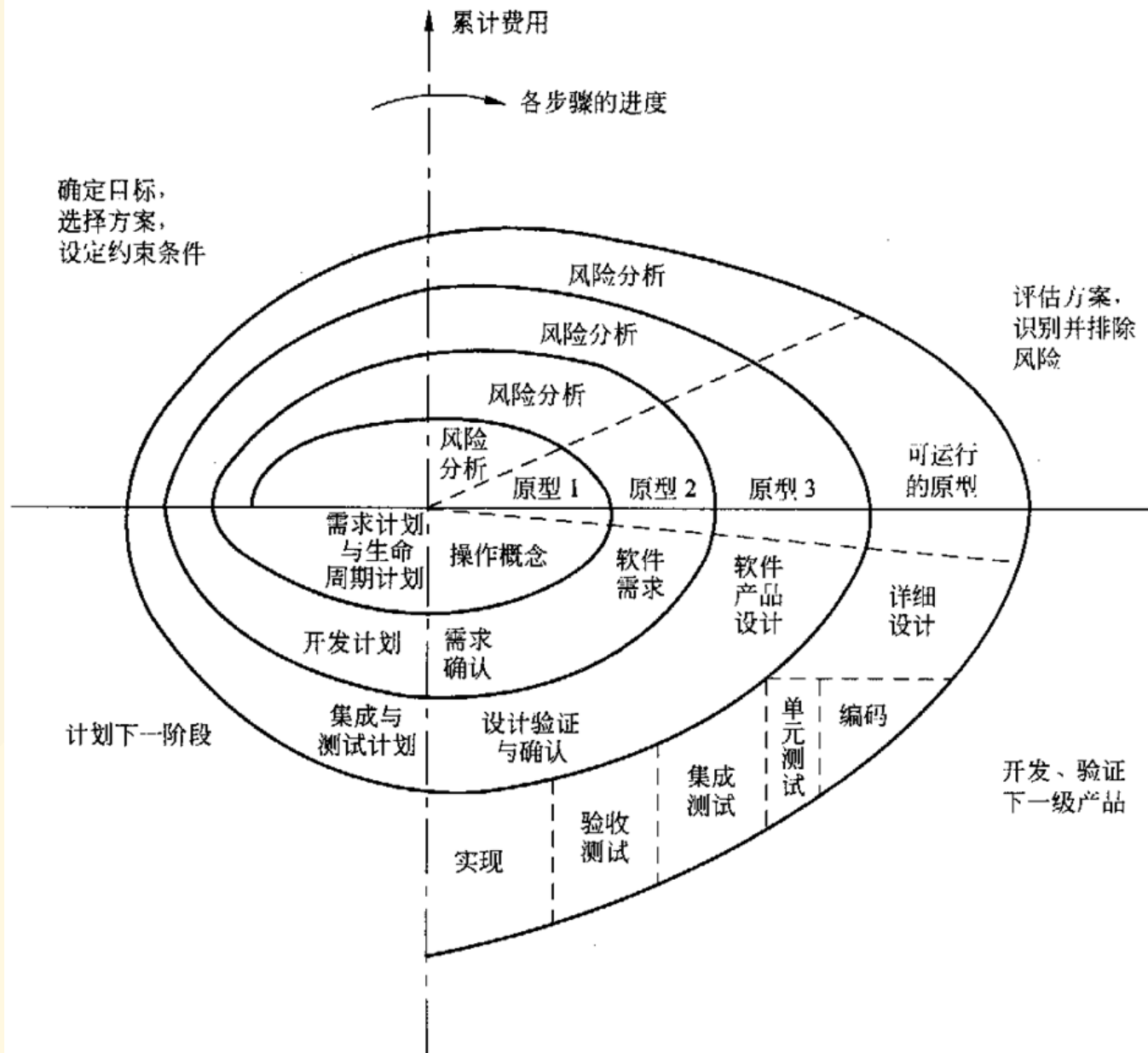
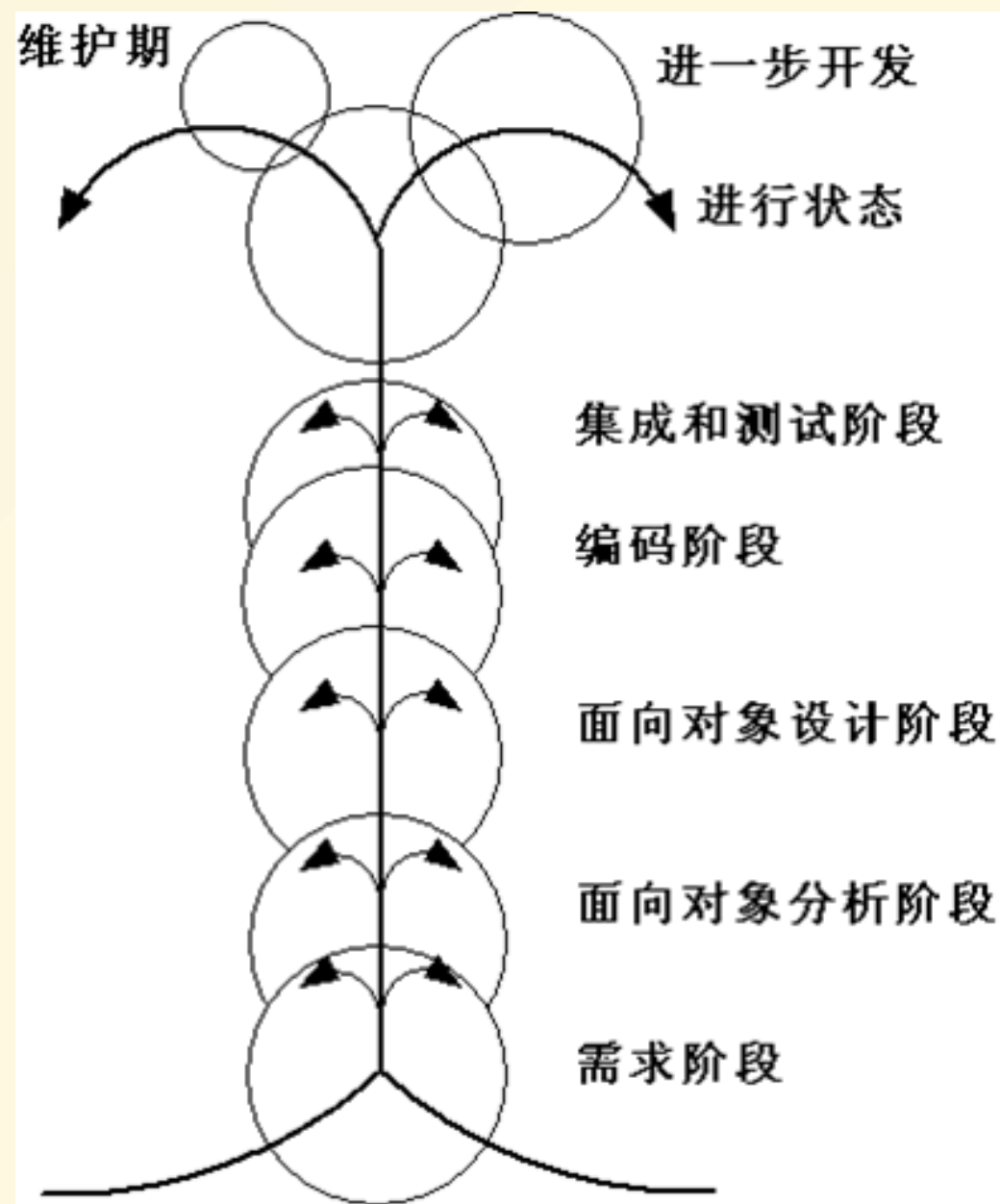


图 1.8 完整的螺旋模型

喷泉模型

- 面向对象生命周期模型，体现迭代和无缝特性。
 - 迭代：求精，系统某部分被逐步细化，相关功能在每次迭代中逐渐加入。
 - 无缝：分析、设计、编码各阶段间不存在明显边界。



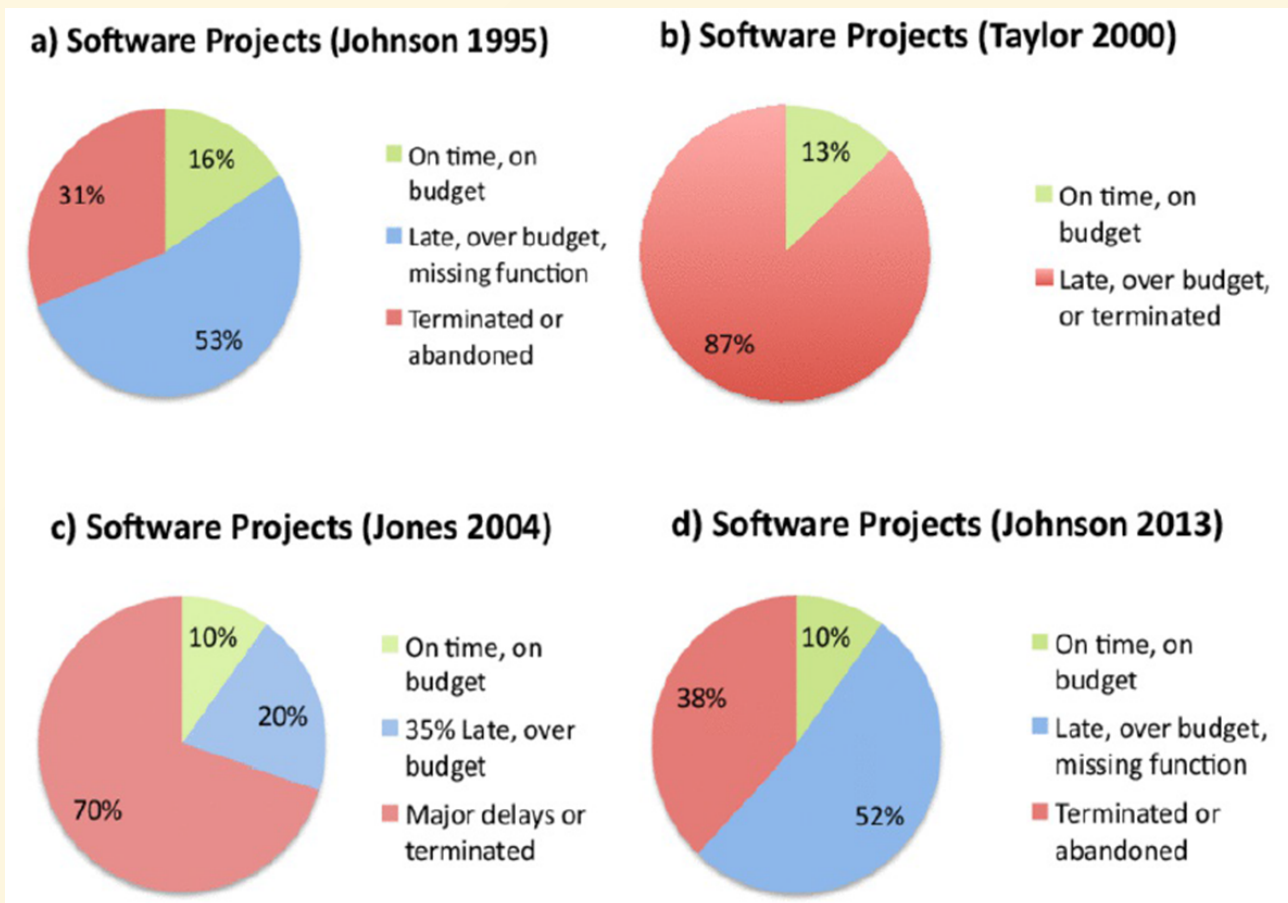
喷泉模型的优缺点

- 优点：无缝，可同步开发，提高开发效率，节省开发时间，适应面向对象软件
- 缺点：可能随时加各种信息、需求与资料，需严格管理文档，审核的难度加大。

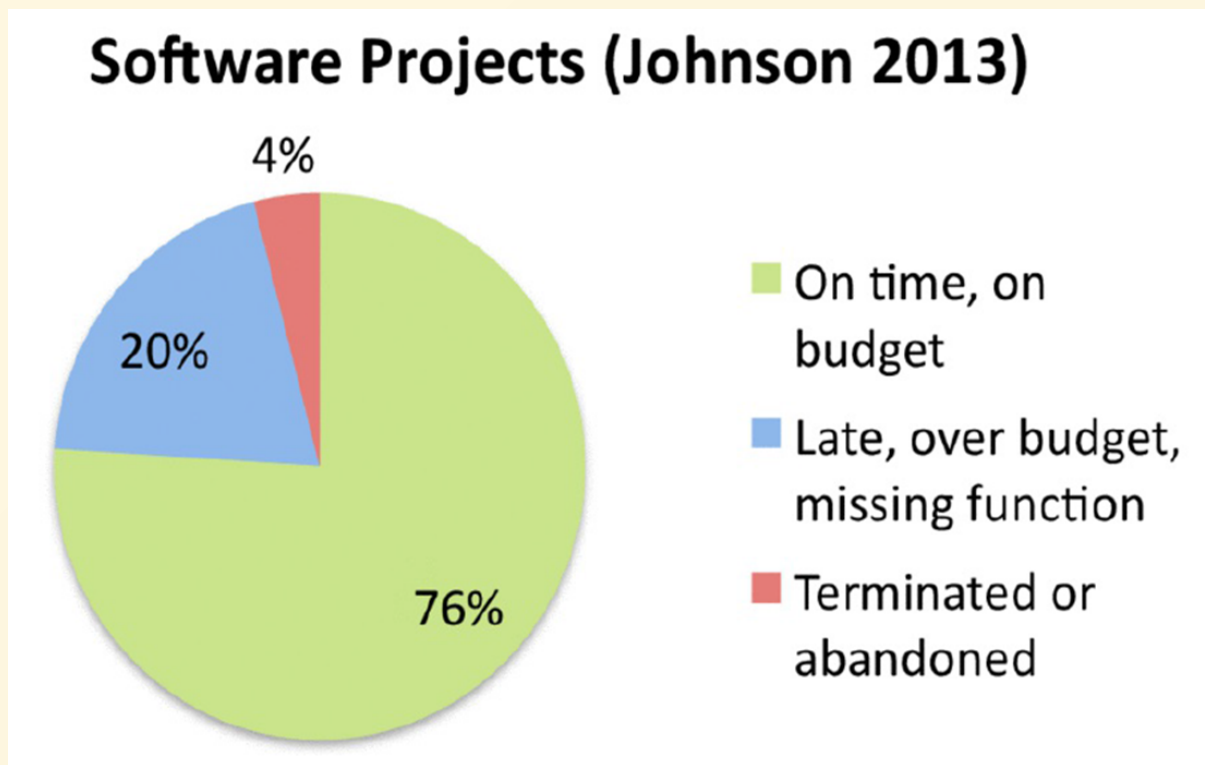
敏捷过程与极限编程

- 敏捷过程：具有高效、快速响应变化的开发过程。
 - （1）个体和交互胜过过程和工具；
 - （2）可以工作的软件胜过面面俱到的文档；
 - （3）客户合作胜过合同谈判；
 - （4）响应变化胜过遵循计划。
- 极限编程
敏捷过程中最著名的一种，指把好的开发实践运用到极致，多应用于软件需求模糊的场合。

软件开发实践调查（传统过程）

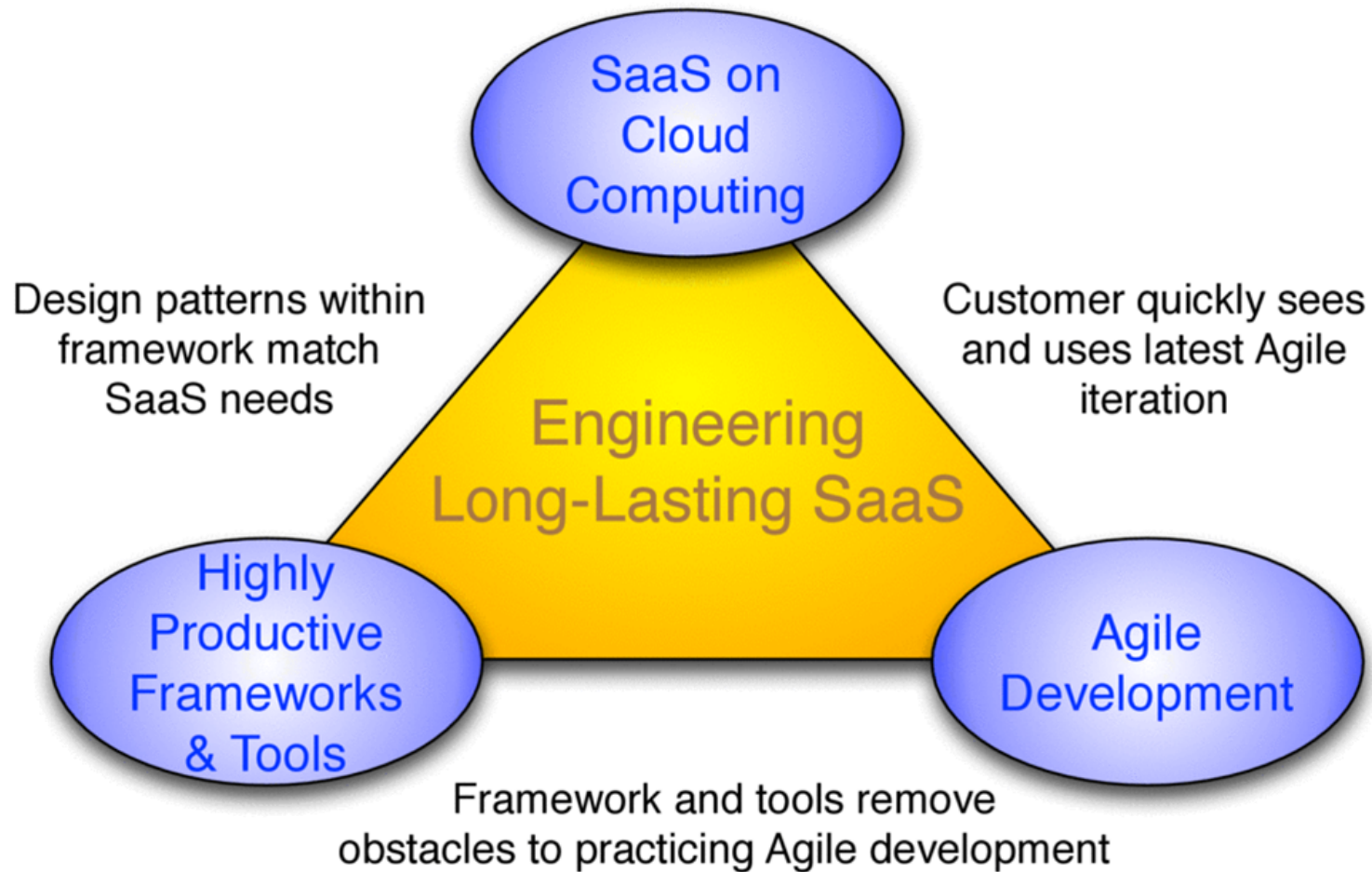


软件开发实践调查（敏捷过程）



- Survey of small projects using agile process (\leq \$1M each)
- 图片来源：Armando Fox, "Engineering Software as a Service"

敏捷过程成功的原因



本章重点

- 软件、软件危机和软件工程的含义
- 软件生命周期各个阶段的任务
- 各个软件过程模型的特点、优缺点和适用场合