软件工程

第一部分 软件工程概述

1.1 软件危机
1.2 软件工程概念与原则
1.3 软件生命周期
1.4 软件过程

教学要求

- 了解软件工程学科发展的源起，即软件危机特点、出现原因与解决途径
- 掌握软件工程的定义与基本原理
- 理解软件生命周期概念及其各个阶段的任务
- 掌握不同的软件过程模型及其比较

计算机软件

✓计算机系统由软件系统和硬件系统组成。
✓计算机软件包括程序，数据及其相关文档的完整集合。

✓软件≠程序
✓软件工程 ≠程序设计

1.1 软件危机

计算机软件开发与维护中遇到一系列的严重问题。主要特征：

- 软件开发成本严重超标；
- 软件成本在计算机系统总成本所占比例逐年上升；
- 软件开发周期大大超过规定日期；
- 软件系统达不到用户要求；
- 软件质量难于保证；
- 软件修改、维护困难；
- 缺少适当的文档资料；
- 软件开发生产率无法满足需求。

软件危机的产生原因

- 软件本身的特点
  - 计算机系统中的逻辑部件，缺乏“可见性”。质量难评价，难管理和控制、难维护。
  - 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。
- 软件开发与维护的方法
  - 对软件开发和维护的糊涂观念，如忽视软件需求分析的重要性、轻视软件配置、轻视软件维护等
  - 在实践过程中或多或少地采用了错误的方法和技术

软件危机的消除途径

从管理（组织措施）和技术（方法与工具）两方面研究如何更好地开发和维护计算机软件。

- 对计算机软件概念的正确认识 (软件由计算机程序、数据及文档组成;IEEE给出的软件配置定义);
- 认识软件开发中管理重要性（是“工程活动”）；
- 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（经验软件工程）；
- 开发和使用更好的软件工具（CASE）。

1.2 软件工程

- 软件工程概念的出现源自软件危机。
- 概括地讲，软件工程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工程学科。
- 采用工程的概念、原理、技术和方法来开发与维护软件，把经过时间考验而证明正确的管理技术和当前能够得到的最好的技术方法结合起来，以经济地（按进度）开发出高质量的软件并有效地维护它。

软件工程定义

- 1968 年德国人 Bauer 在北大西洋公约组织会议上的定义："建立并使用完善的工程化原则，以较经济的手段获得能在实际机器上有效运行的可靠软件的一系列方法"。
- 1983 年 IEEE 的软件工程定义："软件工程是开发、运行 、维护和修复软件的系统方法"。
- 1993 年 IEEE 的一个更加综合的定义："将系统化的、规范的、可度量的方法应用于软件的开发、运行和维护的过程，即将工程化应用于软件中"。

软件工程的本质特性

软件工程一些共性的特点：

- 关注于大型程序的构造
- 控制复杂性
- 有效应对变化
- 提高软件开发效率
- 组织和管理软件开发
- 有效地支持用户需求
- 有效地与不同领域的专家合作

软件工程的基本原理

著名的软件工程专家Barry Boehm 综合了软件工程的若干准则，提出了七条基本原理:

◆按软件生存期分阶段制定计划并认真实施
把整个软件开发过程视为一项工程，把工程划分为若干阶段，分别制定每个阶段的计划，逐个实施。

◆坚持进行阶段评审　前一阶段的结果将成为下一阶段的依据。坚持阶段的评审才能保证错误不传播到下一阶段。

3) 坚持严格的产品控制　将影响软件质量的因素在整个过程中置于严格控制之下（采用基线配置技术）

4) 使用现代程序设计技术　先进的程序设计技术带来的是生产率和质量的提高。使用合适的开发模式和工具可以有效地建立功能强大的系统。

5) 明确责任，使得工作结果能够得到清楚的审查　开发组织严格划分责任并制定产品的标准，使得每个成员的工作有据可依，确保产品的质量。

6)用人少而精　开发组织不在人多，在于每个人的技能适合要求。同时用人少而精，可减少沟通路径，提高生产率。

7)不断改进开发过程　在开发的过程中不断总结经验，改进开发的组织和过程，有效地通过过程质量的改进，提高软件产品的质量。

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。
- 管理就是通过计划、组织和控制等一系列活动，合理地配置和使用各种资源，以达到既定目标的过程。
- 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，也称为范型。
- 软件工程方法学包含3个要素：方法、工具和过程。

- 软件工程方法学包含3个要素：
  - –方法　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　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　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。

- 传统方法学：结构化方法学（生命周期方法学）
  - 采用结构化技术(结构化分析、结构化设计与结构化编程)及其工具完成软件开发的各项任务。
  - 将软件开发过程划分成若干个阶段，阶段之间有严格的技术审查和管理复审。从技术和管理两方面对各个阶段的开发成果进行检查。
  - 使用得十分广泛、也是非常成熟的软件工程方法学。在开发中小规模的软件时比较有效。

- 面向对象方法学
  - 把对象（数据与操作的融合）作为统一的软件构件。用对象分解取代了传统方法的功能分解。
  - 把所有对象都划分成类。类是对具有相同数据和相同操作的一组相似对象的定义。
  - 采用继承机制将把若干个相关类组成一个层次结构的系统。
  - 对象彼此间仅能通过发送消息互相联系。

- 面向对象方法学与结构化方法学的比较
  - 结构化方法学强调自顶向下顺序地完成软件开发的各阶段任务。
  - 面向对象方法学的出发点和基本原则，尽量模拟人类习惯的思维方式，使开发软件的方法与过程尽可能接近人类认识世界解决问题的方法与过程，从而使描述问题的问题空间与实现解法的解空间在结构上尽可能一致。具有：概念和表示方法上的一致性；多次反复迭代的演化过程；促进了软件重用等优点。

1.3 软件生命周期

- 软件生命周期包含三个阶段：软件定义、软件开发及软件运行维护。
  - ✓定义阶段 又称为系统分析，由系统分析员负责完成。确定总体目标，确定可行性，功能需求，估计资源与成本，进度安排等。可以进一步划分为问题定义、可行性研究和需求分析。
  - ✓开发阶段 具体设计与实现前一时期定义的软件。通常包括总体设计，详细设计，编码与单元测试，综合测试。
  - ✓运行维护阶段 执行不同类型的维护活动，使软件持久地满足用户需求。

软件生命周期每个阶段的基本任务。

- 问题定义

  回答“要解决的问题是什么?”

  通过对客户的访问调查，写出关于问题性质、工程目标和工程规模的书面报告。

- 可行性研究

  回答“确定的问题有行得通的解决办法吗?”

  在较抽象的高层次上进行分析和设计过程，探索这个问题是否值得去解，是否有可行的解决办法。

- 需求分析

  准确地回答“为了解决这个问题，目标系统必须做什么?”

  确定目标系统必须具备哪些功能和性能需求，建立需求分析模型和需求规格说明书。

- 总体设计（又称概要设计）

  回答“概括地说，应该怎样实现目标系统?”

  描述多种方案，并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；确定软件系统的体系结构（确定程序由哪些模块组成以及模块间的关系）。

- 详细设计（又称模块设计）

  回答“应该怎样具体地实现这个系统呢?”

  把解决问题的办法具体化，设计出程序的详细规格说明。即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。

- 编码和单元测试

  写出正确的容易理解、容易维护的程序模块。选取适当的程序设计语言，并对模块进行测试。

- 综合测试

  回答“实现的系统正确吗？是用户需要的吗？”

  通过各种类型的测试使软件达到预定的要求。通常进行集成测试和验收测试。保存测试计划、详细测试方案以及实际测试结果，作为软件配置的一个组成部分。

- 软件维护

  通过各种必要的维护活动使系统持久地满足用户的需要，包括改正性维护，适应性维护，完善性维护，预防性维护。

1.4 软件过程

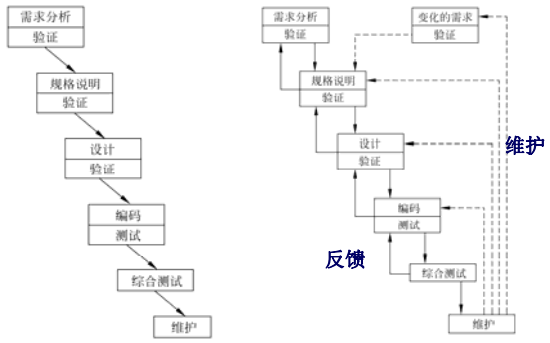
- 为了获得高质量软件所需要完成的一系列任务的框架，规定了完成各项任务的工作步骤。
  - ✓ISO 9000定义：软件过程是把输入转化为输出的一组彼此相关的资源和活动。
  - ✓从软件开发的观点看，它就是使用适当的资源（包括人员、软件工具、时间等），为开发软件进行的一组开发活动，在过程结束时将输入（用户要求）转化为输出（软件产品）。

- 软件过程定义了方法使用的顺序、要求交付的文档资料、为保证质量和适应变化所需要的管理、软件开发各个阶段完成的里程碑。
- 软件过程包含四种基本的过程活动 (包含在软件生命周期中):
  - ❖ plan： 软件规格说明
  - ❖ do： 软件开发
  - ❖ check： 软件确认
  - ❖ action：软件演进

- 软件生命周期模型来描述软件过程。
- 为了简洁地描述软件过程，把总体设计和详细设计合并在一起称为“设计”；把问题定义和可行性研究归并到需求分析中；规格说明作为一个单独阶段。
- 常见的软件过程模型有：
  - 瀑布模型，快速原型模型，增量模型，螺旋模型
  - 敏捷过程，微软过程
  - 喷泉模型，Rational 统一过程

1. 瀑布模型

- 各项活动按自上而下，相互衔接的固定次序，如同瀑布逐级下落，每项活动均处于一个质量环（输入-处理-输出-评审）中。
- 阶段间具有顺序性和依赖性
- 推迟实现的观点
- 质量保证的观点
  - 每个阶段必须完成规定的文档；
  - 每个阶段结束前完成文档审查。



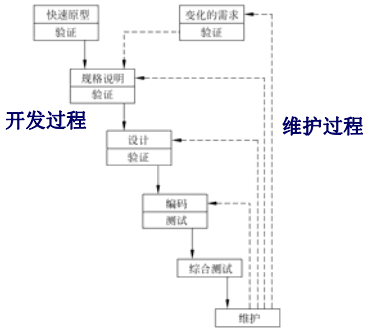
传统的瀑布模型　　实际的瀑布模型

瀑布模型是一种文档驱动的模型。

- 优点：
  - ✓ 可强迫开发人员采用规范的方法；
  - ✓ 严格地规定了每个阶段必须提交的文档(可维护性)；
  - ✓ 要求每个阶段交出的所有产品都必须经过质量保证小组的仔细验证；
- 缺点：
  - ✓ 几乎完全依赖于书面的规格说明，很可能导致最终开发出的软件产品不能真正满足用户的需要；
  - ✓ 缺乏灵活性。

2. 快速原型模型

- 所谓快速原形模型，就是快速建立起来的可以在计算机上运行的程序，它所能完成的功能往往是最终产品能完成的功能的一个子集。
- 快速原型模型的第一步是快速建立一个能反映用户主要需求的原型系统，让用户在计算机上试用它，通过实践来了解目标系统的概貌。原型的用途是获知用户的真正需求。
- 软件产品的开发基本上是线性顺序进行的，不带反馈环，这也是这种过程模型的主要优点。



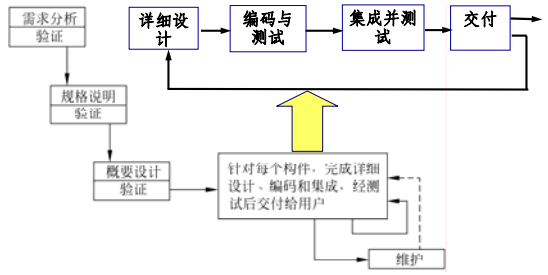
快速原型模型

快速原形模型是对瀑布模型的改进。主要优点是：有助于保证用户的真实需求得到满足。此外，

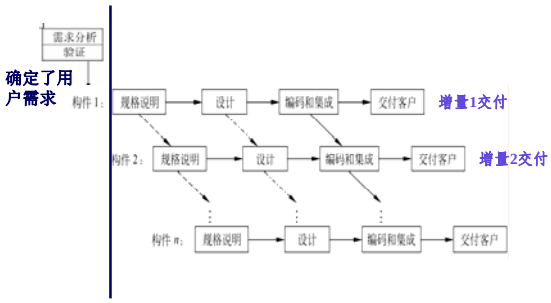
- ✓根据原型系统产生的规格说明文档正确地描述了用户需求，因此不存在较大的返工(线性的过程)。
- ✓开发人员通过建立原型系统已经学到了许多东西。
- ✓开发人员应该尽可能快地建造出原型系统，以加速软件开发过程，节约软件开发成本(体现快速的本质)。
- ✓快速原型的某个部分可以用到最终的软件产品中(复用)。

3. 增量模型

- 增量模型是迭代和演进的过程。增量模型把软件产品分解成一系列的增量构件，在增量开发迭代中逐步加入。
  - ✓分解时必须遵守约束条件：当把新构件集成到现有软件中时，所形成的产品必须是可测试的。
  - ✓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。
  - ✓第一个构件实现软件的基本需求，提供核心功能。早先完成的增量可以为后期的增量提供服务。



增量模型



风险更大的增量模型

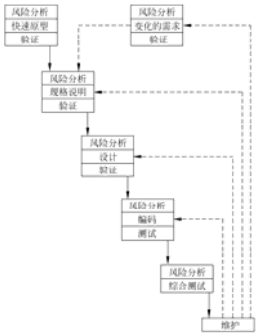
分批地逐步向用户提交产品，整个软件产品被分解成许多个增量构件，开发人员一个构件接一个构件地向用户提交产品。

- 优点
  - ✓能在较短时间内向用户提交可完成部分工作的产品。
  - ✓逐步增加产品功能可以使用户有较充裕的时间学习和适应新产品。
- 缺点：
  - ✓对体系结构的设计要求高（必须开放的、具有可扩充性），要求更精心地设计。

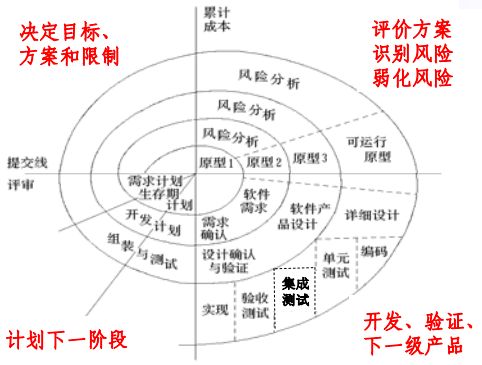
4. 螺旋模型

- 软件风险是任何软件开发项目中都普遍存在的实际问题。
- 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。将瀑布模型与迭代过程结合起来，并且加入风险分析；或者说，在每个阶段之前都增加了风险分析过程的快速原型模型。
- 螺旋模型沿着螺线旋转，自内向外每旋转一圈便开发出更完善的一个新版本。

  制定计划； 风险分析； 实施工程； 客户评估



简化的螺旋模型



完整的螺旋模型

螺旋模型是一种风险驱动的过程模型。主要适用于内部开发的大规模软件项目（风险评估的代价与可能的中止）。

- 主要的优点在于：
  - ✓ 减少了过多测试或测试不足所带来的风险;
  - ✓ 维护只是模型的另一个周期，在维护和开发之间并没有本质区别;
  - ✓ 对可选方案和约束条件的强调有利于软件的重用。
- 缺点:
  - ✓ 要求软件开发人员具有丰富的风险评估经验和这方面的专门知识.

5. 敏捷过程

为了使软件开发团对具有高效工作和快速响应变化的能力，17位著名的软件专家于2001年联合起草了敏捷软件开发宣言。

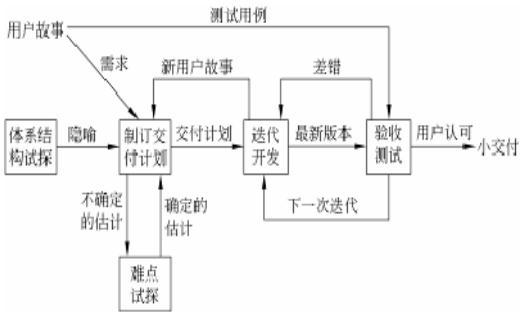
敏捷软件开发由4个观点:

- ✓个体和交互胜过过程和工具
- ✓可以工作的软件胜过面面俱到的文档
- ✓客户合作胜过合同谈判
- ✓响应变化胜过遵循计划

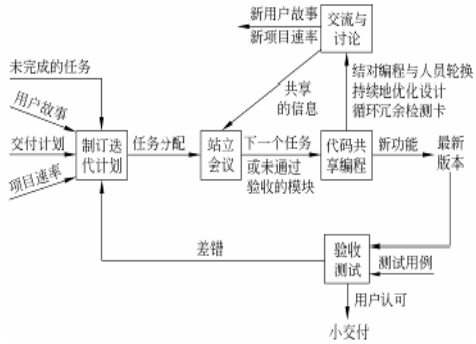
极限编程

极限编程（XP）是敏捷过程一个典型的开发方法。适用于需求模糊且经常变化的场合。有效实践包括:

- ✓客户作为开发的团队
- ✓使用用户素材
- ✓短交付周期
- ✓验收测试
- ✓结对编程
- ✓测试驱动开发
- ✓集体所有
- ✓持续集成
- ✓可持续发展
- ✓开放的工作空间
- ✓及时调整计划
- ✓简单的设计
- ✓重构
- ✓使用隐喻



XP项目的整体开发过程



XP迭代开发过程

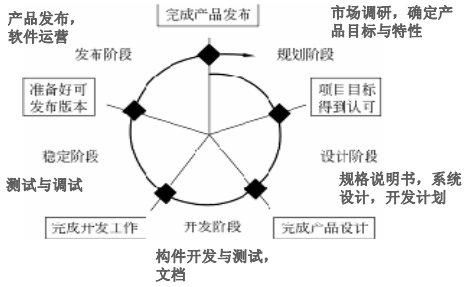
敏捷过程具有对变化和不确定性的更快速、更敏捷的反映特性，而且能够保持可持续的开发速度。

敏捷过程能够较好地适应商业竞争环境下对小型项目提出的有限资源和有限开发时间的约束。

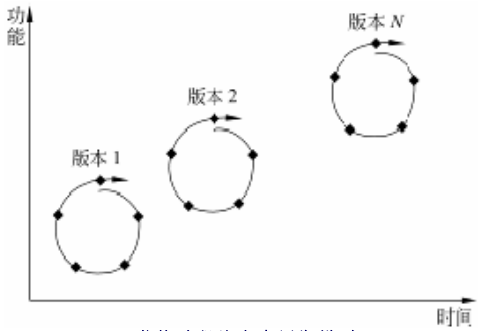
6. 微软过程

微软公司采用的软件开发过程，已经被实践证明是非常成功和行之有效的。包括如下准则：

- ✓项目计划兼顾不确定因素
- ✓有效的风险管理
- ✓经常生成并快速地测试软件过渡版本
- ✓采用快速循环、递进的开发过程
- ✓平衡产品特性和产品成本
- ✓项目进度表应具有稳定性和权威性
- ✓使用小型项目组
- ✓软件配置项基线化
- ✓使用原型验证概念
- ✓追求零缺陷
- ✓里程碑评审会



微软软件生命周期段划分



微软过程的生命周期模型

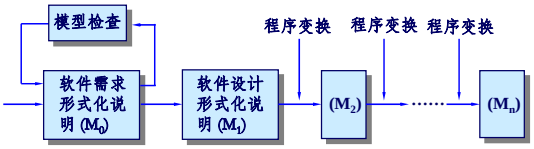
微软过程综合了RUP和敏捷过程的许多优点，是对很多成功项目的开发经验总结。

适用于商业环境下具有有限资源和有限开发时间约束的项目。

7. 变换模型

- 变换模型是一种基于形式化规格说明语言及程序变换的软件开发模型。
- 它采用形式化的软件开发方法，对形式化的软件规格说明进行一系列自动的或半自动的程序变换，最终映射成为计算机系统能够接受的程序系统。
- 多步程序变换过程的重要性质是：每一步程序变换的正确性仅与该步变换所依据的规范 M_i 以及对变换后的假设 M_{i+1} 有关。

- 在此意义上，变换步骤独立于其他变换步骤。这称为变换的独立性。
- 该模型只适合于软件的形式化开发方法；需要严格的数学理论和形式化技术支持；需要一整套开发环境（如程序变换工具、定理证明工具等）的支持。



软件过程模型比较

- 针对不同的开发环境、组织环境、项目特点，选择合适的软件过程模型。
  - ✓瀑布模型 历史悠久、规范的、文档驱动的方法；最终开发出的软件产品可能并不是用户真正需要的。
  - ✓快速原型模型 改进瀑布模型的缺点，让用户试用原型并收集用户反馈意见的办法，获取用户的真实需求。

- ✓增量模型 早期阶段使投资获得明显回报和较易维护；要求软件具有开放的结构。
- ✓螺旋模型 风险驱动的过程，适用于内部开发的大型软件项目，但要求开发人员具有风险分析和排除风险的经验及专门知识。
- ✓敏捷过程 一种有效应对需求不清和快速变化的小型项目。开发过程不是很规范。
- ✓微软过程 一种适用于商业竞争环境下软件项目开发。

- ✓喷泉模型* 面向对象方法学常用的过程模型，强调不同阶段的迭代和无缝的特性。
- ✓RUP* 一个非常规范、全面的过程模型，并在实践中广泛应用。