**软件工程手册 VIP MASTER版**

SW01 概述

SW02 系统工程

SW03 软件过程

SW04 项目管理

SW05 软件需求

SW06 需求工程过程

SW07 系统模型

SW08 原型系统开发

SW09 软件设计概述

SW10 分布式系统

SW11 白盒测试

SW12 黑盒测试

SW13 软件测试策略

SW14 检验和有效性验证

SW15 项目变更

其他一 敏捷开发

其他二 git表

1

8

12

17

20

24

32

35

39

49

53

54

60

66

70

73

78

SW 01概述

一、软件

1.1 什么是软件

* 软件是由计算机程序、数据及文档组成。
* 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

1.2 软件特点

* 抽象性
* 无磨损性
* 对计算机硬件依赖性
* 软件的手工开发方式
* 软件本身复杂性
* 软件的高成本

1.3 软件分类

* 系统软件

操作系统、数据库管理系统(DBMS)、设备驱动程序、通信处理程序等

* 支撑软件

文本编辑程序、文件格式化程序、磁盘向磁带数据传输的程序、支持需求分析、设计、实现、测试和支持管理的软件

* 应用软件

商业数据处理软件、工程与科学计算软件、计算机辅助设计／制造软件、系统仿真软件、智能产品嵌入软件、医疗、制药软件、办公自动化软件

1.4 软件的发展

* 软件的发展经历了三个阶段
  + 程序设计阶段 — 50至60年代
  + 程序系统阶段 — 60至70年代

这一时期主要围绕软件项目，开展了开发模型、支持工具以及开发方法的研究。如：瀑布模型、结构化方法（自顶向下）等

* + 软件工程阶段 — 80年代以后

开展了有关软件生产技术、软件复用技术、软件生产管理的研究和实践；提出具有广泛应用前景的面向对象方法和相关的语言（Smalltalk、C++）；近年来，软件工程的研究从过程转向产品更加注重程序的开发范型和软件生产。

1.5 软件危机

* 对软件开发成本和进度的估算很不准确
* 用户不满意
* 质量不可靠
* 没有适当的文档
* 软件维护困难
* 软件成本比重上升
* 软件开发生产率不能满足人们对合格软件的需求
* 软件工程概念的出现源自软件危机

危机根源

* 软件本身特点

规模庞大：Windows2000：5000万行代码

* 开发方法的缺陷

忽视需求分析、软件开发 = 程序编写、轻视软件维护

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

二、软件工程

2.1 软件工程

* 所有发达国家的经济都是依赖于软件的
* 越来越多的系统是由软件控制的
* 在软件工程方面的开支在发达国家 GNP 中占有很大比例
* 软件工程致力于研究低成本的软件开发
* 软件工程致力于专业化软件开发的理论、方法和工具研究

软件工程方法学的三要素

* 方法Methods: 开发软件的技术，即为软件开发提供了 “如何做” 的技术.
* 工具Tools: 在方法中使用的自动化或半自动化支撑环境，即为软件工程方法提供了自动的或半自动的软件支撑环境
* 过程Process: 获得高质量软件所需要的一系列活动、约束和资源.

2.2 什么是软件工程?

* 软件工程是工程学科，涉及到软件生产的各个方面
* 软件工程师应该有系统的和科学的工作方法，根据要解决的问题,开发上的限制和可利用的资源等因素，选择使用适当的工具和技术

*软件工程是应用一组规范化的方法,采用工程的概念、原理、技术和方法,把经过时间考验证明的正确技术和当前能够得到的先进技术和方法结合起来开发、维护和管理软件*

2.3 软件工程的特点

* 规范化: 多人共同开发和维护软件时,人人都必须遵循相同的约束规范.
* 文档化: 在软件开发的各阶段都要有相关的文档,用于描述设计思想、设计过程和具体实现技术等信息.
* 工程化: 按照工程化的思想开发和维护软件产品.

2.4 软件工程的本质

* 关注于大型软件产品的构造
* 控制软件的复杂性
* 提高开发软件的效率
* 在开发软件过程中合作
* 满足用户的需求
* 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

2.5 软件工程的目标

* 软件工程需要达到的基本目标：
  + 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，及时交付使用

2.6 软件工程的基本原理

著名软件工程专家B.W.Boehm综合软件专家、学者们给出出的关于软件开发的意见,并结合其多年开发软件的经 验,于1983年在一篇论文中提出了软件工程的7条基本原则

1. 用分阶段的生命周期计划严格管理

将软件生命周期划分为若干个阶段，制订实施计划并根据计划进行严格管理

(2) 坚持进行阶段评审

在每个阶段完成时都要进行严格的评审;尽早发现错误，尽量避免将错误带入下一阶段

(3) 实行严格的产品版本控制

基线配置：经过阶段评审后的软件配置成分（文档、源代码）。实行基线配置管理，不得随意变动。

(4) 采用现代程序设计技术

* + 采用先进的技术（结构化程序设计技术、面向对象程序设计技术、构件程序设计技术），提高软件开发的效率和软件产品的质量。

(5) 结果应能清楚地审查(文档支持)

* + 软件开发的管理和软件产品的结果应有利于审查。

(6) 开发人员应该少而精

* + 开发组人员素质高;
  + 人数越少越好.

(7) 承认不断改进软件工程实践的必要性

* + 积极主动地采纳新的软件技术;
  + 不断总结经验.

三、软件工程中的常见问题

3.1 软件工程和计算机科学的区别?

* 计算机科学研究的是构成计算机和软件系统的基础的有关理论和方法，而软件工程研究软件生产中的实际问题
* 计算机科学理论现在已经不够做为软件工程学的完全支撑
* 专业，职业教育中有软件工程专业。
* 计算机科学在本科教育中。

3.2 软件工程和系统工程的区别?

* 系统工程是基于计算机的系统工程，包括硬件开发、软件开发、过程设计、系统实施等。软件工程是系统工程的一部分
* 系统工程师需要负责系统描述、体系结构设计、整合和部署

3.3 什么是软件过程?

* 软件过程是生产软件产品的一系列活动及其结果
* 软件过程包含的4项基本活动:
  + 软件描述 - what the system should do and its development constraints
  + 软件开发 - production of the software system
  + 软件有效性验证 - checking that the software is what the customer wants
  + 软件进化 - changing the software in response to changing demands

3.4 什么是软件过程模型?

* 软件过程模型是从一特定的角度提出的软件过程的简化描述
* 一般的软件过程模型
  + Waterfall
  + Evolutionary development
  + Formal transformation
  + Integration from reusable components

3.5 软件工程的成本包括哪些?

* 60%的成本是开发成本, 40% 是测试成本. 对于定制软件, 软件进化成本会超过开发成本
* 软件开发成本是依赖于系统的开发方法、系统特性的要求
* 软件成本的分布依赖于所采用的开发模型

3.6 什么是软件工程方法?

* 软件工程方法是一种软件开发的结构化方法，其目的是使高质量的软件生产性价比较高

3.7 什么是CASE (Computer-Aided Software Engineering)

* CASE是计算机辅助软件工程，它包括支持软件过程活动（需求分析、系统建模、调试和测试等）的软件工具
* 高端CASE工具

Tools to support the early process activities of requirements and design

* 低端CASE工具

Tools to support later activities such as programming, debugging and testing

3.8 软件包括哪些属性?

* 软件除了提供相应的功能外，作为一个产品它还有一系列相关的反映质量的属性
* 可维护性

Software must evolve to meet changing needs

* 可依赖性

Software must be trustworthy

* 有效性

Software should not make wasteful use of system resources

* 可用性

Software must be usable by the users for which it was designed

3.9 软件工程面临的主要问题?

* 遗留系统的挑战

Old, valuable systems must be maintained and updated

* 异构性的挑战

Systems are distributed and include a mix of hardware and software

* 交付上的挑战

There is increasing pressure for faster delivery of software

四、软件生存周期

4.1 软件生存周期

* 软件有一个孕育、诞生、成长、成熟、衰亡的生存过程。这个过程即为软件的生存期
* 生命周期(life cycle)：当过程涉及到建造某种产品时,常将过程称之为“生命周期”.
* 软件生命周期SLC(Software Life Cycle): 软件开发和维护的过程称之为软件生命周期,它描述了软件产品从设想到实现、提交、使用和维护的过程.

软件生存期六个步骤:

* 制定计划(Planning)：问题定义与可行性研究
* 需求分析和定义(Requirement Analysis and Definition)
* 软件设计( Software Design)
* 程序编写(Coding, Programming)
* 软件测试(Testing)
* 运行/维护(Running/Maintenance)
* (1) 问题定义
  + 问题是什么
  + 确定软件开发工程的总体目标
  + 确定问题的性质、项目规模和系统边界
  + 提交《问题定义报告》
* (2) 可行性研究
  + 有可行解吗？
  + 成本/效益分析
  + 确定工程的可行性
  + 提交《可行性研究报告》
  + 对《可行性研究报告》进行评审
* (3) 项目计划
  + 估计可利用的资源(计算机硬件，软件，人力等)、成本、效益、开发进度。制定《软件项目开发计划》
  + 制定出完成开发任务的实施计划,连同可行性研究报告，提交管理部门审查。对《软件项目开发计划》进行评审
* (4) 需求分析
  + 获取用户的需求,明确用户要求系统做什么，做到什么程度；确定系统的功能需求、性能需要和其他需求
  + 编写《用户需求报告》
  + 根据《用户需求报告》编写《需求规格说明书》
  + 对《用户需求报告》和《需求规格说明书》进行评审
* (5) 总体(概要)设计
  + 确定系统实现方案
  + 对各种方案进行评价,选择一个合适的方案
  + 系统流程图＋层次图或结构图
  + 提交《总体设计报告》
  + 对《总体设计报告》进行评审
* (6) 详细设计
  + 设计每个模块内部结构
  + 设计数据结构和模块内部的算法
  + 提交《详细设计说明书》
  + 对《详细设计说明书》进行评审
* (7) 编码和单元测试
  + 编写程序代码
  + 进行单元测试
  + 提交源程序清单和目标程序
  + 对源程序进行评审

Code+测试方案和结果

* (8) 系统(集成,综合)测试
  + 集成测试：将经过单元测试的模块组装，并进行部件测试.
  + 功能测试：测试集成后系统的功能.
  + 性能测试：测试系统的性能.
  + 安装测试：在用户环境进行测试.
  + 验收测试 ： 验收通过后,由用户签字确认.

符合要求的软件

* (9) 产品发布
  + 完善软件文档
  + 用户培训
  + 软件产品包装
  + 软件产品发布
* (10) 维护
  + 改正性维护 运行中发现了软件中的错误需要修正
  + 适应性维护 为了适应变化了的软件工作环境，需做适当变更
  + 完善性维护 为了增强软件的功能需做变更

持久地满足用户需要

五、软件工程人员的职业道德

5.1 职业和道德上的责任

* 软件工程人员必须承认他们的工作不仅仅是技术的应用，还要担负许多责任
* 软件工程人员要受人尊敬，其行为必须符合道德，必须有责任心
* 某些行为没有法律加以规范，只能靠职业道德来约束

5.2 软件工程人员的职业道德

* 机密

工程师应该尊重他们雇主和客户的机密，无论是否签署过正式的机密协议

* *工作能力*

工程师不应该夸大自己的工作能力。不应该故意接受超出自己能力的工作

* 知识产权

工程师应该知道地方性的有关知识产权的使用方面的法律，例如专利权, 版权等。应该认真履行职责确保顾主和客户的知识产权不受侵害。

* *计算机滥用*

软件工程师不应该在他人的机器上滥用他们的技术。计算机的滥用包括从不起眼的小事 (在老板机器上打游戏,等) 到极端严重的事件 (发布病毒)。

5.3 ACM/IEEE 职业道德规范

* ＡＣＭ和ＩＥＥＥ推出了关于职业道德职业行为的规范
* 规范包含与行为和意识相关的八项原则， 包括从业者、教育家、经理、管理者和政策制定人 , 也包括职业的实习生和学生

5.3.1 道德规范 – 序言

* + 规范的简化版本在较高的层次上提出了要求；在完整的版本中条款举例说明了这些要求如何改变我们作为 软件工程专业人士的生存方式。没有要求,这些细节就编成了生硬和琐细的法律条文； 缺少细节,要求只能是高调和空谈； 二者的结合使之成为完整的规范
  + 软件工程师应当做出承诺，使软件的分析、描述、设计、开发、测试和维护等工作对社会有益且受人尊敬。 基于对公众健康、安全和福利的考虑，软件工程人员应当遵守以下八条原则：
    1. 道德规范 – 八条原则

1. 公众感：软件工程师应该始终与 公众的利益保持一致。

2. 客户和雇主：软件工程师在公众利益的前提下满足客户和雇主的最大利益。

3. 产品：软件工程师应当保证他们的产品和相关的附件达到尽可能高的行业标准。

4. 判断力：软件工程师应该具有公正和独立的判断能力。

5. 管理：软件工程经理和领导者应该拥护并倡导合乎道德的有关软件开发和维护的管理方法。

6. 职业感：软件工程师应当弘扬职业正义感，尊重社会公众利益。

7. 同事：软件工程师应该公平地对待和协助他们的同事。

8. 自己：软件工程师应当毕生学习专业知识，倡导合乎职业道德的职业生活方式。

SW01概述 要点

* 软件工程是一门工程学科，涉及软件生产的各个方面
* 软件产品由开发的程序、相关文档及数据构成。软件产品的基本属性是可维护性、可依赖性、有效性、可用性
* 软件过程由开发软件产品的一系列活动组成。基本活动有：软件描述、开发、有效性验证和进化
* 软件工程方法是软件生产的组织方式，包括软件过程的建议、使用的标记法、进行系统描述的规律和设计指南
* CASE工具是一些软件系统，被设计成支持软件过程中的常规活动、如编辑设计图表、检查图表的连贯性、跟踪已经运行的程序测试等
* 软件工程人员对软件工程这一职业和社会负有责任，不应该只关心技术问题
* 职业协会颁布的行为准则规定了一系列协会成员应该遵守的行为准则

SW02 基于计算机的系统工程

一 、系统、系统工程

* 1. 什么是系统?
* 一个系统是一组相互关联、能一起工作从而达到某个目标的相关组件的集合
* 一个系统可能包括软件, 机械, 电气和电子硬件和人的操作
* 系统组件之间是相互依赖的

1.2 什么是系统工程?

* 系统工程是描述、设计、实现、有效性验证、实施和维护等一系列活动
* 系统工程师不仅要关心系统的硬件和软件，还要注意系统与周围环境的关系以及人机交付问题

1.3 软件和系统工程

* 软件在系统中的比例正在增加
* 软件被视为系统工程中的一个问题，(很遗憾） 很多大型系统项目由于软件方面的原因而推迟移交
* 软件工程和系统工程紧密相关

二 、系统总体特性

2.1 系统总体特性

* 系统的总体特性是系统整体上的特性，是不能从系统某个组成部分得到的特性
* 整体特性是系统组件之间关联关系的结果
* 整体特性是当组件全部集成到系统中之后才能评估和测量

整体特性的例子

* *系统的全部重量*
* *系统的*可信度
  + 这依赖于组件的可信度和组件之间的关系。
* *系统的可用性*
  + 这是一个复杂特性，不仅依赖于系统硬件和软件，而且也依赖系统操作员和它使用的环境。
  + System usability scale

总体特性的类型

* 功能特性
  + 当系统的所有组件一起工作完成一项任务的时候, 就表现出功能特性。 例如, a bicycle has the functional property of being a transportation device once it has been assembled from its components.
* 非功能特性
  + 例子是可靠性、性能、安全和信息安全性。这些特性表现为在特定环境中系统的表现行为。

2.2 系统可靠性工程

* 可靠性是在系统层面上必须被考虑的一个复杂概念
* 组件之间是相互依赖的,缺陷会经过系统在组件间传播

影响可靠性的因素

* 硬件可靠性

硬件组件失败的可能性有多大，多长时间对组件维修一次?

* 软件可靠性

软件组件产生不正确输出的可能性有多大？软件失败与硬件失败有明显的不同，因为软件没有磨损

* 操作员可靠性

系统操作员产生错误操作的可能性有多大？

可靠性关系

* 硬件失败能产生软件预期输入范围之外的假信号
* 软件错误能引起警报，刺激引起操作员紧张，而且导致操作员操作错误
* 一个系统安装的环境能影响它的可靠性

三、系统及其环境

3.1 系统及其环境

* 系统是在一定环境中存在的
* 环境影响系统的功能和性能，举例来说，系统需要从环境获取电源
* 组织环境和物理环境可能都是很重要的
* 环境包含一系列相互作用的其它系统

3.2 系统层次结构

3.3 影响系统设计的人和组织因素

* *过程变化*
  + 系统需要对环境中的工作过程发生变化吗?
* *工作*改变
  + 系统是否降低了环境中使用者的技术，或是引起他们改变他们的工作方式?
* *机构的变化*
  + 系统变化改变了机构中政治力量的构成吗?

系统建模

* 系统体系结构模型给出子系统构成系统的抽象视图
* 系统建模是系统需求和设计的一部分
* 系统是被建成一系列组件和组件间的关系
* 系统体系结构常以方块图来描述，展现主要的子系统及这些子系统之间的关系
* 子系统间的关系用带箭头的连线表示，关系包含数据流、使用和被使用等关系

四、系统工程过程

4.1 系统工程过程

系统工程过程与软件工程过程的主要区别：

* 学科间的相互渗透

不可避免包括来自不同学科的工程师一起工作

* 在系统开发期间的反攻余地在缩小

通常遵循‘瀑布’模型，因为需要并行地执行系统不同组件的开发，各阶段之间只有很小的反复余地，因为硬件改变代价非常高。 软件可以弥补硬件问题。

4.2 系统需求定义

* 需求定义阶段重点导出三种类型的需求

1.抽象功能需求. System functions are defined in an abstract way

2.系统特性. Non-functional requirements for the system in general are defined

3.运行要求和其他

* 需求定义阶段的一个重要部分是建立系统需要达到的一些总的目标

*系统需求问题*

* 系统在定义过程中的变更
* 必须预期硬件/ 通信在系统的生命期中的发展
* 难以定义非功能性需求

4.3 系统设计

* 分割需求
  + Organise requirements into related groups
* 标识子系统
  + Identify a set of sub-systems which collectively can meet the system requirements
* 为子系统分配需求
  + Causes particular problems when COTS are integrated
* 描述子系统功能
* 定义子系统接口
  + Critical activity for parallel sub-system development

4.4 子系统开发

* 一般硬件、软件和通信是并行开发的
* 可能包括一些COTS ( 商业现货产品) 系统采购
* 由于官僚和管理机制僵化，系统变更经常提出的较晚，意味着开发时间进度需要延长，因为需要经常返工

4.5 系统整合

* 系统整合就是将一个个独立开发的子系统整合为一个完整的大系统
* 应该采用渐增式整合，即一次只增加一个子系统
* 子系统之间的接口问题通常在这个阶段发现

4.6 系统安装

* 安装环境与开发环境可能不一致
* 新的系统可能会受到人的抵制
* 系统可能与被替代的系统共存一些时间
* 可能有物理安装问题 ( 例如电缆问题)
* 需要确定操作员培训

4.7 系统操作

* 系统存在的问题被发现(Ariane 5's first test flight )
* 产生新的需求
* 系统的设计不完全符合用户的期望
* 会暴露出与其他系统交互作用中的问题
  + 不相容问题(数据传输)
  + 因为不一致的操作界面而增加操作错误

4.8系统进化

* 大型和复杂系统通常有很长生命期。它们必须经过进化来满足不断变化的需求
* 进化注定是需要付出昂贵代价的
  + 必须从技术和商业角度分析变化
  + 子系统之间的交互作用会出现无法预料的问题
  + 最初设计决策的理由很少被记录
  + 随着系统不断运行，系统进化的成本进一步增加
* 需要维护的现存系统也称为遗留系统

4.9 系统退役

* 系统在生存期之后退出系统服务
* 系统中的数据需要重构和转换，以便在其他一些系统中使用

SW02 系统工程 要点

* 系统工程是一个复杂而又困难的过程，系统工程学包括很多学科的内容
* 系统总体特性是系统作为总体表现出来的特性，而不是任何部分所能表现出来的特性
* 系统体系结构模型一般都用方块图来表示，主要说明子系统极其之间的关系。
* 系统工程过程通常包括描述、设计、开发和集成

SW03 软件过程

一、软件过程

1.1 软件过程概述

* 软件过程是软件产品生产过程中互相连贯的一组活动（描述、设计、实现和测试等）：
  + Specification
  + Design
  + Validation
  + Evolution

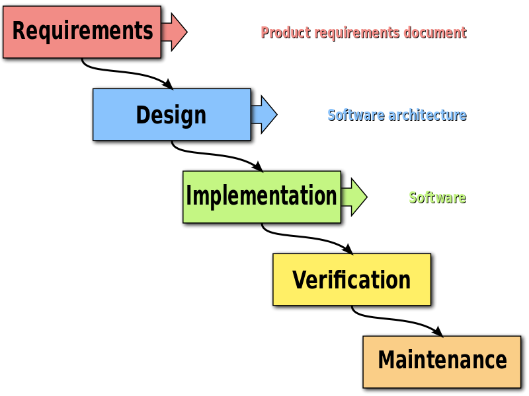
二、软件过程模型

2.1 软件过程模型

* 软件过程模型是软件过程的一个抽象表示法，它从一个特定的角度表现一个过程：
  + 瀑布模型、V模型
  + 增量式开发
  + 面向复用的开发
  + 进化式开发

2.2.1 瀑布模型

* 瀑布模型就象瀑布一样,一级一级地实现.下一个阶段开始之前,前一个阶段必须完成.
* 瀑布模型从一种较高的观点看待软件开发,必须事先将每一个阶段都安排好,开发时只需按模型的顺序进行即可.
* 每个阶段都是一个里程碑(Milestone),在每个里程碑处都有明确的评价标准,并进行严格的评审与审计,通过后才能进入下一个阶段,这样项目经理就能控制好项目的开发进度、成本和质量.

*瀑布模型的基本活动*

* 需求分析和定义
* 系统和软件设计
* 实现和单元测试
* 集成和系统测试
* 运行和维护

*瀑布模型的优点:*

* 结构简单,强调软件开发过程的阶段性和顺序性;
* 对软件开发管理严格,文档齐全(里程碑驱动或称文档驱动),注重复审,保证了软件的质量;
* 支持推迟实现的观点,尽可能将编码向后推迟.

*瀑布模型的缺点:*

* 将开发活动生硬地分解成确切的阶段，对用户需求变更的响应较困难过程不可逆转,前面阶段隐藏下来的问题会在后面放大;
* 重视需求规格说明,然而获得一个完整的规格说明是非常困难的,导致了开发出来的系统最终需求与最初需求不一致,大大地降低了实用性.

*瀑布模型使用指南*

* 开发过程
  + 需求过程
  + 设计过程
  + 实施过程
* 开发后的过程
  + 安装过程
  + 支持过程
  + 维护过程
  + 抛弃过程

*瀑布模型适合的项目*

* 项目的需求在项目开始前很明确
* 解决方案在项目开始前也很明确
* 类似的项目如：公司的财务系统、库存管理系统、短期项目

尽管有许多不足,尽管有许多新的过程模型出现,瀑布模型仍是使用最广泛的过程模型之一.

也是其他软件过程模型的基础。

*支持瀑布模型的原因*

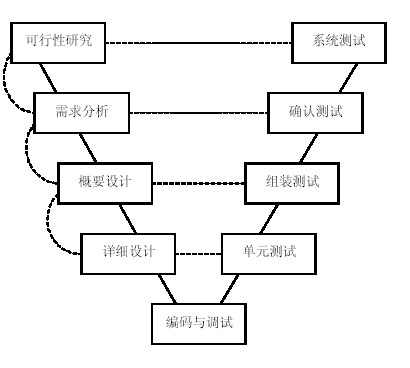
* + 前阶段花的时间可以减少后阶段的成本。
  + 充分的文档使得关于项目的信息不会丢失。理想情况下，整个小组换人也可以继续项目的进行。
  + 模型简单明了，容易使用。

2.2.2 V型模型

* V型模型是瀑布模型的一个变种。它同样需要一步一步进行，前一段的任务完成之后可以进行下一阶段的任务。这个**模型强调测试的重要性**，它将开发活动与测试活动紧密地联系在一起。

*V型模型特点*

* 简单易用
* 强调测试过程与开发过程的对应性和并行性

*V型模型使用指南*

* 严格按照顺序进行开发
* 注意开发过程和测试过程的并行性

*V型模型适合的项目*

* 项目的需求在项目开始前很明确
* 解决方案在项目开始前也很明确
* 对系统的性能安全很严格的项目
* 类似的项目如：
  + 航天飞机等
  + 公司的财务系统

2.2.3 增量模型

增量模型(Incremental Model)根据需求规模说明书将系统按功能分解成若干个子系统.开始开发一个子系统的功能,经过设计、实现、集成、测试后交付使用,然后再开发其他子系统,再加入已有的系统中,逐渐构成能完成全部功能的完整系统.

*增量模型的优点*

* 可以循序渐进，避免一次投入太大的风险
* 更快地开发出可以操作的系统
* 可以减少开发过程中用户需求的变更
* 有些增量可能需要重新开发;

*增量模型的缺点*

* 不适合各部分联系紧密的项目;
* 系统整体结构的一致性可能较差;
* 各子系统风格可能不一致.

*增量模型使用指南*

* 首先构建整个系统的一部分
* 然后逐步地增加功能和性能

*增量模型适合的项目*

* 项目开始，明确了需求的大部分，但是需求可能会发生变化
* 对于市场和用户把握不是很准，需要逐步了解
* 对于有庞大和复杂功能的系统进行功能改进，就需要一步一步实施的。

2.2.4 面向复用的开发

* 基于系统的复用，将已存在的组件或COTS ( 商业现货产品) 系统整合成所要的系统
* 该方法正在变得越来越重要，但是仍然缺乏经验积累

*面向复用的开发的特点*

* Process stages(初始需求描述阶段和有效性阶段同其它过程差不多)，中间阶段：
  + 组件分析
  + 需求修正
  + 利用复用的系统设计
  + 开发和集成
* 面向复用的模型减少了需要开发的软件数量，降低了软件开发成本，降低了风险。然而，需求妥协是不可避免的，这有可能导致一个不符合用户真正需要的系统。

2.2.5 进化式开发

* 探索式开发
  + 目标是和客户一起工作，共同探索系统需求，直到最后交付系统，应该先从理解的比较好的需求开始
* 抛弃式原型
  + 目标是理解用户需求，然后给出一个较好的需求定义

三、软件描述

3.1 软件描述

* 软件描述是确定系统需要哪些服务以及开发和运行期间受到哪些约束(活动：需求工程)
* 需求工程过程
  + Feasibility study
  + Requirements elicitation and analysis
  + Requirements specification
  + Requirements validation

四、软件设计和实现

4.1 软件设计和实现

* 将系统描述转换成一个可执行的程序的过程
* Software design
  + 软件设计是对实现软件的结构、系统的数据、系统组件间的接口以及所用的算法的描述
* Implementation
  + 实现是将软件设计转换为可执行的代码
* 设计和实现的活动是紧密关联的且相互交织在一起

*设计过程活动*

* 体系结构设计
* 抽象描述
* 接口设计
* 组件设计
* 数据结构设计
* 算法设计

4.2 设计方法

* 进行软件设计的系统方法
* 设计文档通常用一组图解式的模型来表示
* 可能的模型
  + Data-flow model
  + Entity-relation-attribute model
  + Structural model
  + Object models

4.3 程序设计和调试

* 将设计转变成程序，并从中除去错误
* 编程规范
* 程序员通过执行程序测试来发现其中的缺陷，并用调试过程来修正程序中的缺陷

五、软件有效性验证

5.1 软件检验和有效性验证

* 检验和有效性验证是看系统是否符合它的描述及系统是否符合客户的预期
* 包括检查和评审以及系统测试过程
* 系统测试是采用具有真实数据的测试用例进行运行系统，检查是否与用户的期望一致

*测试阶段*

* 单元测试

Individual components are tested

* 模块测试

Related collections of dependent components are tested

* 子系统测试

Modules are integrated into sub-systems and tested. The focus here should be on interface testing

* 系统测试

Testing of the system as a whole. Testing of emergent properties

* 接收测试

Testing with customer data to check that it is acceptable

六、软件进化

6.1 软件进化

* 软件具有柔性，容易改变。
* 商业环境发生改变的时候,需求也要发生改变，软件因而一定要因需而变
* 虽然系统可以划分为开发和进化 (维护)两个阶段，当越来越少的系统完全是新的时候，这种划分逐渐变得无关紧要了

七、自动化的过程支持

7.1 自动化的过程支持(CASE)

* 计算机辅助软件工程 (CASE) 是用来支持软件过程活动的软件
* CASE自动化活动实例：
  + Graphical editors for system model development
  + Data dictionary to manage design entities
  + Graphical UI builder for user interface construction
  + Debuggers to support program fault finding

7.2 CASE 技术

* CASE技术对软件过程有重要的推动作用
* 但，CASE工具只能在一定程度上给软件工程带来影响，原因：
  + Software engineering requires creative thought this is not readily automatable
  + Software engineering is a team activity and, for large projects, much time is spent in team interactions. CASE technology does not really support these

7.3 CASE分类

* 分类帮助我们了解CASE工具的不同类型以及它们对软件过程的支持
* 功能角度

Tools are classified according to their specific function

* 过程角度

Tools are classified according to process activities that are supported

* 集成角度

Tools are classified according to their organisation into integrated units

7.4 基于功能的CASE工具分类

7.5 基于活动的CASE工具分类

7.6 基于集成的CASE工具分类

* 工具
  + 支持单个过程活动，例如 设计一致性检查,本文编辑等
* 工作平台
  + 支持一整个过程阶段，例如描述或设计阶段,通常包括多个集成工具
* 环境
  + 支持整个软件过程的全部或某个实质部分。 通常包括一组工作平台

SW03 软件过程 要点

* 软件过程是生产一个软件产品的一系列活动。软件过程模型是这些过程的抽象表示
* 软件过程包括：**软件描述、软件设计和实现、软件有效性验证以及软件进化**
* **软件过程模型主要包括瀑布模型、进化式开发、面向复用的开发**
* 需求工程是开发软件描述的过程
* 设计和实现过程是转换软件描述为一个可运行的软件系统的过程
* 软件有效性验证是检查系统是否与它的描述相一致以及是否符合系统的用户的真实需求的过程
* 软件进化是修改已存在的软件系统以适应用户新的需求的过程
* CASE 技术为软件过程提供自动化的支持

SW04 项目管理

一 、软件项目管理概述

* 软件项目管理：是组织,规划和进度安排软件项目
* 软件项目管理的目标是确保项目符合预算和进度要求，并确保交付的软件能够达到既定的目标
* 需要软件项目管理是因为专业的软件工程总是受预算和工程进度的制约,而这些是开发机构预先安排好的

*软件项目管理的特征*

* 软件产品(项目)是无形的
* 没有标准的软件开发过程
* 大型软件项目常常是“一次性” 的

二、管理活动

* 提出书面建议
* 项目规划和进度
* 项目成本估算
* 项目监督和评审
* 人员选择和评价
* 写作并陈述工作报告

三、项目规划

3.1 项目规划

* 或许是最耗时的项目管理活动
* 是从初始概念到系统交付的连续活动。当有新的信息到来的时候，项目规划可能需要修订
* 以便支持软件项目开发活动的不同活动，如进度、财务状况等，相应的各种类型的开发计划将被开发

3.2 项目计划

* 项目计划包括可用的资源、工作分解以及完成工作的进度安排，项目计划书主要包括：
  + 介绍
  + 项目组织
  + 风险分析
  + 硬件和软件资源需求
  + 任务明细分类
  + 项目进度
  + 监控和报告机制

3.3 项目里程碑和可交付的文档

* 在一个项目中活动应该有确切的输出来判断项目的进展情况
* 里程碑 是过程活动的结束点
* 交付内容 是项目向客户移交的项目结果
* 瀑布过程允许对进展里程碑给出直接定义

四、项目进度

4.1 项目进度

* 项目管理者要估算完成各项活动所需的时间和资源，并按照一定的顺序把它们严密地组织起来
* 项目进度是包括一个项目所有工作分解为若干独立的活动，以及判断完成这些活动所需的时间
* 一定要避免出现因一项关键的任务没有完成而使整个项目延期交付的情形
* 项目进度的估算依赖于项目管理者的直觉和经验

*进度问题*

* 很难估计问题的难度和开发费用
* 软件开发效率不是同人员的数目成比例的
* 增加人员到延迟的项目中会导致项目更大的延迟，因为需要额外的沟通负担
* 意外无处不在！因此，在计划中为偶然性留些余地

4.2 条形图和活动网络图

* 条形图和活动网络图是表示项目进展的图形表示法
* 项目的各项活动应该至少保持一个星期，更细的划分意味着在项目进度的估算和进度表的修订上花太多时间
* 活动网络图表示构成一个项目的不同活动之间的依赖关系
* 条形图可以表示每项活动的负责人是谁以及该项活动预计的开始和结束时间

*活动网络 活动条形图*

*人员分配及时间表*

五、风险管理

5.1 风险管理

* 风险管理是识别风险，并制定计划以最大限度降低风险对项目的影响
* 风险是一些不利因素实际发生的可能性，风险种类：
  + 项目风险 影响进度或资源
  + 产品风险 影响正被开发的软件质量或性能
  + 商业风险 影响组织开发或采购软件

*风险管理过程*

* 风险识别
  + 识别项目，产品和商业风险
* 风险分析
  + 评估这些风险的可能性和后果
* 风险规划
  + 拟订计划避免或将风险的影响减到最少
* 风险监控
  + 在项目活动中监控风险

5.2 风险识别

* 技术风险
* 人员风险
* 组织风险
* 需求风险
* 预算风险
* 估算风险

5.3 风险分析

* 评估风险的可能性和严重性
* 风险出现可能性包括非常小、小、中等、大、非常大
* 风险出现导致的严重性有灾难性的、严重的、可以容忍的、可以忽略

5.4 风险规划

* 项目管理者要考虑已经识别出的每一个重大风险、并确定处理这个风险的策略，策略包括：
  + 规避策略: 风险将会出现的可能性被减少
  + 减到最小限度策略:项目 或产品 上的风险冲击将会被减少
  + 突发事件规划:如果风险发生,突发事件规划是处理 风险的计划

5.5 风险监控

* 风险监控就是要对每一个识别的风险定期进行评估
* 风险监控就是确定风险出现的可能性是变大还是变小，风险的影响是否有所提高或改变
* 每一个重大风险都应该单独评审并在会上进行讨论

SW04 项目管理 要点

* 好的软件项目管理是软件项目成功的重要基础
* 由于软件产品是无形，导致软件项目管理同其它项目管理有明显的区别
* 软件项目管理者承担多种任务，但其最重要的活动是项目规划、估算和进度
* 项目规划和估算是反复的过程，贯穿项目始终的过程
* 项目里程碑是一个项目活动可以预期的结果，到达某一里程碑就要把某些进展报告提交到管理层
* 风险的种类包括项目风险、产品风险、业务风险
* 风险管理是识别风险，并制定计划以最大限度降低风险对项目的影响

SW05 软件需求

一、软件需求概述

1.1 需求工程

* 软件需求是指用户对软件的功能和性能的要求，就是用户希望软件能做什么事情，完成什么样的功能，达到什么性能
* 被对系统应该提供的服务和所受到的约束的描述是系统需求关心的内容。对服务和约束的发现、分析、建立文档、检验的过程—需求工程
  1. 需求类型
* 用户需求
  + 用自然语言加图表的形式描述系统所提供的服务和它的操作约束。 是面向客户的
* 系统需求
  + 用结构化文档详细描述系统的服务。作为客户和承包商之间的合同的一部分

二 、功能需求和非功能需求

2.1功能需求和非功能需求

* 功能的需求
  + Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.
* 非功能的需求
  + constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
* 领域需求
  + Requirements that come from the application domain of the system and that reflect characteristics of that domain

2.2 功能需求

* 描述系统提供的功能性或系统服务
* 依赖软件的类型、软件未来的用户以及开发的的系统类型
* 功能性用户需求，用一般的描述给出，而功能性系统需求应该详细地描述系统服务（输入、输出、异常等）

*功能需求的例子*

* 软件必须提供表达和访问外部文件的手段，这些外部文件是由其他工具创建的
* 系统能提供适当的浏览器供用户阅读馆藏文献

*不严谨的需求*

* 当需求陈述不精确的时候问题就会发生
* 开发者和用户对二义性的需求可能以不同的方式解释
* 注意术语“适当的浏览器”
  + 用户意图 -为每种不同类型的文档设定特殊的浏览器
  + 开发者解释 - 提供一个阅读文本文件内容的浏览器

*需求的全面性和一致性*

* 理论上，系统的功能需求描述应该既全面又具有一致性
* 全面性
  + They should include descriptions of all facilities required
* 一致性
  + There should be no conflicts or contradictions in the descriptions of the system facilities
* 在实际中，是几乎不可能产生一个全面且一致的需求文档

2.3 非功能需求

* 非功能需求定义了系统的特性和约束，如可靠性、反应时间和存储空间等
* 非功能需求不只是与软件系统本身有关，还与系统的开发过程有关。过程需求的例子包括对在过程中一定要用质量标准的描述、设计中必须使用的CASE工具集的描述以及过程所必须遵循的原则（程序语言、开发方法）
* 非功能需求关心的是系统整体特性而不是个别的系统特性，因此，非功能需求比功能需求对系统更关键. 如果非功能需求不能满足，系统将无法使用

*非功能需求的分类*

* 产品需求
  + 叙述被移交的产品所必需具有的特别的行为方式，举例来说执行时间, 可靠性等
* 组织的需求
  + 是组织的政策和程序的结果，所使用的过程标准
* 外部需求
  + 从系统和开发过程外部因素引起的需求，互操作需求和立法需求等

*非功能需求实例*

* 产品需求
  + 它应该能将所有APSE和用户之间的必须的通用标准的Aada字符集表达
* 机构需求
  + 系统开发过程和交付的文档将遵照XYZCo-SP-STAN-95的相关定义
* 外部需求
  + 系统不应该对系统的操作人员公开客户除名字和索引代码外的任何个人信息

*非功能性需求的特点*

* 非功能性需求的常见问题是检验起来非常困难
* 非功能性可能是对系统的易用性、系统的可恢复性和对用户输入的快速反应性能的要求，对这些需求的不详细和不确定会给开发者带来许多问题，比如在系统交付之际会在客户和开发者之间引起争议

2.4 领域需求

* 领域需求起源于系统的应用领域而不是系统的用户需要
* 领域需求是一个新的特有的功能需求，对已存在的功能需求的约束或者是需要实现的一个特别计算
* 领域需求时常反映应用领域的基本问题，如果这些需求不被满足，系统将不能正常运转

*领域需求实例*

* 需求：因为版权限制，一些文档只能根据用户需要输出到本地打印机或网络打印机上。
* 这一需求是版权法对图书馆中资料保护的需要，它描述了对一类文档打印完立即删除的功能

*领域需求问题*

* 可理解性
  + 需求使用应用领域的语言来表达
  + 时常这些不被开发系统的软件工程师所理解
* 隐式描述
  + 领域专家了解该领域，所以他们不会去考虑怎样使领域需求显示给出

三、用户需求

3.1 用户需求

* 用户需求是从用户角度来描述系统功能和非功能需求
* 应该描述功能的和非功能的需求以便让那些没有专业技术知识的系统用户可以理解
* 用户需求使用自然语言，表格和图的形式定义

*用自然语言来描述用户需求时出现的问题*

* 不够清楚

为使文件不难读而失去了精确性

* 需求混乱

功能的和非功能的需求容易混乱

* 需求融合

一些不同的需求可能混在一起表达

3.2 用户需求书写的原则

* 设计一个标准的格式作为对所有需求的描述形式
* 使用一致的语言方式。对于强制性需求用“必须”,对于希望性的需求用“应该”
* 使用文本加亮标识主要需求部分
* 避免使用计算机行话

四、系统需求

4.1 系统需求

* 更详细的用户需求描述
* 作为系统设计的基础
* 会被当作系统的合同的一部分

*系统需求实例*

* 为用户提供外部文件类型的工具
* 每种外部文件类型具有一个相关联的工具
* 每种外部文件类型在界面上用一种专门的图标来表示
* 提供一种工具，使用图标表示由用户定义的外部文件类型
* 当用户选择一个代表外部文件的图标时，选择的效果是将与该外部文件类型相关联的工具启动起来

*需求和设计*

* 原则上讲，系统需求应该陈述系统应该做什么而不包括系统如何实现
* 在实践中，需求和设计总是不可分的
  + 系统体系结构可能用来组织需求
  + 系统可能需要与其它系统统一考虑来产生设计需求
  + 使用特定的设计可能是一个领域需求

*自然语言描述系统需求的问题*

* 不明确
  + 需求的读者和作者对相同文字一定要有同样的解释。 自然语言是不明确的，因此很难做到这一点
* 弹性太大
  + 相同的事物可能以许多不同方式阐述
* 缺乏模块化
  + NL 结构没有充分组织系统需求

五、需求文档

5.1 需求文档(软件需求描述)

* 需求文档是系统开发者所需要的官方陈述
* 应该包括需求的定义和描述
* 它不是一份设计文件。应该尽**最大可能描述清楚什么是系统应该完成的**，而不是描述系统如何完成它

5.2 软件需求文档的要求

* 叙述外部系统行为
* 叙述实现约束
* 容易改变
* 作为维护的参考工具
* 记录对有关系统生命周期的前瞻性考虑，也就是对变更的预见

5.3 IEEE 需求文档的结构标准

* 介绍
* 一般的描述
* 特别需求
* 附加材料
* 索引
* 这是一个一般的结构，对特定的系统还要特别处理

5.4 需求文档结构

* 介绍
* 词汇
* 用户需求定义
* 系统体系结构
* 系统需求描述
* 系统模型
* 系统进化
* 附加材料
* 索引

SW05 软件需求 要点

* 软件的系统需求描述了系统应该做什么以及定义系统运行时和实现时的约束
* 功能需求是有关系统一定要提供的服务
* 非功能需求包括所有开发的系统的约束的产品需求、用于开发过程和外部需求
* 用户需求是为面向购买和使用系统的用户而写的
* 用户需求是用自然语言、表格和图形的形式描述的
* 系统需求要达到能以准确的方式沟通系统必须提供的功能
* 软件需求文档是经过认可的系统需求描述

SW06 需求工程过程

一 、需求工程过程概述

1.1 需求工程的重要性

输入：《合同》/《立项建议书》

输出：《用户需求报告》《需求规格说明书》

*需求分析为什么重要？*

(1)大型系统的失败，最后均归结到需求分析。

(2)《用户需求报告》是一个里程碑/基线。

(3) 需求分析占软件开发工作量的30%左右。

(4) 需求获取中的错误，会发散式的传播。

1.2 需求工程概述

* 软件需求工程是在计算机系统的软件功能分配和软件设计之间起着重要桥梁作用的一项软件工程活动。
* 需求分析（工程）是发现、求精、建模和规约目标系统的过程，即指出软件目标产品必须“做什么”，描述软件系统提供的服务和所受到的约束，是一个对服务和约束的发现、分析、建立文档和检验的过程。

1.3 需求分析的任务

* 需求分析的任务是包括建立并在软件项目计划中细化的软件范围、创建所需数据信息和控制流、操作行为的模型，以及分析可选择的解决方案，并将它们分配到各软件元素中去。

*需求的类型包括哪些？*

1.4 需求工程过程

* 需求过程是一个创建和维持系统需求文档所必需的一切活动的过程(是发现, 分析和验证系统需求的过程)
* 需求过程有4个通用的、高层需求工程的过程活动
  + Feasibility studies
  + Requirements elicitation & Requirements analysis
  + Requirements validation
  + Requirements management

二、可行性研究

2.1 可行性研究概述

* 可行性研究决定所规划的系统是否值得开发
* 可行性研究主要集中在：
  + 研究系统是否符合机构的总体目标；
  + 研究系统能否在现有技术条件、预算和时间限制内完成；
  + 研究系统能否把已存在的其他系统集成

2.2 可行性研究任务

* 可行性研究任务是信息评估、信息汇总和可行性报告。
* 信息评估是找出和分析相关的信息；信息汇总是建立系统的逻辑模型，并从技术可行性、经济可行性、操作可行性和时间可行性等方面探索解决方案；可行性研究报告给出是否要开发系统的意见和建议。

2.3 可行性研究的步骤

* 基于信息的评估(什么是必需)，信息汇总和报告写作
* 对组织中的人们提出下面的一些问题(分析信息)
  + 如果系统没有实现，那将会怎样?
  + 当前的业务处理过程中的问题是什么?
  + 被规划的系统会如何帮助?
  + 系统集成将会有什么问题?
  + 需要新的技术吗? 什么技术?
  + 所规划系统的需要哪些支持支持?

三、需求导出与分析

3.1 需求导出与分析

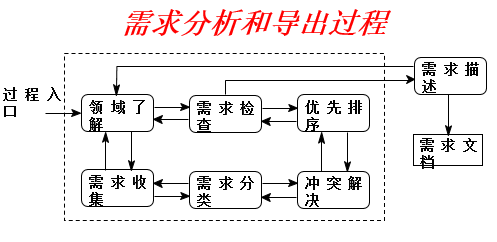
* 需求导出和分析这个活动是软件开发技术人员和客户及系统最终用户一起调查应用领域、即系统应该提供什么服务、系统应该具有升么样的性能以及硬件约束.从一个活动到另一个活动会有持续的反馈，是一个重复的过程
* 所有对系统需求有直接或间接影响力的人员统称项目相关人员，如最终用户、管理者、维护人员、领域专家等，统称项目相关人员

*需求导出与分析面临的困难*

* 用户不知道需要系统要干什么
* 用户用他们自己的专业术语来描述需求
* 不同的用户可能产生矛盾的需求
* 组织的和政治上的因素可能影响系统需求
* 需求在分析过程期间改变。 新的用户出现，系统会有新的需求

3.2 需求导出与分析过程活动

* 领域了解
* 需求收集
* 分类
* 冲突解决
* 优先排序
* 需求检查



3.3 需求导出与分析方法

* 面向视点的方法
* 基于场景的分析方法
* 深入实际的方法

3.3.1 面向视点的导出

* 对于任何中、大型系统，通常有多个不同类型的用（客）户，必然会有不同的视点考虑。从不同视点观察一个问题，可以得到不同的解决方法。
* 多视点的分析是很重要的，因为没有单一正确的分析系统需求的方法

*实例：ATM系统*

* 一个自动柜员机系统，提供自动化的银行服务
* 采用非常简化了的银行客户服务系统, 提供比较狭窄的服务范围
* 具体服务包括现金提取，信息传递 (送一个信息请求一个服务),余额查询和资金划转

*实例：ATM系统相关人员*

* 银行客户
* 其他的银行代表
* 硬件和软件维护工程师
* 银行经理和柜台职员
* 数据库管理员和信息安全管理者
* 市场开发部

1. 视点的类型

* 数据来源或接收者
  + 视点负责产生或消费数据。分析过程包括对数据产生和消费的检查，并假设数据的源和接收者都是有效的
* 表示框架
  + 视点表现特别的系统模型类型。不同的分析方法会对被分析的系统有不同的理解
* 服务的接收者
  + 视点是位于系统的外部并接受从它来的服务。特别适合于交互式系统

*外部视点*

* 交互系统提供服务给最终用户，将视点看做系统服务接收者对交互系统分析是有效的：
  + 自然把最终用户当做系统服务的接收者
  + 视点是构造需求导出的一个自然的方法
  + 比较容易决定一个视点是否是有效的

1. 面向视点的需求过程

* 视点识别
  + 发现接受系统服务的视点，并识别向每个视点提供的服务
* 视点组织
  + 聚集相关的视点成一个层次。 共同服务在高层中提供
* 视点文档编写
  + 精炼被识别出的视点和服务的描述
* 视点-系统映射
  + 将分析转换为系统的设计

*面向视点的导出总结*

对于有多视点（客户）的需求分析过程，关键是发现众多视点的存在，考察不同视点接收的服务，收集这些信息并提供一个框架以解决不同视点提出的需求冲突。具体分为视点识别、视点组织、视点文档编写和视点系统映射四部分内容。

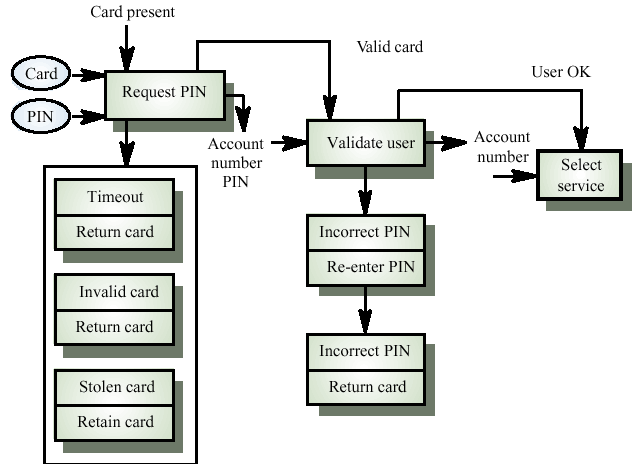
3.3.2 基于场景的方法

* 场景是描述一个系统在实际中如何使用
* 场景对需求导出是有帮助的，因为人们能很自然地将系统映射为这些行为，比从一个系统中抽象出所需要的描述容易得多
* 场景对于我们将导出概概括性的需求描述显得特别重要

1. 场景描述

* 场景开始时的系统状态
* 场景中的一般事件流程
* 哪些会出错以及如何处理
* 同其他协作活动
* 场景完成时的系统状态

1. 事件场景

* 事件场景可以用来描述系统如何响应一些特别的事件的发生，例如 “开始交易”
* 为事件场景给出了一个图形的约定
  + 发送和接收的数据
  + 控制信息
  + 异常处理
  + 下一个预期的事件

*数据和控制分析的符号*

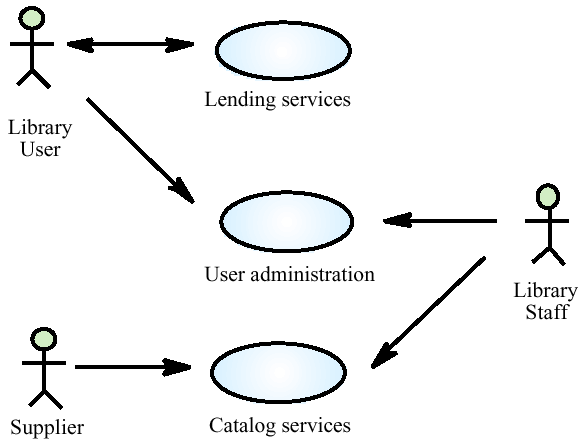
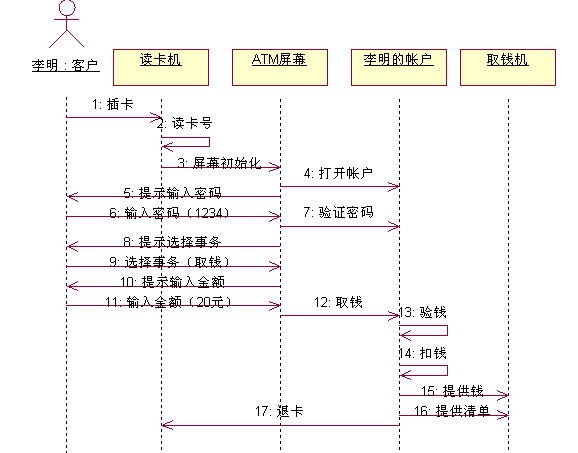
* 椭圆：来自于视点的数据和移交到视点的数据
* 控制信息从方块的顶端进入和离开
* 数据从每个方块的右侧离开
* 异常标记在每个方块的底部
* 下个事件的名字在一个有阴影的方块中给出

*异常描述*

* 在这个例子中，异常是
  + 超时：用户在规定时间内没有能够输入个人身份号码
  + 无效的卡：卡无法得到识别并退回
  + 偷来的卡：识别出卡是偷来的，因而机器将其没收

1. 用例（USE-CASE）

* 用例是一种基于场景的需求导出技术
* 用例是基于 UML 技术的场景，识别出交互的参与者并描述交互本身
* 可以使用序列图把更详细的需求加入到用例中

3.3.3 深入实际的方法

* 一位社会学家花相当的时间观察和分析人们是如何实际工作的
* 因为人们无法解释或清楚地说出他们的工作
* 重要的社会和组织的因素可能得到观察
* 深入实际研究揭示了这样一项内容，工作通常比简单的系统模型所描述的更丰富和更复杂

*原因：* 深入实际

* 需求来自于人们实际的工作而非过程定义中对它们的要求
* 需求来源于合作和对其他人们活动的了解

*例子*

* 软件系统在社会和组织的环境中使用，从而社会和组织会影响甚至是支配系统的需求
* 考虑一个允许高级管理者不经过中层管理者而直接存取数据的系统
  + 管理的状态:资深管理者可能感到使用键盘是太重要的了，这可能限制了系统所使用的界面的类型
  + 管理的责任:管理者可能没有一段完整的时间用于学习系统的使用
  + 组织的抵抗:那些将被架空的中间管理层人员会不遗余力地提供造成误导的和不完全的信息，这样系统将会失败

3.4 需求导出与分析总结

* 获取需求的注意事项
  + 识别真正的客户.
  + 正确理解客户的需求
  + 具备较强的忍耐力和清晰的思维
  + 说服和教育客户
* 获取需求的模型

四、需求有效性验证

4.1 需求有效性验证

* 致力于论证需求定义的系统是客户真的想要的
* 需求错误的代价是很高的，因而需求有效验证是非常重要的
  + 在移交之后修补一个需求错误可能花费高达 100 倍于一个实现错误的费用

4.2 需求有效性验证方法

* 需求检查
  + 对需求文档中定义的需求执行多种类型检查
* 需求评审
  + 对需求进行系统的手工分析
* 设计原型
  + 使用系统的一个可运行的模型检查需求。 Covered in Chapter 8
* 测试用例产生
  + 为需求开发测试检查 易测性

*需求检查*

* 有效性：Does the system provide the functions which best support the customer’s needs?
* 一致性：Are there any requirements conflicts?
* 完备性：Are all functions required by the customer included?
* 现实性：Can the requirements be implemented given available budget and technology
* 可检验性检查：Can the requirements be checked

*需求评审*

* 当需求定义被明确表达出来之后 , 评审应该进行
* 客户应该参与到评审中
* 评审可能是正式的或非正式的。

*评审内容*

* 可测试性。 Is the requirement realistically testable?
* 可理解性。 Is the requirement properly understood?
* 可跟踪性。 Is the origin of the requirement clearly stated?
* 适应性。 Can the requirement be changed without a large impact on other requirements?

4.3 需求有效性验证总结

* 需求是正确的吗？
* 需求是一致的吗？
* 需求是完全的吗？
* 需求是实际可行的吗？
* 需求是客户需要的吗？
* 需求是可检验的吗？
* 需求是可跟踪的吗？
* 最后的签字

五 、需求管理

5.1 需求管理与项目管理的关系

* 项目需求是制定项目计划，开发项目产品和从事项目活动的依据。
* 项目的计划、项目的开发活动及开发的产品应与项目需求保持一致，随需求的变化而调整。

5.2 需求管理

* 需求管理是在需求工程过程和系统开发期间对需求管理的过程，主要是管理变更需求的过程
* 需求不可避免会出现不完全的和不一致的
  + 当业务需要变化，新的需求浮现了
  + 不同的视点有不同的需求

5.3 需求变化原因

* 来自不同用户群的需求优先级在开发期间是会改变的
* 在开发期间系统的业务和技术环境将发生改变

5.4 需求分类

* 从需求进化角度，需求的分类:
  + 持久的需求:从客户机构中的核心活动中导出的需求。举例来说，医院总是有医生、护士和病人。主要来源于领域模型
  + 易变的需求：在开发和使用过程中变更的需求，如在一所医院中，需求来自于医疗政策的需求

*易变需求分类*

* 易变的需求
  + 源于系统环境改变的需求
* 整体需求
  + 对系统的了解加深后所浮现的需求
* 信息化需求
  + 起因于计算机系统的导入的需求
* 兼容性需求
  + 依赖其他的系统或组织过程的需求

5.4 需求管理规划

在需求管理阶段，必须做好以下计划：

* 需求识别
  + 需求标识
* 变更管理过程
  + 需求变更所遵循的过程
* 可追溯性策略
  + 维护的需求关系的策略
* CASE工具支持
  + 需求变更管理所需要的支持工具

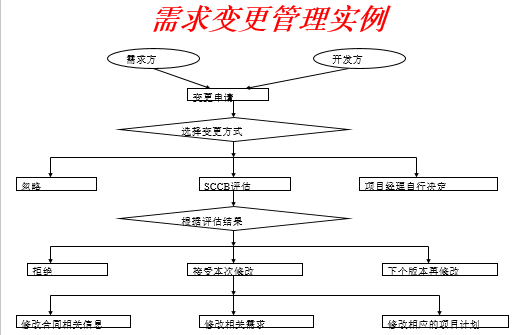
*可追溯性信息*

* 可追溯性是关注需求的来源和系统设计之间的关系
* 来源的可追溯性
  + 需求到提出这些需求的信息持有者之间的关联
* 需求可追溯性
  + 相互依赖的需求之间的关联
* 设计可追溯性
  + 从需求到设计的关联

*CASE工具支持*

* 需求保存
  + 需求如何安全地存储
* 变更管理
  + 变更管理的工作流程，(现实：电子流实现)
* 可追溯性管理
  + 需求之间的关联

5.5 需求变更管理

* 管理和控制需求变更的过程
* 需求变更管理应该处理全部的需求变更
* 主要阶段
  + 问题分析：Discuss requirements problem and propose change
  + 变更分析和成本计算：Assess effects of change on other requirements
  + 变更落实：Modify requirements document and other documents to reflect change

六 、需求描述

6.1 需求描述

* 需求描述是对系统开发需求的正式表述，以需求规格说明文档（或称为软件需求说明书）的形式给出。
* 需求规格说明文档（SRS，Software Requirement Specification）是需求分析任务的最终“产品”。SRS是软件工程项目的第一份，也是最重要的一份文档。
  + 相当于客户和开发商之间的一项合同；
  + 精确地描述产品做什么，以及产品的约束条件等；
  + 给软件设计提供了一个蓝图，给系统验收提供了一个验收标准集。
* SRS 的读者范围广泛，有系统用（客）户、管理者、需求分析工程师、系统测试工程师和系统维护工程师等。

6.2 SRS的写作指南

IEEE/ANSI 830-1993标准和我国GB 856D-1988国家标准，给出了需求规格说明文档（SRS）的内容框架。

6.3 需求描述总结

需求规格说明文档（SRS）是需求分析的“产品”，是软件工程项目的第一份，也是最重要的一份文档。它是系统需求的全面描述，是客户与开发商之间的合同，是系统验收、开发和维护的基础。

SW06 系统需求工程 要点

* 需求工程过程包括一个可行性研究、需求导出和分析、需求有效性验证以及需求管理
* 需求分析是一个包括领域了解、需求收集、分类、组织、优先排序和有效性验证的重复过程
* 不同的项目相关人员对系统有不同的需求
* 社会和机构的因素对系统需求有强大的影响
* 需求有效性验证是检查需求的有效性、一致性、完备性、现实性和可检验性的过程
* 业务上、机构上和技术上的变更不可避免地导致软件系统需求的变更
* 需求管理过程包括管理规划和变更管理

SW07 系统模型

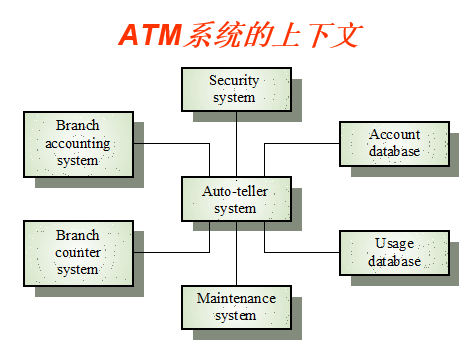
一、系统模型概述

1.1 系统模型

* 系统模型：系统经过需求分析后的抽象描述
* 系统模型能帮助非专业人士理解系统的功能，有利于同客户人员交流
* 不同的模型从不同的角度展现系统
  + 外部角度：展现的是系统的上下文或环境(建模)
  + 行为角度：展现的是系统的行为(建模)
  + 结构角度：展现的是系统的体系架构和数据结构(建模)

1.2 模型类型

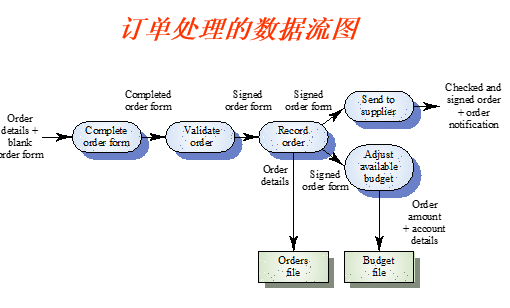
* 数据处理模型：展现数据如何在不同的阶段被处理的
* 组成模型：展现实体是如何由其他实体组成的
* 体系结构模型：展现整个系统的主要的子系统
* 分类模型：展现实体是如何拥有共性特性的
* 刺激/ 响应模型：展现系统是如何对事件作出反应的

二、系统上下文模型

* 上下文模型通常说明系统的边界
* 社会和组织的因素,系统的边界充满了非技术因素
* 体系结构模型展现一个系统和它其他系统的关系(定义系统上下文和系统与环境之间的依赖关系)
* 简单的体系结构模型一般要以其他模型共同来描述

三、行为模型

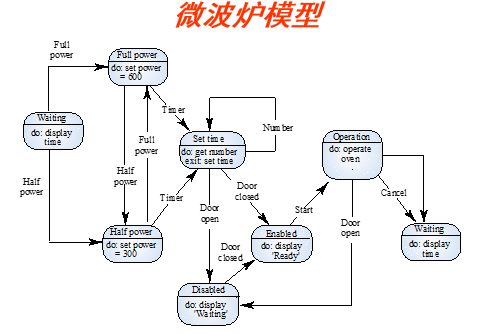
* 行为模型用来描述系统的全部行为
* 行为模型的两种类型
  + 数据处理模型：展现数据在系统中移动时如何被处理的
  + 状态机模型：说明系统如何响应事件的
* 这两个模型对系统的行为描述是必需的

3.1 数据流模型

* 数据流模型是描述数据处理的直观的方式
* 数据流模型用来描述数据是如何在系统中流动的
* 数据流模型是需求分析方法的重要组成部分
* 数据流模型采用简单和直观的符号，以利于客户的理解

*数据流图*

* 上述模型说明了货物订单是如何在各个过程之间流动的，同时描述了其中的数据存储
* 数据流模型的价值主要体现在它对系统中的数据和数据存储在特定的过程中的流动的跟踪和记录，有助于系统分析人员理解系统中到底发生了什么
* 数据流模型是从功能角度来看待系统而得到的模型，对数据的每一个变换用一个处理过程来描述。它不仅可以用来描述系统内的处理过程，有时还能有效地描述系统的上下文。数据流模型可以描述不同系统间以及子系统之间是如何交换信息的

3.2 状态机模型

* 状态机模型是一种描述系统对外部和内部事件响应的行为建模
* 这中系统多是由外界的激励而驱动的，通常用于对实时系统的建模
* 状态机模型中系统状态用节点表示，而事件用这些节点之间的弧来表示。当事件发生时，系统从一个状态转移到另一个状态

四、数据模型

4.1 数据模型

* 用来描述被系统处理的数据的逻辑结构
* 实体-关系-属性模型，它描述数据实体、关联属性以及实体之间的关系
* 广泛使用于数据库设计。

*数据字典*

* 数据字典列出系统模型中所使用的所有名字。包括对实体的描述，对关系的描述以及对属性的描述
* 优点
  + 支持名字管理避免重复
  + 存储了机构关于分析，设计和实现的信息

五、对象模型

* 对象模型用对象类来描述系统
* 一个对象类是对一组对象的抽象化，这些对象具有公有属性和公有的服务 (操作)
* 对象模型
  + 继承模型
  + 聚合模型
  + 交互模型

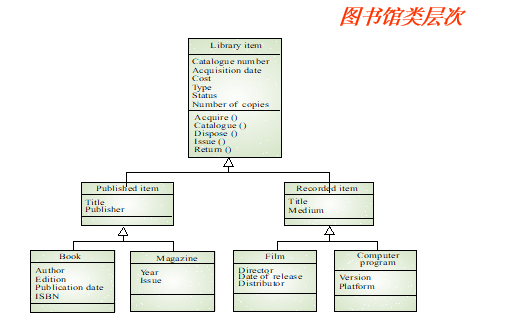
*对象模型*

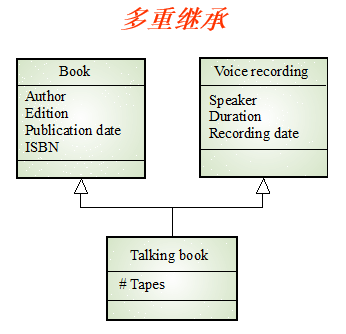
* 是反映被系统操纵的真正 世界实体的自然方式
* 使用这种方法，越是抽象的实体越是难以建模
* 对象类的识别是个困难的过程，需要对应用领域有深刻理解
* 反映领域实体的对象类在系统中是可以重复使用的

1. 继承模型

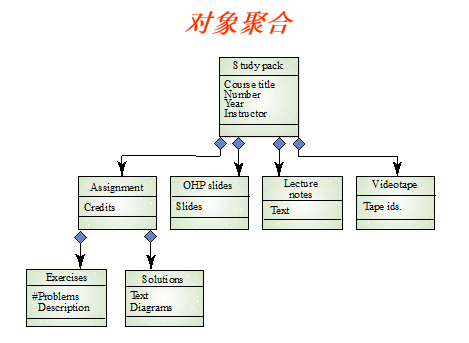
* 面向对象建模的一个任务是找出所研究领域中那些重要的对象，然后将这些对象进行分类
* 为显示对象分类，对象类被组织到一个类继承图中，图中最一般的对象类居于层次关系的顶端，较专门的对象类继承通用类中的属性和服务，又具有自身的属性和服务
* 类层次结构的设计是困难的

*UML*

* 是由使用面向对象分析和设计方法的开发者所设计的
* 已经变成面向对象 建模的一个有力的标准
* 记号法
  + 对象类用一个长方形来表示,在顶端是名字的，属性在中间，在底部的区段中是操作
  + 在对象类 之间的关系( 关联)用对象之间的连线表示
  + 继承是一个泛化，在层次结构中表现为“向上”而不是“向下”

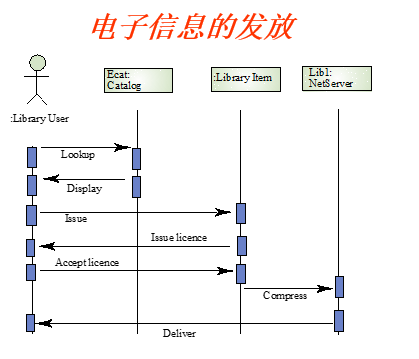


*多重继承*

* 不是从一个单个父类继承属性和服务,一个支持多重继承的系统允许对象类从一些超类中继承
* 能导致语义冲突，在不同的超类中相同名字的属性/ 服务有不同的语义
* 使类层次结构重新组织是更困难的事

1. 对象聚合

* 聚合模型展现如何由其他的类组合成一个新的类
* 很类似于语义数据模型当中的part-of关系

1. 对象行为建模

* 行为模型描述对象之间的交互，来产生系统行为
* 在 UML 中序列图 (或合作图) 用来对对象之间的交互作用建模

SW07 系统建模 要点

* 模型是系统的抽象视图
* 上下文模型给出了要建模的系统在整个环境中与其他系统和过程间的位置关系
* 数据流模型可以用来描述系统中数据的加工过程
* 状态机模型用来描述系统的行为，以响应内部和外部的事件
* 语义数据模型描述系统导入和导出的数据的逻辑结构
* 对象模型描述系统中实体及其分类关系和聚合关系

SW08 原型系统开发

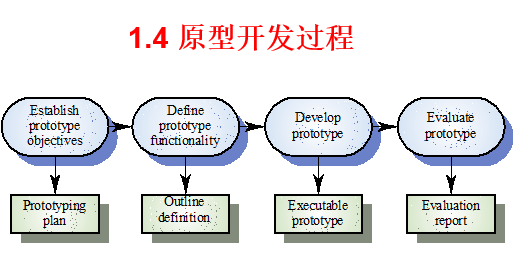
一、软件原型

1.1 系统原型

* 软件客户和最终用户觉得他们的需求很难表达
* 系统需求分析会减少系统开发目标的不确定性，然而，要真正使需求分析得到对方的认可，必需要有一个系统原型
* 原型是系统的初始版本，是一个快速开发的系统

1.2 软件原型支持需求工程

* 软件原型支持需求工程的两个活动：
  + Requirements elicitation. Users can experiment with a prototype to see how the system supports their work
  + Requirements validation. The prototype can reveal errors and omissions in the requirements
* 原型开发可以作为风险分析和降低风险的技术
* 原型系统是需求工程过程的一个组成部分



* 1. 原型系统的好处
* 软件开发人员和用户之间的理解偏差在功能展现时显露出来
* 在原型设计中发现需求的不完善和不一致
* 可迅速地展现一个简单的应用系统
* 通过原型系统可以导出系统需求
* 原型系统可以支持用户培训和系统测试

1.5 原型系统在软件过程的优点

* 提高了系统的实用性
* 使系统需求与用户需求更贴近
* 提高了系统的设计质量

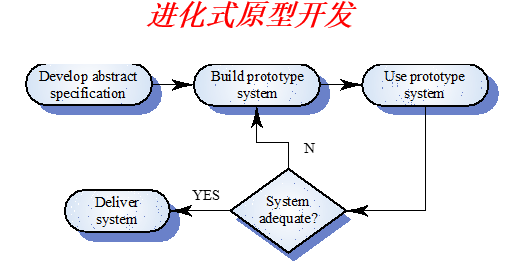
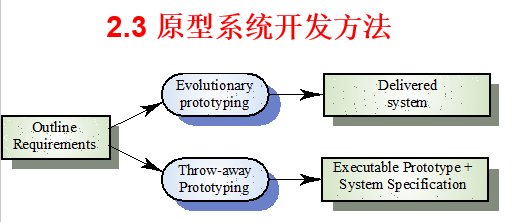
二、软件过程中的原型开发

2.1 软件过程中的原型开发

* 进化式模型(Evolutionary Model)
  + 进化式模型开发从一些主要简单的开始，在对原型的讨论过程中不断发现新的需求、添加新的功能、逐步完善原型、最终该原型变成了一个完善的、满足所有需求的系统
* 抛弃式模型(Throwaway Model)
  + 抛弃式模型是帮助提炼和澄清系统描述，一旦需求描述完成，原型就不再使用，而是被抛弃掉。

2.2 原形系统开发的目标

* 进化式开发的目标是给用户一个实用的系统。原型开发必须从对用户需求把握最准确的需求做起，最优先处理这部分工作
* 抛弃式开发的目标是导出和验证需求。应该从理解得不够好的那部分需求开始实现，因为你的目标是从中发现问题，对明确的需求就没必要去做原型



1. 进化式原型开发

* 进化式开发的系统的系统需求事先不能完全确定
* 首先设计一个全局性框架,然后建立一个初始的稳定模型,运行该原型,并反复添加、修改和完善,逐步向目标系统进化.每一次进化,都产生一个逼近目标的新版本.
* 该模型在进化过程中,始终强调可靠性、可维护性、适应性和效率等软件质量问题.
* 该模型的最终成果就是投入使用的系统

*进化式开发的优势*

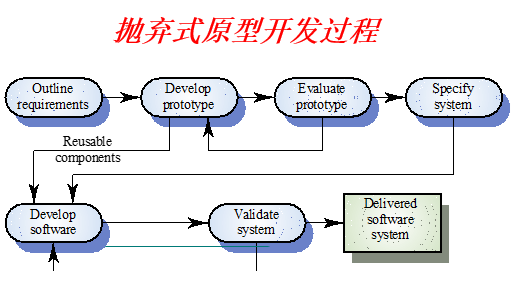
* 加快系统交付的进度
  + 快速的交付比提供完备功能或保证长期可维护性更重要
* 用户的参与
  + 这种方法的优点是客户可跟踪开发过程,及时了解并建议修改开发的过程,最终系统能令用户满意

*进化式开发的基本特性*

* 系统描述、设计和实现交叉进行的
* 系统是逐渐递增的
* 采用了快速开发技术（CASE工具和4GL ）
* 系统用户界面都是用交互式开发系统来实现的

*进化式开发的主要问题*

* 管理问题
  + 现有的软件管理是参照瀑布模型的过程进行管理的
  + 可能涉及到一些不熟悉的技术，管理者会感到有的开发团对使用起来有困难
* 维护问题
  + 连续不断的修改可能导致系统结构的崩溃，增加维护的成本
* 契约问题
  + 客户和软件开发商之间正规的契约是基于系统描述的

1. 抛弃式原型开发

* 抛弃式原型开发降低了需求风险
* 首先制定一个开发计划,然后进行需求分析、设计、编码、测试,由用户或客户评价,如果不满意再返回需求分析部分或设计部分或编码部分,直到满意为止
* 该模型开发的结果不是为最后使用,而是为了理解需求和开发方法,验证那些模糊的、没有把握的需求.
* 任务完成后该模型被抛弃,真正的系统开发将依据该模型的结果从头开始

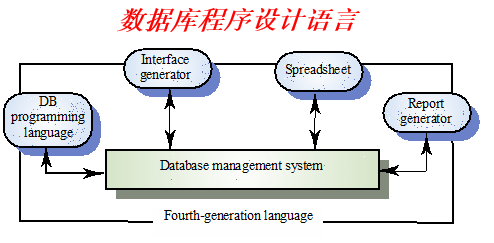
*原型交付*

* 开发人员来自管理者的压力，有时不得不交付抛弃式原型给用户使用
* 这种做法是不明智的：
  + 在原型开发过程中，不太可能使原型的非功能需求满足用户
  + 在原型开发过程中，缺少开发文档
  + 原型开发中的变更可能破坏系统的结构
  + 机构内的质量标准对原型往往不加限制

三、快速原型开发技术

3.1 快速原型开发技术

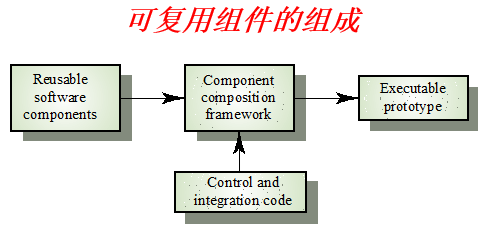
* 常用的快速原型开发技术：
  + 数据库编程
  + 组件和应用集成
* 在实践中，不同的技术可能同时使用
* 许多原型开发系统支持可视化设计方法，在这种开发环境中，原型开发是交付进行的

1. 数据库程序设计

* 绝大多数商业应用处理的是对数据库数据的操作以及包括组织数据、格式化数据的输出
* 为了支持这类应用的开发，商用的数据库管理系统都支持数据管理程序设计
* 数据库程序设计语言使用专门的语言，4GL，指的是数据库程序设计语言和支撑环境

1. 组件和应用集成

* 如果系统中许多部分都可以复用而且不需要重新设计和实现，那么开发的时间将会缩短
* 采用这种组件机制必须包括控制设施以及组件之间通信的机制
* 利用可复用组件的原型开发需要在系统描述中说明哪可复用组件是可利用的



*基于复用的原型开发*

* 应用层开发
  + 整个应用系统与原型结合在一起，功能模块可以共享
  + 例如：如果原型需要一个文本处理，它可以通过在其中集成一个标准的文字处理系统来达到这种功能
* 组件层的开发
  + 单个组件集成进标准的框架从而完成系统构造
  + 这个框架可以是脚本语言和通用的集成框架

四、用户界面原型开发

4.1 用户界面原型开发

* 图形化用户界面目前已经成为交付系统的标准界面系统
* 用户界面的描述、设计和实现在应用系统开发成本中占有相当大的比重
* 从软件工程的角度看，原型开发是用户界面设计过程的基本部分
* 界面生成器是一个图形化屏幕设计系统，界面组件包括菜单、图标、按钮等
* 越来越多的用户界面设计成基于Web的界面

需求分析的原型技术总结

* 采用原型法快速建立原型系统是需求分析工程行之有效的一种需求分析策略。
* 采用快速原型法的需求分析增加了客户和开发者之间的相互交流，尤其是客户和快速原型开发组之间的交流，并几乎一直保持这种交流，直到原型系统得到客户的许可。

SW08 原型系统开发 要点

* 系统原型能给最终用户关于系统功能的一个直观印象
* 随着对软件交付时间的需求越来越紧，原型开发技术逐渐成为中小规模系统开发的标准开发技术
* 抛弃式原型开发是为理解系统需求而做的原型开发；进化式原型开发是对原型不断改进直到成为最终系统
* 快速开发对原型系统非常重要，为了尽快交付原型，必须先甩掉一些功能，或者是放松一些非功能约束
* 原型开发的技术包括使用数据库编程语言以及利用可复用的组件的原型构造
* 用户界面开发总需要使用原型开发方法，因为界面不可能通过静态模型有效地定义

SW09 软件设计概念

一、软件设计概述

* 软件需求分析解决“做什么”的问题，软件设计过程则解决“怎么做”的问题
* 软件设计是把软件需求变换成软件表示的过程，它主要包含两个阶段：软件体系结构设计阶段和部件级设计
* 对比：早期的软件设计局限于数据结构和算法层面上，抽象层次比较低

*为什么要进行软件系统设计*

* 软件系统实现的高层方案
  + 软件系统越来越复杂，需要将其划分为若干部分分而治之—模块化
    - 不同的小组或开发者负责不同的部分
    - 然后在系统层面上进行集成
  + 负责不同部分的开发者对于其它模块需要了解的信息越少越好—抽象与信息隐藏
  + 这些部分之间还需要定义清晰、明确的接口—接口设计

*软件系统设计的重要性*

* *软件系统设计是开发者之间的分工和合作的基础*
* *设计方案是决定系统质量的主要因素*
  + *设计方案可以借鉴成熟的设计经验*
  + *设计方案完成后需要经过质量分析和评审*
* *好的设计方案不仅能很好地支持当前需求的实现，而且能减小未来的系统维护(理解、修改、扩展)成本*

*软件设计的任务*

在软件分析模型中的需求信息(数据、功能和行为)基础上，产生软件系统各个层次上的设计方案

* 数据/类设计：将分析-类模型变换成类的实现和软件实现所需要的数据结构
* 体系结构设计：体系结构设计定义了软件的主要结构元素以及相互之间的关系
* 接口设计：接口设计描述了软件内部、软件和协作系统之间以及软件同人之间如何通信
* 部件级设计：部件级设计将软件体系结构的结构性元素变换为对软件部件的过程性描述

*设计任务1—数据/类设计*

* 基础：在需求中定义的数据对象和关系(实体及实体关系)以及数据字典中描述的详细数据内容(详细信息)
* 数据设计的过程
  + 为在需求分析阶段所确定的数据对象选择逻辑表示
  + 确定对逻辑数据结构所必需的那些操作的程序模块

*设计任务2—体系结构设计*

* 定义软件的整体结构：软件部件、外部可见的属性和它们之间的关系
* 体系结构设计：可以从系统规约、分析模型和分析模型中定义的子系统的交互导出

*设计任务3—接口设计*

* 接口设计主要包括三个方面
  + UI：用户和计算机间的接口(界面)
  + 和外部系统、设备、网络或信息生产者和消费者(比如外部实体)之间的接口
    - 外部系统接口：银行网上支付接口
    - 设备接口：读卡器、扫描枪、传感器接口
    - 信息接口：需要导入/导出的数据接口
  + 设计软件内部各个部件间的接口

*设计任务4—部件级设计*

* 部件级设计完整地描述每个软件部件的内部细节
  + 部件整体的处理和执行流程
  + 部件内本地数据对象的数据结构
  + 部件内处理过程的算法
* 从类为基础的模型、流模型、行为模型中得到的信息是部件设计的基础

*软件设计的(质量)目标*

* 满足用户需求
  + 必须实现分析模型中所有的显式需求
  + 必须满足用户希望的所有隐式需求
* 可读、可理解：设计必须是可读、可理解的，使得将来易于编程、易于测试、易于维护
* 全面性：应从实现角度出发，给出与数据、功能、行为相关的软件全貌

*衡量设计的技术原则*

* 分层的设计结构：从而建立软件系统的分层控制结构
* 模块化：从逻辑上将软件划分为完成特定功能或子功能的部件
* 数据与过程：设计既包含数据抽象，也包含过程抽象
* **高内聚**：建立具有独立功能特征的模块
* **低耦合**：尽量降低模块与外部环境之间的接口复杂度
* 系统化/方法化：设计应能根据软件需求分析获取的信息，建立可驱动、可重复的方法

*软件设计的过程*

* 制定设计规范
  + 通过阅读系统需求说明书确定设计目标以及这些目标的优先顺序
  + 设计方法：结构化、面向对象…
  + 设计文档的编制标准
  + 基本的实现规范：代码的信息形式、与硬件及操作系统的接口规约、命名规则
* 迭代展开、不断细化的系统设计
  + 体系结构和接口设计
  + 数据/类设计
  + 部件级(过程)设计
* 编写设计文档
* 设计评审

二、软件设计原则

2.1 软件设计的基本原则

* 抽象：考虑高层问题时忽略低层细节
* 逐步求精：不断接近问题的完整解
* 模块化：将系统划分为相对独立但又有所关联的多个部分
* 信息隐藏：对其它模块隐藏内部细节
* 模块独立：模块功能相对独立—高内聚/低耦合

**(1) 抽象**

* 控制复杂性的基本策略
* 抽象过程：从特殊到一般的过程，上层概念是下层概念的抽象，下层概念是上层概念的精化和细化
* 软件开发过程的每一步都是对较高一级抽象的解作一次具体化的描述
  + 与实现无关的需求—考虑特定实现技术的设计、将每个模块逐一编码实现……

*过程抽象与数据抽象*

* 过程抽象：功能角度的抽象
  + 使用者将功能体作为单个功能看待
  + 这些功能实际上是由一系列更低级的功能或代码来实现的
  + 过程抽象例：函数、功能性的类/对象
* 数据抽象
  + 定义数据类型和施加于该类型对象的操作
  + 限定了对象的取值范围，只能通过这些操作修改和观察数据
    - 例如面向对象的person类封装实际年龄，public接口上开放的是转换为老年/中年/青年/少年…后的年龄段信息

**(2) 逐步求精**

* 把问题的求解过程分解成若干步骤或阶段，每步都比上步更精化，更接近问题的解法
* 常与分层抽象的思想相结合
  + 抽象使得设计者能够描述过程和数据而忽略低层的细节
  + 求精有助于设计者在设计过程中揭示低层的细节
  + 高层抽象将在下层不断精化，最终得到软件实现

**(3) 模块化(分解)**

* 按照设计原则将系统划分为若干个较小的模块
  + 相互独立但又相互关联
  + 实际上是系统分解和抽象的过程
* 模块是相对独立的程序体
  + 是数据说明、可执行语句等程序对象的集合
  + 单独命名的，并且可以通过名字来访问
    - 例如：类、过程、函数、子程序、宏等

*通过模块化降低开发复杂度*

* C(x)：问题x的复杂性
* E(x)：解决问题x所需工作量
* 对于两个问题p1 和p2：
* 1) 如果C(p1)>C(p2) 那么 E(p1)>E(p2)
* 问题越复杂解决问题所需要的花费更多
* 2) C(p1+p2)>C(p1)+C(p2)因此E(p1＋p2)>E(p1)+E(p2)
* 将复杂问题分解成可以多个子问题分别解决会更加容易(模块化思想的依据)

*模块化例：智能灌溉*

* 总体设计：将系统划分为传感器、灌溉决策和灌溉设备控制三个模块，定义好相互之间的数据和控制接口
  + 传感器模块：接受传感器收集的温度、湿度、土壤等各种数据
  + 灌溉决策：根据种植经验和收集到的数据判断灌溉决策
  + 灌溉设备控制：根据决策控制灌溉设备进行灌溉

降低整体复杂度：可以分而治之而且

可以由具备不同专业知识的开发者分别承担

*误区：无限制地进行模块化分?*

* 如果我们无限制地划分软件，开发它所需的工作量会变得小到可以忽略?
* 结论：NO
  + 模块数量增加时，只是使各个子模块的工作量之和有所减小
  + 然而开发工作量还有很大一部分来自于模块间的接口和集成
    - 除了技术上的接口和集成，还包括人与人之间的沟通
    - 集成和沟通的开销到了一定程度就会成为开发工作量的主要部分

*结论：适度的模块化*

* 模块数增加时，模块间的关系也随之增加，接口和集成的工作量也随之增加
* 结论：寻找最佳模块化程度平衡点

1. **信息隐藏**

* 每个模块都尽量对其他模块隐藏自己的内部实现细节
  + 模块内部的数据和过程不允许其它不需要这些信息的模块使用
  + 定义和实施对模块的过程细节和局部数据结构的存取限制
  + 典型的信息隐藏：面向对象的访问控制符
* 信息隐藏是实现抽象/模块化机制的基本支撑

1. **模块独立(高内聚/低耦合)**

* 模块独立是模块化的根本要求
  + 模块完成独立的功能：明确可辨识
  + 高内聚：内部结构紧密
  + 低耦合：模块间关联和依赖程度尽可能小，与其他模块的接口简单
  + 符合信息隐蔽和信息局部化原则

模块独立的重要性

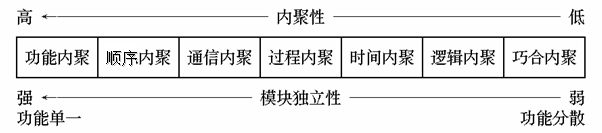
* 模块的开发者专注于某一个相对独立的部分
* 不用过多关心其他模块
* 修改和bug影响的范围被局部化
* 单个模块更容易复用
* 独立的模块更易于维护和测试

模块独立性的指标

* 内聚度与耦合度
  + 内聚(cohesion)：一个模块内部各个元素彼此结合的紧密程度  
    —尽量高
  + 耦合(coupling)：模块之间相互关联的程度  
    —尽量低

内聚度的七个层次

* 巧合内聚(偶然内聚)：将几个模块中的相同程序代码段独立出来建立的模块(无明显独立性)
* 逻辑内聚：完成一组逻辑相关任务的模块，由控制型参数来确定执行哪一种功能
* 时间内聚：模块中的多个任务必须在一段时间内先后执行(无明确的过程约束)
* 过程内聚：模块内的多个任务必须按指定的过程执行
* 通信内聚：模块内所有处理元素都集中在某个数据结构的一块区域中(例如对课程进行选、退课和查询)
* 顺序内聚：指一个模块完成多个功能，这些功能又必须顺序执行(更加单一的过程内聚)
* 功能内聚 ：指一个模块中各个部分都是为完成一项具体功能而协同工作，紧密联系，不可分割的(单个功能)



模块独立性：高内聚低耦合

* 模块独立性的模块：高内聚低耦合的模块
  + 一个模块内部各个元素之间的联系越紧密，则它的内聚性就越高
  + 模块之间的连接越紧密，联系越多，耦合性就越高，而其模块独立性就越弱

三、软件体系结构设计

3.1 软件体系结构设计概述

* 体系结构设计是识别组成系统的子系统和子系统控制和通信框架的设计过程
* 软件体系结构关注系统的一个或多个结构，包含软件部件、部件对外可见的属性以及部件间的关系
* 输出是软件体系结构的一个描述

*软件体系结构设计*

* 体系结构设计的过程是为软件系统建立一个基本框架
* 体系结构设计的过程包括要识别出系统的主要组件以及这些组件之间的通信
* 是系统设计过程的一个早期阶段
* 是需求描述和系统设计之间的桥梁
* 常与一些需求描述活动齐头并进

*清晰的体系结构带来好处*

* 项目相关人员之间的沟通
  + 体系结构可能被当作系统信息持有者之间讨论焦点(桥梁)
* 系统分析
  + 实际是对系统的分析过程，系统体系结构的设计对系统能否满足非功能需求具有极深的影响,有利于系统设计的前期决策
* 大规模复用
  + 体系结构可能在一定的系统范围内再度复用,可传递、易于理解的系统级抽象

*体系结构的设计过程*

* 系统结构化
  + 系统被分解成一些主要的子系统，同时识别出他们之间的通信
* 控制建模
  + 建立了系统不同部分之间的控制关系的一般模型
* 模块分解
  + 将被识别的子系统进一步分解成模块(组件)

*子系统和模块*

* 一个子系统独立构成一个系统，不依赖于其它子系统提供的服务。
* 一个模块是一个系统组件，它能提供服务给其他组件，但是不能认为是一个独立的系统

*体系结构的属性*

* 性能
  + 局部化操作将子系统通信减到最少
* 保密性
  + 使用一种分层的体系结构，让关键性的部分位于内层中
* 安全性
  + 隔离安全性要求极高的成份
* 可用性
  + 包括体系结构的冗余成份
* 可维护性
  + 使用细粒度，自我包含的成份

3.2 体系结构类型

* 单主机结构
  + 界面、数据和程序集中在单台主机上
  + 不需要考虑多用户并发操作的问题
* *客户/服务器体系结构* (Client/Server)结构
  + 常见的服务器端体现为关系数据库
  + 客户端负责显示和业务逻辑处理
  + 在部署和扩展性方面存在不足：系统升级需要一一更新所有客户端

*C/S体系结构*

* 客户/*服务器体系结构模型是一个分布式系统模型，数据和处理过程是分布在多个组件上*
* 一组给*其它子系统提供服务的服务器，例如:打印服务，数据管理等*
* 一组向服务器请求服务的客户机
* 网络， 允许客户存取服务器

*客户/服务器特性*

* *优点*
  + 数据的分布是简单的
  + 对网络化系统的有效利用。 要求有更便宜的硬件
  + 容易增加新的服务器或升级已存在的服务器
* *缺点*
  + 没有一个共享的数据模型，因此子系统可能使用不同的数据模型。 数据交换将是低效的
  + 在每个服务器中的冗余管理(数据)

*B/S(Browser/Server)结构*

* B/S(Browser/Server)结构
  + 瘦客户端：浏览器或客户端程序(Applet等)
  + 三层结构
    - 客户端：处理用户接口和用户请求
    - Web服务器端：处理Web服务和运行业务代码
    - 数据端：关系型数据库和其他后端数据资源
  + 新的发展：多层结构
    - Web服务器分离为Web端和业务服务端(例如J2EE应用服务器)

3.3 软件体系结构风格

* 体系结构风格定义了一系列系统的结构组织的模式，它是对一类具有相似结构的系统体系结构的抽象
  + - 一些实现系统所需的功能的部件
    - 连接各个部件，负责部件间通信和协作的连接器
    - 定义部件之间怎样整合的系统约束
    - 使设计者能够理解整个系统属性并分析已知属性的语义模型

(1) 容器模型

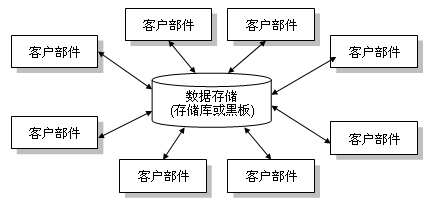
* 系统的子系统一定要通过交换数据才能有效地一起工作，有两种方式可以实现:
  + 全部共享的数据被放在中央数据库或容器中，所有子系统从中存取（容器模型）
  + 子系统维持其自身的数据库，显式地传递数据到其他的子系统，完成子系统之间的数据交换
* 大量数据被共享的时候,共享的容器模型使用的最普遍

*容器模型特性*

* 优点
  + 共享大量数据的有效方式
  + 子系统不需要关心数据如何产生出来的，集中的管理，举例来说，备份, 信息安全, etc.
  + 共享模型一般就是容器模式
* 缺点
  + 子系统一定服从于容器数据模型。 不可避免，需要妥协
  + 数据进化很困难而且很昂贵(数据模型改变)
  + 没有可能采取特定的管理政策
  + 有效的分布是困难的(数据冗余和不一致)

*实例：数据为中心体系结构*

* 一些数据(比如一个文件或者数据库)保存在整个结构的中心，并且被其他部件频繁地使用、添加、删除、或者修改



(2) 抽象机模型

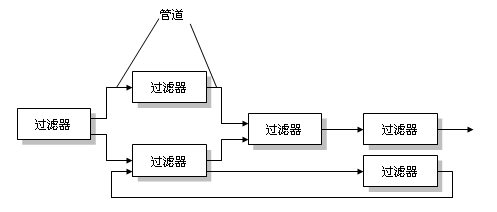
* 体系结构的抽象机模型是用来建立子系统接口模型
* 将系统组织成一组分层 (或抽象机)，每一层提供一组服务

*抽象机模型特性*

* 优点
  + 支持层中的增量式开发。
  + 当一层的接口发生了改变，受影响的只有相邻层
* 缺点
  + 用这种方式构成系统时常是困难的
  + 性能也将是一个问题

1. 数据流风格

* 又称管道/过滤器体系结构风格
* 这种结构适用于输入数据被一系列的计算或者处理部件变换成输出数据。

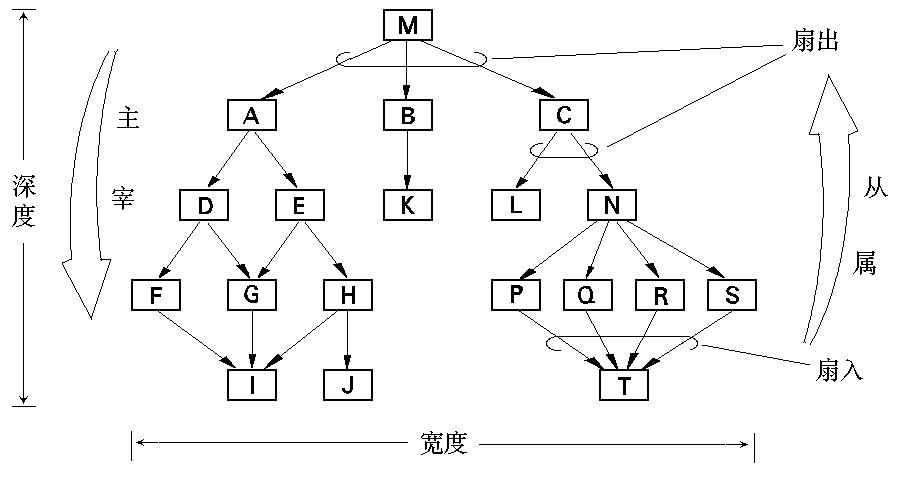


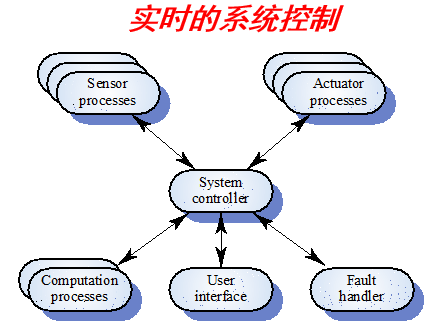
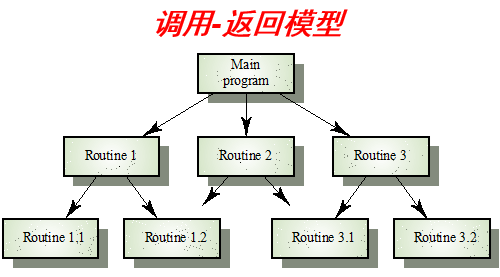
(4) 控制(调用和返回风格)

* 关注在子系统之间的控制流。不同于系统组件模型
* 集中控制
  + 一个子系统总体负责，负责其他子系统的开始和停止
* 基于事件的控制
  + 每个子系统能接收外部产生的事件，事件来自其他的子系统或系统的环境

*集中控制*

* 一个子系统被定义为系统控制器，负责管理其他的子系统的执行
* 调用-返回模型
  + 由上而下的调用的子程序模型，控制开始于子程序层次结构的顶端而且向下移动。 适应于顺序系统
* 管理者模型
  + 一个系统组件控制其它进程的停止、开始和协同。 能在顺序的系统中实现.(适合并发模型)
* 这种风格使一个软件设计者设计出非常容易修改和扩充的体系结构
  + 主程序/子程序风格体系结构
  + 远程过程调用风格的体系结构





*基于事件的控制*

* 基于事件的控制模型常识由外部产生的事件驱动系统，事件的到来不受处理它的那个过程控制
* 事件驱动模型：
  + 广播模型

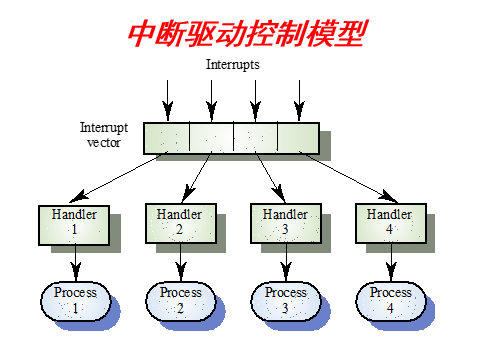
一个事件被广播到所有的子系统，任何能处理事件的子系统都可以这么做

* + 中断驱动模型

使用在实时系统中，中断是通过中断处理器来检测的，并将它传递给某个其它组件来处理

*广播模型*

* 广播模型在基于网络的分布式系统中很有效
* 子系统注册感兴趣的特定事件。当这些事件发生时，控制就被移交给能处理该事件的子系统
* 该模型与集中式模型的主要区别是该模型的控制策略不在事件和信息处理器中。子系统决定他们需要哪些的事件



*中断驱动模型*

* 在实时系统中使用，对一件事件的快速响应是必要的
* 允许快速的响应，但是编程将是困难的，且难以验证

（5） 面向对象风格

* 系统部件封装数据和操作数据的方法
* 部件之间的交互和协调通过消息来传递

（6） 层次式风格的体系结构

* 在这种结构中，定义不同的层次，每层都完成了相对外层更靠近机器指令的操作

（7） 小结

* 在这里要了解几个概念：
  + 程序结构的深度：程序结构的层次数称为结构的深度。结构的深度在一定意义上反映了程序结构的规模和复杂程度。
  + 程序结构的宽度：层次结构中同一层模块的最大模块个数称为结构的宽度。
  + 模块的扇入和扇出：扇出表示一个模块直接调用（或控制）的其它模块数目。扇入则定义为调用（或控制）一个给定模块的模块个数。多扇出意味着需要控制和协调许多下属模块。而多扇入的模块通常是公用模块。

四、部件级设计

4.1 部件级设计概述

* 在结构化分析和设计方法时部件往往被称为模块
* 在面向对象分析和设计时部件被称为类，在基于构件的开发方法中，部件被称为构件。

4.2 部件级设计任务

在部件级设计阶段，主要完成如下工作：

* 1. 为每个部件确定采用的算法，选择某种适当的工具表达算法的过程，编写部件的详细过程性描述；
  2. 确定每一部件内部使用的数据结构；
  3. 在部件级设计结束时，应该把上述结果写入部件级设计说明书，并且通过复审形成正式文档，作为下一阶段（编码阶段）的工作依据。

五、设计规约与设计评审

5.1 设计规约

5.2 设计评审

* 软件设计的最终目标是要取得最佳方案
* “最佳”是指在所有候选方案中，就节省开发费用，降低资源消耗，缩短开发时间的条件，选择能够赢得较高的生产率、较高的可靠性和可维护性的方案

*设计评审的内容*

1．可追溯性：即分析该软件的系统结构、子系统结构，确认该软件设计是否覆盖了所有已确定的软件需求，软件每一成分是否可追溯到某一项需求。

2．接口：即分析软件各部分之间的联系，确认该软件的内部接口与外部接口是否已经明确定义。部件是否满足高内聚和低耦合的要求。部件作用范围是否在其控制范围之内。

3．风险：即确认该软件设计在现有技术条件下和预算范围内是否能按时实现。

4．实用性：即确认该软件设计对于需求的解决方案是否实用。

5．技术清晰度：即确认该软件设计是否以一种易于翻译成代码的形式表达。

6．可维护性：从软件维护的角度出发，确认该软件设计是否考虑了方便未来的维护。

7．质量：即确认该软件设计是否表现出良好的质量特征。

8．各种选择方案：看是否考虑过其它方案，比较各种选择方案的标准是什么。

9．限制：评估对该软件的限制是否现实，是否与需求一致。

10．其它具体问题：对于文档、可测试性、设计过程等等进行评估。

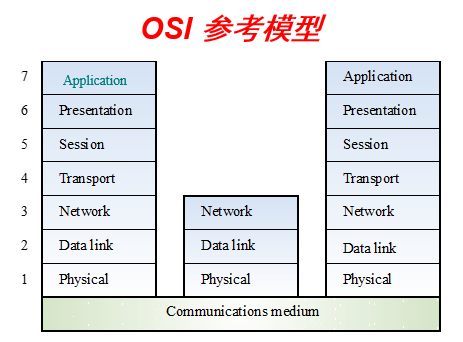
*设计评审*

* 评审分正式评审和非正式评审两种
* 正式评审除软件开发人员外，还邀请用户代表和领域专家参加，通常采用答辩形式
* 非正式评审多少有些同行切磋的性质，不拘泥于时间和形式

五、领域相关的体系结构

5.1 领域相关的体系结构

* 是特定应用领域的体系结构模型
* 参考模型, 更抽象的和 理想化的模型。
* 参考模型是由上而下的模型



*参考体系结构*

* 参考模型起源于对应用领域的研究，而不是对已存在的系统的研究
* 可以被用来作为系统实现的基础或用来对不同的系统做比较。它被当作标准，系统借此得以评估
* OSI 模型是通信系统的一个分层模型

SW10 分布式系统体系结构

一、分布式系统体系结构概述

1.1 系统类型

* 个人系统，非分布式的和设计在个人计算机或工作站上运行的，非分布式的系统。
* 嵌入式系统，运行于单一处理器或在集成在一起的一群处理器上的系统。
* 分布式系统，软件运行于网络相连的一组松散集成在一起一群处理器上的系统。

1.2 分布式系统

* 事实上,所有大型计算机系统现在都用分布式系统
* 分布式信息处理分布在一群计算机上而非限制在单一机器上
* 分布式软件工程学现在变得越来越重要了

1.3 分布式系统特性

* 优点
  + 资源共享
  + 开放性
  + 并发性
  + 伸缩性
  + 容错性
  + 透明性
* 缺点
  + 复杂性
  + 保密性
  + 管理上的问题
  + 不可预见性

1.4 分布式系统体系结构类型

* 客户机-服务器体系结构

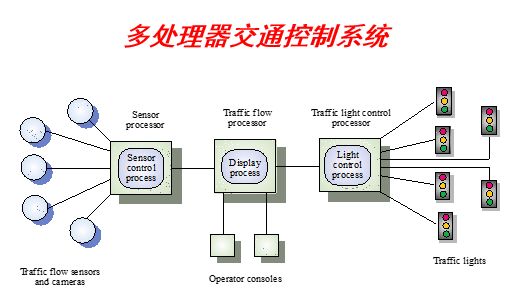
由客户调用的分布式服务

* 分布式对象体系结构

不再区分客户和服务器

*中间件*

* 管理和支持分布式系统不同组件的软件。本质是，确保组件之间的通信和交换数据，它位于系统的中间位置
* 中间件是一种通用软件，而不是编写的软件
* 例子
  + 事务管理器
  + 数据转换器
  + 通信控制器

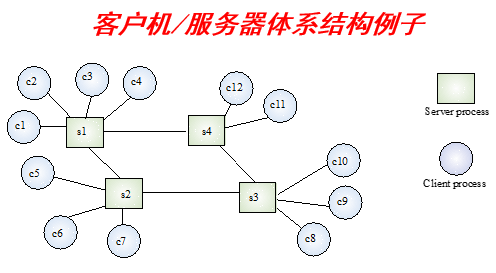
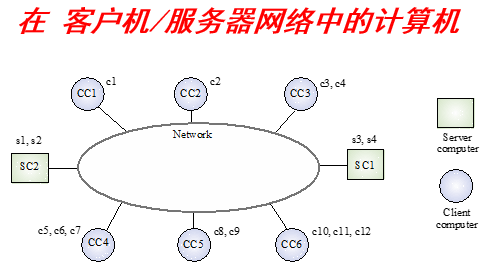
二、多处理器体系结构

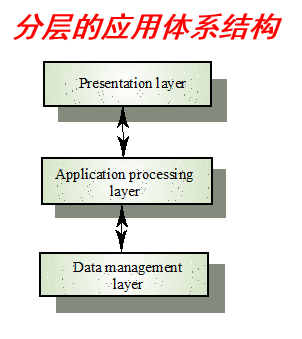
* 多处理器系统是最简单的分布系统
* 系统由多个进程组成，这些进程可能在不同的处理器上运行的
* 进程的分布是预先分配好的或是在控制器控制之下的

三、客户机/服务器体系结构

3.1 客户机/服务器体系结构

* 应用程序被建模成一组服务器，这些服务由服务器提供，并由客户使用这些服务
* 客户知道服务器的存在，但服务器不需要知道客户
* 客户和服务器是不同的进程

3.2 分层的应用体系结构

* 表示层

关注将信息表达给用户和同用户的交互

* 应用处理层

关注实现应用逻辑, 举例来说，在一个银行业系统中，银行业，例如打开帐户和关帐户动作等

* 数据管理层

关注数据库管理(数据的操作)

1. 二层的客户机/服务器体系结构

* 瘦客户机模型
  + 在瘦客户模型中，所有的应用处理和数据管理都是在服务器上执行的。客户机只负责表示部分
* 胖客户模型
  + 在这个模型中，服务器只对负责数据的管理。 客户机上软件实现应用逻辑与用户交互。

*瘦客户模型*

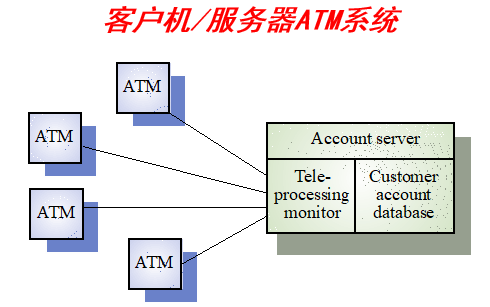
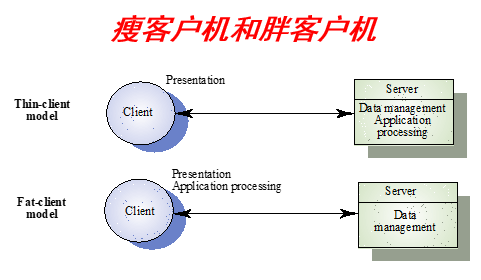
* 应用情况：将集中式遗留系统迁移到客户服务器体系结构的时候使用。

遗留系统本身作为一个服务器，用一个图形界面作为客户机

* 主要的缺点是将繁重的处理负荷放在服务器和网络上

*胖客户模型*

* 将应用逻辑处理和表示交给客户机来执行，所以处理是在本地运行的;服务器是一个事务服务器，即管理所有的数据库事务
* 最适用于新的C/S系统，因为客户机系统的能力是预先已知的，在分布处理上更有效
* 软件管理上的缺陷:应用软件变更时，新的应用版本必须安装在所有的客户机上

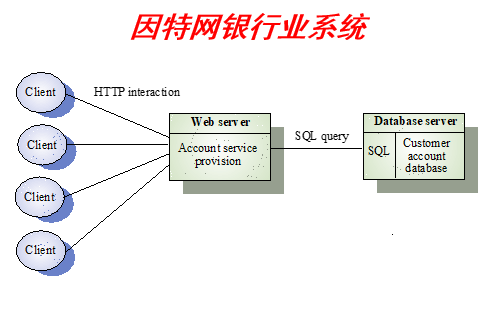
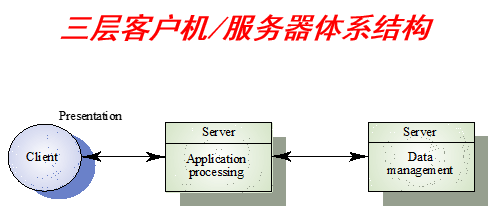


*二层的客户机/服务器体系结构特点*

* 二层的客户机/服务器体系结构的根本问题是将3个逻辑层映射在两个计算机系统
* 若选择瘦客户模型,则可能有伸缩性和性能的问题
* 若选择胖客户模型,则可能有系统管理上的问题

1. 三层的体系结构

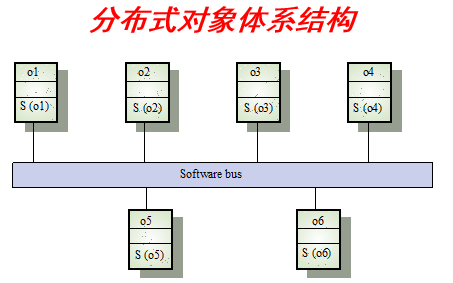
* 在三层的体系结构中，每一层可能运行在不同的处理器上
* 比瘦客户方式有较好的表现(伸缩性和性能)，而且从处理上比胖客户方式更简单
* 有更大伸缩性的体系结构 – 随着需求的迅速增加,额外的服务器可能需要附加进来



四、分布式对象体系结构

4.1 分布式对象体系结构

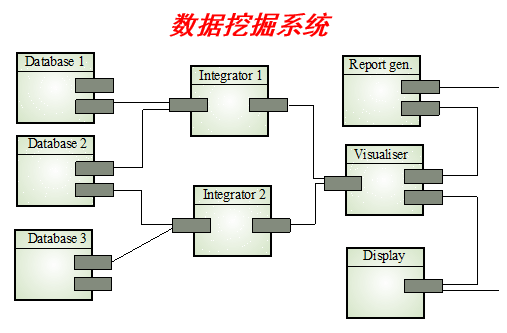
* 分布式系统的客户和服务器的地位是同的
* 分布式对象体系结构中无客户和服务器之分
* 分布式对象体系结构中，基本系统组件是对象，它能提供服务给其他对象，而且接受来自其他对象服务的对象
* 对象通信是经过称为一个对象请求代理的一个中间件 (软件总线)完成的
* 比客户和服务器系统设计要复杂的多

*分布对象体系结构的优点*

* 它允许系统设计者延迟决定应该在哪里和如何提供服务
* 它是一种非常开放的系统体系结构，允许新的资源根据需要被增加进来
* 系统具有很好的柔性的和可伸缩的
* 通过对象在网络上的迁移达到对系统动态地重新配置

*分布对象体系结构的使用*

* 作为一种逻辑模型来构造和组织系统。在这种情况下，要考虑如何独立提供应用程序功能，把这些功能按照服务和服务组合的形式给出
* 作为客户机/服务器系统的一种伸缩实现。在这种情况下，系统的逻辑模型是客户机服务器模型，客户机和服务器被当作分布对象，他们通过软件总线进行通信

*数据挖掘系统*

* 系统允许添加数据库的数量，而不会造成系统大的修改
* 系统通过添加新的集成对象挖掘出新知识

五、分布式对象计算标准

* CORBA（通用对象请求代理体系结构）
* DCOM(分布式组件对象模型)

SW10 分布式系统体系结构 要点

* 事实上，所有新的大型系统都是分布式系统
* 分布式系统支持资源共享，开放性，并发性，伸缩性，容错性和透明性
* 客户机/服务器模型时一个分布式系统
* 在客户机/服务器体系结构模型中，用户界面总是运行于客户机上,而服务器上总是数据管理
* 分布式对象体系结构中,没有客户和服务器之间的明显界限
* 分布式对象系统需要中间件来处理对象通信

SW11 白盒测试

一 、白盒测试概述

* 白盒测试（又称为结构测试）把测试对象看作一个透明的盒子，测试人员根据程序内部的逻辑结构及有关信息设计测试用例，检查程序中所有逻辑路径是否都按预定的要求正确地工作。
* 白盒测试主要是检查程序的内部结构、逻辑、循环和路径。
* 白盒测试主要用于对模块的测试，包括：
  + 程序模块中的所有独立路径至少执行一次
  + 对所有逻辑判定的取值（“真”与“假”）都至少测试一次
  + 在上下边界及可操作范围内运行所有循环
  + 测试内部数据结构的有效性等
* 白盒测试法检查程序内部逻辑结构，对所有逻辑路径进行测试，是一种穷举路径的测试方法。但即使每条路径都测试过了，仍然可能存在错误。因为：
  + 穷举路径测试无法检查出程序本身是否违反了设计规范，即程序是否是一个错误的程序。
  + 穷举路径测试不可能查出程序因为遗漏路径而出错。
  + 穷举路径测试发现不了一些与数据相关的错误。

二、逻辑覆盖测试方法

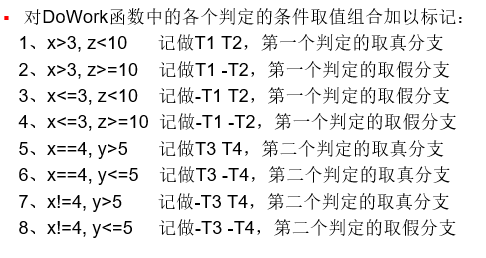
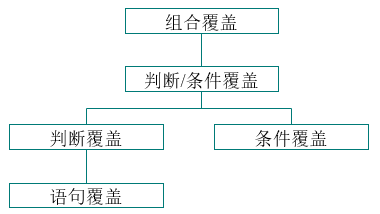
2.1 测试覆盖率

* 测试覆盖率：用于确定测试所执行到的覆盖项的百分比。其中的覆盖项是指作为测试基础的一个入口或属性，比如语句、分支、条件等。
* 测试覆盖率可以表示出测试的充分性，在测试分析报告中可以作为量化指标的依据，测试覆盖率越高效果越好。但覆盖率不是目标，只是一种手段。
* 测试覆盖率包括功能点覆盖率和结构覆盖率：
  + **功能点覆盖率**大致用于表示软件已经实现的功能与软件需要实现的功能之间的比例关系。
  + **结构覆盖率**包括语句覆盖率、分支覆盖率、循环覆盖率、路径覆盖率等等。

2.2 逻辑覆盖法

* 根据覆盖目标的不同，逻辑覆盖又可分为语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、组合覆盖和路径覆盖。
  + 语句覆盖：选择足够多的测试用例，使得程序中的每个可执行语句至少执行一次。
  + 判定覆盖：通过执行足够的测试用例，使得程序中的**每个判定**至少都获得一次“真”值和“假”值， 也就是使程序中的每个取“真”分支和取“假”分支至少均经历一次，也称为“分支覆盖”。
  + 条件覆盖：设计足够多的测试用例，使得程序中每个判定包含的**每个条件的可能取值（真/假）**都至少满足一次。
* 判定/条件覆盖：设计足够多的测试用例，使得程序中每个判定包含的每个条件的所有情况（真/假）至少出现一次，并且每个判定本身的判定结果（真/假）也至少出现一次。
  + 满足判定/条件覆盖的测试用例一定同时满足判定覆盖和条件覆盖。
* 组合覆盖：通过执行足够的测试用例，使得程序中每个判定的所有可能的**条件取值组合**都至少出现一次。
  + 满足组合覆盖的测试用例一定满足判定覆盖、条件覆盖和判定/条件覆盖。

逻辑覆盖主要考察使用测试数据运行被测程序时对程序逻辑的覆盖程度。通常希望选择最少的测试用例来满足所需的覆盖标准。



判断：1,3,5,7 条件：1,4,5,8 判断/条件：1,4,5,8 组合覆盖：1,2,3,4,5,6,7,8

SW12 黑盒测试

一、测试用例设计概述

1.1 测试用例的定义和特征

* 测试用例的定义：
  + 测试用例是为特定的目的而设计的一组测试输入、执行条件和预期的结果。
* 测试用例的特征：
  + 最有可能抓住错误的；
  + 不是重复的、多余的；
  + 一组相似测试用例中最有效的；
  + 既不是太简单，也不是太复杂。

1.2 设计测试用例的基本准则

* 测试用例的代表性

能够代表并覆盖各种合理的和不合理的、合法的和非法的、边界的和越界的以及极限的输入数据、操作和环境设置等。

* 测试结果的可判定性

即测试执行结果的正确性是可判定的，每一个测试用例都应有相应的期望结果。

* 测试结果的可再现性

即对同样的测试用例，系统的执行结果应当是相同的。

1.3 设计测试用例的着眼点

* 根据产品规格，测试基本功能；
* 与系统其他组成部分的配合；
* 极端情况（如内存泄漏、破坏性测试等）；
* 好的测试用例集能花费最小的代价（人力、物力、财力、时间）做最好的测试。

1.4 测试用例设计书写标准

* 在ANSI/IEEE829-1983标准中列出了和测试设计相关的测试用例编写规范和模板。标准模板中主要元素如下：
  + 标识符——惟一标识每一个测试用例
  + 测试项——准确的描述所需要测试的项及其特征
  + 测试环境要求——表征执行该测试用例需要的测试环境
  + 输入标准——执行测试用例的输入需求（这些输入可能包括数据、文件或者操作）
  + 输出标准——按照指定的环境和输入标准得到的期望输出结果
  + 测试用例之间的关联——标识该测试用例与其它的测试（或其它测试用例）之间的依赖关系

二、黑盒测试法的概念

* 黑盒测试是依据软件的需求规约，检查程序的功能是否符合需求规约的要求。
* 黑盒测试被称为功能测试或数据驱动测试。在测试时，把被测程序视为一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下进行。
* 黑盒测试可用于各种测试，它试图发现以下类型的错误：
  + 不正确或遗漏的功能
  + 接口错误，如输入/输出参数的个数、类型等
  + 数据结构错误或外部信息(如外部数据库)访问错误
  + 性能错误
  + 初始化和终止错误
* 主要的黑盒测试方法有：
  + 等价类划分
  + 边界值分析

三、等价类划分法

3.1 等价类划分法概述

* 等价类划分法是一种重要的、常用的黑盒测试方法，它将不能穷举的测试过程进行合理分类，从而保证设计出来的测试用例具有完整性和代表性。
* 举例：设计这样的测试用例，来实现一个对所有实数进行开平方运算（ y = sqrt(x)）的程序的测试。
* 思考方向：由于开平方运算只对非负实数有效，这时需要将所有的实数（输入域x）进行划分，可以分成：正实数、0 和 负实数。假设我们选定+1.4444代表正实数，-2.345代表负实数，则为该程序设计的测试用例的输入为+1.4444、 0 和 -2.345。
* 等价类划分法是把所有可能的输入数据，即程序的输入域划分为若干部分（子集），然后从每一个子集中选取少数具有代表性的数据作为测试用例。
* 所谓等价类是指某个输入域的子集合。在该子集合中，各个输入数据对于揭露程序中的错误都是等效的，它们具有等价特性，即每一类的代表性数据在测试中的作用都等价于这一类中的其它数据。这样，对于表征该类的数据输入将能代表整个子集合的输入。
* 因此，可以合理的假定：测试某等价类的代表值就是等效于对于这一类其它值的测试。

3.2 等价类的划分原则

* 等价类是输入域的某个子集合，而所有等价类的并集就是整个输入域。因此，等价类对于测试有两个重要的意义：
  + 完备性——整个输入域提供一种形式的完备性
  + 无冗余性——若互不相交则可保证一种形式的无冗余性
  + 如何划分？——先从程序的规格说明书中找出各个输入条件，再为每个输入条件划分两个或多个等价类，形成若干的互不相交的子集。
* 采用等价类划分法设计测试用例通常分两步进行：
  + 确定等价类，列出等价类表。
  + 确定测试用例。
* 划分等价类可分为两种情况：
  + 有效等价类
    - 是指对软件规格说明而言，是有意义的、合理的输入数据所组成的集合。利用有效等价类，能够检验程序是否实现了规格说明中预先规定的功能和性能。
  + 无效等价类
    - 是指对软件规格说明而言，是无意义的、不合理的输入数据所构成的集合。利用无效等价类，可以鉴别程序异常处理的情况，检查被测对象的功能和性能的实现是否有不符合规格说明要求的地方。
* 进行等价类划分的依据：
  + 按照区间划分 在输入条件规定了取值范围或值的个数的情况下，可以确定一个有效等价类和两个无效等价类。
  + 按照数值划分 在规定了一组输入数据（假设包括 n个 输入值），并且程序要对每一个输入值分别进行处理的情况下，可确定 n 个有效等价类（每个值确定一个有效等价类）和一个无效等价类（所有不允许的输入值的集合）
  + 按照数值集合划分 在输入条件规定了输入值的集合或规定了“必须如何”的条件下，可以确定一个有效等价类和一个无效等价类（该集合有效值之外）
  + 按照限制条件或规则划分 在规定了输入数据必须遵守的规则或限制条件的情况下，可确定一个有效等价类（符合规则）和若干个无效等价类（从不同角度违反规则）
  + 细分等价类 在确知已划分的等价类中各元素在程序中的处理方
* 例：程序输入条件为小于100大于10的整数x，则有效等价类为10＜x＜100，两个无效等价类为x≤10和x≥100。
* 例：程序输入x取值于一个固定的枚举类型{1,3,7,15}，且程序 中对这4个数值分别进行了处理，则有效等价类为x=1、x=3、x=7、x=15，无效等价类为x≠1,3,7,15的值的集合。
* 例：程序输入条件为取值为奇数的整数x，则有效等价类为x的值为奇数的整数，无效等价类为x的值不为奇数的整数。
* 例：程序输入条件为以字符‘a’开头、长度为8的字符串，并且字符串不包含‘a’~ ‘z’之外的其它字符，则有效等价类为满足了上述所有条件的字符串，无效等价类为不以‘a’开头的字符串、长度不为8的字符串和包含了‘a’~ ‘z’之外其它字符的字符串。

3.3 等价类划分法的测试用例设计

* 在设计测试用例时，应同时考虑有效等价类和无效等价类测试用例的设计。
* 根据已列出的等价类表可确定测试用例，具体过程如下：
  + 首先为等价类表中的每一个等价类分别规定一个唯一的编号。
  + 设计一个新的测试用例，使它能够尽量覆盖尚未覆盖的有效等价类。重复这个步骤，直到所有的有效等价类均被测试用例所覆盖。
  + 设计一个新的测试用例，使它仅覆盖一个尚未覆盖的无效等价类。重复这一步骤，直到所有的无效等价类均被测试用例所覆盖。

3.4 等价类划分设计测试用例的步骤

*确定等价类*根据软件的规格说明，对每一个输入条件（通常是规格说明中的一句话或一个短语）确定若干个有效等价类和若干个无效等价类。可使用如下表格

|  |  |  |
| --- | --- | --- |
| 输入条件 | 有效等价类 | 无效等价类 |
|  |  |  |

*确定等价类的规则：*

* (1) 如果输入条件规定了取值范围，则可以确定一个有效等价类（输入值在此范围内）和两个无效等价类（输入值小于最小值及大于最大值）

例如，规定输入的考试成绩在0..100之间，则有效等价类是“0 ≤ 成绩 ≤ 100”，无效等价类是“成绩 < 0”和“成绩 > 100”。

* (2) 如果输入条件规定了值的个数，则可以确定一个有效等价类（输入值的个数等于规定的个数）和两个无效等价类（输入值的个数小于规定的个数和大于规定的个数）

例如，规定输入构成三角形的3条边，则有效等价类是“输入边数 = 3”，无效等价类是“输入边数 < 3”和“输入边数 > 3”。

* (3) 如果输入条件规定了输入值的集合（即离散值），而且程序对不同的输入值做不同的处理，那么每个允许的值都确定为一个有效等价类，**另外还有一个无效等价类（任意一个不允许的值）。**

例如，规定输入的考试成绩为优、良、中、及格、不及格，则可确定5个有效等价类和一个无效等价类。

* (4) 如果输入条件规定了输入值必须遵循的规则，那么可确定一个有效等价类（符合此规则）和若干个无效等价类（从各个不同的角度违反此规则）。

例如，在Pascal语言中对变量标识符规定为“以字母开头的……串”。那么有效等价类是“以字母开头的串”，而无效等价类有“以数字开头的串”、“以标点符号开头的串”…等。

* (5) 如果输入条件规定输入数据是整型，那么可以确定三个有效等价类（正整数、零、负整数）和一个无效等价类（非整数）。
* (6) 如果输入条件规定处理的对象是表格，那么可以确定一个有效等价类（表有一项或多项）和一个无效等价类（空表）。

以上只是列举了一些规则，实际情况往往是千变万化的，在遇到具体问题时，可参照上述规则的思想来划分等价类。

*设计测试用例*  
在确定了等价类之后，建立等价类表，列出所有划分出的等价类。并为每个有效等价类和无效等价类编号。

*利用等价类设计测试用例的步骤：*

* (1) 设计一个新的测试用例，使其尽可能多地覆盖尚未被覆盖的有效等价类，重复这一步，直到所有的有效等价类都被覆盖为止；
* (2) 为每个无效等价类设计一个新的测试用例。

3.5 常见等价类划分测试形式

* 针对是否对无效数据进行测试，可以将等价类测试分为 标准等价类测试和健壮等价类测试。
  + **标准等价类测试**——不考虑无效数据值，测试用例使用 每个等价类中的一个值。
  + **健壮等价类测试**——主要的出发点是考虑了无效等价类。对有效输入，测试用例从每个有效等价类中取一个值； 对无效输入，一个测试用例有一个无效值，其他值均取有效值。
* 健壮等价类测试存在两个问题：
  + 需要花费精力定义无效测试用例的期望输出
  + 对强类型的语言没有必要考虑无效的输入

四、边界值分析法

4.1 边界值分析法概要

* 边界值分析法就是对输入或输出的边界值进行测试的一种黑盒测试方法。通常边界值分析法是作为对等价类划分法的补充，这种情况下，其测试用例来自等价类的边界。
* 为什么使用边界值分析法？

无数的测试实践表明，大量的故障往往发生在输入定义域或输出值域的边界上，而不是在其内部。因此，针对各种边界情况设计测试用例，通常会取得很好的测试效果。

* 怎样用边界值分析法设计测试用例？
  + 首先确定边界情况。通常输入或输出等价类的边界就是应该着重测试的边界情况。
  + 选取正好等于、刚刚大于或刚刚小于边界的值作为测试数据，而不是选取等价类中的典型值或任意值。
* 边界值分析使用与等价类划分法相同的划分，只是边界值分析假定错误更多地存在于划分的边界上，因此在等价类的边界上以及两侧的情况设计测试用例。

*例：测试计算平方根的函数*

* + 输入：实数
  + 输出：实数
  + 规格说明：当输入一个0或比0大的数的时候，返回其正平方根；当输入一个小于0的数时，显示错误信息“平方根非法-输入值小于0”并返回0；库函数Print-Line可以用来输出错误信息。
* 等价类划分：
  + 可以考虑作出如下划分：
    - 输入 (i)<0 和 (ii)>=0
    - 输出 (a)>=0 和 (b) Error
  + 测试用例有两个：
    - 输入4，输出2。对应于 (ii) 和 (a) 。
    - 输入-10，输出0和错误提示。对应于 (i) 和 (b) 。
* 边界值分析：
  + 划分(ii)的边界为0和最大正实数；划分(i)的边界为最小负实数和0。由此得到以下测试用例：
    - 输入 {最小负实数}
    - 输入 {绝对值很小的负数}
    - 输入 0
    - 输入 {绝对值很小的正数}
    - 输入 {最大正实数}
* 通常情况下，软件测试所包含的边界检验有几种类型：
  + 数字、字符、位置、大小、尺寸、空间等
* 相应地，以上类型的边界值应该在：
  + 最大/最小、首位/末位、上/下、最高/最低、最短/最长、空/满等情况下

假设一个文本输入区域允许输入1个到255个 字符，输入1个和255个字符作为有效等价类；输入0个和256个字符作为无效等价类，这几个数值都属于边界条件值。

*内部边界值分析*

* 在多数情况下，边界值条件是基于应用程序的功能设计而需要考虑的因素，可以从软件的规格说明或常识中得到，也是最终用户可以很容易发现问题的。然而，在测试用例设计过程中，某些边界值条件是不需要呈现给用户的，或者说用户是很难注意到的，但同时确实属于检验范畴内的边界条件，称为内部边界值条件或子边界值条件。
* 内部边界值条件主要有下面几种：
  + 数值的边界值检验
  + 字符的边界值检验
  + 其它边界值检验

小结：在实际的测试用例设计中，需要将基本的软件设计要求和程序定义的要求结合起来，即结合基本边界值条件和内部边界值条件来设计有效的测试用例

4.2 边界值分析法测试用例

* 采用边界值分析测试的基本思想是：故障往往出现在输入变量的边界值附近。
  + 边界值分析法利用输入变量的最小值(min)、略大于最小值(min+)、输入值域内的任意值(nom)、略小于最大值(max-)和最大值(max)来设计测试用例。
* 边界值分析法是基于可靠性理论中称为**“单故障”**的假设，即有两个或两个以上故障同时出现而导致软件失效的情况很少，也就是说，软件失效基本上是由单故障引起的。
  + 在边界值分析法中获取测试用例的方法是：
    - 每次保留程序中一个变量，让其余的变量取正常值，被保留的变量依次取min、min+、nom、max-和max。
    - 对程序中的每个变量重复 (1) 。

*选择测试用例的原则*

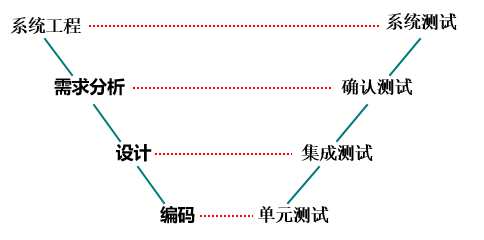
* 如果输入条件规定了值的范围，则应取刚达到这个范围的边界值以及刚刚超过这个范围边界的值作为测试输入数据。
* 如果输入条件规定了值的个数，则用最大个数、最小个数和比最大个数多1个、比最小个数少1个的数作为测试数据。
* 根据程序规格说明的每个输出条件，使用原则 (1)。
* 根据程序规格说明的每个输出条件，使用原则 (2) 。
* 如果程序的规格说明给出的输入域或输出域是有序集合 （如有序表、顺序文件等），则应选取集合中的第一个和 最后一个元素作为测试用例。
* 如果程序中使用了一个内部数据结构，则应当选择这个内部数据结构的边界上的值作为测试用例。
* 分析程序规格说明，找出其它可能的边界条件。

*健壮性测试*

健壮性测试是作为边界值分析的一个简单的扩充，它除了对变量的5个边界值分析取值外，还需要增加一个略大于最大值(max+)以及略小于最小值(min-)的取值

SW13 软件测试策略

一、软件测试策略概述

* 一种测试策略就是将测试分为单元测试、集成测试、确认测试和系统测试。
* 单元测试是针对程序中的模块或构件，主要揭露编码阶段产生的错误。
* 集成测试针对集成的软件系统，主要揭露设计阶段产生的错误。
* 确认测试是根据软件需求规约对集成的软件进行确认，主要揭露不符合需求规约的错误。
* 对于基于计算机系统中的软件，还需将它集成到基于计算机系统中，并进行系统测试，以揭露不符合系统工程中对软件要求的错误。
* Tom Gilb指出实现一个成功的软件测试策略必须涉及如下问题：
  + 在着手开始测试之前的较长时间，就要以量化的形式确定产品的需求。
  + 显式地陈述测试目标。
  + 了解软件的用户并为每一类用户建立剖面（profile）图
  + 建立一个强调“快速循环（rapid cycle）测试”的测试计划。
  + 构造“健壮”的软件，它被设计成可测试自身。
  + 使用有效的正式技术评审作为测试之前的过滤器。
  + 使用正式技术评审来评估测试策略和测试用例本身。
  + 为测试过程建立一种持续改进的方法。
* V模型：描述软件开发各阶段与测试策略之间的对应关系。
* 软件测试过程与软件工程的开发过程是相对的。
* 采用“V”形图表示软件开发与软件测试的对应关系。
* 单元测试的目的是保证每个模块单独运行正确
* 经单元测试后的模块，对软件包进行集成测试，主要测试软件结构问题，因测试建立在模块间的接口上。
* 系统测试主要是检验软件是否满足功能、行为和性能方面的要求
* 验收测试是检验软件产品的最后一道工序。

二、单元测试

2.1 单元测试概述

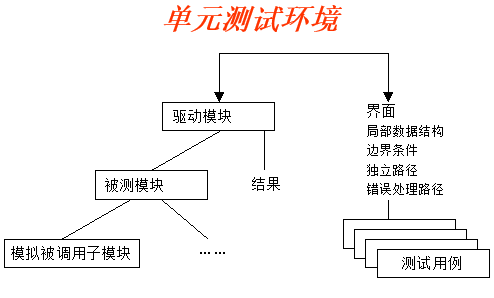
* 单元测试是对软件设计的最小单元——模块进行正确性检验的测试工作，主要测试模块在语法、格式和逻辑上的错误。
* 单元测试应对模块内所有重要的控制路径进行测试，以便发现模块内部的错误。
* 单元测试是检查软件源程序的第一次机会，通过孤立地测试每个单元，确保每个单元工作正常，这样比单元作为一个更大系统的一个部分更容易发现问题。
* 在单元测试中，每个程序模块可以并行、独立地进行测试工作。

2.2 单元测试的任务

* 单元测试是针对每个程序模块进行测试，单元测试的主要任务包括：
  1. 模块接口测试
  2. 模块局部数据结构测试
  3. 模块中所有独立执行路径测试
  4. 各种错误处理测试
  5. 模块边界条件测试

*模块接口测试*

* 针对模块接口测试应进行的检查，主要涉及以下几方面的内容。
  + 模块接受输入的实际参数个数与模块的形式参数个数是否一致。
  + 输入的实际参数与模块的形式参数的类型是否匹配。
  + 输入的实际参数与模块的形式参数所使用单位是否一致。
  + 调用其他模块时，所传送的实际参数个数与被调用模块的形式参数的个数是否相同。
  + 调用其他模块时，所传送的实际参数与被调用模块的形式参数的类型是否匹配。
  + 调用其他模块时，所传送的实际参数与被调用模块的形式参数的单位一致。
  + 调用内部函数时，参数的个数、属性和次序是否正确。
  + 在模块有多个入口的情况下，是否有引用与当前入口无关的参数。
  + 是否会修改了只读型参数。
  + 出现全局变量时，这些变量是否在所有引用它们的模块中都有相同的定义。

3.3 单元测试的执行过程

* 在完成了程序编写、复查和语法正确性验证后，就应进行单元测试。
* 测试用例设计应与复审工作相结合，根据设计信息选取数据，将增大发现上述各类错误的可能性。
* 在进行单元测试时，需设置若干辅助测试模块。

3.4 单元测试技术和测试数据

* 用于单元测试的主要技术
  + - 静态测试
    - 白盒测试

3.5 单元测试数据

* 单元测试中使用的数据，通常不使用真实数据。
* 在创建测试数据时，应确保数据充分地测试单元的边界条件。
* 测试时还要考虑往样本数据中引入一些手工制作的数据，以便测试单元的某个具体特性，例如对错误条件的响应等。
* 当测试一个单元要从远程数据源接收数据时，有必要在单元测试中使用测试辅助程序，来模拟对这些数据的访问。

3.6 单元测试人员

* 单元测试一般由开发设计人员本身完成
* 由编写该单元的开发设计者设计所需的测试用例和测试数据，来测试该单元并修改缺陷。
* 开发组组长负责保证使用合适的测试技术，在合理的质量控制和监督下执行充分的测试。

三、集成测试

3.1 集成测试概述

* 将经过单元测试的模块按设计要求连接起来，组成所规定的软件系统的过程称为“集成”。

3.2 集成测试的主要任务

* 集成测试是组装软件的系统测试技术，按设计要求把通过单元测试的各个模块组装在一起之后
* 集成测试的主要任务是要求软件系统符合实际软件结构，发现与接口有关的各种错误。
* 将各模块连接起来，检查模块相互调用时，数据经过接口是否丢失。
* 将各个子功能组合起来，检查能否达到预期要求的各项功能。
* 一个模块的功能是否会对另一个模块的功能产生不利的影响。
* 全局数据结构是否有问题，会不会被异常修改。

3.3 集成测试方法

* 集成测试包括两种不同方法
  + - 非增量式集成
    - 增量式集成

(1) 非增量式测试方法

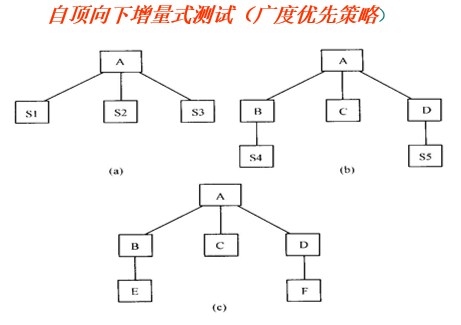
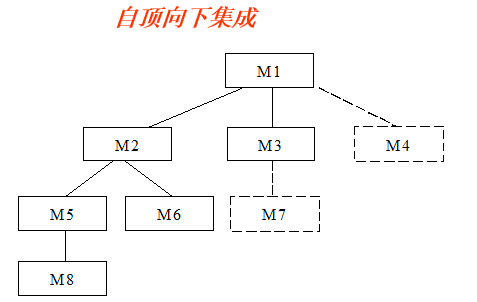
* 非增量式测试方法是采用一步到位的方法来进行测试，即对所有模块进行个别的单元测试后，按程序结构图将各模块连接起来，把连接后的程序当做一个整体进行测试。

(2) 增量式测试方法

* 自顶向下增量式测试
* 自底向上增量式测试

(2.1) 自顶向下增量式测试

自顶向下增量式测试表示逐步集成和逐步测试是按结构图自上而下进行的。即模块集成的顺序是首先集成主控模块（主程序），然后按照软件控制层次结构向下进行集成。



* 集成测试的整个过程由下列步骤完成：
  + 主控模块作为测试驱动器，把对主控模块进行单元测试时引入的被调用模拟子模块用实际模块替代。
  + 依照所选用的模块集成策略（深度优先和广度优先），下层的被调用模拟子模块一次一个地被替换为真正的模块。
  + 在每个模块被集成时，都必须立即进行测试一遍。
  + 回到第2步重复进行，直到整个系统结构被集成完成。

(2.2) 自底向上增量式测试

* 自底向上增量式测试是从最底层的模块开始，按结构图自下而上逐步进行集成和测试。

3.4 集成测试技术和测试数据

* 集成测试主要测试软件的结构问题，因为测试建立在模块的接口上，所以多为黑盒测试，适当辅以白盒测试。
* 执行集成测试采用的方法：
  + 确认组成一个完整系统的模块之间的关系
  + 评审、确认模块间的接口
  + 产生测试用例。
* 此外，在测试过程中尤其要注意关键模块，所谓关键模块一般都具有下述一个或多个特征。
  + 具有高层控制功能
  + 复杂，易出错
  + 有特殊的性能要求
* 集成测试的主要目的是验证组成软件系统的各模块的接口和交互作用。
* 集成测试一般也不使用真实数据
* 在单元测试时，根据需要生成了一些测试数据，在集成测试时可适当地重用这些数据，这样可节省时间和人力。

3.5 集成测试技术遵循的原则

* 集成测试需要遵循以下原则：
  + 所有公共接口都要被测试到
  + 关键模块必须进行充分的测试
  + 集成测试应当按一定的层次进行
  + 当接口发生修改时，涉及的相关接口必须进行再测试
  + 测试执行结果应当如实记录

3.6 集成测试人员

* 由于集成测试不是在真实环境下进行，而是在开发环境，或是一个独立的测试环境下进行的，所以集成测试所需人员一般从开发组中选出，在开发组长的监督下进行，开发组长负责保证在合理的质量控制和监督下使用合适的测试技术执行充分的集成测试。
* 在集成测试过程中，测试过程由一个独立测试小组长来监控测试工作。
* 集成测试过程中应考虑邀请一个用户代表非正式地观看集成测试 。

四、系统测试

4.1 系统测试概述

* 集成测试通过以后，软件已经组装成一个完整的软件包，这时就要进行系统测试。

4.2 系统测试任务

* 系统测试一般要完成以下几种测试：
  + 功能测试
  + 性能测试
  + 安全测试
  + 强度测试

4.3 系统测试技术和测试数据

* 系统测试完全采用黑盒测试技术
* 系统测试尽可能地采用真实数据进行测试

4.4 系统测试人员

* 系统测试由独立的测试小组进行
* 在系统测试过程中，测试过程由一个独立测试组长来监控测试工作
* 系统测试过程也应有用户代表参加

五、验收测试

5.1 验收测试概述

* 系统测试完成后，并使系统试运行了预定的时间，应进行验收测试。

5.2 验收测试任务

* 软件验收测试应完成的主要测试工作包括以下几方面：
  + 文档资料的审查验收
  + 功能测试
  + 性能测试
  + 安装测试
  + 用户操作测试

5.3 验收测试技术和测试数据

* 验收测试完全采用黑盒测试技术，主要是用户代表通过执行其在平常使用系统时的典型任务来测试软件系统，根据业务需求分析，检验软件是否满足功能、性能等方面的要求。
* 在验收测试中就应该使用真实数据。
  + 用户代表被允许使用这些数据，或者合理地组织测试使测试组长不必看到这些数据也可进行测试。

5.4 验收测试人员

* 验收测试一般在测试组的协助下，由用户代表执行。测试组长负责保证在合理的质量控制和监督下使用合适的测试技术执行充分测试。测试人员在验收测试工作中将协助用户代表执行测试，并和测试小组长一起向用户解释测试用例的结果。

5.5 α和β测试

* 软件开发设计人员在软件开发设计时，不可能完全预见用户实际使用软件系统的情况。
* 另外，一个软件产品可能拥有众多用户，不可能由每个用户验收，此时多采用称为α、β测试的过程，以发现那些似乎只有最终用户才能发现的问题。
* α测试是在软件开发公司内模拟软件系统的运行环境下的一种验收测试，即软件开发公司组织内部人员，模拟各类用户行为对即将面市的软件产品（称为α版本）进行测试，试图发现并修改错误。
* 经过α测试调整的软件产品称为β版本。紧随其后的β测试是指软件开发公司组织各方面的典型用户在日常工作中实际使用β版本，并要求用户报告异常情况，提出批评意见。然后软件开发公司再对β版本进行改错和完善。
* 所以，一些软件开发公司把α测试看成是对一个早期的、不稳定的软件版本所进行的验收测试，而把β测试看成是对一个晚期的、更加稳定的软件版本所进行的验收测试。

六、回归测试

6.1 回归测试概述

* 回归测试是指软件系统被修改或扩充后重新进行的测试，是为了保证对软件所做的修改没有引入新的错误而重复进行的测试。
* 每当软件增加了新的功能，或者软件中的缺陷被修正，这些变更都有可能影响软件原有的功能和结构。
* 为了防止软件的变更产生无法预料的副作用，不仅要对修改内容进行测试，还要重复进行过去已经进行过的测试，以证明修改没有引起未曾预料的后果，或证明修改后的软件仍能满足具体的需求。

6.2 回归测试技术和测试数据

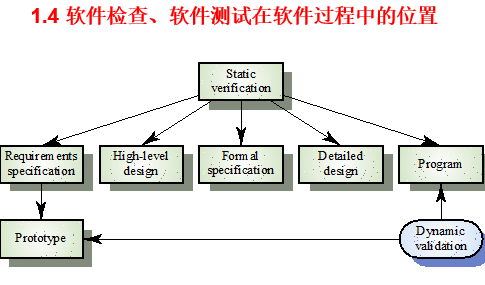
* 原来的测试数据
* 尽可能覆盖与相关功能的数据

6.3 回归测试范围

* 局限在修改范围内的测试
* 在受影响功能范围内回归
* 根据一定的覆盖率指标选择回归测试

6.4 回归测试人员

* 独立的测试小组
* 由测试组长负责
* 在回归测试过程中，测试过程由一个测试小组长来监控测试工作。在回归测试完成时测试组组长负责整理并归档大量的回归测试结果，包括测试结果记录、回归测试日志和简短的回归测试总结报告。

SW14 检验和有效性验证

一、检验和有效性验证概述

1.1 检验和有效性验证含义

* 检验和有效性验证是保证软件满足用户的需要
* 检验:   
   “Are we building the product right"
* 软件应该满足定义的功能和非功能需求
* 有效性验证:  
   "Are we building the right product"
* 应该保证软件满足客户的期待

1.2 检验和有效性验证过程

* 检验和有效性验证过程是被应用到软件开发周期的各个阶段

1.3 静态和动态检验、有效性验证

* 软件检查 Concerned with analysis of the static system representation to discover problems *(*static verification)

May be supplement by tool-based document and code analysis

* 软件测试 Concerned with exercising and observing product behaviour (dynamic verification)

The system is executed with test data and its operational behaviour is observed

* 1. 检查技术
* 检查技术包括程序检查、自动化的源代码分析和形式化检验
* 需然，软件检查得到广泛的使用，然而程序测试仍然是起决定作用的软件检验和有效性验证技术
  1. 软件测试
* 程序测试仍然是软件检验和有效性验证起决定作用的技术
* 测试过程来发现程序中的缺陷和不足
* 非功能需求是通过有效性技术来验证的
* 静态检验将被应用到所有的检验和有效验证过程中

*测试类型*

* 缺陷测试

Tests designed to discover system defects.

A successful defect test is one which reveals the presence of defects in a system.

* 统计性测试

tests designed to reflect the frequence of user inputs. Used for reliability estimation.

1.6 检验和有效性验证的目标

* 最终目标是要确信软件系统是“达到设计目标的”
* 这并不意味着程序完全没有缺陷
* 而是表明系统足以承担要求的任务，

1.7 检验和有效性验证确信程度

* 系统需要的信任程度依赖于系统的设计目标、系统用户对它的期望和目前的市场环境

软件功能

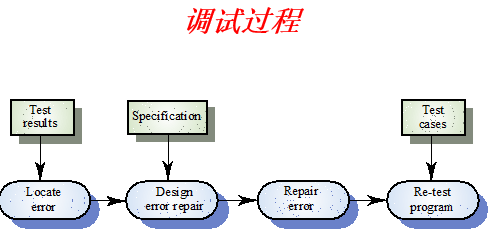
* + - The level of confidence depends on how critical the software is to an organisation

用户期望值

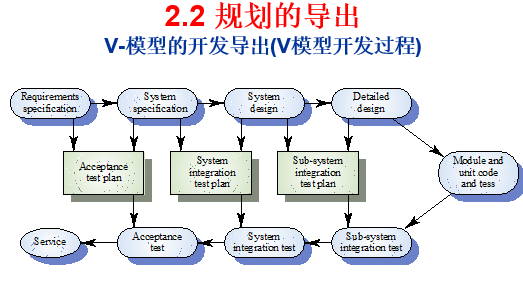
* + - Users may have low expectations of certain kinds of software

市场环境

* + - Getting a product to market early may be more important than finding defects in the program

1.8 检验,有效性验证和调试

* Defect testing and debugging are distinct   
  processes
* Verification and validation is concerned with establishing the existence of defects in a program
* Debugging is concerned with locating and   
  repairing these errors

二、检验和有效性规划

2.1 检验和有效性规划

* 仔细规划能从程序检查和测试中得到更多回报
* 规划应该在开发过程的早期就开始
* 应该在检验和有效性验证的静态和动态方法之间进行均衡
* 测试的规划主要不仅是制定测试过程标准，而且应该是描述具体的测试内容

2.3 软件测试计划的结构

* The testing process
* Requirements traceability
* Tested items
* Testing schedule
* Test recording procedures
* Hardware and software requirements
* Constraints

三、软件检查

3.1 软件检查

* 软件检查不需要运行程序，所以在程序完成之前可以进行检查
* 软件检查的对象是系统的一些表示形式 (requirements, design, test data, etc.)
* 软件检查被证明是一种有效的错误检查技术

3.2 软件检查的好处

* 在检查过程能发现很多不同的缺陷
* 复用了领域和程序语言知识

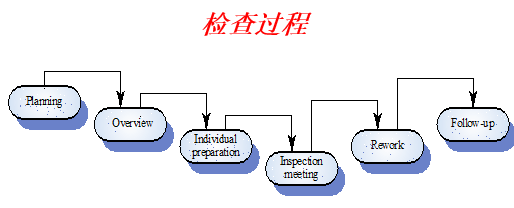
3.3 检查和测试

* 检查和测试不是两个相互竞争的技术，是互为补充
* 二者各有优点，应该在检验和有效性验证过程中一起使用
* 检查能检查是否和需求描述一致性，而不能检查是否同用户的真实需求一致
* 检查不能检查非功能性特性,such as performance, usability, etc.

3.4 程序检查

* 正式化的方法是文档评审
* 程序检查的目标是评审程序中的缺陷
* 缺陷可能是逻辑错误，可能是错误的条件或者是与机构和项目不相符的问题 (e.g. an uninitialised variable) or non-compliance with standards

*程序检查之前的准备工作*

* 被检查的代码有一个精确的描述
* 检查小组的成员应该熟悉机构的标准
* 有一个最新的、语法正确的代码版本
* 应该准备一个检查列表
* 管理者认同检查需要增加成本
* 管理者不应该将软件检查结果作为职员的考评

*检查过程*

* 准备好检查材料，保证要审查的代码及其描述是完整的
* 代码及相关的文档将分发给检查小组，并由程序的作者说明该程序的设计目的
* 记录检查中发现的问题
* 在检查过程之后，程序由其作者修改检查出来的问题
* 可以反复进行检查过程

*检查过程小组*

* 检查小组至少由4人构成
* 检查者发现相关的问题（错误、不符合规范等）
* 阅读者向检查小组阅读代码
* 协调者主持会议并记录发现的相关问题

*检查过程列表*

* 常规的错误检查列表
* 针对具体程序语言的错误检查列表
* 如：初始化、常量命名、循环终止、数组边界 etc.

四、自动静态分析

4.1 自动静态分析

* 静态分析器是一个软件工具，扫描程序源代码、检查可能存在的缺陷和不规则处
* 静态分析器分析程序源代码，试图发现潜在的错误，并引起检查者对程序中的异常的注意
* 静态分析器对检查者是非常有效的，是检查者的极好辅助工具

4.2 静态分析包括以下阶段

* 控制流分析*.*

Checks for loops with multiple exit or entry points, finds unreachable code, etc.

* 数据使用*分析*

Detects uninitialised variables, variables written twice without an intervening assignment, variables which are declared but never used, etc.

* 接口分析

Checks the consistency of routine and procedure declarations and their use

* 信息流分析

Identifies the dependencies of output variables. Does not detect anomalies itself but highlights   
information for code inspection or review

* 路径分析

Identifies paths through the program and sets out the statements executed in that path. Again, potentially useful in the review process

4.3 静态分析方法的使用

* 对于像C语言，静态分析是发现错误的有效技术，C语言不含有严格的类型规则，C编译器所进行的检查是有限的
* 对于像Java语言，静态分析的作用是有限的，Java语言编译器具有很强的功能

SW14 检验和有效性验证 要点

* 检验和有效性验证是不一样的
* 测试计划需要描述测试过程的各个环节
* 静态检验技术包括对程序源代码的检查和分析以发现存在的错误
* 程序检查对程序发现错误是非常有效的
* 程序代码由一个小组系统地进行检查
* 静态分析器是一个软件工具，它处理程序源代码并将程序中的不规则之处显示出来

SW15 软件变更

一、软件变更概述

1 .1 软件变更

* 软件变更是管理软件变更的过程
* 软件变更是不可避免的：
  + 软件进入使用期，新的需求会出现
  + 已有需求随着业务环境的变化而变化
  + 软件运行中的错误要进行修改
  + 软件运行必须适应新的设备
  + 其他性能和非功能特性需要修改
* 机构的一个主要问题是实现和管理他们的遗留系统变更，使之能继续支持他们的业务运作

1.2 软件变更策略

* 软件维护
  + 对软件做局部修改，但软件的基本结构保持不变
* 体系结构的转换
  + 通常系统从集中式的体系结构转向为分布式体系结构
* 软件再工程
  + 这种策略是没有新的功能的添加，系统通过修改以后更容易理解和改变
* 上述策略并不是相互排斥的，它们可以单独使用，也可以组合使用

二、程序进化的动态特性

* 程序进化的动态特性是研究系统的变更过程
* 这个领域的大部分工作是由Lehman和Belady完成的，他们提出了关于系统变更的一组定律，并认为这些定律是不变的、广泛实用的
* 这些定律是通过对多个大型软件系统的成长和进化的基础上而提出的，因此，这些定律可能不适应小规模的系统

三、软件维护

3.1 软件维护

* 软件维护，是在软件投入使用之后，对程序进行变更的过程， 软件维护就成为软件变更的一个常规过程
* 软件维护不包括重大的体系结构上的改变
* 变更的实现是修改已有的系统组件以及在必要的地方添加新组件到系统中

*软件维护是不可避免的*

* 由于环境的改变，当系统正处于开发阶段，系统需求将可能发生改变。如果不进行软件变更，系统将不满足用户需求
* 系统是和环境紧密相联系的。当系统安装在一个环境的时候，它可能影响环境，同时环境也会影响系统需求
* 系统本身缺陷的存在

*维护的类型*

* 修补软件缺陷的维护
  + 修改系统的缺陷，满足系统的需求
* 使软件适应不同的操作环境的维护
  + 在 系统环境（操作系统、硬件）发生改变的时候，需要进行这种维护
* 增加或修改系统功能的维护
  + 修改系统满足新的需求

*维护成本*

* 通常来将，维护成本高于开发成本
* 影响软件维护存在着技术与非技术的因素
* 维护会破坏软件的体系结构，从而影响进一步的软件维护（变得更加困难）
* 软件使用越长，维护成本会增加(e.g. old languages, compilers etc.)

*影响维护成本因素*

* 团队稳定性
  + 如果开发团对能够保持一定的时间，将会降低维护成本
* 合同责任
  + 系统的维护合同一般是独立于系统开发合同的。
* 人员技术水平
  + 维护人员一般都缺乏经验，而且不熟悉应用领域
* 程序年龄和结构
  + 随着程序年龄的增长，其结构受到变更的破坏，由此变得越来越不容易理解和变更

3.2 软件维护过程

*变更请求*

* 变更请求来自于系统用户、管理层或客户
* 在进行版本规划时，所有提出的变更请求都要仔细分析，最后决定哪些变更需要在系统的具体哪一个版本实现
* 实际上，一些变更请求必需紧急实现（与必须马上解决的问题有关）
  + 软件的缺陷
  + 环境的变化对系统产生了意想不到的影响
  + 新的业务发生变化

3.3 维护预测

* 维护预测主要是预测哪些系统变更请求会发生；系统的哪些部分容易给维护人员造成困难；在给定的时间内系统总的维护成本将是多少。上述不同方面的预测之间存在着紧密的关联：
  + 是否应该接受一个系统变更在某种程度上取决于受该变更影响的组件的可维护性；
  + 系统变更的实现容易使系统结构退化，降低了系统的可维护性
  + 维护费用取决于变更的数目，而变更实现的成本取决于系统组件的可维护性

*变更预测*

* 预测变更请求的数目需要了解系统和外部环境的关系
* 许多系统与外部环境之间存在着复杂的关系，对环境所做的改变不可避免地导致系统变更的发生。
* 影响系统和外部环境的关系的因素：
  + 系统接口的数目和复杂性
  + 本质上易变的系统需求数目
  + 系统所处的业务过程

*过程度量*

* 使用过程数据来帮助预测可维护性，对维护性评估有用的过程度量有；
  + 纠正性维护请求的数目
  + 影响分析所需的平均时间
  + 实现一个变更的平均时间
  + 突出的变更请求数目
* 根据变更请求的预测信息和对系统可维护性的预测来预测维护费用

3.4 体系结构的进化

* 许多公司纷纷将原有系统从集中式大型机系统转变为分布式C/S体系结构的系统
* 许多因素在影响和加速这种变化：
  + 硬件成本
  + 对用户界面的期待
  + 对系统分布式的访问

*遗留系统的结构*

* 理想情况下，分布式体系结构应该将用户接口、系统服务和系统数据管理严格分开
* 实际上，遗留系统将上述三部份完全融为一体

*分布式选择*

* 最简单的选项中，客户机只有用户界面的表示，所有其他功能都在服务器上
* 最复杂的选项中，服务器只管理数据，所有其他功能都分布在客户机上

*3.4.1 用户界面分布*

* 用户界面的分布利用了桌面PC机的局部处理能力优势，提供了更好的图形界面给系统用户
* 如果遗留系统是结构化的，那么用户界面服务就能清楚地识别出来，这样遗留系统就容易被修改从而实现用户界面的分布
* 否则，屏幕管理中间件将文本接口转变为图形接口

SW15 软件变更 要点

* 软件变更策略包括软件维护、体系结构进化和软件再生工程
* 软件维护有三种基本类型：
  + 软件缺陷的维护
  + 适应不同操作环境的维护
  + 实现新的需求的维护
* 软件变更成本常常超过开发成本
* 高额的维护费用来源于人员的不稳定性、开发合同没有鼓励可维护代码的生产、缺乏维护系统所需的技能以及由于系统老化和频繁维护导致的系统结构退化
* 体系结构进化包括从集中的、以数据为中心的体系结构到分布式体系结构的转变
* 遗留系统的体系结构进化的常用策略是将遗留系统封装为一个服务器，并实现用户界面的分布，用户界面通过专用的中间件来访问系统的功能

其他一 敏捷开发

* 软件开发的新挑战
  + 快速的市场进入时间，要求高生产率
  + 快速变化的需求
  + 快速发展的技术
* 传统的软件开发方法
  + 强调过程
  + 强调文档
  + 开发人员负担过重

称为重载(Heavyweight)方法

*针对上述问题，产生了一系列轻载(Lightweight)方法，如XP、SCRUM等。*

*2001年2月，新方法的一些创始人在美国犹他州成立了敏捷软件开发联盟 ，简称Agile 联盟。*

*Agile 联盟起草了一个敏捷软件开发宣言，该宣言由四个价值观声明组成，并提炼出敏捷软件开发方法必须遵循的12条原则。*

*Agile方法是在保证软件开发有成功产出的前提下，尽量减少开发过程中的活动和制品的方法。笼统的讲就是，“刚刚好”（Just enough），即开发中的活动及制品既不要太多也不要太少。*

一、Agile

1.1 Agile方法的价值观

* 个人和交互高于过程和工具

不是否定过程和工具的重要性，而是更强调软件开发中人的作用和交流的作用。

软件是由人组成的团队来开发的，与软件项目相关的各类人员通过充分的交流和有效的合作，才能成功地开发出得到用户满意的软件。

如果光有定义良好的过程和先进的工具，而人员的技能很差，又不能很好地交流和协作，软件是很难成功地开发的。

* 可运行软件高于详尽的文档

通过执行一个可运行的软件来了解软件做了什么，远比阅读厚厚的文档要容易得多。

敏捷软件开发强调不断地快速地向用户提交可运行的软件（不一定是完整的软件），以得到用户的认可。

好的必要的文档仍是需要的，它能帮助我们理解软件做什么，怎么做以及如何使用，但软件开发的主要目标是创建可运行的软件。

* 与客户协作高于合同（契约）谈判

只有客户才能明确说明需要什么样的软件，然而，大量的实践表明，在开发的早期客户常常不能完整地表达他们的全部需求，有些早期确定的需求，以后也可能会改变。

要想通过合同谈判的方式，将需求固定下来常常是困难的。

敏捷软件开发强调与客户的协作，通过与客户的交流和紧密合作来发现用户的需求。

* 对变更及时做出反应高于遵循计划

任何软件项目的开发都应该制订一个项目计划，以确定各开发任务的优先顺序和起止日期。然而，随着项目的进展，需求、业务环境、技术等都可能变化，任务的优先顺序和起止日期也可能因种种原因会改变。

因此，项目计划应具有可塑性，有变动的余地。当出现变化时及时做出反应，修订计划以适应变化。

1.2 Agile方法的指导原则

（1）最优先的是通过尽早地和不断地提交有价值的软件使客户满意

（2）欢迎变化的需求，即使该变化出现在开发的后期，为了提升对客户的竞争优势，Agile过程利用变化作为动力

（3）以几周到几个月为周期，尽快、不断地发布可运行软件

（4）在整个项目过程中，业务人员和开发人员必须天天一起工作

（5）以积极向上的员工为中心建立项目组，给予他们所需的环境和支持，对他们的工作予以充分的信任

（6）项目组内效率最高、最有效的信息传递方式是面对面的交流

（7）测量项目进展的首要依据是可运行的软件

（8）敏捷过程提倡可持续的开发，项目发起者、开发者和用户应能长期保持恒定的速度

（9） 应时刻关注技术上的精益求精和好的设计，以增强敏捷性

（10）简单化是必不可少的，这是尽可能减少不必要工作的艺术

（11）最好的构架、需求和设计出自于自我组织的团队

（12）团队要定期反思怎样才能更有效，并据此调整自己的行为

1.3 Agile方法的适用范围

Martin Fowler认为：新方法不是到处可适用的

适合采用Agile方法的情况：

* 需求不确定、易挥发（Volatile,意指今天的要求明天就不需要了）
* 有责任感和积极向上的开发人员
* 用户容易沟通并能参与

1.4 Agile的典型方法

* Extreme Programming (简称XP)
* SCRUM
* Crystal Methodologies (简称Crystal)
* Feature Driven Development(简称FDD)
* Dynamic Systems Development Methodology(简称DSDM)
* Adaptive Software Development(简称ASD)
* Pragmatic Programming等

二、XP方法

* 由Kent Beck提出，是Agile方法中最引人注目的一个
* XP最初实践于1997年Crysler公司的C3项目 (Smalltalk开发）
* 适用于10人以下项目组、开发地点集中的场合
* 广泛用于需求模糊和挥发性强的场合
* IONA公司的Obix技术支持小组在采用了XP方法后，软件生产率提高了67%

2.1 XP方法的4个价值观

* 交流（Communication）
  + 实践表明，项目失败的重要原因之一是交流不畅，使得客户的需求不能准确地传递给开发人员，造成开发人员不能充分理解需求；模型或设计的变动未能及时告知相关人员，造成系统的不一致和集成的困难
  + 所有项目相关人员之间充分的有效的交流是软件开发成功所必不可少的
  + XP方法提倡面对面的交流，这是一种有效的也是效率最高的交流方式
* 简单（Simplicity）
  + 指在确保得到客户满意的软件的前提下，做最简洁的工作（简单的过程、模型、文档、设计和实现）
  + 在开发中不断优化设计，时刻保持代码简洁、无冗余
  + 体现了敏捷开发的“刚刚好(Just enough)”思想，即开发中的活动及制品既不要太多也不要太少，刚好即可
* 反馈（Feedback）

及时有效的反馈能确定开发工作是否正确，及时发现开发工作的偏差并加以纠正。

强调各种形式的反馈，如非正式的评审（走查，Walkthrough）、小发布等

* 勇气（Courage）

采用敏捷软件开发需要勇气：

* + 信任合作的同事，也相信自己
  + 做能做到的最简单的事
  + 只有在绝对需要的时候才创建文档
  + 让业务人员制定业务决策，技术人员制定技术决策
  + 用可能的最简单的工具，例如白板和纸，只有在复杂建模工具能提供可能的最好价值时才去使用它们
  + 相信程序员能制定设计决策，不需要给他们提供过多的细节
  + 需要勇气来承认自己是会犯错误的，需要勇气来相信自己明天能克服明天出现的问题。

2.2 XP方法的12个核心实践

* 1.完整的团队（Whole Team）
  + 所有的小组成员应在同一个工作地点工作
  + 成员中必须有一个现场用户（On-site User）由他提出需求，确定开发优先级
  + 通常还设一个“教练”（Coach）角色

教练指导XP方法的实施，以及与外部的沟通和协调

* 2.计划对策（Planning Game）

包括两类：发布计划和迭代（Iteration）计划

* 3. 系统比喻 （Metaphor）

系统比喻是待开发软件的一个每个成员都熟悉的形象化比喻，相当于一个粗略的软件体系结构

* 4. 小发布（Small release）

经常、不断地发布可运行的、具有商业价值的小软件版本，供现场用户评估或最终使用

* 5. 测试（testing）

XP方法提倡测试优先，即先写测试后编代码（testing then coding）

* 6. 简单设计（Simple Design）
  + 设计只考虑当前定义的功能而不考虑以后需求的变化
  + 该设计是完成目前功能所需的最简洁的设计
* 7. 结对编程（Pair Programming）

一个程序员编程的同时，另一个程序员负责检查程序的正确性和可读性

结对的伙伴可以动态调整

* 8. 设计改进（Design Improvement）

在不影响程序的外部可见行为的情况下，按高内聚低耦合的原则对程序结构进行改进，保持代码简洁、无冗余

* 9. 持续集成（Continuous Integration）

每完成一个模块的开发（包括该模块的单元测试）后，立即将其组装到系统中，并进行集成测试，完成该集成测试后才能进行下一次集成

* 10. 代码全体共有（Collective code Ownership）

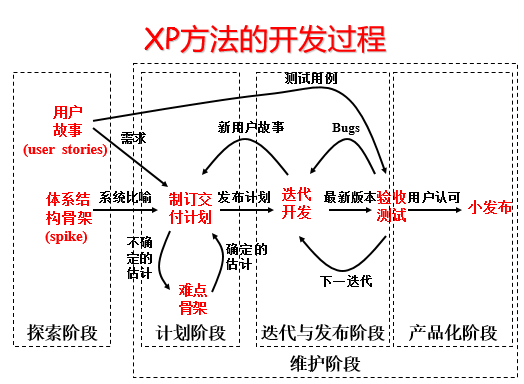
团队中的任何人可以在任何时候修改系统任何位置上的任何代码

团队的成员都可以参加模型的开发，又有系统比喻、结对编程、编码标准、持续集成等实践，这些都为代码全体共有提供了支持

* 11. 编码标准（Coding Standard）

XP方法强调制订一个统一的编码标准，包括命名、注释、格式等编程风格

* 12. 可持续步调（Sustainable Pace）

 每周40小时工作制

探索阶段

* 探索阶段的主要工作是开发初始的用户故事（User Stories ）和体系结构骨架（architecture spike）。
* 用户故事描述了系统高层的需求，它是制订发布计划的输入。
* 在探索阶段，试探找到系统中固定不变的部分（体系结构骨架），并找出一种形象的比喻，这种比喻描述了你打算如何构建系统，起到概念框架的作用。
* 探索阶段还应根据用户故事编制相应的测试用例，供以后验收测试时使用。

计划阶段

* 计划阶段的任务是根据用户故事描述的需求、系统体系结构骨架和系统比喻来制订迭代计划和发布计划。
* 使用你最熟悉的形式为用户故事建模，这个模型描述了用户故事的任务以及这些任务之间的关系。
* 通常图形方式（可以是草图）比文字描述更直观。
* 尽可能精确地估算工作量，这是制订计划的重要依据。对于那些不能确切估算其工作量的难点部分，要进一步作分析，直至能确定其工作量估算。

迭代到发布阶段

* 迭代到发布阶段根据迭代和发布计划，开发满足指定用户故事需求的软件，并与前面已完成的软件版本集成，得到软件的一个新版本。
* 根据在探索阶段编写的测试用例，进行验收测试。一旦发现错误或者通过验收测试想进入下一轮迭代时，就重复迭代开发的工作。
* 在这一阶段当客户提出新的用户故事，或者根据项目的进展情况认为有必要时，可以回到计划阶段，对迭代和发布计划做出修改或调整。

产品化阶段

* 产品化阶段的工作主要是确认迭代开发的软件已经做好进入产品化的准备。
* 在此阶段可进行更多的测试，如系统测试、负载测试、安装测试等。
* 另一个工作就是整理文档。虽然敏捷软件开发的价值观中强调“可运行软件高于详尽的文档”，但是，必要的文档仍是需要的。

可能要写的文档：

* 系统文档

系统文档的目的在于为系统提供一个总览，来帮助人们理解它。主要包括：系统技术体系结构和业务体系结构的总览、高层次的系统需求、关键设计决策的总结、体系结构图以及重要的设计模型（如果有的话）等。

* 操作文档

操作文档的内容包括：系统涉及的依赖关系，与其他系统、数据库以及文件文互的特性，对备份流程的参考引用，系统的联系人列表以及联系方法，系统的适用性及可靠性需求的总结，系统预期负载情况概况，以及排错指导原则。

* 支持文档

支持文档的内容包括：支持人员专用的培训教材，解决问题时作为参考的用户文档，排错指导原则，解决疑难问题时的上报流程，以及维护团队的联系列表。

* 用户文档

参考手册用于快速查询；用户指南用于指明系统的工作方式；支持指南用于指导如何获取其他的帮助；培训资料则主要用于培训。

维护阶段

* 维护阶段涵盖了计划阶段、迭代到发布阶段和产品化阶段
* 通常这个阶段主要包括面向产品的活动，如系统的运行和支持。

其他二 git表

**GitHub创建仓库提示代码**

echo "# 项目名" >> README.md

git init

git add README.md

git commit -m "first commit"

git remote add origin git@github.com:qiubaiying/项目名.git

git push -u origin master

**若仓库存在直接push**

git remote add origin git@github.com:qiubaiying/test.git

git push -u origin master

**创建仓库（初始化）**

# 在当前指定目录下创建

git init

# 新建一个仓库目录

git init [project-name]

# 克隆一个远程项目

git clone [url]

**添加文件到缓存区**

添加所有变化的文件

git add .

添加名称指定文件

git add text.txt

**配置**

设置提交代码时的用户信息

git config [--global] user.name "[name]"

git config [--global] user.email "[email address]"

**提交**

# 提交暂存区到仓库区

git commit -m "msg"

# 提交暂存区的指定文件到仓库区

$ git commit [file1] [file2] ... -m [message]

# 提交工作区自上次commit之后的变化，直接到仓库区

$ git commit -a

# 提交时显示所有diff信息

$ git commit -v

# 使用一次新的commit，替代上一次提交

# 如果代码没有任何新变化，则用来改写上一次commit的提交信息

$ git commit --amend -m [message]

# 重做上一次commit，并包括指定文件的新变化

$ git commit --amend [file1] [file2] ...

**远程同步**

# 下载远程仓库的所有变动

$ git fetch [remote]

# 显示所有远程仓库

$ git remote -v

# 显示某个远程仓库的信息

$ git remote show [remote]

# 增加一个新的远程仓库，并命名

$ git remote add [shortname] [url]

# 取回远程仓库的变化，并与本地分支合并

$ git pull [remote] [branch]

# 上传本地指定分支到远程仓库

$ git push [remote] [branch]

# 强行推送当前分支到远程仓库，即使有冲突

$ git push [remote] --force

# 推送所有分支到远程仓库

$ git push [remote] –all

分支

# 列出所有本地分支

$ git branch

# 列出所有远程分支

$ git branch -r

# 列出所有本地分支和远程分支

$ git branch -a

# 新建一个分支，但依然停留在当前分支

$ git branch [branch-name]

# 新建一个分支，并切换到该分支

$ git checkout -b [branch]

# 新建一个分支，指向指定commit

$ git branch [branch] [commit]

# 新建一个分支，与指定的远程分支建立追踪关系

$ git branch --track [branch] [remote-branch]

# 切换到指定分支，并更新工作区

$ git checkout [branch-name]

# 切换到上一个分支

$ git checkout -

# 建立追踪关系，在现有分支与指定的远程分支之间

$ git branch --set-upstream [branch] [remote-branch]

# 合并指定分支到当前分支

$ git merge [branch]

# 选择一个commit，合并进当前分支

$ git cherry-pick [commit]

# 删除分支

$ git branch -d [branch-name]

# 删除远程分支

$ git push origin --delete [branch-name]

$ git branch -dr [remote/branch]

**标签Tags**

添加标签 在当前commit

git tag -a v1.0 -m 'xxx'

添加标签 在指定commit

git tag v1.0 [commit]

查看

git tag

删除

git tag -d V1.0

删除远程tag

git push origin :refs/tags/[tagName]

推送

git push origin --tags

拉取

git fetch origin tag V1.0

新建一个分支，指向某个tag

git checkout -b [branch] [tag]

**查看信息**

# 显示有变更的文件

$ git status

# 显示当前分支的版本历史

$ git log

# 显示commit历史，以及每次commit发生变更的文件

$ git log --stat

# 搜索提交历史，根据关键词

$ git log -S [keyword]

# 显示某个commit之后的所有变动，每个commit占据一行

$ git log [tag] HEAD --pretty=format:%s

# 显示某个commit之后的所有变动，其"提交说明"必须符合搜索条件

$ git log [tag] HEAD --grep feature

# 显示某个文件的版本历史，包括文件改名

$ git log --follow [file]

$ git whatchanged [file]

# 显示指定文件相关的每一次diff

$ git log -p [file]

# 显示过去5次提交

$ git log -5 --pretty --oneline

# 显示所有提交过的用户，按提交次数排序

$ git shortlog -sn

# 显示指定文件是什么人在什么时间修改过

$ git blame [file]

# 显示暂存区和工作区的差异

$ git diff

# 显示暂存区和上一个commit的差异

$ git diff --cached [file]

# 显示工作区与当前分支最新commit之间的差异

$ git diff HEAD

# 显示两次提交之间的差异

$ git diff [first-branch]...[second-branch]

# 显示今天你写了多少行代码

$ git diff --shortstat "@{0 day ago}"

# 显示某次提交的元数据和内容变化

$ git show [commit]

# 显示某次提交发生变化的文件

$ git show --name-only [commit]

# 显示某次提交时，某个文件的内容

$ git show [commit]:[filename]

# 显示当前分支的最近几次提交

$ git reflog

**撤销**

# 恢复暂存区的指定文件到工作区

$ git checkout [file]

# 恢复某个commit的指定文件到暂存区和工作区

$ git checkout [commit] [file]

# 恢复暂存区的所有文件到工作区

$ git checkout .

# 重置暂存区的指定文件，与上一次commit保持一致，但工作区不变

$ git reset [file]

# 重置暂存区与工作区，与上一次commit保持一致

$ git reset --hard

# 重置当前分支的指针为指定commit，同时重置暂存区，但工作区不变

$ git reset [commit]

# 重置当前分支的HEAD为指定commit，同时重置暂存区和工作区，与指定commit一致

$ git reset --hard [commit]

# 重置当前HEAD为指定commit，但保持暂存区和工作区不变

$ git reset --keep [commit]

# 新建一个commit，用来撤销指定commit

# 后者的所有变化都将被前者抵消，并且应用到当前分支

$ git revert [commit]

# 暂时将未提交的变化移除，稍后再移入

$ git stash

$ git stash pop