软件工程复习

第一章　概述

1. 软件概念

软件是计算机系统中与硬件相互依存的另一部分，它是包括程序，数据及其相关文档的完整集合。

程序是按事先设计好的功能性能要求执行的指令序列(instructions)

数据是使程序能正常操纵信息的数据结构(data structures)

文档是与程序开发，维护和使用有关的图文材料(documents)

软件 = 程序+数据+文档

程序 = 算法+数据结构

软件的特点：复杂性、一致性、可变性、不可见性

2. 软件危机概念

软件危机是指在计算机软件的开发和维护过程中所遇到的一系列严重问题。

典型表型：

对软件开发成本和进度的估计常常很不准确。

用户对“已完成的”软件系统不满意的现象经常发生。

软件产品的质量往往靠不住。

软件常常是不可维护的。

软件通常没有适当的文档资料。

软件成本在计算机系统总成本所占的比例逐年上升。

软件开发生产率提高的速度，远远跟不上计算机应用普及的速度。

产生原因：

客观原因：软件本身的复杂性

主观原因：不正确的开发、管理方法

3. 软件工程概念

IEEE：

① 把系统的、规范的、可度量的途径应用于软件开发、运行和维护的过程，也就是把工程应用于软件；

② 研究①中提到的途径。

软件工程 7 条基本原理:

(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——基线配置管理

(4) 采用现代程序设计技术

(5) 结果应能清楚地审查

(6) 开发小组的成员应该少而精

(7) 承认不断改进软件工程实践的必要性

4. 软件职业道德

八项原则:

公众、客户和雇主、产品、判断、管理、专业、同行、自身

应遵守的规则:

从不为了个人利益而窃取数据;
从不散布或售卖你所工作的软件项目的专利信息;
从不恶意地破坏或修改别人的程序、文件或数据;
从不侵犯别人、别的团队或组织的隐私;
从不为了某种利益而非法入侵别人的系统;
从不创建或传播计算机病毒。

第二章　软件过程

1．软件过程概述

定义

软件过程是开发高质量软件产品的一系列相关活动(activities)的集合。

4 个基本活动

Software specification （软件规格说明）– 定义系统应该做什么;
Software design and implementation （软件设计与实现）– 定义系统的架构和实现;
Software validation （软件确认）– 检验系统是否满足用户的需求;
Software evolution （软件演变）– 随用户需求的变化变更系统。

2．软件过程描述

4 种组织形式

线性、迭代、增量、并行

其他描述

除了四个基本活动以外，通常还包括:

软件项目追踪和控制 – 控制项目进度;
风险管理 – 评估项目风险;
软件质量保证
技术审查– 评估工作产品;
软件配置管理– 管理文档、源代码和变更

除了活动以外，过程描述通常还包括:

Products – 过程活动的工作产品;（体系结构设计的工作产品是软件体系结构模型）
Roles – 软件过程中相关人员的职责;（项目经理、程序员等）

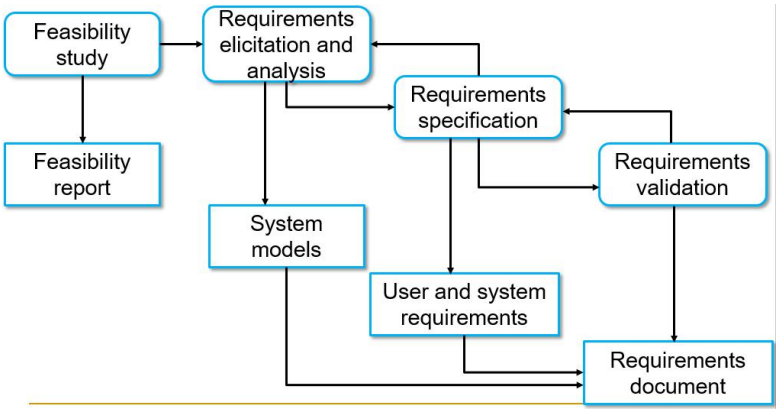
Pre- and post-conditions – 表达式，在过程活动进行前后都为真。(体系结构设计之初，前置条件是经过用户确认的所有需求；体系结构设计完成后，后置条件是经过评审的体系结构的 UML 模型。)

3．软件过程活动

①需求工程过程

软件需求规格说明:理解和定义用户需要什么样的服务,确定系统运行和开发时的约束。
需求工程过程:

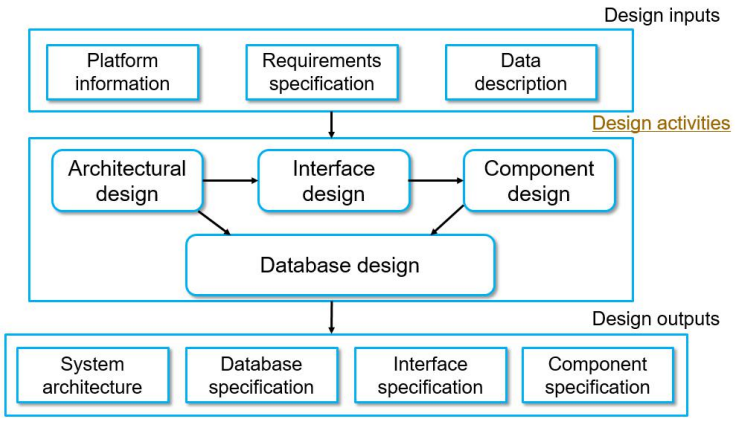
Feasibility study 构建系统在技术和经济上可行吗?
Requirements elicitation and analysis 用户对系统有哪些期望和要求? （模型）
Requirements specification 详细的定义需求
Requirements validation 验证需求的一致性、正确性



②设计与实现

把软件的规格说明转变为可执行的系统
软件设计：设计出实现了需求规格说明的软件结构
实现：把软件的结构转变为可执行的程序

设计过程



设计活动

Architectural design 确定系统的总体结构，子系统，子系统之间的关系，以及它们是如何分布的。

Interface design 定义系统构件之间的接口，接口的定义要无歧义。

Component design 定义系统构件。

Database design 定义系统的数据结构，以及如何用数据库来表示。

两阶段设计

总体设计

“概括的说，应该怎样实现目标系统？”

体系结构

详细设计

“应该怎样具体地实现这个系统？”

任务：细化概要设计所生成的各个模块，并详细描述程序模块的内部细节(算法，数据结构等)，形成可编程的程序模块。

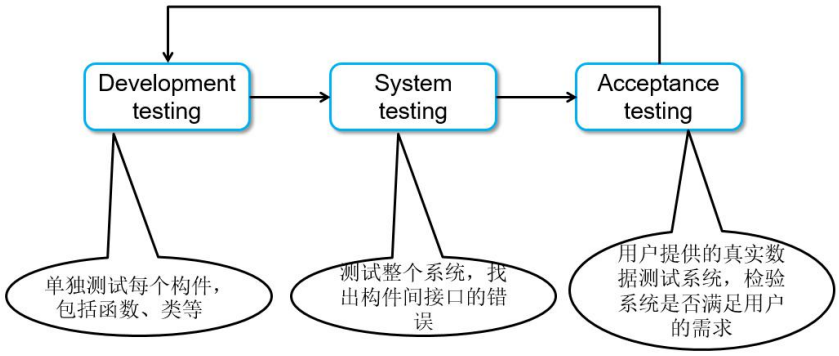
③软件确认

Verification and validation (V & V) ： 系统符合规格说明，满足用户的期望。

包括检验 （ checking ）、审查 （ review ） 和系统测试 （ system testing ）。

系统测试是使用测试用例来执行系统，检验系统的正确性。

测试是执行 V & V 活动的最常用的手段。



④软件演化

软件具有灵活性，经常变更。

业务环境变更，导致需求变更，支持业务的软件系统也要随之变更。

软件开发是创造性的活动，软件维护（maintenance）相比较缺少挑战性。

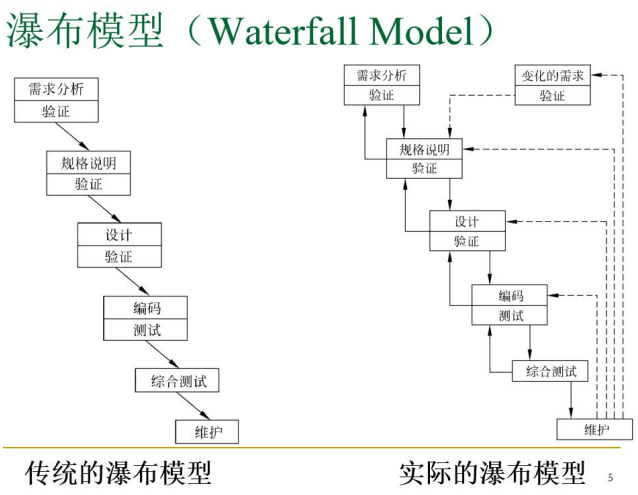
软件维护的开销占整个项目总成本的比例较大。

软件开发和维护是一个连续的过程，整个软件过程是一个演变的过程。

4．软件过程模型

软件过程模型是一个软件过程的抽象表示。

① 瀑布模型



特点：

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

优缺点：

1. 优点：

   规范的方法

   文档驱动

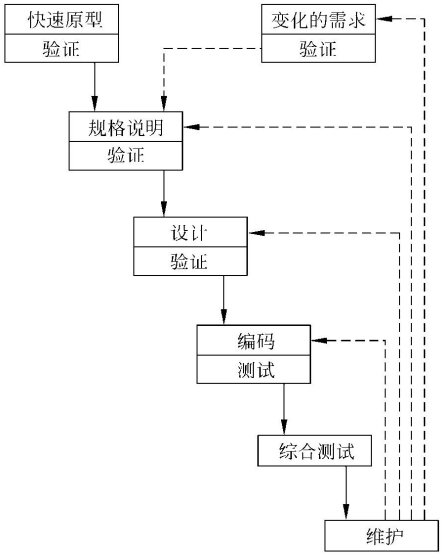
   质量保证

2. 缺点：

   文档驱动（需求规格说明）

适用项目：适用于需求已明确定义且固定的项目

② 快速原型模型



特点：

软件系统的初始版本

快速、迭代开发

常用于用户界面设计、验证系统功能需求

原型系统可以不考虑响应时间、错误处理、可靠性、程序质量等

优缺点：

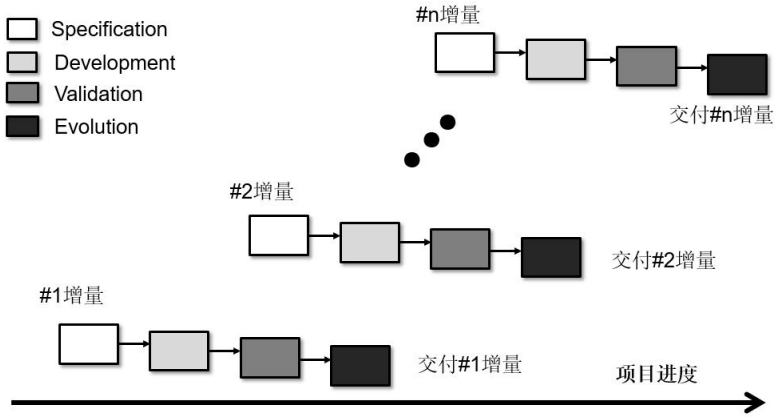
优点：

基本上是线性开发

确保交付的产品满足用户的需求

适用项目：适用于用户驱动的系统（需求模糊或随时间变化）

③ 增量模型



特点：

增量模型混合线性和并行过程流

每一个增量过程内部线性（原型）

增量过程之间并行

第一个增量为核心产品：基本需求

优缺点：

优点：

用户可以更早的使用软件

软件的核心功能能够得到更充分的测试

适用项目：

初始的软件需求明确

需要向用户快速提供一个有限功能的软件，在后续版本中精化和增加功能

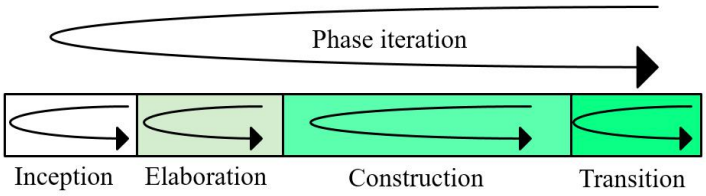
④ Rational 统一过程（RUP）

RUP 三个视图：

动态视图、静态视图、实践视图

RUP 四个阶段：

Inception（初始阶段）、Elaboration（精化阶段）、Construction（构建阶段）、Transition（移交阶段）



- 阶段内迭代（In-phase iteration）
- 跨阶段迭代（Cross-phase iteration）

RUP 工作流：

核心过程工作流

业务建模：使用业务用例对业务过程建模

需求：捕获用户的需求

分析与设计：分析模型、设计模型

实现：编程实现构件（自动代码生成）

测试：单个子系统的测试，各个子系统的集成测试

部署：可运行的产品移交给最终用户

核心支持工作流

配置与变更管理：管理系统的变更

项目管理：管理系统的开发过程

环境：软件开发工具

最佳实践：

迭代式开发、管理需求、使用基于构件的体系结构、可视化建模（UML）、验证软件质量、控制软件变更

特点：iterative and incremental

优缺点：融合传统软件过程模型的优点（混合过程模型）

适用项目：多功能和广泛适用

5. 敏捷软件开发

敏捷方法是一种思维方式和软件过程方法论

敏捷方法是一种以人为核心、迭代、增量式的开发方法

拥抱变化的开发流程

1．敏捷宣言

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

敏捷方法的原则：

客户参与、增量交付、人胜过过程、拥抱变化、保持简单

敏捷方法适用于：

小型或中等规模的商用软件产品的开发

组织内部的系统定制开发

敏捷=理念+优秀实践+具体应用

极限编程（eXtreme Programming）

应用最广泛的敏捷方法

把好的开发实践（迭代式开发）运用到极致

2.技术实践（XP）

①用户故事

用户是 XP 团队的一部分，负责提出需求

用户需求表示为场景或用户故事

用户故事写在卡片上，开发团队把用户故事分解为一系列任务

任务可作为进度安排和工作量估算的基础

②重构

在不改变代码外在行为的前提下，对代码做出修改，以改进程序的内部结构。

重构方法：

提炼函数（Extract Method）

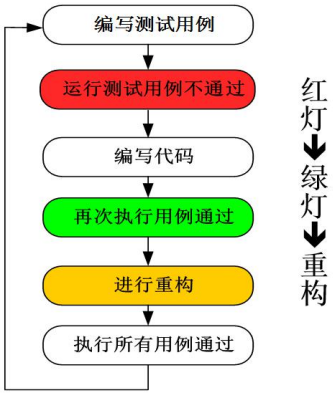
函数改名（Rename Method）

搬移函数（Move Method）

内联函数（Inline Method）

内联临时变量（Inline Temp）……

③TDD（测试驱动开发）



测试是 XP 方法的重要特点

XP 测试特性：

测试驱动开发（Test-driven development，TDD）

用户参与测试开发和确认的过程

使用自动化测试框架

④结对编程

两位程序员在一台电脑前工作，一个负责敲入代码，而另外一个实时检视每一行敲入的代码

⑤持续集成

一种软件开发实践，即团队的成员经常集成他们的工作，通常每个成员每天至少集成一次，每次集成通过自动化的构建（包括编译，发布，自动化测试）来验证，从而尽快地检测出集成错误。

3.管理实践

SCRUM 敏捷团队：PO（项目负责人）、Scrum Master（Scrum 教练）、Team

每日站立会议：每日工作前，团队成员的例行沟通机制，由 Scrum Master 组织，Team 成员全体站立参加

关键要点：准时开始，高效会议，问题追踪

第三章　需求工程

1.需求工程概述

①需求工程概念

需求工程：对系统应该提供的服务和所受到的约束进行理解、分析、建立文档、检验的过程。

②需求工程包含两个方面

需求开发：需求获取、需求分析、编写需求规格说明书、需求确认

需求管理：需求跟踪、需求变更控制、版本管理、需求复用

③需求的两个层次

用户需求：

用户需求是从用户角度描述的系统功能需求和非功能需求，通常只涉及系统的外部行为，而不涉及系统的内部特性。

系统需求：

系统需求是关于软件系统功能和运行约束的更详细的描述。定义了系统中需要实现的功能，描述了开发人员需要实现什么 。

一个用户需求可以扩展为多个系统需求

④需求的两种类型

功能需求：

描述系统应该提供的功能或服务。

功能需求应当具备完整性和一致性

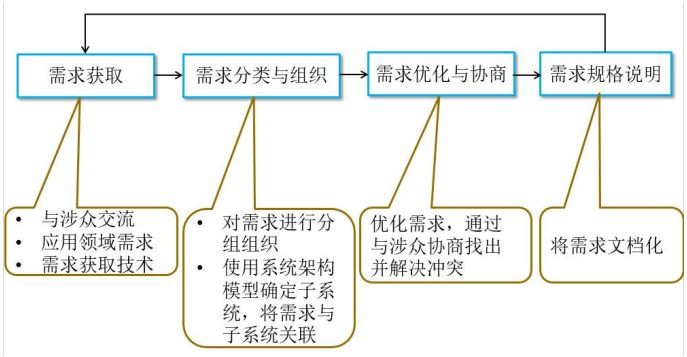
非功能需求：

系统的特性和约束

分类：产品需求、组织需求、外部需求

2.需求获取和分析

①过程



②技术

交流的对象：

涉众（Stakeholders）：最终用户、管理人员、工程师、领域专家等

需求获取技术：

访谈（Interviews） 封闭式（正式）、开放式（非正式）

情景（Scenarios） 用户将来如何使用系统的现实例子

用例（Use cases） 用例描述的是一种交互 每个用例就是一个情景

原型（prototypes）

3.需求规格说明

将用户需求和和系统需求文档化的过程
软件需求规格说明（software requirements specification, SRS）
　　用户需求不包含技术信息
　　系统需求包含技术信息，是系统设计的基础
SRS 依赖于所开发系统的类型和开发过程模型

Chapter	Description
序言	文档的预期读者、版本信息等
引言	简略的描述系统的功能，系统和其它系统的关系，描述系统如何满足组织的战略目标等。
词汇表	文档的技术词汇
用户需求	系统提供的服务，非功能需求，使用自然语言、图或者其它用户理解的符号，定义产品和过程标准。
系统架构	系统体系结构，子模块组织等。
系统需求	详细描述系统功能和非功能需求，定义与其它系统的接口。
系统模型	系统的图形模型，展示系统的构件之间的关系，以及系统和环境之间的关系。对象模型、数据流模型等。
附录	和当前开发的应用相关的详细的、具体的信息，包括硬件需求，数据库描述等。
索引	字母顺序索引、图索引、函数索引等。

SRS 结构

4.需求验证

一致性：任何一条需求不能和其它需求互相矛盾。
完整性：SRS 应包括用户需要的每一个功能或性能。
现实性：现有的软硬件技术可以实现需求。
有效性：需求确实满足用户的实际需要
需求验证技术：人工审查、原型

5.系统建模

建立系统抽象模型的过程，其中每个模型从不同的视角描述该系统。
1.模型分类
①环境模型
②交互模型：用例图、顺序图
③结构模型：类图
④行为模型：状态图
⑤数据驱动模型：数据流图
2.系统建模方法
①结构化分析
　　用抽象模型的概念，按照软件内部数据传递、变换的关系，自顶向下逐层功能分解的系统分析方法来定义系统的需求。（数据与处理过程分离）
　　数据模型（实体-联系图 ERD）P62

功能模型（数据流图 DFD）P40
行为模型（状态转换图）P65
数据字典　P47
②面向对象分析

第四章　面向对象分析

1.面向对象方法学概论

①面向对象方法学的要点
认为客观世界是由各种**对象**组成的，任何事物都是对象。
把所有对象都划分成各种对象**类**(类， class)，每个对象类都定义了一组数据和一组方法。
按照子类与父类的关系，把若干个对象类组成一个层次结构的系统(**继承**)。
对象彼此之间仅能通过传递**消息**互相联系。
OO =objects+class+inheritance+communication
②面向对象的基本概念
对象(object)
类(class)
消息(message)
封装(encapsulation)
继承(inheritance)
多态(polymorphism)
③面向对象建模
功能模型：描述系统功能—用例图 P224、顺序图
对象模型：描述系统组织结构—类图 P217
动态模型：描述系统控制结构—状态图 P65

2.面向对象分析的基本过程

OOA 就是抽取和整理用户需求并建立问题域精确模型的过程
①3 个子模型
静态结构（对象模型）最基本、最重要、最核心
交互次序（动态模型）
数据变换（功能模型）
②5 个层次
主题层、类与对象层、结构层、属性层、服务层

面向对象分析的过程:
寻找类与对象
识别结构
识别主题
定义属性

建立动态模型
建立功能模型
定义服务

3.需求陈述&分析建模

需求陈述是阐明“做什么”，而不是“怎样做”。
需求陈述应该描述用户的需求而不是提出解决问题的方法。
分析建模：ATM 机的例子 P235

第五章　设计

1.设计概述

①技术角度——4 部分
数据/类设计、体系结构设计、接口设计和构件设计
②管理角度——2 阶段
总体设计、详细设计
③设计过程
从回答“做什么”到回答“怎样做”
由两个主要阶段组成：
系统设计阶段，确定系统的具体实现方案
设想供选择的方案
选取合理的方案
推荐最佳方案
结构设计阶段，确定软件结构
功能分解
设计软件结构
设计数据库
制定测试计划
书写文档
审查和复审
④设计原理
模块化(Modularity)
把程序划分成独立命名且可独立访问的模块，每个模块完成一个子功能，把这些模块集成起来构成一个整体，可以完成指定的功能满足用户的需求。
抽象(Abstraction)
抽出事物的本质特性而暂时不考虑它们的细节。
逐步求精(Stepwise Refinement)
为了能集中精力解决主要问题而尽量推迟对问题细节的考虑。
信息隐藏(Information Hiding)
应该这样设计和确定模块，使得一个模块内包含的信息(过程和数据)对于不需要这些信

息的模块来说，是不能访问的。

模块独立

每个模块完成一个相对独立的子功能，并且和其他模块间的接口简单

耦合：是对一个软件结构内不同模块之间互连程度的度量。

非直接耦合（ Good）

数据耦合

特征耦合

控制耦合

外部耦合

公共耦合

内容耦合（ Bad）

内聚：标志一个模块内各个元素彼此结合的紧密程度。简单地说， 理想内聚的模块只做一件事情。

偶然内聚（ bad）

逻辑内聚

时间内聚

过程内聚

通信内聚

顺序内聚

功能内聚（ good）

2.面向对象设计

①设计准则

弱耦合

交互耦合：消息连接（要松散） 继承耦合：互为基类和派生类（要紧密）

强内聚

服务内聚、类内聚、一般-特殊内聚

可重用

With reuse 尽量使用已有的类

For reuse 如果确实需要创建新类，则在设计这些新类的协议时，应该考虑将来的可重复使用性。

②设计过程

数据/类设计

完成类图（确定属性和方法）、完成类的详细设计、数据设计

体系结构设计

软件体系结构给出软件的全貌图

接口设计

软件接口描绘信息流是如何输入输出系统的，以及构件间是如何通信的

三方面：用户界面、与外部系统的接口、内部不同系统间的接口

构件设计

完整的描述每个构件的内部细节

三方面：数据结构、算法、访问所有服务的接口

3.体系结构设计

体系结构设计是确定组成软件系统的子系统，以及子系统间通信的设计过程。
体系结构风格（style）：不同系统的体系结构设计方案存在着许多共性问题，把这些共性部分抽象出来，就形了具有代表性的和可广泛接受的体系结构风格
体系结构模式（ Pattern）：是一种设计模式，是针对于具体问题的解决方案，是对以前某一类问题的经验总结，在某种场景下可以套用。
经典体系结构风格：

数据为中心体系结构　数据库（被动）、黑板（主动）　例子：剪贴板
管道-过滤器　例子：编译器、shell 命令
主程序-子程序
分层体系结构　例子：操作系统、计算机网络、图书馆管理系统
客户机/服务器　例子：FTP 系统
MVC

4.构件设计

①构件的定义

系统中的一个模块化的、可部署的、可替换的部件，它封装了内部实现并提供了一系列的对外接口，这个部件称作构件

②构件设计过程

构件设计过程是一个迭代、不断精化的过程

第一次迭代：构件中的每个类都包括和实现相关的所有属性和方法

第二次迭代：为每个属性定义合适的数据结构；为每个方法设计算法（活动图、过程设计方法）

定义每个类和其它类进行通信的所有接口

③软件重用

重用级别：代码重用、设计结构重用、分析结果重用
重用成分：项目计划、成本估计、体系结构、需求模型和规格说明、设计、源代码、用户文档和技术文档、用户界面、数据、测试用例
类构件重用方式：实例重用（封装性）、继承重用（继承性）、多态重用（多态性）

④基本设计原则

开闭原则（ OCP）　抽象是关键
针对接口编程原则　针对抽象编程
类的单一职责原则（ SRP）
Liskov 替换原则（ LSP）
接口隔离原则（ ISP）
迪米特法则（ LoD）
合成/聚合复用原则（CARP）

⑤构件详细设计

图形工具　程序流程图、盒图（N-S 图）、问题分析图（PAD 图）
列表工具　判定表、判定树
语言工具　过程设计语言（PDL，也称伪码）

⑥基于构件的软件工程

2 个子过程：领域工程、基于构件的开发

构件标准：Sun's Enterprise Java Beans（ EJB）、Microsoft's COM and .NET、 CORBA Component Model（ CCM）

5.用户界面设计

黄金规则：置于用户控制之下、减少用户记忆负担、保持界面一致

6.基于模式的设计

①设计模式的概念

设计模式（ Design pattern）是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。

原理： 面对接口编程，而不是面对实现。

目标原则是： 降低耦合,增强灵活性。

②分类

创建型模型（有关对象创建的模式）、结构性模型（描述对象构造和组成的方式）、行为型模型（描述一组对象交互的方式）

③工厂模式

工厂模式专门负责将大量有共同接口的类实例化，工厂模式可以动态决定将哪一个类实例化。

简单工厂模式、工厂方法模式、抽象工厂模式

④模板方法模式

定义一个操作中的算法的骨架，而将这些步骤延迟到子类中。 Template Method 模式使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤。

第六章　实现

1.编码

把软件设计结果翻译成用某种程序设计语言书写的程序。

①良好的编程实践

变量的命名要有意义且一致

代码的 self-documenting

使用参数

②编码风格

程序内部的文档

数据说明

语句构造

2.配置管理

1.概念

①配置管理（SCM）

变更管理

一组管理变更的活动（ umbrella activity）

贯穿于整个软件过程中的保护性活动

“如果你不控制变更，那么变更将控制你”

一个未受控制的变更流可以很容易的将一个运行良好的软件项目带入混乱，结果会影响软件质量并且会推迟软件交付。

目标：

每个工作产品都可以标识、跟踪和控制

每个变更都可以跟踪和分析

每个需要知道变更的人员都得到通知

②软件配置项（SCI）

计算机程序（源代码和可执行程序）

描述计算机程序的文档（针对技术开发者和用户）

数据（包含在程序内部和程序外部）

③基线（baseline）

已经通过了正式复审的规格说明或中间产品，它可以作为进一步开发的基础，并且只有通过正式的变化控制过程才能改变它。简而言之，就是通过了正式复审的软件配置项。

④配置管理库（ SCM Repository）

配置管理库也称受控库，用于存储软件配置项以及相关配置管理信息。

配置管理库特征：

版本控制：保留所有历史版本，能够回溯

关联跟踪

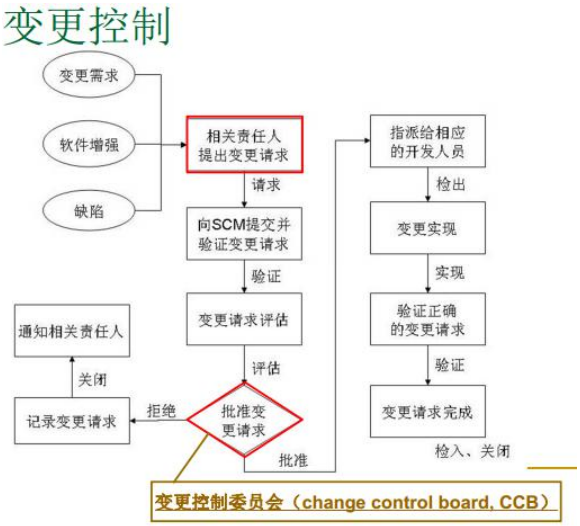
需求跟踪：正向跟踪、逆向跟踪

审计跟踪：变更源（ when, why, who）

2.配置管理过程

标识软件配置中的对象　基本对象、聚集对象　（名字、描述、资源表、实现）

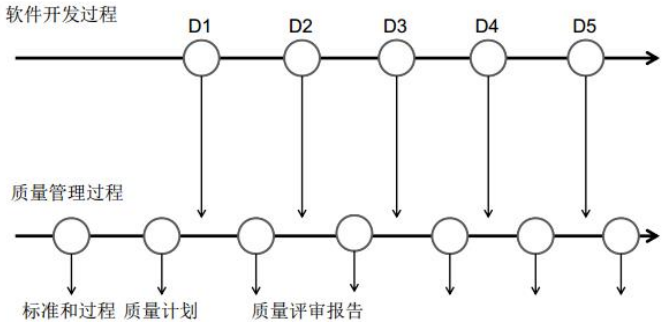
变更控制



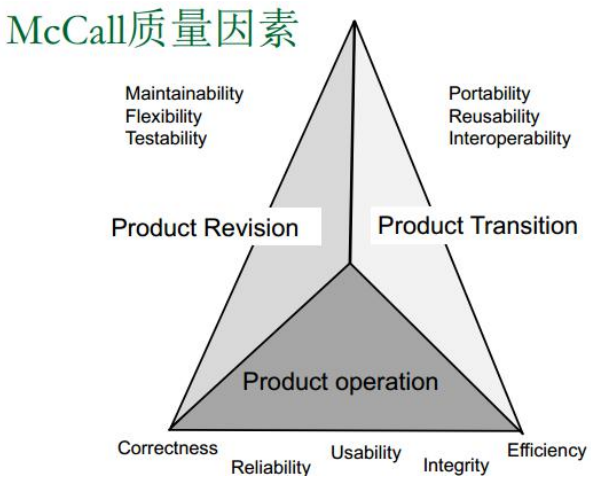
版本控制　分支与合并
配置审计
状态报告

3.质量管理

①原则
在组织层上，质量管理关注于建立标准组织过程
在项目层上，质量管理关注于标准过程的实施
质量管理关注于建立项目的质量计划
②过程



③软件质量
有效的软件过程
有用的产品
为软件公司和软件的使用者增加显著的价值



④软件质量保证（SQA）
评审
　　技术评审　非正式评审（桌面检查）、正式技术评审（走查、审查）
　　代码审查
软件测试
程序正确性证明
　　Floyd 不变式断言
　　　　输入断言
　　　　循环不变式

输出断言

4.软件度量

软件度量（ Metrics）是指对软件产品、软件开发过程及软件开发项目的定量描述。
软件度量三维度

产品度量：软件开发过程中所生成的各种文档和程序

需求模型度量：检测需求模型，预测将来系统的规模

功能点度量（FP）：度量系统提供的功能

$$FP = UFP \times TCF = \sum_{i=1}^{5} a_i * wf \times \left[0.65 + 0.01 \times \sum_{i=1}^{14} F_i\right]$$

（输入、输出、查询、主文件、外部接口）

设计模型度量

Card 软件设计复杂性度量

Fenton 简单形态度量

OOD 度量

源代码度量

Halstead 方法

程序中使用的不同运算符(包括关键字)的个数n_1，以及不同操作数(变量和常数)的个数n_2。预测程序长度的公式如下：

$$H = n_1 \log_2 n_1 + n_2 \log_2 n_2$$

代码行

- 由多名有经验的软件工程师分别做出估计。
- 每个人都估计程序的最小规模(a)、最大规模(b)和最可能的规模(m)，
- 分别算出这3种规模的平均值、和之后，再用下式计算程序规模的估计值：

$$L = \frac{\bar{a} + 4\bar{m} + \bar{b}}{6}$$

单位：

LOC或KLOC。

过程度量：与软件开发有关的各种活动，如软件设计等

项目度量：度量项目规模、项目成本、项目进度等

第七章　软件测试

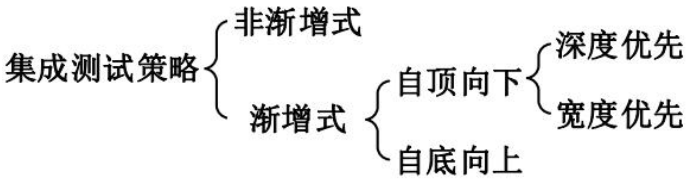
1.软件测试基础

①定义
证明程序中有错误
②目标
测试是为了发现程序中的错误而执行程序的过程；
好的测试方案是极可能发现迄今为止尚未发现的错误的测试方案；
成功的测试是发现了至今为止尚未发现的错误的测试。
③软件测试准则
所有测试都应该能追溯到用户需求
应该远在测试开始之前就制定出测试计划
把 Pareto 原理应用到软件测试中（80%/20%原理）
应该从“小规模”测试开始，并逐步进行“大规模”测试
穷举测试是不可能的
为了达到最佳的测试效果，应该由独立的第三方从事测试工作
④测试方法
黑盒测试(功能测试)
白盒测试(结构测试)

2.软件测试策略

①概念
指导测试活动的执行
②常用测试策略
在系统开发完成后对整个系统进行测试
采用增量的方式进行测试：先测各个模块，再测模块的集成，再测整个系统
每天系统完成部分功能后都进行测试
③测试步骤
单元测试
集成测试
确认测试
系统测试
④单元测试（人工测试“代码审查”或计算机测试　白盒）
5 个测试重点：
　　模块接口、局部数据结构、重要执行通路、出错处理通路、边界条件
单元测试过程：
　　驱动程序：接收测试数据，传送给被测试的模块，并且打印出有关的结果。
　　存根程序：代替被测模块所调用的模块
自动化测试
　　JUnit

⑤集成测试



自顶向下需要存根程序，不用驱动程序
自底向上不需要存根程序，需要驱动程序。
改进：混合法，上层使用自顶向下，下层使用自底向上
回归测试：

指重新执行已经做过的测试的某个子集，以保证测试过程中的变化没有带来非预期的副作用。可以人工或使用自动化的捕获回放工具。

⑥确认测试

确认测试必须有用户积极参与，或者以用户为主进行。
软件配置复查
Alpha 测试：

由用户在开发者的场所进行，并且在开发者对用户的“指导”下进行测试。在受控的环境中进行。

Beta 测试（验收测试）：

由软件的最终用户们在一个或多个客户场所进行。开发者通常不在现场。用户记录测试过程中遇到的问题，并报告给开发人员。

3.软件测试技术

①测试方案概念

②白盒测试技术

5 种逻辑覆盖

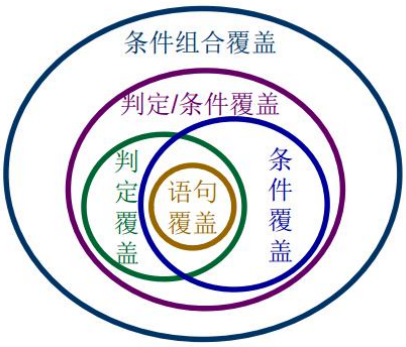
语句覆盖：选择足够多的测试数据，使被测程序中每个语句至少执行一次。

判定覆盖：不仅每个语句必须至少执行一次，而且每个判定的每种可能的结果都应该至少执行一次。

条件覆盖：不仅每个语句至少执行一次，而且使判定表达式中的每个条件都取到各种可能的结果。

判定/条件覆盖：使得判定表达式中的每个条件都取到各种可能的值，每个判定表达式也都取到各种可能的结果。

条件组合覆盖：要求选取足够多的测试数据，使得每个判定表达式中条件的各种可能组合都至少出现一次。



基本路径测试

画出流图（ Flow graph）

计算环形复杂度

- 定量度量程序的逻辑复杂度
- 环形复杂度$V(G)$计算方法
  - $V(G)$=流图中的区域数
  - $V(G)=E-N+2$，其中E是流图中的边数，N是结点数
  - $V(G)=P+1$，其中P是流图中判定结点的数目

以该复杂度为指南定义独立路径的基本集合

从该基本集合导出测试用例

③黑盒测试技术

等价划分:

把程序的输入域划分成若干个数据类，每一类以一个代表性的测试数据进行测试，这个数据就等价于这一类中的其它数据。

边界值分析:

处理边界情况时程序最容易发生错误。

选取的测试数据应该刚好等于、刚刚小于和刚刚大于边界值。

通常设计测试方案时总是联合使用等价划分和边界值分析两种技术。

错误推测:

列举出程序中可能有的错误和容易发生错误的特殊情况，并且根据它们选择测试方案。

By Vodka