**Chapter\_01 Software & Software Engineering**

什么是软件Software is: (1) instructions (2) data (3)documentation

软件的特点

1. Software is developed or engineered, it is not manufactured in the classical sense.

2. Software doesn't "wear out."

3. Although the industry is moving toward component-based construction, most software continues to be custom-built.

遗留软件Legacy Software，有商业价值，必须进行变更

软件危机Software crisis

Software Engineering，为了解决软件危机，需要用工程化的思想对待软件，所以产生软件危机

IEEE定义：Software Engineering:

(1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.

(2) The study of approaches as in (1).

A Layered Technology，从技术角度说，软件工程包括四个层次

Tools、methods、process model、a quality focus

A Process Framework

Framework activities(work tasks/work products/milestones & deliverables/QA checkpoints)

Umbrella Activities

Framework Activities框架活动内容

Communication/ Planning/ Modeling/ Construction / Deployment

Hooker’s General Principles

1: The Reason It All Exists

 2: KISS (Keep It Simple, Stupid!)

 3: Maintain the Vision

 4: What You Produce, Others Will Consume

 5: Be Open to the Future

 6: Plan Ahead for Reuse

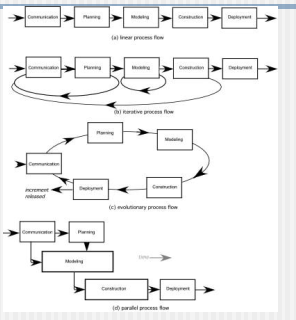
 7: Think!

**Chapter\_02 Process Models过程模型**

The life cycle of Software软件生命周期：是指一个软件项目被提出并着手实施开始，到该软件报废或停止使用为止的一段时间。

Software Process软件过程定义了软件生产的一系列活动， 这些活动贯穿于软件开发的整个过程。

Process Flow 四种过程流



Prescriptive Models

The Waterfall Model 最典型的代表

瀑布模型的优点

通过设置里程碑，明确每阶段的任务与目标;可为每阶段制定开发计划，进行成本预算，组织开发力量;通过阶段评审，将开发过程纳入正确轨道;强迫开发人员采用规范化的方法。

瀑布模型的缺点

缺乏灵活性，不能适应用户需求的改变;开始阶段的小错误被逐级放大，可能导致软件产品报废;返回上一级的开发需要十分高昂的代价;随着软件规模和复杂性的增加，软件产品成功的机率大幅下降.

文档推动，按部就班，缺乏灵活性，需求明确

Evolutionary Models: Prototyping原型模型

原型模型的优点

开发者与用户充分交流，可以澄清模糊需求，需求定义比其他模型好得多;

开发过程与用户培训过程同步;

为用户需求的改变提供了充分的余地;

开发风险低，产品柔性好;开发费用低，时间短;系统易维护，对用户更友好;

原型模型的缺点

开发者在不熟悉的领域中不易分清主次，原型不切题;

产品原型在一定程度上限制了开发人员的创新;

随着更改次数的增多，次要部分越来越大， “淹没” 了主要部分;

原型过快收敛于需求集合，而忽略了一些基本点;

资源规划和管理较为困难，随时更新文档也带来麻烦;

只注意原型是否满意，忽略了原型环境与用户环境的差异;

Evolutionary Models: The Spiral螺旋模型，risk driving 风险驱动，适合需求动态变化的

优点

 支持用户需求的动态变化。 具有良好的可扩充性和可修改性。 也支持软件系统的可维护性， 每次维护过程只是沿螺旋模型继续多走一两个周期。

 原型易于用户和开发人员共同理解需求，还可作为继续开发的基础，并为用户参与所有关键决策提供了方便。

 螺旋模型为项目管理人员及时调整管理决策提供了方便，进而可降低开发风险。

 支持需求不明确、特别是大型软件系统的开发，并支持面向规格说明、面向过程、面向对象等多种软件开发方法，是一种具有广阔前景的模型。

缺点：

1. 果每次迭代的效率不高， 致使迭代次数过多， 将会增加成本并推迟提交时间；
2. 使用该模型需要有相当丰富的风险评估经验和专门知识， 要求开发队伍水平较高

Evolutionary Models: Concurrent 协同模型

优点：

1、可用于所有类型的软件开发，尤其多个团队共同参与一个工程，对于C/S结构更加有效。

2、可以随时查阅到开发的状态。

The Incremental Model增量模型

融合了线性顺序(瀑布)模型的基本成分和原型的迭代特征。采用随着日程时间的进展而交错的线性序列。每一个线性序列产生软件的一个可发布的“增量” 。

适用：迫切需要为用户提供一套功能有限的产品，在后续产品中再细化或拓展。

RAD Model

RAD是一个线性顺序的软件开发模型，是线性顺序模型的一个“高速”变种，强调极短的开发周期。通过使用基于构件的建造方法获得了快速开发,过程使得一个开发组能够在很短时间内(如60到90天)创建出“功能完善的系统”

Other Process Models

Component based development

Formal methods

AOSD

Unified Process

**Chapter\_03 Agile Development敏捷开发**

The Manifesto for Agile Software Development敏捷宣言的内容

•Individuals and interactions over processes and tools

•Working software over comprehensive documentation

•Customer collaboration over contract negotiation

•Responding to change over following a plan

以人为本

What is “Agility”?什么是敏捷

Effective (rapid and adaptive) response to change

 Effective communication among all stakeholders

 Drawing the customer onto the team

 Organizing a team so that it is in control of the work performed

Yielding …

 Rapid, incremental delivery of software

什么是敏捷开发？一种以人为核心、迭代、循序渐进的开发方法。

敏捷方法

XP -eXtreme Programing极限编程:

SCRUM：是一种迭代的增量化过程，用于产品开发或工作管理 。

水晶方法Crystal:把不同类型的项目采用不同的方法。

FDD－特性驱动 Feature Driven Development，

DSDM-Dynamic System Development Methodology，

ASD-Adaptive Software Development，

误区

误区一:敏捷是"一个”过程

误区二:敏捷仅是个软件过程

误区三:敏捷是反文档的

误区四:为了敏捷而敏捷

误区五:重做就是重构

What is Spike solution of XP?

有困难，先设计出原型，然后再进行困难解决

What are the agile process model ? 敏捷过程模型有哪一些

**Chapter\_04 Understanding Requirements**

Requirements Engineering-I，需求工程的任务

Inception

Elicitation

Elaboration

Negotiation

Specification

Validation

Requirements management

Quality Function Deployment 质量功能部署QFD 三种需求

Function deployment

Information deployment

Task deployment

Value analysis

Building the Analysis Model建立分析模型的四种元素

Elements of the analysis model

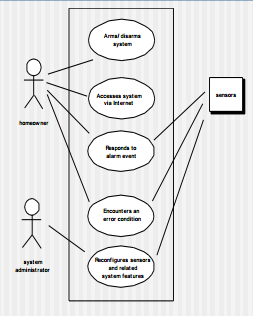
Scenario-based elements

Class-based elements

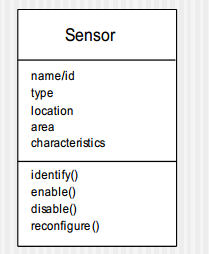
Behavioral elements

Flow-oriented elements

Use-Case Diagram基于场景的典型代表



Class Diagram类图

名称，属性，操作

State Diagram状态图 名称，变量，活动

**Chapter\_05 Requirements Modeling Scenarios, Information, and Analysis Classes**

分析模型是系统描述和设计模型的桥梁

Use-Case Diagram

Activity Diagram

System类的类图

CRC Models

 Class-responsibility-collaborator (CRC)

**Chapter\_06 Design Concepts**

Design 软件设计宣言的内容

Firmness/Commodity/Delight

Design Model主要包括的设计

Data / Class Design

Architectural Design

Interface Design

Component -Level Design

设计的目标：质量

Fundamental Concepts

 Abstraction—data, procedure, control

 Architecture—the overall structure of the software

 Patterns—”conveys the essence” of a proven design solution

 Separation of concerns—any complex problem can be more easily handled if it is subdivided into pieces

 Modularity—compartmentalization of data and function

 Hiding—controlled interfaces

 Functional independence—single-minded function and low coupling

 Refinement—elaboration of detail for all abstractions

 Aspects—a mechanism for understanding how global requirements affect design

 Refactoring—a reorganization technique that simplifies the design

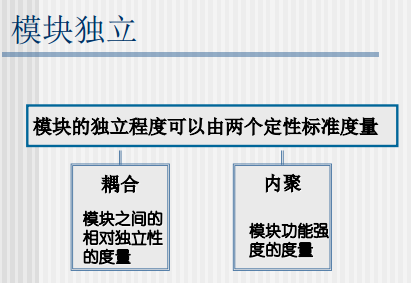
 OO design concepts—Appendix II

 Design Classes—provide design detail that will enable analysis classes to be implemented

Functional Independence独立性判断因素

Cohesion is an indication of the relative functional strength of a module.

Coupling is an indication of the relative interdependence among modules.

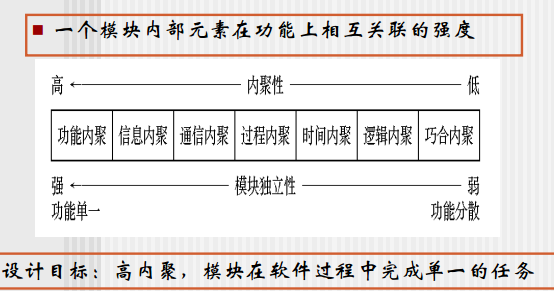


Coupling耦合



内容耦合，耦合性最高，独立性最弱

Cohesion内聚



功能内聚，内聚性最高，独立性最强

高内聚，低耦合

Refactoring

Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code [design] yet improves its internal structure.

从工程管理的角度看，软件设计包括：概要设计，详细设计

从技术的角度看，软件设计包括：数据设计体系，结构设计，构件设计，接口设计

**Chapter\_07 Architectural Design 体系结构设计**

Why Architecture

(1) analyze the effectiveness of the design

(2) consider architectural alternatives

(3) reduce the risks

Why is Architecture Important?

Representations of software architecture are an enabler

The architecture highlights early design decisions

Architecture “constitutes a relatively small, intellectually graspable mode

Architectural Styles 体系结构的风格

Data-centered architectures

 Data flow architectures

 Call and return architectures

 Object-oriented architectures

 Layered architectures

Architectural Patterns

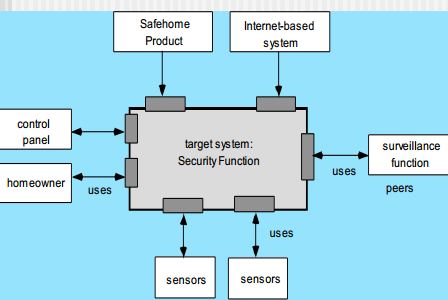
Concurrency

Persistence

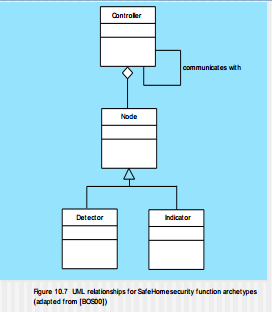
Distribution

Architectural Design

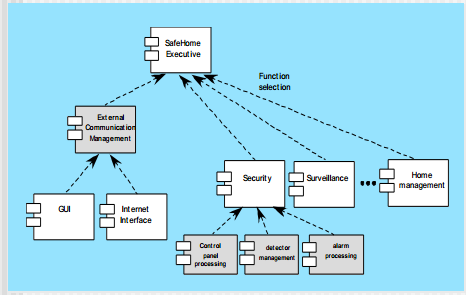
1The software must be placed into context



2A set of architectural archetypes should be identified



3The designer specifies the structure of the system by defining and refining software components that implement each archetype



Analyzing Architectural Design，What’s the process of architectural design?

1. Collect scenarios.

2. Elicit requirements, constraints, and environment description.

3. Describe the architectural styles/patterns that have been chosen to address the scenarios and requirements:

• module view

• process view

• data flow view

4. Evaluate quality attributes by considered each attribute in isolation.

5. Identify the sensitivity of quality attributes to various architectural attributes for a specific architectural style.

6. Critique candidate architectures (developed in step 3) using the sensitivity analysis conducted in step 5.

**Chapter\_08 Component-Level Design**

What is a Component?

a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces.（omg）

Basic Design Principles基本设计原则

The Open-Closed Principle (OCP).开闭

The Liskov Substitution Principle (LSP).

Dependency Inversion Principle (DIP).

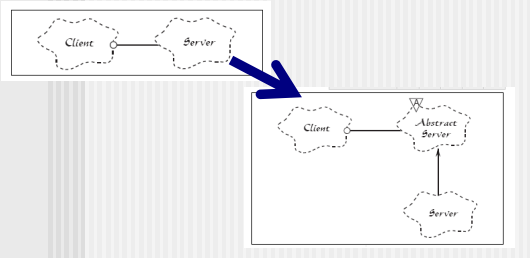
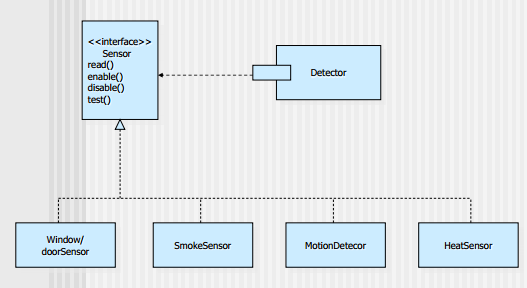
The Interface Segregation Principle (ISP).

The Release Reuse Equivalency Principle (REP).

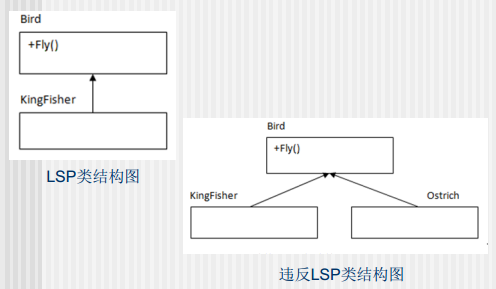
The Common Closure Principle (CCP).

The Common Reuse Principle (CRP).

OCP (Open-Closed Principle )

 接口抽象出来

LSP

违反，鸵鸟不会飞

基本设计原则

REP发布复用等价性原则---对类打包管理，同时升级。

CCP共同封装原则----一同变更的类应该合在一起。

CRP共同复用原则----可能一起被复用的类才能打包到一块。

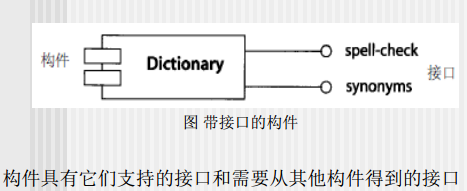
Design Guidelines

Components

Interfaces

Dependencies and Inheritance

构件的UML表示



内聚与耦合密切相关，同其它模块强耦合的模块意味者弱内聚，强内聚模块意味着与其它模块间松散

耦合。设计目标：力争强内聚、弱耦合

算法设计模型

 以细节的层次表示算法， 它能对质量进行评审

 选择：

 图解 (例如：流程图、盒图)

 伪代码(例如： PDL) ... 很多选择

 编程语言

 决策表

**Chapter\_09 Software Testing Strategies测试策略**

Strategic Approach

To perform effective testing, you should conduct effective technical reviews. By doing this, many error swill be eliminated before testing commences.

Testing begins at the component level and works "outward" toward the integration of the entire computer based system. Different testing techniques are appropriate for different software engineering approaches and at different points

in time.

Testing is conducted by the developer of the software and (for large projects) an independent test group.

Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.

Testing Strategy测试策略

1Unit test

2Integration test

3Validation test

4System test

High Order Testing高阶测试

Validation testing——Focus is on software requirements

System testing—— Focus is on system integration

Alpha/Beta testing——Focus is on customer usage，根据场景不同，开发者而发，在实际自己环境白塔

Recovery testing——forces the software to fail in a variety of ways and verifies that recovery is properly performed

Security testing——verifies that protection mechanisms built into a system will, in fact, protect it from improper penetration

Stress testing——executes a system in a manner that demands resources in abnormal quantity, frequency, or volume

Performance Testing——test the run-time performance of software within the context of an integrated system

调试不是测试，测试中包含调试Debugging

Debugging Techniques

brute force / testing

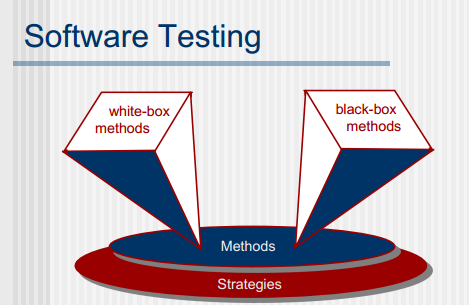
backtracking

induction

deduction

集成测试有哪些What are the strategies of the integration testing?

**Chapter\_10 Testing Conventional Applications**

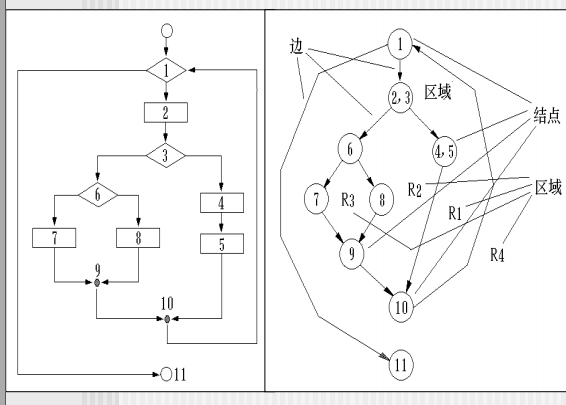
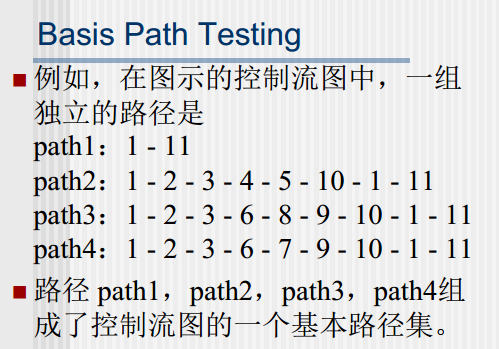
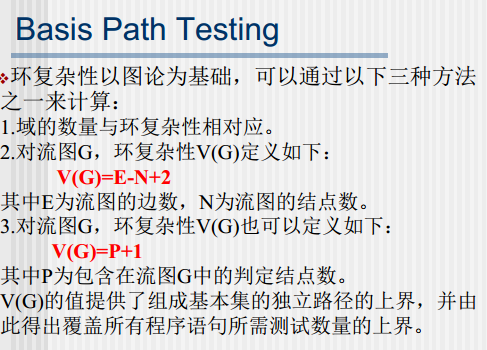


从内部关注程序流程，路径对错，指白盒测试

从外部看，关注功能性测试，黑盒测试

基本路径测试 典型白盒测试

程序流图

黑盒测试——等价类划分

Equivalence Partitioning

等价类

 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示程序中的错误都是等效的。

等价类划分

 等价类划分是一种典型的黑盒测试方法。

 这一方法完全不考虑程序的内部结构，只依据程序的规格说明来设计测试用例

测试用例

不能出现相同的测试用例

**Chapter\_11 Quality Concepts**

Quality 什么是质量 a characteristic or attribute of something. 软件质量困境

For software, two kinds of quality may be encountered:

Quality of design

Quality of conformance

User satisfaction = compliant product + good quality + delivery within budget and schedule

Software Quality

An effective software process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it.

The Software Quality Dilemm

一个我们想做到非常非常好的软件，达到客户需求，会花费很多的人力物力财力；另一方面，为了追求利益，为了赶工，交付足够好的就好了

If you produce a software system that has terrible quality, you lose because no one will want to buy it.

If on the other hand you spend infinite time, extremely large effort, and huge sums of money to build the absolutely perfect piece of software, then it's going to take so long to complete and it will be so expensive to produce that you'll be out of business anyway.

Either you missed the market window, or you simply exhausted all your resources.

So people in industry try to get to that magical middle ground where the product is good enough not to be rejected right away, such as during evaluation, but also not the object of so much perfectionism and so much work that it would take too long or cost too much to complete. [Ven03]