目录

第一章：软件工程概述.....7
软件：.....7
软件工程：.....7
软件过程：.....7
软件过程模型.....7
软件工程方法.....7
CASE：.....7
软件基本属性：.....7
软件工程面临的主要挑战：.....7
职业道德：.....7
第三章 软件过程.....8
1、通用软件过程模型.....8
瀑布模型.....8
瀑布模型问题和适用性.....8
增量式开发/交付.....8
增量开发/交付要点：.....8
增量开发/交付优点：.....8
极限编程.....8
进化式开发（探索式）.....8
进化式开发（抛弃式）.....8
进化式开发问题和适用性.....8
形式化系统开发.....9
形式化系统问题和适用性.....9
面向复用的开发.....9
过程反复.....9
螺旋式开发.....9
2、软件描述.....9
需求工程过程（软件描述主要过程）.....9
3、软件设计实现.....9

设计.....9
实现.....9
设计方法.....9
4、软件有效性验证.....10
5、软件进化.....10
第五章 软件需求.....10
需求工程.....10
需求.....10
需求类型.....10
功能和非功能需求.....10
需求的完整性和一致性.....10
需求的相互作用.....11
需求度量.....11
1、用户需求.....11
自然语言问题.....11
2、系统需求：标准的格式、一致的语言.....11
结构化自然语言描述.....11
基于格式的描述.....11
基于 PDL 的需求定义（伪代码）.....11
基于 PDL 的缺点和适用.....11
3、需求文档（SRS）.....11
第六章 需求工程过程.....12
需求工程过程.....12
1、可行性研究.....12
可行性研究焦点.....12
2、需求导出和分析（需求导出、需求发现）.....12
系统需求导出和分析的困难.....12
基于方法的分析.....12
(1)面向视点的导出.....12
视点类型.....12
面向视点的需求定义方法（VORD）.....13

VORD 模板 13
(2)场景（脚本） 13
场景描述 13
事件场景 13
VORD 中数据和控制分析的符号 13
(3)用例 13
序列图 14
3、需求有效性验证 14
需求检查 14
需求有效性验证技术 14
4、需求管理 14
需求变更 14
持久和易变的需求 14
可追溯性 15
第七章 系统模型 15
系统建模 15
结构化方法 15
结构化方法的不足 15
不同角度描述 15
系统模型类型 15
上下文模型 16
行为模型 16
数据流模型 16
状态机模型（摩尔机） 16
结构模型 16
(1)语意数据模型（E-R） 16
数据字典 16
(2)对象模型 16
对象类 16
继承模型（泛化） 16
多重继承 17

对象聚合.....17
交互模型（对象行为建模）.....17
第八章 软件原型.....17
系统原型.....17
作用和优点.....17
软件原型开发.....17
进化式原型开发（探索式进化开发）.....17
进化式原型开发过程.....18
进化式原型优点.....18
抛弃式原型开发（抛弃式进化开发）.....18
快速原型开发技术（原型开发重要基础）.....18
动态高级语言开发.....18
数据库编程语言.....18
组件和应用集成（复用）.....18
可视化编程.....18
第十章 体系结构设计.....19
体系结构的好处.....19
体系结构设计过程.....19
子系统和模块.....19
体系结构模型.....19
容器模型.....19
容器模型优缺点.....19
客户端-服务器模型（C/S 模型）.....20
C/S 模型优缺点.....20
抽象机模型（分层体系结构）.....20
控制模型.....20
(1)集中式控制.....20
调用-返回模型.....20
管理者模型.....20
(2)基于事件的控制（事件驱动系统）.....20
广播模型.....20

中断驱动模型.....20
模块分解.....21
对象模型.....21
数据流模型（管道模型/过滤器模型）.....21
领域相关的体系结构.....21
类模型.....21
参考模型.....21
第十九章 检验和有效性验证（V&V）.....21
检验.....21
有效性验证.....21
检验和有效性验证过程.....21
软件检查（静态检验）（昂贵的）.....21
程序检查的预处理.....22
检查团队.....22
检查清单.....22
自动静态分析.....22
软件测试（动态检验）.....22
测试类型.....22
测试和调试.....22
软件开发的倒 V 模型.....22
净室软件开发.....22
特征.....23
第二十章 软件测试.....23
测试过程.....23
缺陷测试.....23
测试的优先级：.....23
测试数据和测试用例.....23
(1)黑盒测试.....23
等价划分.....23
(2)结构化测试（白盒测试/路径测试/结构测试）.....24
路径测试.....24

环路复杂性.....24
集成测试.....24
集成测试方法.....24
测试方法比较.....24

第一章：软件工程概述

软件：计算机程序和相关的文档。

通用产品、定制产品

软件工程：关于软件生产各个方面的一门工程学科

计算机科学：

研究计算机组成和软件系统基础的理论和方法；

系统工程：

以计算机为基础的系统开发的方方面面，包括软硬件，包含软件工程

软件过程：软件系统的规格描述、设计、实现和测试的互相连贯的一系列活动

软件描述→软件设计和开发→软件有效性验证→软件维护和进化

软件过程模型：从一**特定角度**提出的软件过程的**简化、抽象化**模型

软件工程方法：

软件开发的**结构化**的方法，目的在于提高软件的生产性价比。包括系统模型、符号、规则、设计建议和过程指南

CASE：计算机辅助软件工程

为软件活动提供自动支持的软件系统。主要是**方法**上

软件基本属性：可维护性、可依赖性、可用性、有效性

软件工程面临的主要挑战：

遗留系统（维护和升级）、缩短的交付周期、需求多样化

职业道德：机密、工作能力、知识产权、计算机滥用

第三章 软件过程

1、通用软件过程模型

瀑布模型、进化式开发（增量式开发）、形式化系统开发、面向复用的开发

瀑布模型

需求定义→系统和软件设计→实现和单元测试→集成和系统测试→操作和维护

瀑布模型问题和适用性

项目被生硬地分割、对需求响应困难

适合需求能被较好理解的情况，大规模、需求不易变更、要求较高的系统

增量式开发/交付

系统不是一次交付，是先建立一个大致的框架，然后将要求的功能分为多次增量进行开发和交付

增量开发/交付要点：

需求有优先顺序，从**最高优先级**开始

一旦一个增量开始开发，就要**冻结**这部分的需求

增量开发/交付优点：

系统可以较早看到、早的增量可以为后续增量导出需求、降低总体失败风险、最高优先级别的增量得到最多的测试

极限编程

基于非常小的功能增量开发和交付的一种新方法，用户参与开发团队之中

进化式开发（探索式）

和客户一起工作，从最初进化到最后的需求。应该从**理解最清楚**的需求开始

进化式开发（抛弃式）

目的是理解需求，从**理解较差**的需求开始

进化式开发问题和适用性

体系结构通常较差、过程不可见、需要特殊工具

适合中小型交互系统，大型系统一部分，生命周期短的系统

形式化系统开发

用形式化数学转换将系统描述成一个可执行程序（IBM 净室系统），保证正确性

形式化系统问题和适用性

需要专门技能和训练、难以描述系统某些方面（如界面）

适合要求特别严格（安全性、保密性等）的系统

面向复用的开发

基于已有组件或者 COTS 系统（商业现货系统）的系统重用

组件分析→需求修正→基于重用的系统设计→开发和集成

过程反复

系统需求在项目期间总是进化的，将软件开发描述为周期性过程。

两个混合式模型：增量式开发、螺旋式开发

螺旋式开发

过程表示为螺旋线，每个回路表示过程中的一个阶段

2、软件描述

建立系统需要哪些服务（功能性需求）以及系统操作和开发的约束（非功能性需求）的过程

需求工程过程（软件描述主要过程）

可行性研究→需求分析和导出→需求描述→需求有效性验证

可行性报告　　系统模型　　用户和系统需求　　需求文档

将系统描述转换为一个可运行的系统的过程

3、软件设计实现

设计：对软件结构、系统数据、系统组件间的借口、算法的描述

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

实现：将结构转化为可执行程序

设计和实现往往是紧密相连，可能是交叉进行的

设计方法：设计通常是**图形化**的文档

4、软件有效性验证

检验和有效性验证是为了说明系统符合它的描述并且满足系统用户的需求

测试阶段

单元测试→模块测试→子系统测试→系统测试→接收测试

5、软件进化

软件具有**灵活、可以改变**的固有特性

第五章 软件需求

需求工程

建立用户**需求**以及使用和开发的**约束**的过程，生成的需求是系统服务和约束的描述

需求：高层抽象描述到底层数学形式化描述，可能是合同标书的基础，也可能是合同本身

需求类型

用户需求：关于系统的服务和约束的**自然语言加上方块图**标书，为**客户**撰写

系统需求：一个**结构化的文档**写出系统的服务，**合同**内容

软件描述：作为设计和实现的基础，为**开发人员**撰写

功能和非功能需求

功能需求：系统需要提供的**服务**的表述

功能性用户需求（高层表述） 功能性系统需求（详细描述）

非功能需求：系统提供的服务或功能上的**约束**

产品需求：描述交付的**产品**的行为，执行速度、可靠性等

机构需求：机构**政策**和程序的结果，如过程标准、实现要求

外部需求：系统**外部因素**和**开发过程**，如互操作性要求、法律要求等

领域需求：从系统应用领域中得出，反映了领域的特征

问题：易懂性不够（领域专业术语，软件工程师理解困难）

不够清晰明了（领域专家无法或者不愿表达得非常清晰明了）

需求的完整性和一致性

完整：所有服务都应该给出描述

一致：对系统服务的描述应该没有冲突和矛盾

理论上应该完整且一致，实际做不到

需求的相互作用

不同的非功能需求可能会冲突

需求度量

速度、规模、易用性（培训时间、帮助页数）、可靠性、鲁棒性（故障后重启时间、故障时数据损坏的可能性）、可移植性

1、用户需求

应该描述功能性需求和非功能性需求，用**自然语言、方块图、表**定义

自然语言问题

不够清晰、需求混乱（功能性需求与非功能性需求混合）、需求合并（不同需求一起表达）

二义性、随意性、模块化不够

2、系统需求：标准的格式、一致的语言

结构化自然语言描述：用格式限制的自然语言来表达需求

基于格式的描述：一系列定义好的格式

基于 PDL 的需求定义（伪代码）

需求用一种类似程序设计语言来描述，但有更大的表达能力

基于 PDL 的缺点和适用

适用：操作可分解为一系列动作且**顺序非常关键**的情况、硬件和软件**接口必须被定义**

缺点：表达能力不够、只有懂得编程人员才懂、更多作为设计描述而不是帮助理解模型

3、需求文档（SRS）

对系统开发者**要求**的**正式表述**，应该包括经过认可的**系统定义**和**需求描述**，陈述**应该做什么**

第六章 需求工程过程

需求工程过程：发现、分析和验证系统需求的过程

可行性研究→需求分析和导出→需求描述→需求有效性验证

1、可行性研究

决定一个系统是否值得做

可行性研究焦点：是否符合机构的**总体目标**

是否可能在**现有技术、预算和时间**限制下完成

能否与已经存在的其它系统**集成**

2、需求导出和分析（需求导出、需求发现）

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

相关的**技术人员和客户、最终用户**一起工作调查应用领域，以发现系统提供的**服务**、系统的运行**限制**。

系统需求导出和分析的困难

项目相关人员**不知道真正想要**什么

项目相关人员用**自己的语言表达**，软件工程师理解困难

不同人员提出的需求可能**冲突**

政治上的因素、分析过程中需求改变

基于方法的分析

基于使用一个**结构化方法**来理解系统，广泛用于需求分析

(1)面向视点的导出

视点：一种收集和组织需求的方式，需求来自一组有**共同认知的信息持有者**。

多视点分析是重要的

视点类型

数据源或数据接收器：生产或消费数据的视点，识别**产生或消费什么数据**和**采取什么处理过程**

表示框架：视点代表特定类型的系统模型。对比**多个**这样的视点以发现需求

服务的接受者：系统外的视点，从系统接受服务，主要适合**交互系统**

外部视点：**系统服务的接受者**，对终端用户比较自然

面向视点的需求定义方法（VORD）

视点识别→视点组织→视点文档→视点系统映射（每个步骤都可以反复回任何一个阶段）

VORD 模板

视点模板（名称，属性，事件，要求的服务，子视点）

服务模板（名称、原理（提供服务的理由）、描述、所服务的视点、非功能需求、提供者）

(2)场景（脚本）

描述系统如何实际使用

比起抽象描述，场景更容易让人关联起来

场景描述

场景开始时的系统状态（**前置条件**）

关于**常规事件流**的描述

可能出错的位置以及如何处理（**差错控制及错误处理**）

可能同时发生的活动的信息（**突/并发活动**）

场景完成后的系统状态

事件场景

可用来描述系统对特定事件**如何响应**。

VORD 中数据和控制分析的符号

方块图：数据**提供和交付**

控制信息

例外处理（分上下部分，上为例外名称，下为对应操作）

下一个预期的**事件**

椭圆：来自视点和交付给视点的**数据**

控制信息从每个方框的**顶部**出入

数据从每个方框的**右侧**出来

异常显示在方框的**底部**

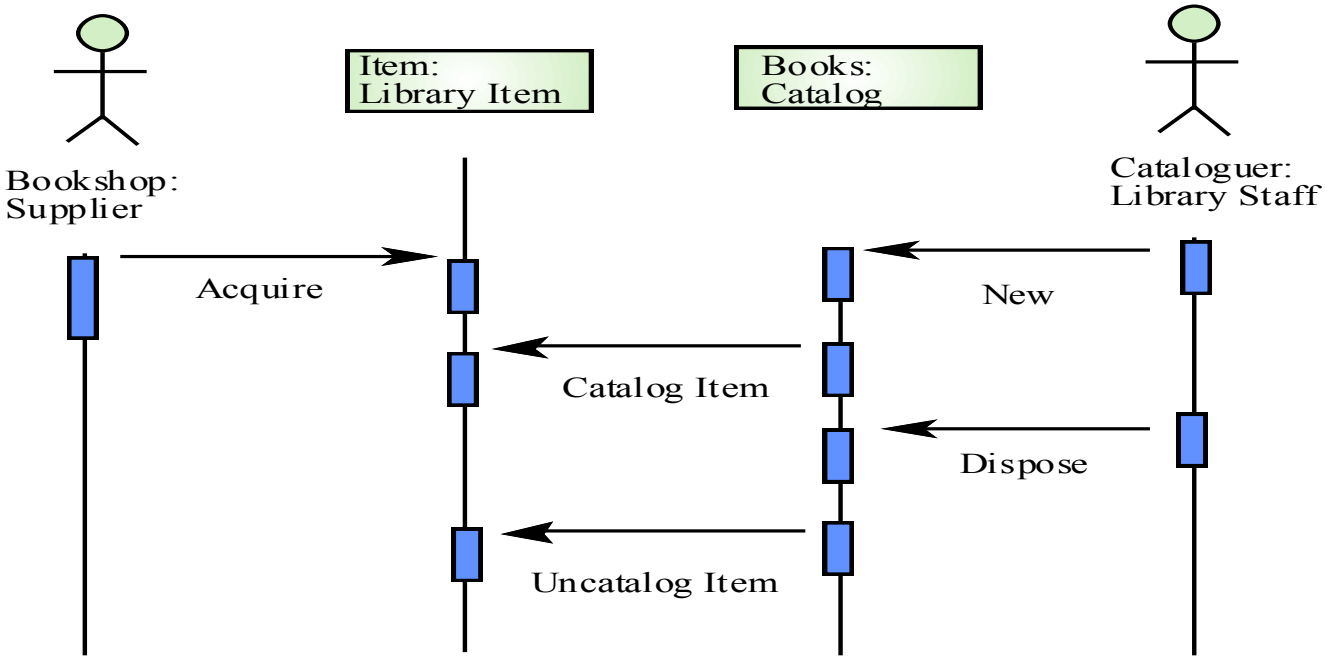
场景完成后**预期的下一个事件**的名称表示在一个**粗线框**中

(3)用例

用例是**基于场景**的需求导出技术，用 **UML** 标记。用例确定**交互中的角色**，描述**交互本身**

一组用例描述系统所有可能的**交互**，用 **UML 序列图或状态图**添加细节表述系统中时间处理序列

序列图



3、需求有效性验证

证明系统中定义的需求是客户真正想要的。

需求错误的代价是高昂的。

需求检查

有效性（是否客户需要）、精确性、一致性、完整性、现实性（可行性）、**可验证性**（需求是否可检验）

需求有效性验证技术

需求评审（可检验性、可理解性、可追溯性、适应性）、原型开发、测试案例、自动一致性分析

4、需求管理

在需求工程过程和系统开发过程中**管理需求变更**的过程，包括需求管理**规划**和需求**变更管理**

需求变更

开发过程中不同视点的需求的**优先级**是变化的

购买者与最终使用者的**需求冲突**

商务和技术**环境**变化

持久和易变的需求

持久的需求：稳定的需求来自客户**核心**的活动

易变的需求，开发或者系统**使用中**变化的需求

可追溯性

需求、需求的来源和**设计**之间的关系必须被记录下来

来源可追溯性（需求和需求提出者）、**需求**可追溯性（需求之间）、**设计**可追溯性（需求和设计）

第七章 系统模型

需求分析系统的抽象描述

系统建模

有助于分析者理解**系统功能**，模型可用来和客户**沟通**

结构化方法

定义了一系列模型和导出模型、规则、指南的过程

结构化方法的不足

看不出性能（非功能性需求）、太具体以至于难以理解、太多文档

不同角度描述

从外部来看，是对系统上下文或系统环境建模

从交互上看，是对交互行为建模

从行为来看，是对系统行为建模

从结构上看，是对系统的体系结构和系统处理的数据的结构建模

系统模型类型

数据处理模型，说明**数据**在不同阶段如何被处理

组成模型，说明系统中的**实体**是如何由其它实体**组成**

体系结构模型，说明构成整个系统的主要**子系统**

分类模型：实体间的**共同特性**

激励/响应模型：对实践的**响应**

上下文模型

上下文模型表示一个系统在环境中的**位置**以及和其它系统、过程之间的**关系**。

首先应该界定系统的**边界**

简单的上下文模型只有方框和线条，方框内是系统名称，通常需要配合其它模型一起

行为模型

描述系统的**所有行为**，系统响应来自外界的刺激（**事件/数据**）时所发生的或有可能发生的事情

数据流模型：描述**数据**如何在系统中流动，从**功能**角度对系统的**数据处理**建模

关注数据流向，**圆角矩形**表示对数据**处理**，箭头及上方文字表示**数据**

状态机模型（摩尔机）：表示系统对**事件**的响应，经常用来对**实时系统**建模

节点表示系统**状态**，节点间的**连线**表示**事件**

节点用**圆角矩形**表示，上方为**状态**，下方为**要做的事情**（和摩尔机相同）

状态表：每个状态操作内容中的动作的**简要描述**

结构模型

(1)语意数据模型（E-R）

描述系统**数据加工的逻辑结构**，实体—关系—属性，广泛应用于数据库设计

方框→实体，上半部分：名字，下半部分：属性　　连线→关系

数据字典

语意数据模型中用到的所有名称的列表

(2)对象模型

描述系统的对象**类**。能比较自然反映系统所处理的真实世界中的实体，但识别对象类很困难

对象类：一系列具有**共同属性和服务**的对象的**抽象描述**，用矩形框表示，顶部为名称，中间为属性，操作在下部

继承模型（泛化）

层级结构。**顶端**的类反映了所有类的**共同特征**。

上下层类之间通过箭头连接，箭头指向**上层**。

多重继承

允许对象类从几个超级类继承。会使系统更加复杂。

对象聚合

层级结构，表示对象的组合类，是如何由其它类**组成**的

上下层类之间用箭头连接，指向**下层**，箭头末端有一个**菱形**

交互模型（对象行为建模）

为对象之间的交互作用建模。

最典型：**UML 交互序列图**（时序）

第八章 软件原型

为了**需求有效性验证**的快速软件开发

系统原型

系统的**快速开发**

作用和优点

帮助**客户和开发人员**理解系统需求，给予最终用户关于系统功能的**直观印象**（可用于需求导出、需求有效性验证）

暴漏出**理解偏差**，更加接近**真正需要**的系统，减少整体开发投入

发现需求的**不完善和不一致**，提高设计的质量

早期能得到一个能工作的系统

可以作为得出**系统描述的基础**

支持**用户培训和系统测试**

软件原型开发

进化式原型开发（探索式进化开发）

开发一个初始的原型，然后通过几个阶段的**精炼修改**，得到**最终系统**。

目标是**交付一个工作系统**，从**理解最好的需求**开始

进化式原型开发过程

基于支持**快速系统开发**的技术（如 CASE 或者 4GL（四代语言）），无法做一致性检验

以一系列**增量**式交付

进化式原型优点

加快系统交付，用户参与度高

抛弃式原型开发（抛弃式进化开发）

用来**发现系统需求**，然后是**抛弃**掉的。从一个**初始描述**开发，交付后**用做实验**，然后**抛弃**

目标是**导出和验证系统需求**，降低需求风险。

快速原型开发技术（原型开发重要基础）

动态高级语言开发

有强大的**数据管理**功能的语言

一般不用于大型系统

数据库编程语言

基于**数据库管理系统**

一般包含*数据查询语言*、*界面生成器*、*报表生成器*

对**中小型商业系统**来说成本效益较好

语言+环境→第四代语言

组件和应用集成（复用）

原型可以从一组**可复用的组件**、加上一些**结合机制**，快速生成

结合机制应包含*控制机制*和*组件通信机制*

应用级：整个应用集成　　组件级：个别组件集成

可视化编程

脚本语言（如 VB）支持可视化编程，通过**标准的项目和相关组件**生成原型的**用户界面**

原型开发是用户界面的**基本方法**

第十章 体系结构设计

组织一个系统和设计系统的**整体结构**。具体来说，就是**识别**出组成系统的子系统，并建立子系统**控制和通信**的框架的过程

是一个系统设计的**早期过程**，是描述和设计过程之间的连接

体系结构的好处

项目相关人员之间的**沟通**

系统分析：对分析系统关键性需求有着深远影响，是能否满足非功能性需求成为可能

大规模**复用**：能在具有相似需求的系统之间互用

体系结构设计过程

系统结构化（分解成基本的子系统）→控制建模（*建立控制关系模型*）→模块分解（*分解子系统*）

子系统和模块

子系统**独立**构成系统，不依赖其他子系统提供的服务

模块通常是一个能提供服务给其它组件的**系统组件**，不能看作**独立**的系统

体系结构模型

容器模型

共享数据存放在一个中央数据库或者是容器中，可以被所有子系统所访问

共享**大量数据**时，容器模型最常用

容器模型优缺点

共享大量数据的有效方法

子系统不需关心数据如何集中管理

某些活动（备份、访问控制等）可以集中进行

清晰看出共享模型

子系统与容器**数据模型一致**

数据**进化困难、昂贵**

特殊化管理困难

数据分布比较困难

客户端-服务器模型（C/S 模型）

说明数据和处理是如何在**一个范围内**的组件间分布的**分布式系统**模型。服务器，客户机，**网络**

C/S 模型优缺点

数据分发**简单明了**

硬件**低廉**

容易**增加和升级**服务器

没有共享数据模型，不同的数据组织，导致数据交换可能效率不高

服务器存在**冗余**

没有**名字和服务的集中登记**，难以知道都有哪些服务器和服务

抽象机模型（分层体系结构）

用来建立子系统的**接口模型**

组织为**一系列层次（抽象机）**，每一层提供一组服务

支持不同层中的子系统的增量开发

控制模型

关注子系统间的**控制流**

(1)集中式控制

一个子系统全面负责控制，负责启动和终止其它子系统

调用-返回模型

自上而下，控制从*子程序层的顶端*开始，向下移动，适用**顺序系统**

管理者模型

*一个系统组件控制*其它系统过程的停止、开始和协调。适用**并发系统**。

(2)基于事件的控制（事件驱动系统）

广播模型

事件和消息管理器来对根据事件决定调用哪个程序

对于将*子系统集成到网络中不同的计算机*的系统非常有效

中断驱动模型：适用于**实时系统**，需要对实践做出**快速响应**

每种中断类型关联到一个*存储单元*（**中断矢量**），由*硬件开关*将中断转到它的处理程序

编程复杂，难以验证

模块分解

子系统分解成模块

对象模型

系统分解成一组松散的互相作用的**对象**，以及良好定义的**接口**

分解关系到对象类位置

数据流模型（管道模型/过滤器模型）

分解成**功能模块**（即数据处理模块），这些功能模块将输入转化为输出

不适合交互系统

领域相关的体系结构

特定于某些应用领域的体系结构模型

类模型：一般是从下往上的模型（聚合模型）

参考模型：更加抽象的**理想化模型**，从上往下的模型（泛化模型）

第十九章 检验和有效性验证（V&V）

确保一个系统满足**用户的需求**（不代表没有缺陷，因此使用类型决定了对软件的要求程度）

检验（注重**过程**）：是否在正确地建立一个产品。即软件应该**符合描述**（但描述可能会偏离需求的）

有效性验证（注重**结果**）：是否建立一个正确的产品。软件应该**满足用户真正的需要**

检验和有效性验证过程

是个**全生命周期**的过程——应用到软件过程的每个阶段

主要目的：发现**缺陷**、苹果系统在实际操作中是否**可用**

软件检查（静态检验）（昂贵的）

分析系统的**静态表述**以发现问题（缺陷、**不规范**），要有相关的文档

一次检查能发现**许多缺陷**，在测试中可能被其它缺陷屏蔽

容易发现**经常出现的错误**

无法检验**非功能特征**

只能检验是否和描述一致，无法检验是否和用户真正需求一致

程序检查的预处理

对被检代码精确的**描述**，熟悉机构**标准**，最新的**语法版本**，检查**清单**，不能用结果评价**员工**

检查团队

至少 **4 人**，被检代码**作者**，**检查者**，**讲解者**，**主席/仲裁**

检查清单

常见错误的清单，与编程语言有关

编程语言的**类型检查**（不同类型的直接转换）越弱，错误清单越长

自动静态分析

分析程序文本发现程序的不规则之处，这些不规则之处可能预示着代码缺陷。是人工的**补充**

软件测试（动态检验）

实际**运行和观察**软件的行为，必须有可运行的软件

可以发现错误的存在，而不是错误的不存在

是非功能性需求的**唯一验证手段**

应该结合静态检验，并由独立人员来测试

测试类型

缺陷测试：发现**系统缺陷**

统计性测试：**可靠性**估计

测试和调试

检验和有效性验证是证明软件系统中**存在错误**，调试是**对缺陷定位和修改**的过程，包含在实现中

软件开发的倒 V 模型

净室软件开发

缺陷**避免**，而不是缺陷一处

基于*增量开发*、*形式化描述*、*使用**正确性论证**进行**静态检验***、*用**统计性测试**决定程序**可靠性*

特征

形式化描述、增量式开发、结构化编程、静态检验（对数学方法论证）、统计性测试（疲劳测试）

第二十章 软件测试

测试程序以**发现**程序中的缺陷

测试过程

组件测试（模块测试）：通常由**组件开发者**负责，测试基于**开发者的经验**

集成测试：通常由**独立的测试团队**负责，测试基于**系统描述**

缺陷测试

目标：**发现**程序中的缺陷

成功的缺陷测试：引起**程序异常动作**的测试

测试的优先级：

只有穷尽的测试才能证明程序没有缺陷

相比组件，测试更应关注**系统的功能和性能**

相比新的功能，测试**旧的功能更加重要**

相比边界值，**典型值**更重要

测试数据和测试用例

测试数据：用来测试系统的输入**数据**

测试用例：测试系统的**输入**（系统实现前就应该写好）

(1)黑盒测试

测试用例基于**系统描述**，被测对象的输入与输出应该一一对应

测试**规划**可以在软件过程的早期开始

等价划分

输入数据和输出结果可以分成几个不同的集合，对**集合中每个成员**，程序的行为都是**等价的**

(2)结构化测试（白盒测试/路径测试/结构测试）

测试用例基于**程序结构**，目标是让所有的程序语句执行一遍（**语句覆盖**）

路径测试

目标：保证程序的**每条路径**都至少执行一次，不代表所有**路径组合**都能测试到

首先需要**程序流图**

程序流图：节点表示程序**无分支的序列**，边代表**控制流**，循环用一个返回到条件节点的箭头表示

计算环路复杂性的**基础**

环路复杂性

环路复杂性=边数-节点数+2

所有语句的**最少测试用例数**=环路复杂性=**程序中的条件数**

集成测试

集成测试应该是**黑盒测试**（白盒测试代价过大，只适合组件测试）

主要困难是**定位错误，增量式集成测试**可以减轻

集成测试方法

自顶向下的测试：从高层系统开始，需要**程序桩**（支撑上层程序的假函数）

自底向上的测试：一层层集成组件，直到产生整个系统，需要假程序（**驱动程序**）调用底部程序

大多是两种方法结合

测试方法比较

自顶向下：容易发现**体系结构**的错误，允许在开发早期有一个有限功能的**演示系统**

自底向上：较容易**执行**测试

测试的**观察**：均需要额外代码