第1章 软件工程概述

1.1.1

软件：是指计算机程序、运行程序所需的数据、与程序有关的文档资料的总称。

1.1.2

软件的特点：抽象性、不是传统意义上被制造、尚未完全摆脱手工艺方式。

1.1.4 软件分类：1.按功能：系统软件、支撑软件、应用软件。

2.按规模：微型、小型、中型、大型、甚大型、极大型。

3.按工作方式：批处理、分时处理、实时处理、交互式处理。

1.4.1

软件工程：研究用工程化方法构建和维护有效的、实用的和高质量的软件的学科。

软件工程的根本在于提高软件的质量与生产率，最终实现软件的工业化生产。

1.4.2 软件工程三要素：过程、方法、工具。

1.4.3

软件生命周期：从软件目标的提出、定义、开发、维护，直到最终被丢弃的整个过程。

计划时期：**问题定义、可行性分析；**

开发时期：**需求分析、概要设计、详细设计、编码、测试；**

运行时期：**运行与维护。**

1.6

软件开发模型：对软件过程的建模，做任务及执行顺序，目标是保证质量和适应需求变化（加强产品控制）。

1.6.1 瀑布模型

问题定义、可行性研究、需求分析、概要设计、详细设计、编码、测试

特点：1.顺序性：完成前一阶段工作后，才能进行下一阶段工作。前一阶段的输出就是后一阶段的输入。

依赖性：只有前一阶段输出正确，后一阶段的工作才可能获得正确结果。

2.推迟实现：把逻辑设计与物理实现清楚划分开，尽可能推迟程序的物理实现。

3.质量保证：各阶段必须完成规定的文档，阶段结束前对文档进行审评。

缺点：1.要求从一开始就清楚地给出所有需求，以后不能发生任何变化。

2.开发模型是线性的，程序的运行版本要等到项目开发晚期得到。

1.6.2 原型模型：1.收集用户需求2.建立原型3.用户评估原型4.逐步调整原型。

优点：1.原型提供了整理、分析和提炼软件需求的机制

2.用户不用等到项目开发周期的晚期才能得到可运行程序

缺点：太慢、太大、难以使用，没有考虑软件的总体质量和长期的可维护性。

1.6.3 增量模型：软件被作为一系列的增量构建来设计、实现、集成和测试。

增量模型融合了瀑布模型、原型模型。
增量模型实质是一系列瀑布模型。
增量模型每一个增量均为发布的产品。
优点：适应需求的变化。
缺点：软件过程的控制失去整体性，容易退化为边做边改模型。

1.6.4 旋转模型：瀑布模型+原型模型，是弯曲了的瀑布模型。

第二章 项目计划

2.1 问题定义：

目的：明确用户要计算机解决的问题是什么。
任务：确定问题的背景、待开发系统的目标和范围。
成果：《软件开发任务书》。
《软件开发任务书》内容：

1.项目名称
2.开发背景
3.项目目标：用户对新系统的主要目标
4.项目范围：功能、性能、输入/输出；相关相连系统；费用、时间
5.初步想法：在用户提出的需求的基础上，可考虑实现其他功能
6.提出可行性研究的计划

2.2 可行性研究：

目的：短时间内，花费最小的代价，确定项目是不是**可能实现**和**值得开发**。
任务：经济、技术、运行、法律可行性
工具：系统流程图
成果：《可行性论证报告》
《可行性论证报告》内容：

1.系统概述：项目背景、目标、范围
2.当前系统分析（系统流程图）
3.目标系统分析（系统流程图）
4.可行性分析：经济、技术、运行、法律可行性
5.综合意见：新系统是否可行
6.项目计划

2.3 系统流程图：

作用：描述系统的物理模型，表达系统各部件之间的信息流动。
基本思想：以黑盒子的形式描绘系统里面的每个部件。
项目计划：指定计划的过程就是一个逐渐了解掌握项目的过程，在范围、进度、资源、质量之间寻求一个平衡的过程。

第三章 需求分析

3.1 **需求分析：**

依据：可行性论证报告

目的：准确回答**系统必须做什么**这个问题，细化系统流程图中计算机部件。

任务：1.建立目标系统的逻辑模型，调查用户确切需求，细化，反复和用户交流，建立原型系统，数据流图、数据字典、加工处理。

2.形成软件需求规格说明书。

步骤：1.分析系统的要求：必须处理的信息和应该产生的信息

2.目标系统的逻辑模型：数据流图、数据字典、加工处理

3.构造原型系统

4.修正开发计划

5.复审、验证

6.得出《软件需求规格说明书》

3.2 数据流图（Data Flow Diagram）：描述软件系统逻辑模型的工具。

功能：抽象了信息在系统中的流动和加工处理的情况。

符号：1.数据的源点、终点：系统以外的人、事物

2.加工处理：系统功能的抽象

3.数据存储：数据库、数据文件

4.数据流：数据及其在系统中流动的方向

步骤：1.识别并从问题中提取数据流图中的元素：源与目的、数据流、加工处理、数据存储；

2.画顶层DFD-

基本逻辑模型：一个加工处理和若干个输入输出数据流；把整个系统当做一个大的加工处理，标明系统的输入、输出、数据源、目的；

3.分层细化DFD：中间层（可进一步分解），底层（基本加工框）；

4.完善补充

DFD没有提供条件逻辑，不需要表示条件处理或循环。

3.3 **数据字典**：数据元素+数据流+数据存储。

1.数据元素：不可再分的数据。

组成：1.名称　2.别名　3.取值类型

4.长度 5.描述 6.位置

2.数据流：数据在系统中流动的方向。

组成：1.名称 2.描述 3.来源 4.去处 5.组成 6流通量

3.数据存储：保存数据的方式和数据组织结构。

组成：1.名称 2.输入数据流 3.输出数据流

4.组成 5.描述 6组织方式

3.3.4 加工逻辑

组成：1.名称 2.编号 3.输入

4.输出 5.功能描述 6.加工处理

3.4 加工逻辑描述工具：

常用技术：1.结构化语言 2.判定表 3.判定树 4.IPO图

作用：描述如何把输入数据流→输出数据流

3.4.1 结构化语言：一种介于自然语言和形式化语言之间的语言。

3.4.2 **判定表**：一组**条件**取值的组合→加工逻辑的**动作**。

3.4.3 **判定树**：判定表的变形，比判定表更加直观。

3.4.4 IPO图（输入/处理/输出图）

3.5 **结构化分析方法/SA**：面向数据流分析方法的一种。

基于计算机的系统→信息变化。输入、变换、输出。
当数据流过基于计算机的系统时会被变换。

指导思想/功能：自顶向下、逐步分解。

画数据流图的原则：

1.父/子图平衡：输入/输出数据流必须相同；

2.掌握分解的速度：一次2~7个加工处理；

3.区分全局文件和外部项；
4.加工框编号：便于引用和追踪。

第四章 概要设计

4.1 **概要设计：**

起点：需求分析阶段得出的DFD图，数据字典

任务：从需求分析阶段的工作结果出发，进行软件结构设计与数据设计

DFD→HIPO/SC

软件结构设计：建立良好的模块结构，确定模块间的关系。

数据设计：将数据字典转换为实现系统需要的数据结构（如ER图）。

成果：《概要设计说明书》

4.2.1 模块：数据说明、可执行语句等程序对象的集合。

内部特性：完成其功能的程序代码和仅供该模块内部使用的数据。

外部特性：模块名和参数表，以及对程序及整个系统造成的影响。（黑盒）

模块化：把系统划分成若干个模块，每个模块完成一个子功能，模块既独立，又互相有一定联系，把他们组成一个有机的整体，完成指定的功能。

模块化是软件结构设计的一个基本准则。

4.2.2 软件结构风格的总体要求-

独立性高，一个模块的功能不是同其他模块紧密联系在一起。

独立性强的模块不容易受其他模块改变的影响。

独立性强的模块应是**高内聚、低耦合**的模块

4.2.3 **耦合：模块之间**的互相依赖的紧密程度的度量。

耦合越松散，模块之间的联系越小。

1.非直接耦合：两模块没有直接关系。 低

2.数据耦合：两模块之间交换的是简单数据。

3.特征耦合：模块间交换的是数据结构。

4.控制结构（中等耦合）：传递的信息中有控制信息。

5.外部耦合：一组模块都访问同一全局变量。

6.公共耦合：一组模块都访问同一全局数据结构。

7.内容耦合（病态耦合关系）：a一个模块直接调用另一个模块中的数据；

b一个模块直接转移到另一个模块中；

c一个模块有多个入口；

d两个模块有一部分代码重叠。 高

内聚：模块内部各个元素彼此结合的紧密程度的量度。

内聚越高，模块内部各成分之间的关联也就越强。

1.偶然内聚：模块内各组成成分在功能上是不相关。 低

2.逻辑内聚：由若干个逻辑功能相同或相似的成份组成。

目的：省去程序中的重复部分。

3.时间内聚：相同的时间执行的成分组合在一个模块内。

4.过程内聚（中等内聚）：一个模块内部包含一组任务并必须以特定次序执行。

5.通信内聚：模块内使用了同一组输入数据，或产生同一组的输出结果。

6.顺序内聚：模块中各成份密切相关，一个组成部分的输出作为另一个组成部分的输入。但功能不单一

7.功能内聚：模块内各成分结合在一起完成单元一的功能。 高

模块化设计准则：一个模块一个功能。

内聚比耦合更重要。

4.2.4 好的软件设计特性

1. 改进软件结构，提高模块独立性：

分解模块减少控制信息的传递。

合并模块减少对全局数据的引用。

宁要塔形，不要饼形，提倡瓮形。

2. 模块规模适中：

最好写在一页纸内/60行。

过少：模块增多，关系复杂，接口代价高。

过多：分解不充分，阅读难，复杂。

3. 深度、宽度、扇出和扇入：

深度：软件结构的层数。

宽度：最多模块层的模块数。

扇出：某模块直接控制的模块数。

扇入：某模块被多少模块调用。

结论：好的软件结构，上层扇出高，中层删除少，低层扇入高（瓮形）。

4. 模块的作用域应该在控制域之内：

控制域：模块本身及下属模块。

作用域：受该模块内判定影响的所有模块。

作用域在控制域内且越近越好。

5. 降低模块接口的复杂性

6. 模块功能可以预测

4.3 图形工具：层次图适合作为文档，结构图适合复审软件结构。

4.3.1

层次图（H图）：描绘软件的层次结构。矩形框表示模块，框间连线表示调用关系。

HIPO：一个带编号的H图和一组模块的IPO图组成。

4.3.2 结构图（SC）：

1.传入模块：下层传数据到上层。

2.传出模块：上层传数据到下层。

3.变换模块：从上层取数据处理后返回上层。

4.源模块：不调用其他模块的传入模块。

5.漏模块：不调用其他模块的传出模块。

6.协调模块：对下属模块进行控制和管理的模块。

附加：简单调用、选择调用、循环调用。

4.4 结构化设计方法（SD）

面向数据流的系统分析、设计方法：SA和SD

结构化设计SD任务：将系统逻辑模型DFD转换为软件结构图HIPO/SC图。

方法：变换型DFD→变换映射，事务型DFD→事务映射。

指导思想：自顶向下、逐步求精。

4.4.1 变换型：数据随时间的推移而流动。

步骤：1.识别输入边界、输出边界和变换中心三部分。

逻辑输入：离物理输入端/起点最远的数据流。

逻辑输出：离物理输出端/终点最远的数据流。

2.进行第一级分解。

设计顶层主控模块和第一层软件结构。

1）输入模块

2）输出模块

3）变换模块：将逻辑输入转换为逻辑输出。

3.完成第二级和下层的映射。

任务：将DFD中的每一个处理映射到程序结构中的模块。

方法：从变换中心的边界开始，沿输入路径和输出路径向外映射。

4.优化及软件结构。

4.4.2 事务型：以事务中心为核心，根据事务的要求去执行不同的动作序列。

步骤：1.识别事务输入、事务中心和若干动作路径三部分。

2.进行第一级分解。

映射顶层和第一层。

输入模块、调度模块。

3.进行第二级分解。

设计中下层模块，对通路再识别、划分、映射直到全部映射完毕。

4.优化软件设计。

一个动作路径可映射为一个事务模块，调度下分为事务层、操作层、细节层。

4.5

降低模块接口的复杂程度：接口简单（低耦合高内聚）接口复杂（高耦合第、低内聚）。

第五章 详细设计

任务：1.确定每个模块的算法

2.确定每个模块的数据组织
3.为每个模块设计一组测试用例
4.编写《详细设计说明书》

5.1 结构程序设计：

自顶向下、逐步求精。

从高级语言中取消GOTO语句。

利用三种基本的控制结构：顺序、选择、循环。
单入口和单出口规则。
作用：开创了一种新的程序设计思想、方法和风格，显著提高软件的生产率、降低软件的维护成本。

优点：1.提高软件开发的成功率和生产率

2.确保模块逻辑结构清晰
3.易编码易测试
缺点：存储容量和运行时间有所增加（相对汇编语言）

5.2 详细设计工具-描述算法：决定各个模块的算法并精确地表达算法。

5.2.1程序流程图/程序框图：

基本符号：开始、结束、处理、控制流。
优点：直观描绘程序的控制流程、灵活方便。
缺点：控制转移无约束、结构混乱、过早考虑控制流程不考虑全局结构。
应用范围：规模小的程序。

5.2.2 N-S图/盒图：

基本结构：顺序型、选择型、while重复型、until重复型、多分支选择型。

优点：1自顶向下逐步求精。

2.结构清晰（只能结构化的）。
3.容易确定全局和局部变量的作用域。
4.容易表示嵌套关系、块内的层次结构。
缺点：受限制，不灵活。

5.2.3 PAD图/问题分析图：利用二维树形结构图表示程序。

控制结构：顺序型、选择型、while重复型、until重复型、多分支选择型。

优点：1.树形结构，克服了流程图不能表达程序结构的缺点，克服了N-S图的限制。

2.支持自顶向下逐步求精的方法。

3.易转换为高级语言源程序。

应用范围：大型软件的开发。

第六章 编码

目标：产生正确可靠、简明清晰、具有较高效率的源程序。

结构化的程序设计：1.严格控制GOTO语句。

2.使用基本控制结构，控制结构只有一个入口和一个出口。

6.1 程序设计语言的特性、程序设计风格会深刻影响软件的质量和可维护性。

源代码越清楚和简明，越便于验证代码和文档的一致性，越容易测试和维护。

清晰和效率常常会产生矛盾。

对大多数模块，编码时把简洁清晰放在第一位。

6.2程序设计语言：指编写计算机程序所用的语言，是人与计算机交流的工具。

1GL：机器语言

2GL：汇编语言

3GL：高级编程语言。C、Java等

4GL：面向问题的语言，以数据库管理系统提供的功能为核心，构造开发高层软件系统的开发环境。Power Builder、Dephi等

5Gl：自然语言/知识库语言/人工智能语言。并没有真正的5GL

6.3.1 程序文档化：

1.标识符的命名：即符号名，包括模块名、变量名、常量名、子程序名等。在一个程序中，一个标识符只应用于一种用途。

匈牙利命名法：变量名=属性+类型+对象描述。

2.安排注释：1）序言性注释：置于每个程序模块的开头部分

2）功能性注释：嵌在源程序中，描述其后的语句或程序段在做什么工作。

3.程序的视觉组织：1）恰当地利用空格、空行

2）移行/向右缩进

4.自文档代码：代码即文档，代码精巧，注释适当。

6.3.2 数据说明：

1.次序规范化　2.安排有序化　3.注释说明复杂数据结构

6.3.3 语句结构：

1.一行一句
2.清晰性

3.直接说明程序员用意

4.清晰第一，效率第二（通过高效算法实现）

5.避免过复杂的条件测试

6.减少使用否定条件的条件语句

7.减少IF语句过多的嵌套（容易产生二义性）

6.3.4 输入/输出：

1.输入：保证每个数据的有效性
2.输入：检查重要组合的合理性
3.输入：步骤和操作尽可能简单
4.输入：简单的输入格式，允许自由格式输入
5.允许缺省值
6.输入：使用输入结束标志
7.提示可使用的选择项、取值范围，屏幕显示状态信息
8.设计输出报表格式

6.3.5 效率

6.3.6编码工具：1）集成开发环境（IDE）：Visual Studio、Eclipse

2）代码管理工具（Source Control）：CVS、SVN、GIT

SVN服务器端：VisualSVN Server、客户端：TortoiseSVN

第七章 测试

7.1 软件测试：为了发现错误而执行程序的过程。

破坏性工作，用各种方法证明软件有错。
目的：尽可能多发现并排除软件中潜在的错误，最终把高质量软件交给用户。

程序测试=狭义的软件测试。

广义的软件测试应贯穿于整个软件开发期间。

7.2 测试步骤：

1.单元测试：对模块测试，编码错误和功能情况

2.集成测试：组装模块后，对模块间接口测试

3.确认测试：检查是否满足需求（功能/性能）

4.系统测试：把软件放进实际运行环境中，与其他系统一起测试

7.3 测试方法：

静态测试（代码复审）：检查代码的静态结构
动态测试（机器测试）：在测试用例上执行被测程序的过程

1）黑盒测试/功能测试：根据程序的功能来设计测试用例

2）白盒测试/结构测试：根据被测程序的内部结构设计测试用例

7.4 测试用例的设计

测试用例=输入数据+期待结果。

7.4.1 **白盒**测试用例设计：以程序**内部逻辑结构**为基础设计测试用例

1）逻辑覆盖法：使用程序流程图
1.语句覆盖：将程序中每个语句至少执行一次
2.判定覆盖：每个判定的每个分支路径至少要执行一次
3.条件覆盖：每个条件的真假两种情况至少执行一次
条件覆盖不一定符合判定覆盖。（判定只在乎结果）

4.判定/条件覆盖：每个判定的每个分支路径至少要执行一次，每个条件的真假两种情况至少执行一次

5.条件组合覆盖：每个判定的所有条件的各种可能组合至少执行一次

2）路经测试法：使用程序图。设计足够的测试用例，覆盖程序中的所有可能的路径

1.点覆盖=语句覆盖：每个结点至少执行一次

2.边覆盖：每条边至少执行一次

3.路径覆盖：每条路径至少执行一次

好的白盒测试：条件组合覆盖、路径覆盖。

7.4.2 **黑盒**测试用例设计：只依据程序的**功能**来设计测试用例

1）等价分类法：将所有可能的输入数据划分成若干个等价类，然后从每一类中选取少数有代表性的数据作为测试用例

包括 有效等价类+无效等价类

步骤：

1.划分等价类：有效+无效

2.设计测试用例：

有效等价类：尽量选取公用测试用例，以减少测试次数

无效等价类：每类一例，以防漏掉错误

1）为每个等价类规定一个唯一编号

2）尽可能多覆盖尚未被覆盖的等效有效类，直到全覆盖

3）仅覆盖一个尚未被覆盖的无效等价类，直到全覆盖

2）边界值分析法：对等价分类法的补充，针对各种边界情况设计

步骤：

1.确定边界情况

2.选取正好等于、刚刚大于、刚刚小于边界的值作为测试数据

3）错误推测法：靠经验和直觉推测

7.5单元测试：针对软件的最小单位模块进行的正确性测试。

依据：详细设计说明书，源程序清单

内容：

1）模块接口测试：参数是否正确

2）局部数据结构测试

3）路径测试

4）错误处理测试

5）边界测试

辅助模块：驱动模块（调用）、桩模块（被调用）

7.6 集成测试

目标：发现并排除在模块组装中可能出现的问题。

内容：1.全局数据结构是否有问题

2.子功能组合后是否达到预期要求的功能

3.单个模块的误差累积后，是否会放大

渐增式组装：单元测试的同时就可以进行集成测试，发现并排除。

1）自顶向下：深度优先（按子系统方向）广度优先（沿层自左向右）
需要桩模块。

2）自底向上：需要驱动模块

3）混合式：对上层模块，自顶向下，对关键模块，由底向上

7.7.1 确认测试/有效性测试

依据：软件需求说明书

明确软件的功能和性能。

确认测试的基础。

任务：验证软件功能、性能及特性是否符合用户的要求

是软件开发单位最后一项开发活动。

1.有效性测试--黑盒测试

2.软件配置复查--用户文档、开发文档、系统配置 齐全且可正确使用

交付文档：确认测试分析报告、最终的用户手册、项目开发总结报告

7.7.2 系统（验收）测试：以用户为主的测试，投入生产，由用户测试

7.7.3 α 测试：用户和开发人员在模拟操作环境下进行的测试

β 测试：多个用户在实际操作环境下进行的测试，应用

7.8 调试Debug：通过现象，找出原因的一个思维分析的过程

7.9.1 测试工具分类：

1.白盒测试工具：静态测试工具：对代码进行语法扫描

动态测试工具：采用插桩的方式，向代码生成的文件中插监测代码

2.黑盒测试工具

3.性能测试工具

4.测试管理工具

第八章 软件维护

8.1 软件交付使用后，为了改正错误或满足新的需求而修改软件的过程。

分类：1）改正性维护：识别和改正软件错误

2）适应性维护：适应新环境

3）完善性维护：扩充软件功能，增强软件性能//占维护工作的50%

4）预防性维护：提高软件的可维护性和可靠性，为改进打好基础

8.2 影响维护代价的非技术因素：

1）开发人员的稳定性
2）软件的生命周期
3）商业操作模式变化对软件的影响

影响维护代价的技术因素：

1）软件对运行环境的依赖性
2）编程语言
3）编程风格
4）测试与调试工作
5）文档的质量

8.3 维护申请表：维护人员给用户的空白的表

第九章 面对对象方法

9.1 面向对象方法（OOA、OOD、OOP）

基于对象的自底向上地进行功能综合。
从内部结构上模拟客观世界。
以对象为中心，把世界看成对象的集合、对象之间通过消息通信。

UML：基于面对对象技术的标准建模语言。

三个要点：

1）统一：

UML的核心，提升开发团队的沟通效率

通过双向工程实现开发人员和开发环境的统一

2）建模：

体现UML的使用价值

3）语言：
建模活动最有效的表述形式

9.2 面对对象=对象+类+继承+通信

三要素：封装继承多态
类是一组具有相同结构、操作，遵守相同规则的对象的抽象
对象是类的实例
对象三要素：属性（对象的特征）事件（对象的动作）方法（操作）

9.3.1 面对对象的分析OOA：运用面向对象的方法进行需求分析

过程：

1）问题分析，建立用例模型
2）发现和定义类与对象
3）识别对象的外部联系
4）建立系统的静态结构模型
5）建立系统的动态结构模型

UML模型的基本组成：

1.要素（点）：模型的核心内容

Use Case、Actor

类Class、接口Interface

2.关系（线）：将要素联系在一起
关联关系：聚合关系、组合关系

依赖关系
泛华关系
实现关系

3.图（面）：将一组要素和关系展现出来

两种静态图：

Use Case图（Use Case Diagram）：展现Use Case Actor及其关系

类图（Class Diagram）：展现类，接口及其关系

4种动态图：

序列图（Sequence Diagram）：按时序展示对象间的消息传递

协作图（Collaboration

Diagram）：收发消息的对象间的协作关系

状态图（Statechart

Diagram）：对象经历的状态及对事件的响应

活动图（Activity

Diagram）：系统从一个活动转移到另个的路径

9.3.2 **Use Case图：**

作用：描述拟建系统和外部环境的关系。

组成：

1）Use Case（用例）：一个完整的功能。

表示从外部用户角度观察的系统功能。

总是由Actor初始化。

为Actor提供值。

2）Actor（执行者）：系统外部和系统进行交互的人、其他硬件系统。

3）通信关联：表示Use Case和Actor之间的双向交互。

作用：

对客户：详细描述系统的功能、使用方法

对开发人员：帮助理解系统的需求

对测试人员：验证最终系统是否与UC图的功能一致

对文档人员：为编写用户手册提供参考

过程：

1）找出系统外的Actor：人、外部系统设备

2）使用UC图描述系统向Actor提供的功能和使用方法

3）绘制UC图并编写描述，功能含义和实现步骤以文本描述

9.3.3 **类图**：建立对象与类模型

作用：描述类、接口之间的关系

组成：

类：描述一组具有相同属性、操作、语义的对象

类名+属性+操作

接口：说明应提供的服务，是一组操作的集合，描述多态性

接口名+操作

1.关联关系：一个类的对象作为另一个类的对象的成员变量

特性：多重性、访问方向、角色

关联关系的强化形式：

聚合关系：表示两类对象间有整体与部分的关系

1）整体的对象消失不会导致的对象的消失

2）部分的对象可以被多个整体的对象共享

组合关系：表示两类对象间有整体与部分的关系

1）整体与部分间皮之不存毛将焉附

2）部分不能被整体共享

2.依赖关系：两个对象之间由于通信形成的关系

3.泛化关系：类A（特殊）到类B（一般）的泛化关系表示“类A是类B的一种”。

即子类对于父类的关系

有助于代码共享和复用

4.实现关系：描述类实现接口

9.3.4 建立系统动态模型

实体类：拟建系统要记录和维护的信息

人员、组织、物品、事件、表格

边界类：拟建系统和外部元素之间交互的边界

用户界面、与外部系统的接口、与其他设备的接口

控制类：拟建系统在运行中的执行逻辑

交互图：对共同工作的对象群体的行为建模

1.**序列图**：强调消息的**时间顺序**的交互图（强调消息的时间顺序）

组成：

对象：对象名=类名

对象生存线
控制焦点：
消息：序号、名称、参数
引起调用、返回、发送、创建、撤销

2.**协作图**：强调发送和接受消息的对象之间的**组织结构**

组成：对象、消息、链接、自链接

状态机：对单个对象的生命期建模

状态图：描述对象可能处于的状态、状态之间的转换（事件、动作）

矩形表示对象的状态。
开始状态必须有，实心黑点。
终止状态可有可无，实心黑点外套圈。
带箭头的直线表示状态迁移的方向，迁移条件写在直线旁边。
组成：状态、初始状态、结束状态、转换

正向工程：根据状态图（模型）生成代码。

逆向工程：根据代码生成模型。

活动图：描述Use Case的控制流结构，关注对象间发生的活动

组成：

1.活动

2.决策点

3.同步棒：对业务过程的并发流建模

同步棒分叉、同步棒会合

4.泳道对活动分组：描述了完成各个活动的一个或多个类

5.对象流：表示活动涉及的对象，用依赖关系连接创建、修改、撤销该对象的活动

活动图对表示并发行为很有用，可进行业务过程建模（Use Case分析）