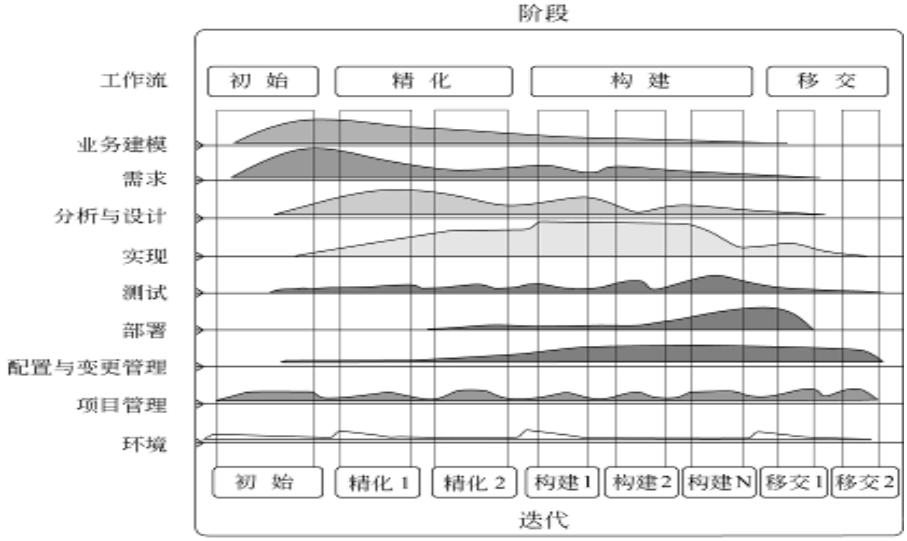
1.4.6

2、RUP 软件开发生命周期



（1） 核心工作流：

其中前六个是核心过程工作流程，后三个为核心支持工作流程

业务建模：深入了解使用目标系统的机构及其商业运作，评估目标系统对使用它的机构的影响

需求：了解客户需求，并使开发人员和客户达成共识

分析与设计：根据需求转化成产品原型

实现：开发，单元测试，把各个单元合起来

测试：检查各个子系统的交互和集成

部署：生成目标系统的可执行版本，交给用户

配置与变更管理：跟踪并维护在软件开发过程中产生的所有制品完整性和一致性

项目管理：制定计划、人员配置、执行和监督方面的实用准则，风险管理

环境：向软件开发人员提供软件开发环境，包括过程管理和工具支持

（2） 工作阶段

RUP 把软件生命周期划分为 4 个阶段

初始阶段：问题定义，确定目标，可行性分析，降低关键风险

细化阶段：制定项目计划，配置各类资源，建立系统架构（各种视图）

构造阶段：开发，确保产品可移交给用户

移交阶段：产品发布、安装、用户培训

每个阶段的目标通过一次或多次的迭代来完成

每个阶段结束之前都有一个里程碑评估工作成果

（3） RUP 迭代式开发

整个项目开发过程由多个迭代过程组成，每次迭代只考虑部分需求

每次循环都是完整的生命周期，每次循环结束都为用户提交一个可运行的版本

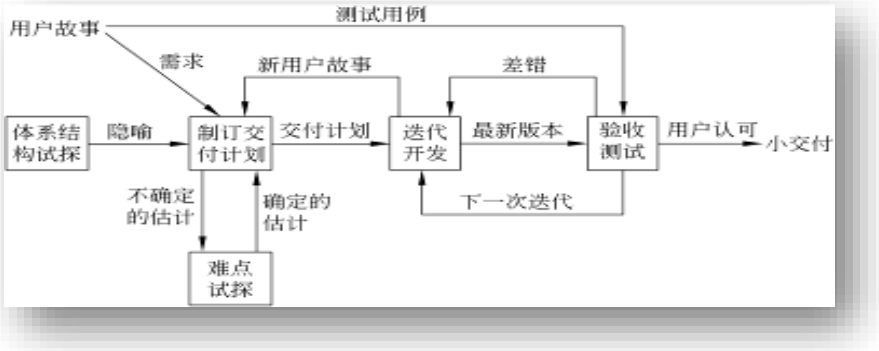
1.4.7 敏捷过程与极限编程

一、敏捷过程

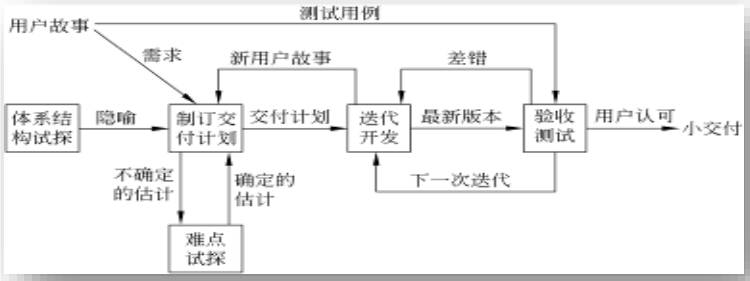
1、敏捷过程

定义：是一种以**人为核心**、迭代、循序渐进的开发方法。在敏捷开发中，软件项目的构建被切分成多个子项目，各个子项目的成果都经过测试，具备集成和可运行的特征。换言之，就是把一个大项目分为多个相互联系，但也可独立运行的小项目，并分别完成，在此过程中软

件一直处于可使用状态。
2、敏捷过程的价值声明
（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　要有灵活性和可塑性
3、敏捷开发的特点
“适应性”而非“预设性”，欢迎需求变化
面向人而非面向过程
敏捷开发特别重视项目团队中的信息交流
二、极限编程
客户作为开发团队的成员
使用用户素材（用户素材就是正在进行的关于需求的谈话内容的助记符）
短交付周期 每两周完成一次的迭代过程
验收测试
结对编程 两人在同一台电脑上开发项目，一人写一人审查
测试驱动开发 编码前要先设计好测试方案
集体所有 每个人都有权更改代码，每个人都要为代码负责
持续集成 一天内多次集成，随着需求的变化，应不断回归测试
可持续的开发速度 匀速开发
开放的工作空间
及时调整计划 计划应该是灵活的
简单的设计 本次迭代不考虑下一轮迭代
重构 重新调整和优化系统内部结构
使用隐喻 要有全局视图：描述系统如何运作，以及怎样把新功能加入西戎
（2）整体开发过程



（4）　迭代开发过程



敏捷过程能很好的适应商业环境下对小型项目提出的有限资源和有限时间的约束

1.5.7 微软过程

1、微软过程准则

项目计划兼顾不确定性因素

风险管理减少不确定因素影响

经常生成并快速测试过渡版本

采用快速循环、递进的开发过程

创造性工作衡量产品特性和成本

小型项目组并行完成开发

项目进度表高稳定和权威

项目早期软件配置项基线化，后期冻结产品

原型验证，对项目进行早期论证

零缺陷作为追求目标

里程碑评审会目的是改进工作

2、微软软件生命周期

分为五个阶段

规划阶段：获得用户情况和客户需求等。需完成：产品目标；获取竞争对手信息；完成调研分析；确定产品主要特征；定义问题及功能。

设计阶段：确定 70%以上的需求。(文档：特性规格说明书；系统设计；子系统及规格说明；制定产品开发计划)

开发阶段 编写程序和文档

稳定阶段 对产品进行测试和调试

发布阶段 将项目移交和支持人员

适用于商业环境的具有有限资源和有限开发时间约束的项目的软件过程模式

综合了 Rational 许多优点，但在方法、工具和产品方面论述不够全面

1.5 软件开发策略

（1）软件复用：类似于 APICloud 的众多 API

确保质量，提高效率

（2）分而治之：降低复杂度，提高效率

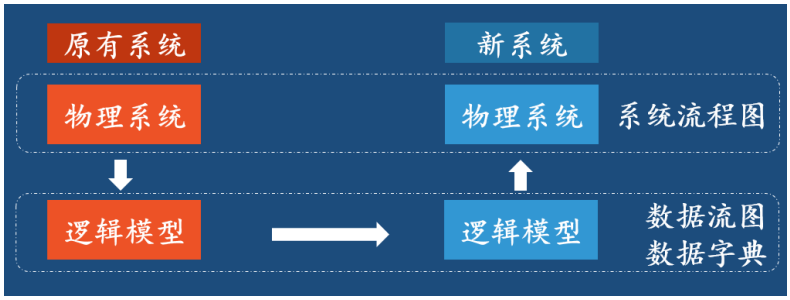
（3）折中优化：指优化软件的各个质量因素，实现整体质量的最优

（提高运行速度；提高对内存资源的利用率；是用户界面更加友好；使三维图形的真实感更强）

1.6 CASE 工具与环境

CASE:计算机辅助软件工程

在软件工程活动中，软件工程师和管理人员按照软件工程的方法和原则，借助于计算机及其软件工具的帮助，开发、维护、管理软件产品的过程称为计算机辅助软件工程。
第二章：可行性研究
2.1 可行性研究的任务
技术可行性：风险分析；资源分析；技术分析
经济可行性：费用估计；效益估计（新软件与旧软件相比有没有提高）
操作可行性：对目标操作系统所规定的运行方式是否能达到
可行性分析的时间长短取决于工程的规模，成本只是预期工程总成本的 5%~10%
2.2 可行性研究过程
1、复查系统规模和目标：我们认为要的是不是用户想要的
2、研究目前正在使用的系统
解决老系统问题，新系统效益要大于老系统效益
3、导出新系统的高层逻辑模型



4、进一步定义问题（此时应考虑成本）
复查问题定义、工程规模和目标
讨论基础：数据流图、数据字典
5、导出和评价供选择的解法
从建议的系统逻辑模型出发，导出若干较高层次的物理解法供比较和选择（数据流图上划分不同的边界、组合的方法）
根据技术可行性排除不可行系统
去掉用户不能接受的方案
考虑经济方面可行性
为每个方面制定实现进度表，估计生命周期每个阶段作量
6、推荐行动方针
向用户推荐方案
7、草拟开发计划
包含：工程进度表，各类开发人员、各种资源需要的时间、估计每个生命周期的成本，给出需求分析的详细进度表和成本估计
8、书写文档提交审查
2.3 系统流程图
基本思想：用图形符号以黑盒子的形式描绘组成系统的每个部件（包括程序文档数据库和人工过程等
表达了数据在系统各个部件之间的流动情况
2.4 数据流图
简称 DFD（Data Flow Diagram）
描绘信息流和数据从输入移动到输出的过程中所经受的变换，反映了数据在软件中流动和被

处理的逻辑过程
P42
第一步：提取数据流图的 4 种成分（源点或终点、处理、据存储和数据流）
第二步：确定“基本系统模型”
第三步：确定“功能级数据流图”
第四步：细化数据流图
画图原则：
1 确定系统的源点和终点
2 确定系统的输入和输出数据流的关系
3 保持分解前后输入/输出数据流必须相同(父子平衡)
4 尽量简化加工之间的联系
5 用“自顶向下”方法，逐层画出数据流图，每张数据流图中加工(处理)的个数不能超过 9 个(7 加减 2) 采用层次结构的数据流图
6 注意分解速度(一般每分解一层增加 2-7 个加工)
7 在画数据流图时应避免线条交叉，必要时可使用重复的外部项(源点或终点)或数据存储符号
8 适当的命名，加工处理要编号

数据流图的用途：
1 作为交流信息的工具
2 作为分析和设计的工具
3 数据流图可以辅助物理系统的设计(图)
4 数据流图对详细设计也有帮助
2.5 数据字典
2.5.1 数据字典的内容
数据字典：是关于数据的信息集合，是对数据流图种包含的所有元素定义的集合
数据流图和数据字典共同构成系统的逻辑模型，两者缺一不可
由对下列 4 类元素的定义组成：
(1) 数据流；　　(2) 数据流分量(即数据元素)；
(3) 数据存储；　(4) 处理
数据字典中记录数据元素的下列信息：
一般信息（名字、别名、描述）
定义（数据类型、长度、结构）
使用特点（值的范围、使用频率、使用方式）
控制信息（来源、用户、程序的改变和使用权）
分组信息（父结构、从属结构、物理位置——记录、文件和数据等）
2.5.2 定义数据的方法
数据的基本类型：顺序、选择、重复，可选
数据字典定义中出现的符号含义：

符号	含义	说明举例
=	被定义为	订书单＝教材ISBN十价格十数量
+	与	X=a+b 表示X由a和b组成
[…|…]	或	X=[a|b] 表示X由a或b组成
{…}	重复	X={a} 表示X由0个或多个a组成
M{…}n	重复	X=2{a}5 表示X中最少出现2次a，最多出现5次a。 5，2为重复次数的上、下限
(…)	可选	X=(a) 表示a可在X中出现，也可不出现
“…”	基本数据元素	X=“a”，表示X是取值为字符a的数据元素
..	连接符	X=1..9，表示X可取1到9中任意一个值

2.6 成本/效益分析

2.6.1 成本分析

软件开发的成本主要表现为人力消耗

1、 代码行技术

T——软件成本

N——软件源代码行数

A——每行源代码的平均成本

T=A*N 其中 A 取决于软件的复杂程度和工资水平

2、 任务分解技术

把软件开发工程分解为若干个相对独立的任务，然后再分别估计成本

表 2.2 典型环境下各个开发阶段需要使用的人力的百分比

任 务	人力(%)
可行性研究	5
需求分析	10
设计	25
编码和单元测试	20
综合测试	40
总计	100

设 T-软件总成本

B-每个单独开发任务的成本

α-系数

$T=\sum \alpha B$

C-完成每个单独任务所需的人力(月)

D-每人每月的平均工资

$B=C\cdot D$

$T=\sum \alpha\cdot C\cdot D$

3、 自动估计成本技术

更加客观，但是需要有大量历史数据且有良好的数据库系统支持

2.6.2 成本/效益分析的方法

估计开发成本、运行费用和新系统将带来的效益

运行费用取决于系统的操作费用(操作员人数，工作时间，消耗的物资等等)和维护费用。

系统的经济效益等于因使用新系统而增加的收入加上使用新系统可以节省的运行费用。

效益分析:

1、 货币的时间价值 即利率

i: 年利率

P: 现在存入的钱

n: 年

n 年后可以获得的钱数为 F=P(1+i)^n
这些钱的现在价值为 P=F/(1+i)^n
2、 纯收入
是指在整个生命周期之内系统的累计经济效益（折合成现在值）与投资之差
（相当于比较投资开发一个软件系统和把钱存在银行中（或贷款给其他企业）这两种方案的优劣）
3、投资回收期
是指累计的经济效益等于最初投资所需要的时间
4、投资回收率
相当于年利率
第三章 需求分析
3.1 需求分析的任务
3.1.1 确定对系统的综合要求
1、功能需求：系统必须要提供的服务
2、性能需求：系统必须满足的定时约束和容量约束，包括速度（响应时间），信息量速率、主存容量、磁盘容量、安全性等方面的需求
3、可靠性和可用性需求
可靠性需求定量的指定系统的可用性
量化了用户可以使用系统的程度
4、出错处理的需求：系统对于环境错误的响应，应尽量少用
5、接口需求：系统描述应用系统与它的环境通信的格式（用户接口需求，硬件接口需求，软件接口需求，通信接口需求）
6、约束 应遵守的限制条件
7、逆向需求：系统不应该做什么
8、将来可能提出的需求：不属于当前开发范围但以后可能提出的需求
3.1.2 分析系统的数据要求
E-R 图，数据字典（缺点：不够直观形象）， Warnier 图
3.1.3 导出系统的逻辑模型
通常用数据流图、E-R 图、状态转换图、数据字典、算法描述等
3.1.4 修正系统开发计划
根据分析过程中获得的新信息，修正以前的开发计划
3.2 与用户沟通获取需求的方法
（1）需求分析困难的原因
客户说不清需求；需求经常变动；分析人员理解偏差
（2）访谈
分为正式访谈和非正式访谈
访谈提纲：目的，对象，访谈维度，汇总问题
经常用到情景分析技术，便于用户理解，使用户化被动为主动，使用户更有积极性
访谈纪要：背景，访谈结果，如有需要附上文件
（3）面向数据自顶向下求精
研究对象：可行性分析阶段的数据流图
从输出端开始向输入端回溯，把涉及到的有关数据元素保存在数据字典中，把涉及到的算法保存在 IPO 图中
（4）简易的应用规格说明书

提倡用户与开发者密切合作，共同标识问题，提出解决方案要素

优点：用户开发者密切合作；及时讨论并求精；有能导出规格说明的具体步骤

（5）快速建立软件原型

第四代技术：数据库查询和报表语言等

可重用构建技术：用已经有的原型来装配原型

形式规格说明与原型技术：即墨刀等

3.3 分析建模与规格说明

数据模型——ER 图 逻辑模型——数据流图 行为模型——状态转换图

软件需求规格说明 P62

3.4 实体——联系图

基本成分：

（1） 数据对象

可以是外部实体、失误、行为、事件、角色、单位、地点、结构等（矩形）

（2） 属性

属性定义了数据对象的性质（圆形或圆角矩形）

（3） 联系

一对一（1:1）、一对多（1：N）、多对多（M：N）（菱形）

3.5 数据规范化

即范式

范式级别越高，即数据间的依赖性（冗余度）越低，但是数据的稳定性较差，每次需要访问的表增多，性能将下降，最合适的应该是 3NF P65

3.6 状态转换图

3.6.1 状态

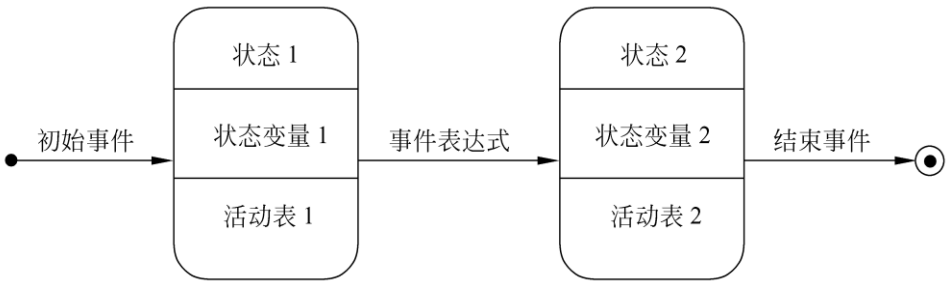
（1）状态

状态规定了系统对时间的响应方式

状态主要有：初态、终态和中间状态。一张状态图中只能有一个初态，可以有 0 到多个终态

（2）符号表示

活动表中三种标准时间：entry、exit、do



事件表达式格式：事件名（参数表）/动作表达式

3.7 其他图形工具

3.7.1 层次方框图
用树形结构的一系列多层侧的矩形框秒回数据的层次结构
3.7.2 Warnier 图
可以指出一个信息或是一个信息元素是重复出现的或是某一类信息是有条件的出现的
3.7.3 IPO 图
能方便的描述输入数据、对数据的处理和输出数据之间的关系
3.8 验证软件需求
3.8.1 从哪方面验证
一致性、完整性、有效性、现实性
3.8.2 验证的方法
一致性：自然语言书写 完整性和有效性：原型 现实性：根据经验
3.8.3 软件工具
RSL：需求陈述语言 PSL/PSA：问题陈述语言/问题陈述分析程序
第五章 总体设计
5.1 设计过程
（1）设想供选择的方案：目的：选出最佳方案
可从数据流图出发
（2）选取合理的方案
每个方案应准备：系统流程图；组成系统的物理元素清单；成本/效益分析；进度计划
（3）推荐最佳方案
选出最佳方案，并制定详细的实现计划
（4）功能分解 复杂功能进一步分解
（5）设计软件结构
程序中一个模块完成一个适当的子功能，模块应被组织成功良好的层次系统，顶层调用下层模块。若是数据流图层次适当，可直接映射成软件结构。软件结构可以用层次图或结构图描绘
（6）设计数据库
（7）制定测试计划　应在早起阶段就考虑测试问题
（8）书写文档 需要系统说明、用户手册、测试计划、详细的实现计划、数据库设计结果
（9）审查和复查
5.2 软件体系结构介绍
软件体系结构是软件系统中最本质的东西
5.2.1 定义
软件体系结构是具有一定形式的结构化元素，即构件的集合
软件体系结构为软件系统提供了一个结构、行为和属性的高级抽象
5.2.2 典型软件体系结构
分层、管道和过滤器、BS、CS、P2P
BT 把 P2P 技术发展到近乎完美
BS 是 CS 的一种，CS 可以基于任何协议，BS 只能基于 http 协议
5.2.3 分层体系结构
上下级关系：上层系统使用下层系统功能，下层系统不能使用上层系统功能
顺序相邻关系：只能相邻两层之间传递
5.2.4 客户机/服务器结构
浏览器——web 服务器——数据库服务器是一种十分流行的客户机/服务器结构

优点：客户机统一采用浏览器，使用户便捷，且客户机端
5.3 设计原理
5.3.1 模块化
模块基本属性：功能、逻辑、状态（模块需要的环境及条件）
模块化理论依据：E(P1+P2）> E(P1)+E(P2)
优点：易于测试哦和调试；提高软件的可修改性；易于阅读和理解；有助于软件开发工程组织与管理
5.3.2 抽象
抽象成层次
处理复杂系统唯一有效方法：用层次的方式构造和分析
5.3.3 逐步求精
定义：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
要求设计者细化原始陈述，随着每个后续求精步骤的完成而提供越来越多的细节
5.3.4 信息隐藏和局部化
信息隐藏原理：
一个模块包含的信息对于不需要这些信息的模块来说是不能访问的（即仅仅交换必要的信息）
5.3.5 模块独立（耦合和内聚）
1、模块的内聚性（7 个）
低内聚：偶然内聚：当几个模块内凑巧有一些程序段代码相同，又没有明确表现出独立的功能，为了减少存储把这些代码独立出来建立一个新的模块，这个模块就是偶然内聚模块。
逻辑内聚：一个模块完成的任务在逻辑上相同或相似
时间内聚：一个模块的所有功能必须在同一时间内执行，又叫经典内聚
中内聚：过程内聚：一个模块内处理元素相关，且必须以特殊次序执行
通信内聚：一个模块内的处理元素使用相同的输入数据或相同的输出数据
高内聚：顺序内聚：一个模块的处理元素和同一个功能密切相关且这些处理需按顺序执行
功能内聚：一个模块中各部分都是完成某一具体功能必不可少的组成部分
2、模块的耦合性
耦合性：非直接耦合、数据耦合、特征耦合、控制耦合、公共环境耦合、内容耦合（6 个）
非直接耦合：两个模块没有直接联系，他们的联系完全是通过主模块的控制和调用实现的
数据耦合：两个模块通过简单的数据参数来交换输入输出信息的
特征耦合：又叫标记耦合，两个模块之间通过传递数据结构，或都与一个数据结构有关系
控制耦合：模块之间传递的信息是控制量（一个过程通过标志或命令直接控制另一个过程）
公共环境耦合：一组模块都访问同一个公共数据环境。耦合的复杂度随模块个数增加而增加
内容耦合：一个模块直接访问另一个模块的内部数据
5.4 启发规则
1、模块功能的完善化 不仅应完成指定功能，还应告诉使用者完成任务的状态及不能完成的原因
2、改进软件结构提高模块独立性：力求高内聚低耦合
3、模块规模应该适中：模块不宜过大，一般不超过 500 行
4、深度、宽度、扇出、扇入都应适当
深度：表示软件结构中控制的层数，标志系统的大小和复杂度
宽度：软件结构内同一个层次上的模块总数的最大值，宽度越大系统越复杂

扇出：是一个模块直接控制（调用） 模块的总数，扇入越大系统越复杂
扇入：表示有多少个上级直接调用它，扇入大是有好处的但是要适度
5、模块的作用域应该在控制域之内
6、力争降低模块接口的复杂程度
7、设计单入口单出口的模块 警告工程师不要使模块间出现内容耦合
8、设计功能可预见的模块 输入数据相同就产生同样的输出
5.5. 描述软件结构的工具
5.5.1 层次图 适合自顶向下设计软件的过程中使用
5.5.2 HIPO 图 层次图+输入/处理/输出
5.5.3 结构图
尾部是空心圆表示传递的是数据;
实心圆表示传递的是控制信息。
总结：层次图和结构图并不表示模块的调用次序
层次图可作为秒回软件结构的文档
5.6 面向数据流的设计方法
面向数据流的方法定义了一些不同的“映射”，利用这些映射可以把数据流图变成软件结构
1、基本理念（概念）

第七章 实现
包括编码和测试
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
测试：检测程序并改正错误的过程
7.1 编码
语法，语义，语用（表示构成语言的各个记号和使用者的关系）
7.1.1 选择程序设计语言
必要性：影响人思维和解题方式，影响人机通信方式和质量，影响代码阅读和理解的难易程度
重要性：根据设计完成编码困难最少；减少程序测试量；得出更容易阅读和维护的程序
为什么选用低级语言：时间空间和环境（单片机）
标准：用户要求；编译程序；可以使用的软件工具；工程规模；开发人员；软件可移植性；软件应用领域
7.1.2 编码风格
（1）源程序文档化 标识符（即起名问题）；注释（占到整个程序 1/3 或 1/2）；视觉组织（空格）
（2）数据说明 应按一定次序常量、变量、数组、公用数据块、所有文件
（3）语句结构 一行只写一条语句；分解复杂表达式；写的程序要清晰易懂；尽量少用否定语句
（4）输入和输出
（5）效率 指程序执行速度及程序占用的内存的存储空间
a、程序运行时间 避免嵌套循环。少用多维数组、少用指针和复杂的表，采用整数表达式和布尔表达式
b、存储器效率 程序功能合理分块 关键：程序的简单性
c、输入输出效率 安排适当缓冲区，成块传送数据，存取方法简单

7.2 软件测试
7.2.1 软件测试的目的
为了发现程序中的错误而执行程序的过程
7.2.2 测试准则
7.2.3 测试方法
黑盒测试：测试功能是否能正常使用
白盒测试：测试产品内部动作是否按照规格说明书的规定正常进行
7.3 单元测试
通常使用白盒测试
7.3.1 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
7.3.2 代码审查
人工检查 优点：可一次性发现多个错误。计算机检查发现错误立刻停止
7.3.3 计算机测试
驱动软件：接受测试数据，相当于调用
存根软件：代替被测试的模块所调用的模块
7.4 集成测试
集成测试是测试和组装软件的系统化技术
方法：渐增式测试方法：每次加一个，好定位（好）
非渐增测试方法：全部合起来一起测试
7.4.1 自顶向下集成
从主控模块开始集成，有 BFS 和 DFS 两种方法
先测试主控模块，用存根程序代替需要调用的模块，测试完之后，每次用一个实际模块代替存根程序，进行测试，在这过程中，为使不引入新错误，要不断对之前的模块进行回归测试
7.4.2 自底向上集成
不需要存根程序，为每个模块设计一个驱动程序
7.4.3 比较
自顶向下优点：不需要驱动程序，能在早期实现并验证系统主要功能
缺点：需要存根程序，底层关键模块错误发现较晚
实际应用：（1）基本使用自顶向下，前期使用自底向上
（2）混合：底层用自底向上，上层用自顶向下
7.4.4 回归测试
重新执行已经做过的部分测试
7.5 确认测试（验收测试）
目的：验证软件有效性
基础：软件规格说明书
7.5.1 确认测试范围 这阶段的问题与需求分析有关
7.6 白盒测试
测试方案：测试目的、输入的数据及预期输出结果
7.6.1 逻辑覆盖
语句覆盖：被测程序中每个语句至少执行一次
判定覆盖：不仅，每个判定的每种可能都要覆盖
条件覆盖：不仅，判定表达式中每个条件都去到各种可能的结果
条件组合覆盖：每个判定表达式中条件的各种可能组合都至少出现一次

点覆盖，边覆盖，路径覆盖 P165
7.7 黑盒测试
7.7.1 等价划分
思想：把素有数据分为若干个等价类，从每个等价类中只取一组代表性数据作为测试数据
划分规则 P172
测试步骤：等价类编号->设计测试用例覆盖未覆盖过得有效等价类->设计测试用例覆盖一个尚未被覆盖的无效等价类
7.8 软件可靠性
程序在给定的时间间隔内，按照规格说明书的规定成功地运行的概率
“错误”：由开发人员造成的软件差错（bug）
“故障”：由错误引起的软件的不正确行为
7.9 软件可用性
程序在给定的时间点，按照规格说明书的规定，成功地运行的概率
两者区别：可靠性是指在某一段时间，系统没有失效的概率，可用性是指在某一时间点系统正常运行的概率
第八章 维护
8.1 软件维护的常识
软件系统交付使用以后，为了改正错误或满足新的需要而修改软件的过程
分类：改正性维护、适应性维护、完善性维护、预防性维护
8.1.1 软件维护的特点
（1）结构化维护与非结构化维护差别巨大
非结构化：只有程序没有文档
结构化：有程序也有文档
（2）维护成本高昂
软件维护活动分为生产性活动和非生产性活动
生产性活动包括分析评价、修改设计、编写程序代码
非生产性活动为理解东西
总工程量：M=P+K*exp（c-d）
M：总工程量 P：标示生产性活动量 K：经验常数 C：复杂度 D：维护人员对软件熟悉程度
（3）维护的问题很多
8.1.2 软件维护过程
（1）维护组织
维护管理员、系统监督员（对软件特别熟悉的技术人员）、配置管理员、维护人员、修改控制决策机构
维护团队可分为短期团队和长期团队
（2）维护报告
维护申请报告和软件修改报告
软件修改报告应包括：需求所需工作量、维护类别、请求优先级、背景数据
（3）维护事件流
请求->分类->排队->取出任务->实施工程
实施工程：修改软件需求说明、修改软件设计、设计评审、重新编码、单元测试、集成测试、确认测试等
最后一步是复审
（4）保存维护记录

（5）评价维护活动

8.2.1 决定软件可维护性因素

可理解性、可测试性、可修改性、可移植性、可重用性

提高可维护性方法：

1、 确定质量管理目标和优先级

2、 使用提高软件质量的技术和工具

3、 选择可维护性搞的程序设计语言 高级语言

4、 改进程序文档

5、 进行质量保证审查

8.2.2 文档

（1）用户文档：功能、安装、使用、参考、操作员指南

（2）系统文档

8.2.3 可维护性复审

需求分析阶段的复审：应该对将来要改进的部分和可能修改的部分进行说明

设计阶段的复审：评价软件结构和过程是否合理

代码复审：编码风格和内部说明文档

设计和编码阶段：尽量使用可重用构件

测试阶段：测试结束好进行最正式的可维护性复审

维护应针对整个软件配置，即对源程序的修改应反映在文档中

7.3 软件再工程

7.3.1 预防性维护

把今天的方法学应用到昨天的系统上，以支持明天的需求。

7.3.2 再工程

逆向工程：在软件生存周期中，将软件转换成更抽象形式的活动

重构：指在同一抽象级别上转换系统的描述形式

设计恢复：指借助工具从程序中抽象出设计信息

再工程：在逆向工程的基础上修改或重构已有系统

再工程主要目的：为遗留系统转化为可演化系统提供一条可行的途径

再工程分为：业务过程再工程：定义目标，评估现有系统，修订业务

软件再工程：对软件实施再工程

7.3.3 再工程过程

1、库存目录分析：包含每个应用系统的基本信息

预防性维护的对象：将使用很久的程序、现在正成功使用的程序、在将来可能会做重大修改的程序

2、文档重构 更新文档

3、逆向工程 比源代码更高的抽象层次上创建出程序的表示过程（实现级、结构级、功能级、领域级）

4、代码重构

5、数据重构 发生在相当底的抽象层次上

6、正向工程 从现有系统中恢复设计信息，并使用信息去改变和重构现有系统