

## 第 3 章

# 系统开发和运行基础知识

### 出题方向提示

#### 1. 考频统计

历年考题知识点分布如表 3-1 所示。

表 3-1 历年考题知识点分布统计表

年 份	试题 分布	分 值	考 核 要 点
2004 上	上午：无	0	
2004 下	上午：9~16	8	软件生存周期、需求分析任务、软件设计、系统测试、PERT 网图、数据字典、CMM 模型、数据流图
2005 上	上午：18~20	3	软件可移植性、系统切换技术
2005 下	上午：6~7	2	开发模型、测试计划
2006 上	上午：4~5、7~10	6	增量模型、Gantt 图、面向对象程序设计语言、软件规模度量、组件

#### 2. 命题要点

- Ä 软件工程概念：软件危机、软件工程研究内容，软件生命周期，软件开发模型，软件开发方法。
- Ä 需求分析：任务，主要工作，软件需求规格说明书，结构化分析方法。
- Ä 结构化设计方法：主要任务，原则，概要设计，详细设计。
- Ä 面向对象设计：面向对象设计的基本概念。
- Ä 开发环境：开发工具（设计工具、编程工具、测试工具、CASE），集成开发环境。
- Ä 软件测试评审方法：测试目的，测试方法（白盒测试、黑盒测试）、测试步骤（单元测试、集成测试、确认测试，系统测试，回归测试），评审方法。
- Ä 项目管理基础知识：制定项目计划，质量计划、管理和评估，过程管理（PERT 图、甘特图、工作分解结构、进度控制、关键路径），配置管理，人员计划和管理，文档管理，开发组织和作用，成本组织和风险管理，能力成熟度模型（CMM）。
- Ä 系统运行和维护：系统转换（直接转换、平行转换、分段转换），软件维护的分类，可维护性软件。

## 考点 1 需求分析和设计方法

### 1. 软件危机

①主要表现：软件需求的增长得不到满足；软件开发成本和进度无法控制；软件质量难以保证；软件不可维护或维护程度非常低；软件成本不断提高；软件开发生产效率的提高赶不上硬件的发展和应用需求的增长。总之，可以将软件危机归结为成本、质量和生产等问题。

②产生原因：一方面是由于软件本身存在着复杂性；另一方面与软件开发所使用的方法和技术相关。

### 2. 软件工程

①为了消除软件危机，通过认真研究解决软件危机的方法，认识到软件工程是使计算机软件走向科学的途径，逐渐形成了软件工程的观念，并开辟工程学的新兴领域，即软件工程学。

②软件工程有以下 3 个要素。

- ┆ 方法：完成软件工程项目的手段。
- ┆ 工具：支持软件的开发、管理、文档生成。
- ┆ 过程：支持软件开发的各个环节的控制、管理。

### 3. 软件生命周期

软件生存周期：软件产品从考虑其概念开始到该软件产品交付使用，直至最终退役为止的整个过程。

①计划阶段：确定待开发系统的总体目标和范围，研究系统的可行性和可能的解决方案，对资源、成本及进度进行合理的估算。软件计划的主要内容包括所采用的软件生命周期模型、开发人员的组织、系统解决方案、管理的目标与级别、所用的技术与工具，以及开发的进度、预算和资源分配。

②分析阶段：分析、整理和提炼所收集到的用户需求，建立完整的分析模型，将其编写成软件需求规格说明书和初步的用户手册。通过评审需求规格说明书，确保对用户需求达到共同的理解与认识。软件需求规格说明书明确地描述了软件的功能，列出软件必须满足的所有约束条件，并定义软件的输入和输出接口。

③设计阶段：决定软件怎么做，设计人员依据软件需求规格说明书，确定软件的体系结构，进而确定每个模块的实现算法、数据结构和接口等，编写设计说明书，并组织进行设计评审。

④实现阶段：将所设计的各个模块编写成计算机可接受的程序代码，与实现相关的文档就是源程序及合适的注释。

⑤测试阶段：在设计测试用例的基础上，再测试软件的各个组成模块。然后，将各个模块集成起来，测试整个产品的功能和性能是否满足软件需求规格说明书。

⑥运行维护阶段：是软件过程的一个组成部分，应当在软件的设计和实现阶段充分考虑软件的可维护性。维护阶段需要测试是否正确地实现了所要求的修改，并保证在产品的修改过程中，没有做其他无关的改动。

#### 4. 软件开发模型

①瀑布模型：将软件生命周期划分为制订计划、需求分析、软件设计、程序编写、软件测试和运行维护等 6 个基本活动，并且规定它们自上而下、相互衔接的固定次序，如同瀑布流水，逐级下落等。其优点是可以规范化过程，有利于评审；缺点是过于理想，缺乏灵活性，容易产生需求偏差。

②快速原型模型：第一步是建造一个快速原型，实现客户或未来的用户与系统的交互，用户或客户对原型进行评价，进一步细化待开发软件的需求。通过逐步调整原型使其满足客户的要求，开发人员可以确定客户的真正需求是什么；第二步则在第一步的基础上开发客户满意的软件产品。快速原型方法可以克服瀑布模型的缺点，减少由于软件需求不明确带来的开发风险，具有显著的效果。

③演化模型：也是一种原型化开发，但与快速原型模型不同的是，快速原型模型在获取真实需求后，将抛弃原型。而演化模型则不然，它将在快速开发一个原型的基础上，逐步演化成最终的软件。

④增量模型：软件被作为一系列的增量构件来设计、实现、集成和测试。每一个构件是由多种相互作用的模块，所形成的提供特定功能的代码片段构成。增量模型在各个阶段并不交付一个可运行的完整产品，而是交付满足客户需求的一个子集的可运行产品。整个产品被分解成若干个构件，开发人员逐个构件地交付产品，这样做的好处是软件开发可以较好地适应变化，客户可以不断地看到所开发的软件，从而降低开发风险。也就是说增量模型有利于快速开发软件。

⑤螺旋模型：综合了瀑布模型和演化模型的优点，还增加了风险分析，特别适合于大型复杂的系统。采用螺旋模型时，软件开发沿着螺旋线自内向外旋转，每转一圈都要对风险进行识别和分析，螺旋线第一圈的开始点可能是一个概念项目。从第二圈开始，一个新产品开发项目开始了，新产品的演化沿着螺旋线进行若干次迭代，一直运转到软件生命期结束。

⑥喷泉模型：对软件复用和生存期中多项开发活动的集成提供了支持，主要支持面向对象的开发方法。“喷泉”一词本身体现了迭代和无间隙特性。系统某个部分常常重复工作多次，相关功能在每次迭代中随之加入演进的系统。所谓无间隙是指在开发活动中，包括分析、设计和编码之间不存在明显的边界。

#### 5. 软件开发方法

①结构化软件开发方法（SASD）：也称为面向功能的软件开发方法或面向数据流的软件开发方法。首先用结构化分析（SA）对软件进行需求分析，然后用结构化设计（SD）方法进行总体设计，最后是结构化编程（SP）。SASD 给出了两类典型的软件结构（变换型和事务型）使软件开发的成功率大大提高。

结构化软件开发方法，是采用结构化技术来完成软件开发的各项任务的。它把软件生命周期划分成若干个阶段，依次地完成每个阶段的任务；它与瀑布模型有很好的结合度，是与其最相适应的软件开发方法。

②面向数据结构的软件开发方法：从目标系统的输入、输出数据结构入手，导出程序框架结构，再补充其他细节，从而得到完整的程序结构图。有 Jackson 方法和 Warnier 方法。

③面向对象的软件开发方法：随着 OOP（面向对象编程）向 OOD（面向对象设计）和 OOA（面向对象分析）的发展，最终形成面向对象的软件开发方法 OMT（Object Modelling

Technique)。这是一种自底向上和自顶向下相结合的方法。它以对象建模为基础，不仅考虑了输入、输出数据结构，还考虑了所有对象的数据结构。

④基于构件化的开发方法：用预先建立的构件和模板，像“搭积木”一样进行建造。

## 6. 需求分析

### (1) 任务

确定软件系统的功能需求和非功能需求；分析软件系统的数据要求；导出系统的逻辑模型；修正项目开发计划；如有必要，还可以开发一个原型。

### (2) 主要工作

需求获取——确定对目标系统的各方面需求。涉及到的主要任务是建立获取用户需求的方法框架，并支持和监控需求获取的过程；需求分析和综合——对问题进行分析，然后在此基础上整合出解决方案；编写需求规格说明书——对已确定的需求进行文档化描述。该文档通常称为“软件需求规格说明书”；需求评审——评审需求分析的正确性、完整性和清晰性。

### (3) 软件需求规格说明书

需求分析阶段的最后成果，是软件开发的重要文档之一。其作用有 3：便于用户、开发人员进行理解和交流；反映出用户问题的结构，可以作为软件开发工作的基础和依据；作为确认测试和验收的依据。软件需求规格说明书的内容主要包括：概述、数据描述、功能描述、性能描述、参考文献、附录等。

## 7. 结构化分析方法

①结构化分析方法（Structured Analysis, SA）是面向数据流进行需求分析的方法，采用自顶向下、逐层分解，建立系统的处理流程，以数据流图和数据字典为主要工具，建立系统的逻辑模型。

②常用工具：数据流图、数据字典、结构化语言、判定树和判定表等。

③数据流图（Data Flow Diagram, DFD）介绍如下。

- Ⅰ 作用：以图形的方式描绘数据在系统中流动和处理的过程，它只反映系统必须完成的逻辑功能，所以是一种功能模型。
- Ⅰ 图形元素：圆或椭圆，表示加工，输入数据经过加工变换产生输出；箭头，表示数据流，沿箭头方向传送数据的通道；双杠，表示存储文件，处理过程中存放各种数据文件；方框，表示源/宿、系统与环境接口，属于系统之外的实体。
- Ⅰ 分层数据流图：分为顶层流图、底层流图和中间层流图。顶层流图仅包含一个加工，它代表被开发系统。它的输入流是该系统的输入数据，输出流是系统所输出数据。底层流图是指其加工不需再做分解的数据流图，处在最底层。中间层流图则表示对其上层父图的细化。它的每一加工可能继续细化，形成子图。
- Ⅰ 建立步骤：由外向里先画系统的输入输出，然后画系统的内部；由顶向下完成顶层、中间层、底层数据流图；逐层分解。

④数据字典：结构化分析方法的核心，是对数据流图中出现的被命名的图形元素的确切解释，通常包括名称、别名、何处使用/如何使用、内容描述、补充信息等内容。

⑤加工规格说明：用来说明 DFD 中的数据加工的加工细节，表达“做什么”，而不是“怎样做”。描述工具有结构化语言、判定表和判定树。

## 8. 结构化设计

### (1) 软件设计过程

对程序结构、数据结构、过程细节和接口细节逐步细化、评审和编写文档的过程。从技术角度上，软件设计分成体系结构设计、数据设计、接口设计、过程设计4个方面的工作。从管理角度上讲，软件设计分为概要设计、详细设计两个阶段。

### (2) 软件设计目标

设计必须实现分析模型中描述的所有显式需求，必须满足用户希望的所有隐式需求；设计必须是可读、可理解的，使得将来易于编程、易于测试、易于维护；设计应从实现角度出发，给出与数据、功能、行为相关的软件全貌。

### (3) 基本原理和相关概念

①抽象化：常用的抽象手段有过程抽象、数据抽象和控制抽象。

┆ 过程抽象：任何一个完成明确功能的操作都可被使用者当作单个的实体看待，尽管这个操作实际上可能由一系列更低级的操作来完成。

┆ 数据抽象：数据抽象与过程抽象一样，允许设计人员在不同层次上描述数据对象的细节。

┆ 控制抽象：与过程抽象和数据抽象一样，控制抽象可以包含一个程序控制机制而无须规定其内部细节。

②自顶向下，逐步细化：将软件的体系结构按自顶向下方式，对各个层次的过程细节和数据细节逐层细化，直到用程序设计语言的语句能够实现为止，从而最后确立整个的体系结构。

③模块化：将一个待开发的软件分解成若干小的简单的部分——模块，每个模块可独立地开发、测试，最后组装成完整的程序。这是一种复杂问题的“分而治之”的原则。模块化的目的是使程序的结构清晰，容易阅读，容易理解，容易测试，容易修改。

④控制层次：表明了程序构件（模块）的组织情况。控制层次往往用程序的层次（树形或网状）结构来表示。

┆ 深度：程序结构的层数，可反映程序结构的规模和复杂程度。

┆ 宽度：同一层模块的最大模块个数。

┆ 模块的扇出：一个模块直接调用（或控制）的其他模块数目。

┆ 模块的扇入：调用（或控制）一个给定模块的模块个数。

⑤信息隐蔽：将每个程序的成分隐蔽或封装在一个单一的设计模块中，定义每一个模块时尽可能少地显露其内部的处理，可以提高软件的可修改性、可测试性和可移植性。

⑥模块独立：每个模块完成一个相对独立的特定子功能，并且与其他模块之间的联系简单。衡量度量标准有两个：模块间的耦合和模块的内聚。模块独立性强必须做到高内聚低耦合。

┆ 耦合：模块之间联系的紧密程度，耦合度越高则模块的独立性越差。耦合度从低到高的次序依次是：非直接耦合、数据耦合、标记耦合、控制耦合、外部耦合、公共耦合、内容耦合。

┆ 内聚是指模块内部各元素之间联系的紧密程度，内聚度越低模块的独立性越差。内聚度从低到高依次是：偶然内聚、逻辑内聚、瞬时内聚、过程内聚、通信内聚、顺序内聚、功能内聚。

#### (4) 模块优化的启发规则

- l 改进软件结构提高模块独立性，提高模块的内聚度，降低模块间的耦合度。
- l 模块规则不应过大，最好编写在一页纸内（不超过 60 行语句），模块规模过小，接口复杂。
- l 深度、宽度、扇出和扇入应适中。
- l 模块的作用域应该在控制域之内。
- l 力争降低模块接口的复杂程度。
- l 设计单入口、单出口的模块。
- l 模块功能应该可以预测。

### 9. 概要设计

#### (1) 设计过程

- l 制订规范：制订在设计时应该共同遵守的标准，以便协调组内各成员的工作。
- l 软件系统结构的总体设计：划分成模块的层次结构、确定每个模块的功能、确定模块间的调用关系、确定模块间的接口、评估模块划分的质量及导出模块结构的规则。
- l 处理方式设计：确定为满足软件系统的性能需求所必需的算法和模块间的控制方式（性能设计）、确定外部信号的接收发送形式（接口设计）。
- l 数据结构设计：确定软件涉及的文件系统的结构，以及数据库的模式、子模式，进行数据完整性和安全性的设计。
- l 可靠性设计：确定软件可靠性和其他质量指标。
- l 编写概要设计阶段的文档：概要设计阶段完成时应编写以下文档——概要设计说明书、数据库设计说明书、用户手册、修订测试计划。对测试的策略、方法和步骤提出明确的要求。
- l 概要设计评审：对概要设计的结果进行严格的技术审查，在技术审查通过之后再由使用部门的负责人从管理角度进行复审。

#### (2) 设计工具

常用的软件结构设计工具是结构图（也称程序结构图）。

- l 作用：描述软件系统的层次和分块结构关系，它反映了整个系统的功能实现，以及模块与模块之间的联系与通信。
- l 图形元素：在程序结构图中，模块用一个矩形表示，矩形内注明模块的功能和名字；箭头表示模块间的调用关系。用带实心圆的箭头表示传递的是控制信息，用带空心圆的箭心表示传递的是数据。

#### (3) 面向数据流的设计方法

①数据流的类型：有变换型和事务型两种。

- l 变换型：信息沿输入通路进入系统，同时由外部形式变换成内部形式，进入系统的信息通过变换中心，经加工处理以后再沿输出通路变换成外部形式，离开软系统。
- l 事务流：当信息沿输入通路到达一个处理，这个处理根据输入数据的类型从若干个动作序列中选择出一个来执行。在一个事务流中，事务中心接收数据，分析每个事务以确定它的类型，根据事务类型选取一条活动通路。

②面向数据流设计方法的实施要点与设计过程如下。

面向数据流的结构设计过程和步骤是：分析、确认数据流图的类型，区分是事务型还是变换型；说明数据流的边界；把数据流图映射为程序结构；根据设计准则把数据流转换成程序结构图。

- ┆ 变换分析：将变换型映射成结构图。
- ┆ 事务分析：将事务型映射成结构图。

## 10. 详细设计

### (1) 任务

为软件结构图中的每一个模块确定实现算法和局部数据结构，用某种选定的表达工具表示算法和数据结构的细节。

### (2) 常用工具

- ┆ 图形工具：程序流程图、N-S、PAD 和 HIPO。
- ┆ 表格工具：判定表。
- ┆ 语言工具：PDL（伪码）。

## 11. 面向数据结构设计——Jackson 方法

①面向数据结构设计：以数据结构作为设计的基础，它根据输入输出数据结构导出程序的结构，适用于规模不大的数据处理系统，Jackson 方法是一种典型的面向数据结构的设计方法。

②Jackson 图如图 3-1 所示。

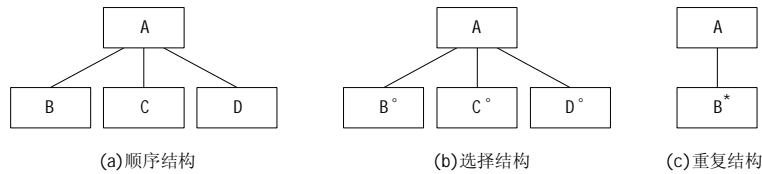


图 3-1 Jackson 图

③Jackson 方法的设计步骤如下。

- ┆ 分析并确定输入和输出数据的逻辑结构，并用 Jackson 图表示。
- ┆ 找出输入数据结构与输出数据结构间有对应关系的数据单元。所谓有对应关系的数据单元是指有直接因果关系，在程序中可以同时处理的数据单元。对于重复结构的数据单元，必须在重复次数和次序都相同时才有对应关系。
- ┆ 从描述数据结构的 Jackson 图导出描述程序结构的 Jackson 图。
- ┆ 列出所有的操作，并把它们分配到程序结构图的适当位置上。
- ┆ 用伪码表示程序。

## 12. 面向对象设计基本概念

①对象：一组属性及这组属性上的专用操作的封装体，通常由对象名、属性和操作这 3 个部分组成。属性表示该对象的状态，用户只能看见对象封装界面上的信息，对象的内部实现对用户是隐蔽的。封装目的是使对象的定义和实现分开。

②类：一组具有相同属性和相同操作的对象的集合。一个类中的每个对象都是这个类的一

个实例 (Instance)。

③继承：在某个类的层次关联中不同的类共享属性和操作的一种机制。一个父类可以有多个子类，这些子类都是父类的特例。父类描述了这些子类的公共属性的操作，子类中还可以定义它自己的属性和操作。一个子类只有唯一的一个父类，这种继承称为单一继承。一个子类有多个父类，可以从多个父类中继承特性，这种继承称为多重继承。

④消息：对象间通信的手段、一个对象通过向另一对象发送消息来请求其服务。消息通常包括接收对象名、调用的操作名和适当的参数（如有必要）。消息只告诉接收对象需要完成什么操作，但不能指示接收者怎样完成操作。消息完全同接收者解释，接收者独立决定采用什么方法来完成所需的操作。

⑤多态性：同一个操作作用不同的对象可以有不同的解释，产生不同的执行结果。

⑥继承性：是面向对象程序设计语言不同于其他语言的主要特点，是否建立了丰富的类库是衡量一个面向对象程序设计语言成熟与否的重要标志之一。

⑦组件：在面向对象的软件工程中，一个组件 (component) 包含了一些协作的类的集合。

## 考点 2 开发环境

### 1. 软件工具

①用于辅助软件开发、运行、维护、管理、支持等过程中的活动的软件，通常也称为 CASE (计算机辅助软件工程) 工具。

②通常可将软件工具分为软件开发工具、软件维护工具和软件管理工具。

### 2. 集成开发环境

①一种把支持多种软件开发方法和开发模型的软件工具，集成在一起的软件开发环境。

②集成型开发环境通常可由工具集成和环境集成机制两部分组成。工具集中还应该包含支持软件生存周期的阶段活动，以及支持各种开发方法和开发模型的工具，它支持软件开发的全过程；而环境集成机制主要包含数据集成机制、控制集成机制和界面集成机制等 3 方面内容。

③集成开发环境应具有开放性和可剪裁性。开放性为环境外的工具集成到环境中来提供方便，可剪裁性可根据不同的应用或不同的用户需求进行剪裁，以形成特定的开发环境。

## 考点 3 软件的测试

### 1. 软件测试的基本概念

①测试工作量约占软件开发总工作量的 40% 以上，特别对一些关系到人的生命安全的软件，测试成本可能相当于开发阶段总成本的 3 倍至 5 倍。

②测试的目的：检验它是否满足规定的需求或是弄清预期结果与实际结果之间的差别。简单地说，软件测试的目的是尽可能多地发现软件产品（主要指程序）中的错误和缺陷。

③测试用例：由测试数据和预期结果构成的。一个好的测试用例是极有可能发现迄今为止

尚未发现的错误。一次成功的测试，能发现至今为止尚未发现的错误。

④测试方法：测试的关键是测试用例的设计，其方法可分成白盒测试和黑盒测试。

## 2. 白盒测试

①白盒测试法需要了解程序内部的结构，测试用例是根据程序的内部逻辑来设计的。白盒测试法主要用于软件的单元测试。

②白盒测试的基本原则是：保证所测模块中每一个独立路径至少执行一次；保证所测模块所有判断的每一个分支至少执行一次；保证所测模块每一个循环都在边界条件和一般条件下至少执行一次；验证所有内部数据结构的有效性。

③白盒测试法常用的技术是逻辑覆盖。主要的覆盖标准有6种，即强度由低到高依次是：语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖、路径覆盖。

## 3. 黑盒测试

①黑盒测试，是对软件已经实现的功能是否满足需求进行测试和验证。黑盒测试不关心程序内部的逻辑，只是根据程序的功能说明来设计测试用例。黑盒测试法主要用软件确认测试。

②测试方法如下。

- Ⅰ 等价类划分：把输入数据划分成若干个有效等价类和若干个无效等价类，然后设计测试用例覆盖这些等价类。
- Ⅰ 边界值分析：对各种输入、输出范围的边界情况设计测试用例的方法。这是因为程序中在处理边界情况时出错的概率比较大。
- Ⅰ 错误猜测：根据经验或直觉推测程序中可能存在的各种错误。
- Ⅰ 因果图：根据输入条件与输出结果之间的因果关系来设计测试用例。

## 3. 软件测试步骤

①单元测试：也称模块测试，主要发现编码和详细设计中产生的错误，通常采用白盒测试。放在编码阶段，由程序员自己来完成，检查它是否实现了详细设计说明书中规定的模块功能和算法。其测试计划是在详细设计阶段完成。单元测试的测试计划是在详细设计阶段完成。

②集成测试：也称组装测试，对由各模块组装而成的程序进行测试，主要检查模块间的接口和通信。集成测试主要发现设计阶段产生的错误，通常采用黑盒测试或灰盒测试。集成的方式可分成非渐增式集成和渐增式集成。非渐增式集成是先测试所有的模块，然后把把这些模块集成在一起对整个程序进行测试。渐增式集成是将单元测试和集成测试合并在一起。它根据模块结构图，按某种次序选一个尚未测试的模块，把它同已经测试好的模块组合在一起对整个程序进行测试，每次增加一个模块，直至所有模块全部集成在程序中。当使用渐增式集成方式把模块结合到程序中去时，有自顶向下和自底向上两种集成策略。其测试计划在概要设计阶段完成，集成测试的测试计划也在概要设计阶段完成。

③确认测试：检查软件的功能、性能及其他特征是否与用户的需求一致，它是以需求规格说明书（即需求规约）作为依据的测试。确认测试通常采用黑盒测试。其测试计划是在需求分析阶段完成。

- Ⅰ Alpha 测试：在开发者的现场由客户来实施的。被测试的软件是在开发者指导下，从用户的角度在常规设置的环境下运行的。

I Beta 测试：在一个或多个客户的现场，由该软件的最终用户实施。开发者通常不在场。

④系统测试：把已经经过确认的软件纳入实际运行环境中，与其他系统成分组合在一起进行测试。主要内容包括恢复测试、安全测试、强度测试、性能测试、可靠性测试、安装测试等。

## 考点 4 项目管理基础知识

### 1. 项目管理和项目计划的制订

#### (1) 项目管理

项目的核心内容就是在成本、质量、进度间寻找平衡。主要包括 POIM 4 个方面：Plan 计划、Organize 组织、Implement 实现、Measurement 度量。

主要活动包括：启动——确定目标和范围，考虑解决方案，根据方案进行社会、经济、技术可行性分析、成本估算、任务分解、进度安排等；度量——把握过程中的实际情况和产品质量；估算——人力、时间、成本的计算，要建立历史项目资料库；风险分析——按发生可能性和影响性排序，制定解决方案和预防措施；进度安排——基于工作任务分解 WBS 之上，对时间、人员、设备分配，用甘特图、PERT 图等 Project 工具；追踪控制——根据实际进度情况进行有效调整，并处理需求变更而引起的项目计划的变更。

#### (2) 项目计划的制订

预测未来、确定任务、估计可能碰到的问题，并提出完成任务和解决问题的有效方案、方针、措施和手段，以及必须的各种活动和工作成果的过程。

项目计划的主要内容包括：估算所需要的人力（通常以人/月为单位）、项目持续时间（以年份或月份为单位）、成本（以元为单位）；做出进度安排，分配资源，建立项目组织及任用人员（包括人员的地位、作用、职责、规章制度等），根据规模和工作量估算分配任务；进行风险分析，包括风险识别、风险估计、风险优化、风险驾驭策略、风险解决和风险监督等。这些步骤贯穿在软件工程过程中；制订质量管理指标；编制预算和成本；准备环境和基础设施等。

### 2. 质量计划、管理和评估

#### (1) 软件质量度量模型

软件质量特性和质量子特性如表 3-2 所示。

表 3-2 软件质量特性和质量子特性

质量特性	描述	子特性	子特性描述
功能性	与一组功能及其指定的性质有关的一组属性。这里的功能是指满足明确或隐含的需求的那些功能	适合性	与规定任务能否提供一组功能及这组功能的适合程度有关的软件属性
		准确性	与能否得到正确或相符的结果或效果有关的软件属性
		互用性	与其他指定系统进行交互的能力有关的软件属性
		依从性	使软件遵循有关的标准、约定、法规及类似规定的软件属性

(续表)

质量特性	描述	子特性	子特性描述
		安全性	与防止对程序及数据的非授权的故意或意外访问的能力有关的软件属性
可靠性	与在规定的一段时间和条件下,软件维持其性能水平的能力有关的一组属性	成熟性	与由软件故障引起失效的频度有关的软件属性
		容错性	与在软件故障或违反指定接口的情况下,维持规定的性能水平的能力有关的软件属性
		可恢复性	与在失效发生后,重建其性能水平并恢复直接受影响数据的能力,以及为达此目的所需的时间和能力有关的软件属性
可用性	与一组规定或潜在的用户为使用软件所需作的努力和对这样的使用所作的评价有关的一组属性	可理解性	与用户为认识逻辑概念及其应用范围所花的努力有关的软件属性
		易学习性	与用户为学习软件应用所花的努力有关的软件属性
		可操作性	与用户为操作和运行控制所花努力有关的软件属性
效率	与在规定的条件下,软件的性能水平与所使用资源量之间关系有关的一组属性	时间特性	与软件执行其功能时响应和处理时间及吞吐量有关的软件属性
		资源特性	与在软件执行其功能时所使用的资源数量及其使用时间有关的软件属性
可维护性	与进行指定的修改所需的努力有关的一组属性	可分析性	与为诊断缺陷或失效原因及为判定待修改的部分所需努力有关的软件属性
		可修改性	与进行修改,排除错误或适应环境变化所需努力有关的软件属性
		稳定性	与修改所造成的未预料结果的风险有关的软件属性
		可测试性	与确认已修改软件所需的努力有关的软件属性
可移植性	与软件可从某一环境转移到另一环境的能力有关的一组属性	适应性	与软件无需采用有别于为该软件准备的活动或手段就可能适应不同的规定环境有关的软件属性
		可安装性	与在指定环境下安装软件所需努力有关的软件属性
		一致性	使软件遵循与可移植性有关的标准或约定的软件属性
		可替换性	与软件在该软件环境中用来替代指定的其他软件的机会和努力有关的软件属性

### (2) 质量管理

软件管理通过制订质量方针、建立质量目标和标准 (Target),并在项目生命期内持续使用质量计划 (Plan)、质量控制 (Do)、质量保证 (Check) 和质量改进 (Action) 等措施来落实质量方针的执行,确保质量目标的实现,最大限度地使客户满意。

### (3) 软件质量评审

主要包括设计质量评审和程序质量评审。

- I 设计质量评审: 在需求分析阶段产生的软件需求规格说明、数据要求规格说明,在软件概要设计阶段产生的软件概要设计说明书等。
- I 程序质量评审: 着眼于软件本身的结构、与运行环境的接口、变更带来的影响而进行的评审活动。通常它是从开发者的角度进行评审,直接与开发技术有关,主要包括软件的结构 (功能结构、功能的通用性、模块层次、模块结构、处理过程的结构)、与运行环境的接口 (包括硬件、其他软件 and 用户)、变更的影响范围。

### 3. 进度管理

#### (1) 主要内容

确立日程计划、规定各开发阶段（工程）的定义和成品、设定工程管理重点——里程碑（milestone）、制作工程作业网络图、掌握各工程进度、及时发现工程时间延误和提前，并及时调整相关部署等。

#### (2) 技术与方法

①Gantt（甘特）图：用水平线段表示任务的工作阶段；线段的起点和终点分别对应着任务的开工时间和完成时间；线段的长度表示完成任务所需的时间。优点：标明了各任务的计划进度和当前进度，能动态地反映软件开发进展情况。缺点：难以反映多个任务之间存在的复杂的逻辑关系。

②PERT 技术和 CPM 方法：PERT 技术叫做计划评审技术，CPM 方法叫做关键路径法，它们都是安排开发进度、制订软件开发计划的最常用的方法。它们都采用网络图来描述一个项目的任务网络，也就是从一个项目的开始到结束，把应当完成的任务用图或表的形式表示出来。通常用两张表来定义网络图。一张表给出与一特定软件项目有关的所有任务（也称为任务分解结构），另一张表给出应当按照什么样的次序来完成这些任务（也称为限制表）。

### 4. 文档管理

#### (1) 文档

文档是软件产品的一部分，没有文档的软件就不称其为软件。文档的编制在软件开发工作中占有突出的地位和相当大的工作量。高质量、高效率地开发、分发、管理和维护文档对于转让、变更、修正、扩充和使用文档，对于充分发挥软件产品的效益有着重要的意义。

#### (2) 文档的类型

软件文档从形式上来看，大致可分为两类：一类是开发过程中填写的各种图表，可称之为工作表格；另一类是应编制的技术资料或技术管理资料，可称之为文档或文件。

按照文档产生和使用的范围，软件文档大致可分为开发文档、管理文档和用户文档等 3 类。

#### (3) 主要文档

国家标准《计算机软件产品开发文件编制指南 GB 8567-88》中规定，在一项软件开发过程中，一般地说应该产生 14 种文件，如表 3-3 所示。

### 5. 成本组织

#### (1) 估算策略

- I 自顶向下。对整个项目的总开发时间和总工作量做出估算，然后把它们按阶段、步骤和工作单元进行分配。
- I 自底向上。自底向上的方法则正好相反，分别估算各工作单元所需的工作量和开发时间，然后相加，就得出总的工作量和总的开发时间。

表 3-3 软件文档

文档	与软件生命周期各阶段关系					与各类人员的使用关系			
	可行性研究与计划	需求分析	软件设计	编码/单元测试	集成/确认测试	管理人员	开发人员	维护人员	用户
可行性研究报告	√					√	√		
项目开发计划	√					√	√		
软件需求说明书		√					√		
数据要求说明书		√					√		
概要设计说明书			√				√	√	
详细设计说明书			√				√	√	
数据库设计说明书			√				√	√	
用户手册			√	√					√
操作手册			√	√					√
模块开发卷宗				√	√	√		√	
开发进度月报	√		√	√	√	√			
测试计划			√				√		
测试分析报告					√		√	√	
项目开发总结报告					√	√			

(2) 估算方法

- I 专家估算法 (Delphi)：依靠一个或多个专家，对要求的项目做出估计。
- I 类推估算法：在自顶向下的策略中，将估算项目的总体参数与类似项目进行直接相比得到结果。在自底向上的策略中，类推是在两个具有相似条件的工作单元之间进行。
- I 算式估算法：专家估算法和类推估算法的缺点在于，它们依靠带有一定盲目性的和主观的猜测对项目进行估算。算式估算法目的是避免主观因素的影响。用于估算的算式方法有两种基本方法，一是由理论导出的，二是由经验得出的。

(3) 软件规模估算

- I LOC (代码行数) 估算法：LOC 是一种自底向上的估算方法，即从模块开始进行估算，其优点是容易计算。
- I FP (功能点) 估算法：依据对软件信息领域特性和软件复杂性的评估结果，估算软件规模。这种方法用功能点为单位度量软件规模。功能点有 5 个信息域特征：输入数 (Inp)、输出数 (Out)、查询数 (Inq)、主文件数 (Maf)、外部接口数 (Inf)。

(4) 工作量估算

- I 工作量可以用人/日、人/月或人/年的数量来表示。知道单位工作量的成本，就可得到估算成本。
- I 工作量估算模型有：IBM 模型 (在 60 个项目的基础上进行统计的静态模型)、普特南模型 (是一个动态多变量模型，通过建立一个资源需求曲线模型导出等式)、COCOMO 模型 (将项目分为组织型、嵌入型、半独立型项目，包括有基本静态、半独立、详细 3 种不同模型)。

## 6. 开发组织和作用（开发组成员、项目经理）

### （1）组织结构的模式

有 3 种方式：按课题划分的模式、按职能划分的模式、矩阵模式。

### （2）程序设计小组的组织形式

- Ⅰ 主程序员制小组：由一位主程序员、二位至五位技术员、一位后援工程师组成。主程序员负责小组全部技术活动的计划、协调与审查工作，还负责设计和实现项目中的关键部分。技术员负责项目的具体分析与开发，以及文档资料的编写工作。后援工程师支持主程序员的工作，为主程序员提供咨询，也做部分分析、设计和实现的工作，并在必要时能代替主程序员工作。主程序员制的开发小组突出了主程序员的领导。这种集中领导的组织形式能否取得好的效果，很大程度上取决于主程序员的技术水平和管理才能。
- Ⅰ 民主制小组：组内成员之间可以平等地交换意见。工作目标的制订及做出决定都由全体成员参加。虽然也有一位成员当组长，但工作的讨论、成果的检验都公开进行。
- Ⅰ 层次式小组：组内人员分为 3 级，组长（项目负责人）负责全组工作，包括任务分配、技术评审和走查、掌握工作量和参加技术活动，直接领导二三名高级程序员；每位高级程序员通过基层小组，管理若干位程序员。

## 7. 风险管理

①风险分析包括 4 个不同的活动：风险识别、风险估计、风险评价和风险驾驭。

②风险识别：系统地确定对项目计划（估算、进度、资源分配）的威胁，通过识别已知的或可预测的风险，就可能设法避开风险或驾驭风险。从宏观上来看，可将风险分为项目风险、技术风险和商业风险。

③风险估计：从两个方面估价每一种风险。一是估计一个风险发生的可能性，二是估价与风险相关的问题出现后将会产生的结果。通常，项目计划人员与管理人员、技术人员一起，进行 4 种风险估计活动。

- Ⅰ 建立一个尺度来表明风险发生的可能性。
- Ⅰ 描述风险的后果。
- Ⅰ 估计风险对项目和产品的影响。
- Ⅰ 指明风险估计的正确性以便消除误解。

## 8. 能力成熟度模型（CMM）简介

能力成熟度模型 CMM 用于衡量软件企业的开发管理水平。它可作为软件发包方评估承包方执行能力的参考标准，也可以被软件企业作为软件过程改进工作的参考模型。CMM 模型将软件过程的成熟度分为 5 个等级。

①初始级：软件过程的特点是无秩序的，有时甚至是混乱的。软件过程定义几乎处于无章法和步骤可循的状态，软件产品所取得的成功往往依赖于极个别人的努力和机遇。

②可重复级：已建立了基本的项目管理过程，可用于对成本、进度和功能特性进行跟踪。对类似的应用项目，有章可循并能重复以往所取得的成功。

③已定义级：用于管理的工程软件过程均已文档化、标准化，并形成了整个软件组织的标

准软件过程。全部项目均采用与实际情况相吻合的、适当修改后的标准软件过程来进行操作。

④已管理级：软件过程和产品质量有详细的度量标准。软件过程和产品质量得到了定量的认识和控制。

⑤优化级：通过对来自过程、新概念和新技术等方面的各种有用信息的定量分析，能够不断地、持续地对促进过程进行改进。

## 考点 5 系统运行和维护

### 1. 系统转换

①直接转换：在原有系统停止运行的某一时刻，新系统立即投入运行，中间没有过渡阶段。

②平行转换：新系统和原系统平行工作一段时间，经过这段时间的试运行后，再用新系统正式替换下原有系统。在平行工作期间，一旦新系统有问题就可以暂时停止而不会影响原有系统的正常工作。

③分段转换：上述两种方式的结合，采取分期分批逐步转换。一般比较大的系统采用这种方式较为适宜，它能保证平稳运行，费用也不太大。

### 2. 软件维护

①在软件交付使用后，为了改正软件中隐藏的错误，或者为了使软件适应新的环境，或者为了扩充和完善软件的功能或性能而修改软件的过程。在整个软件生存周期所花费的代价中，20 世纪 80 年代末用于软件维护的经费约为 75%，到 90 年代初为 90%。

②软件维护的分类如下。

- Ⅰ 改正性维护：在使用过程中发现了隐藏的错误后，为了诊断和改正这些隐藏错误而修改软件的活动。
- Ⅰ 适应性维护：为了适应变化了的环境而修改软件的活动。
- Ⅰ 完善性维护：为了扩充或完善原有软件的功能或性能而修改软件的活动。
- Ⅰ 预防性维护：为了提高软件的可维护性和可靠性，为未来的进一步改进打下基础而修改软件的活动。

在整个软件维护活动中，改正性维护约占 20%，适应性维护约占 25%，完善性维护约占 50% 以上，其他维护约占 4%。

③可维护性软件：理解、改正、改动、改进软件的难易程度，有下面 3 个指标。

- Ⅰ 可理解性：别人能理解系统的结构、界面功能和内部过程的难易程度。模块化、详细设计文档、结构化设计和良好的高级程序设计语言都有助于提高可理解性。
- Ⅰ 可测试性：测试和诊断软件（主要指程序）中错误的难易程度。测试主要是发现软件中的错误，而诊断错误的性质和出错的位置通常是调试的任务。提高软件可测试性的措施有：书写详细正确的文档，采用良好的程序结构，使用测试工具和调试工具，保存以前的测试过程和测试用例等。
- Ⅰ 可修改性：修改软件（主要指程序）的难易程度。