一、简答题范围（30 分左右）

1. 简述软件的定义及特征（第 1 章）（P3）

软件是：（1）指令的集合（计算机程序），通过执行这些指令可以满足预期的特性、功能和性能需求；（2）数据结构，使得程序可以合理利用信息；（3）软件描述信息，它以硬拷贝和虚拟形式存在，用来描述程序的操作和使用。

2. 简述软件工程的定义(IEEE)及软件过程的 5 种框架活动(第 2 章）（P11）

软件工程是：（1）将系统化的、规范的、可量化的方法应用于软件的开发、运行和维护，即将工程化方法应用于软件；（2）对（1）中所述方法的研究。

（1）沟通(与客户沟通与协调，以理解项目目标）

（2）策划（工作、技术任务、风险、资源、产品，进度计划）

（3）建模（用模型来理解软件需求，完成设计）

- 需求分析
- 设计

（4）构建（编码、测试）

- 代码生成
- 测试

（5）部署（软件交付用户，用户测评并反馈）

3. 画图说明软件过程流的各种类型（第 3 章）（P24）

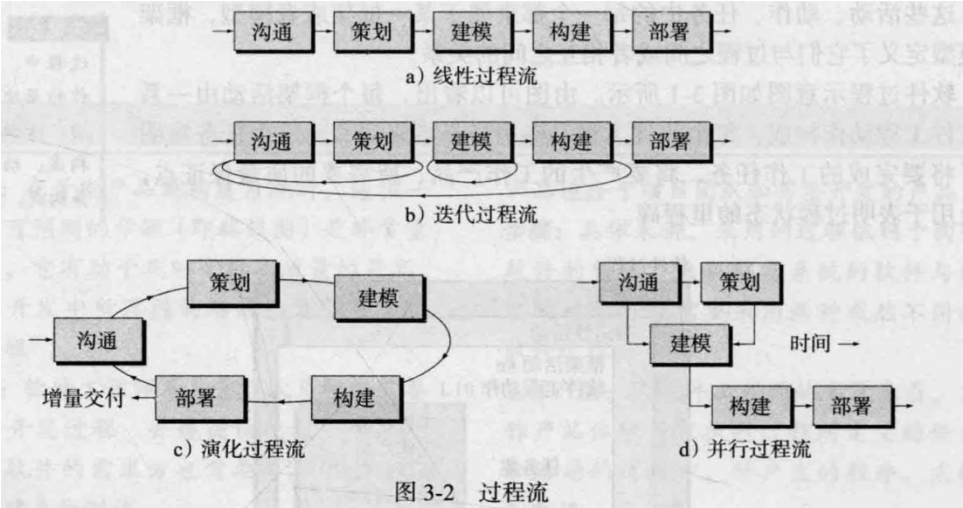


图 3-2　过程流

4. 画图说明软件过程的增量模型及适用情形和特点(第 4 章)(P32)

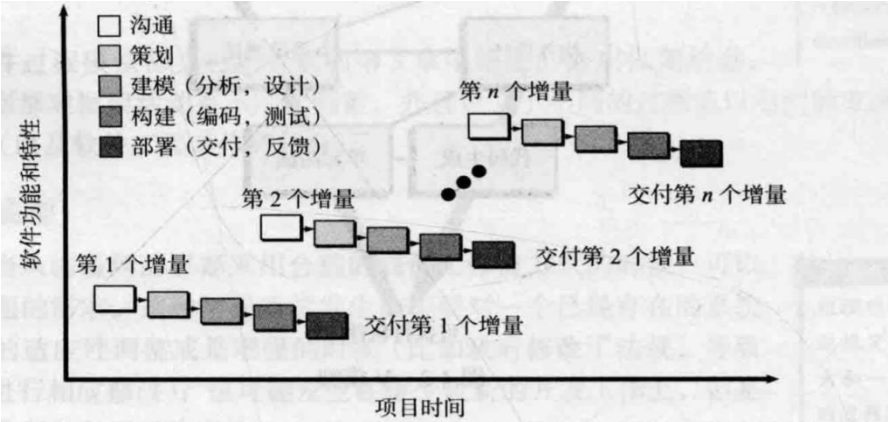
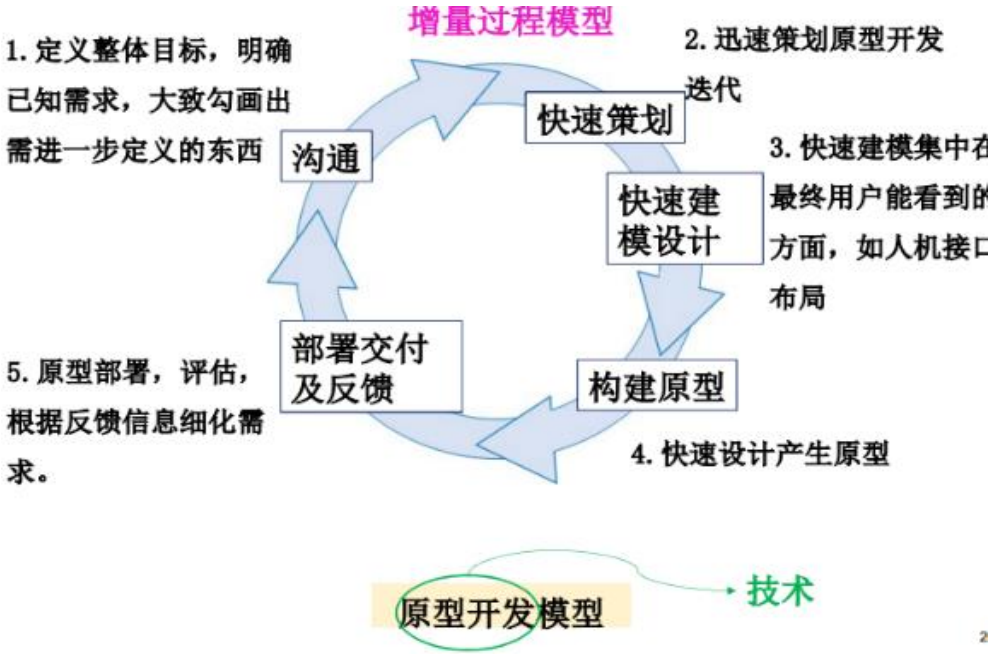


图 4-3　增量模型

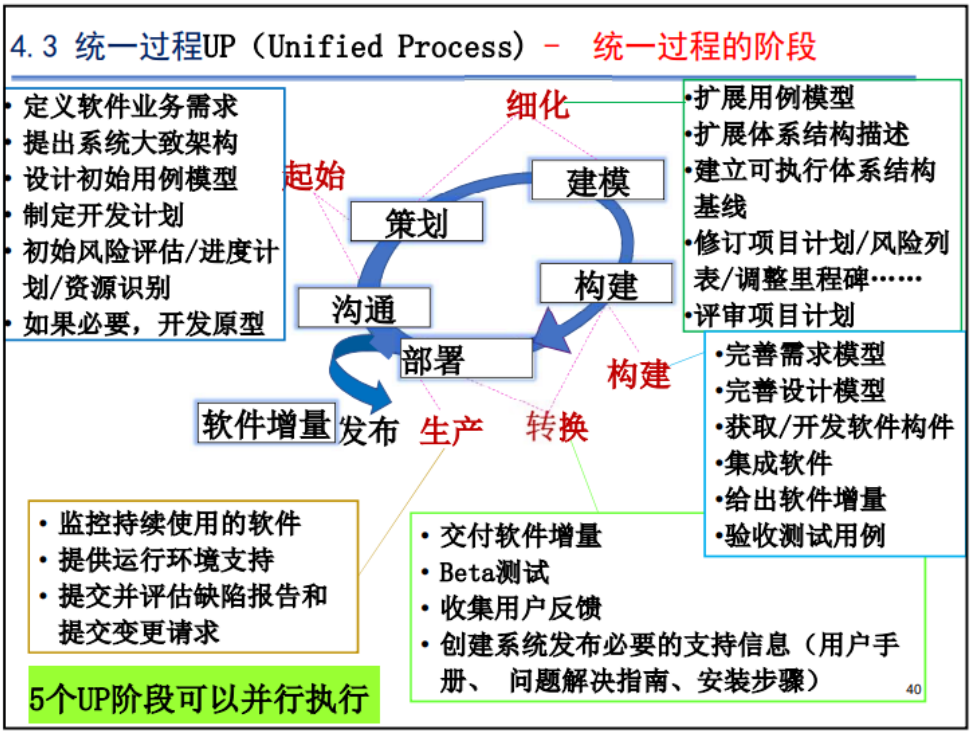
5. 画图说明软件过程的原型模型及适用情形和特点（第 4 章）（P33）



特点：1 很少是好用的，可能太慢太大，难以使用
2 一般作为被丢弃的系统
3 原型开发对于软件工程来说是一个有效的模型，是为定义需求服务的

适用情形：1 客户提出了一些基本功能，但没有详细定义功能和特性需求
2 开发人员可能对算法的效率、操作系统的兼容性和人机交互的形式等情况并不确定

6. 画图说明统一过程的各个阶段（第 4 章）（P41）



7. 简述敏捷原则（第 5 章）（P51）

1、 我们最优先要做的是通过尽早、持续交付有价值的软件来使客户满意。
2、 即使在开发的后期，也欢迎需求变更。敏捷过程利用变更为客户创造竞争优势。
3、 经常交付可运行软件，交付的间隔可以从几个星期到几个月，交付的时间间隔越短越好。
4、 在整个项目开发期间，业务人员与开发人员必须天天都在一起工作。
5、 围绕有积极性的个人构建项目。给他们提供所需的环境和支持，并且信任他们能够完成工作。
6、 在团队内部，最富有效果和效率的信息传递方法是面对面交谈。
7、 可运行软件是进度的首要度量标准。
8、 敏捷过程提倡可持续的开发速度。责任人、开发者和用户应该能够长期保持稳定的开发速度。
9、 不断地关注优秀的技能和好的设计会增强敏捷能力。
10、 简单——使不必做的工作最大化的艺术——是必要的。
11、 最好的架构、需求和设计出自于自组织团队。
12、 每隔一定时间，团队会反省如何才能更有效地工作，并相应调整自己的行为。

8. 简述需求建模的模型（第 8 章）（P122）

- 场景模型：出自各种系统“参与者”观点的需求。
- 面向类的模型：表示面向对象类（属性和操作）的模型，其方式为通过类的协作获得系统需求。
- 基于行为和模式的模型：描述如何将软件行为看作外部“事件”后续的模型。
- 数据模型：描述问题信息域的模型。
- 面向流的模型：表示系统的功能元素并且描述当功能元素在系统中运行时怎样进行

数据变换。

9. 简述 CRC 模型的评审步骤（第 9 章）（P143）

1、 **所有参加（CRC 模型）评审的人员拿到一部分 CRC 模型索引卡。**拆分协作卡片（也就是说每个评审员不得有两张存在协作关系的卡片）。

2、 分类管理所有的用例场景（以及相关的用例图）。

3、 **评审组长细致地阅读用例。**当评审组长看到一个已命名的对象时，给拥有相应类索引卡的人员一个令牌。

4、 **当令牌传递时，索引卡的拥有者需要描述卡上记录的职责。**评审组确定（一个或多个）职责是否满足用例需求。

5、 **如果记录在索引卡上的职责和协作不能满足用例，就需要修改卡片。**修改可能包括定义新类（和相关的 CRC 索引卡），或者在已有的卡上说明新的或修改的职责、协作。

10. 简述行为建模的步骤（第 10 章）（P149）

（1）评估所有的用例，以保证完全理解系统内的交互顺序。（2）识别驱动交互顺序的事件，并理解这些事件如何与特定的对象相互关联。（3）为每个用例生成序列。（4）创建系统状态图。（5）评审行为模型以验证准确性和一致性。

11. 画图说明从需求模型到设计模型的转换（第 11 章）（P164）

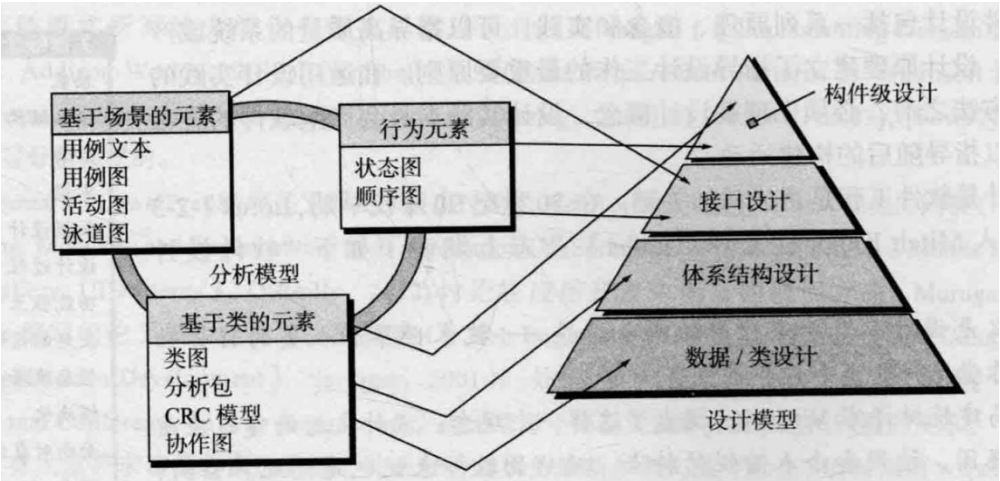


图 12-1 从需求模型到设计模型的转换

12. 简述模块的功能独立及评估标准（第 11 章）（P172）

功能独立的概念是关注点分离、模块化、抽象和信息屏蔽概念的直接产物。

通过开发具有“专一”功能和“避免”与其他模块过多的交互的模块，可以实现功能独立。每个模块仅涉及需求的某个特定的子集。

当从程序结构的其他部分观察时，每个模块只有一个简单接口

功能独立的重要性：

具有独立模块的软件更容易开发：功能分离且接口简单

独立模块更容易维护和测试：修改副作用小，错误不易扩散，模块可复用。

独立性有两条定性标准进行评估：**（1）内聚性显示了某个模块相关功能的强度。**一个内聚的模块执行一个独立的任务，与程序的其他部分构件只需要很少的交互。简单地说，一个内

聚的模块应该（理想情况下）只完成一件事情。**（2）耦合性显示了模块间的相互依赖性。**耦合性依赖于模块之间的接口复杂性、引用或进入模块所在的点以及什么数据通过接口进行传递。

13. 简述重构的定义及重构时的检查要点（第 11 章）（P173）

“重构是使用这样一种方式改变软件系统的过程：不改变代码（设计）的外部行为而是改进其内部结构。”

在重构软件时，检查现有设计的

1 冗余性、
2 没有使用的设计元素、
3 低效的或不必要的算法、
4 拙劣的或不恰当的数据结构、
5 其他设计不足，

修改这些不足以获得更好的设计。

14. 简述体系结构风格描述的 4 个要素及其分类（第 12 章）（P189）

每种风格描述一种系统类别，包括：（1）完成系统所需要的某种功能的一组构件（例如数据库、计算模块）；（2）能使构件间实现“通信、协调和合作”的一组连接件；（3）定义构件如何集成为系统的约束；（4）语义模型，能使设计者通过分析系统组成成分的已知属性来理解系统的整体性质。

分类：

- 以数据为中心的体系结构
- 数据流体系结构
- 调用和返回体系结构
- 面向对象体系结构
- 层次体系结构

15. 简述构件级设计的 7 个基本原则（第 13 章）（P212）

- 开闭原则(OCP)。“模块[构件]应该对外延具有开放性，对修改具有封闭性”。
- Liskov 替换原则(LSP)。“子类可以替换它们的基类”。
- 依赖倒置原则(DIP)。“依赖于抽象，而非具体实现”。
- 接口分离原则(ISP)。“多个客户专用接口比一个通用接口要好”。
- 发布复用等价性原则(REP)。“复用的粒度就是发布的粒度”。
- 共同封装原则(CCP)。“一同变更的类应该合在一起”。
- 共同复用原则(CRP).。“不能一起复用的类不能被分到一组”。

16. 简述黄金规则中把控制权交给用户的规则（第 14 章）（P231）

- 以不强迫用户进入不必要的或不希望的动作的方式来定义交互模式。
- 提供灵活的交互。
- 允许用户交互被中断和撤销。
- 当技能水平高时可以使交互流线化并允许定制交互。
- 使用户与内部技术细节隔离开来。
- 设计应允许用户与出现在屏幕上的对象直接交互。

17. 简述黄金规则中减轻用户记忆负担的原则（第 14 章）（P232）

- 减少对短期记忆的要求。
- 建立有意义的默认设置。
- 定义直观的快捷方式。
- 界面的视觉布局应该基于真实世界的象征。
- 以一种渐进的方式揭示信息。

18. 简述黄金规则中保持界面一致的原则（第 14 章）（P233）

- 允许用户将当前任务放入有意义的环境中。
- 在完整的产品线内保持一致性。
- 如果过去的交互模型已经建立起了用户期望，除非有不得已的理由，否则不要改变它。

19. 简述压力测试并举例说明（第 17 章）（P358）

以非正常的数量、频率或容量的方式执行系统。测试是想要破坏程序。

举例：

举例：（1）在平均每秒出现 1-2 次中断的情况下，可以设计每秒产生 10 次中断的测试用例；

（2）将输入数据的量提高到一个数量级以确定输入功能将如何反应；

（3）执行需要最大内存或其他资源的测试用例；

（4）设计可能在实际的运行系统中产生惨败的测试用例；

（5）创建可能会过多查找磁盘驻留数据的测试用例。

19.1 简述 00 测试中集成测试的两种策略（17 章）

自顶向下集成：自顶向下集成测试是一种构建软件体系结构的增量方法。模块的集成顺序为从主控模块（主程序）开始，沿着控制层次逐步向下，以深度优先或广度优先的方式将从属于（和间接从属于）主控模块的模块集成到结构中去。

自底向上集成：自底向上集成测试是从原子模块（程序结构的最底层构件）开始进行构建和测试。由于构件是自底向上集成的，在处理时所需要的从属于给定层次的模块总是存在，因此，没有必要使用桩模块。

20. 简述单元测试中桩模块和驱动模块的作用？（第 17 章）（P350）

驱动程序是一个“主程序”，它接收测试用例数据，将这些数据传递给（将要测试的）构件，并打印相关结果。桩程序的作用是替换那些从属于被测构件（或被其调用）的模块。桩程序或“伪程序”使用从属模块的接口，尽可能做少量的数据操作，提供入口的操作，并将控制返回到被测模块。

21. 简述测试中症状与原因之间的关系（第 17 章）

1、 症状与原因出现的地方可能相隔很远。也就是说，症状可能在程序的一个地方出现，而原因实际上可能在很远的另一个地方。高度耦合的构件加剧了这种情况的发生；

2、 症状可能在另一个错误被改正时（暂时）消失；

3、 症状实际上可能是由非错误因素（例如舍入误差）引起的；

4、 症状可能是由不易追踪的人为错误引起的；

5、 症状可能是由计时问题而不是处理问题引起的；

6、 重新产生完全一样的输入条件是困难的（如输入顺序不确定的实时应用）；

7、 症状可能时有时无，这在软硬件耦合的嵌入式系统中尤为常见；

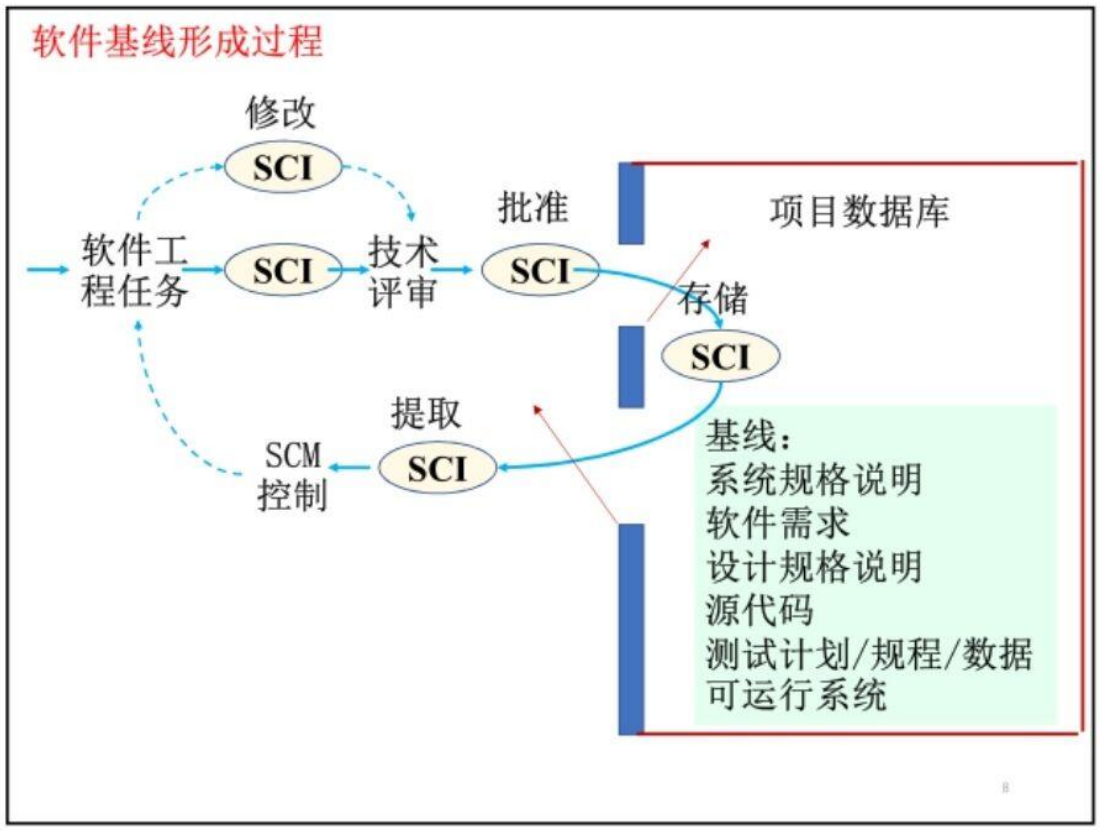
8、 症状可能是由分布运行在不同处理器上的很多任务引起的。

22. 简述软件基线及 SCI 和项目数据库并画图说明（第 21 章）（P463）

基线是一个软件配置管理概念，它能够帮助我们在不严重阻碍合理变更的条件下控制变更。IEEE 是这样定义基线的：已经通过正式评审和批准的规格说明或产品，它可以作为进一步开发的基础，并且只有通过正式的变更控制规程才能修改它。

SCI：软件配置项是在软件工程过程中创建的信息。在现实中，是将 SCI 组织成配置对象，这些配置对象具有自己的名字，并且按类别存储在项目数据库中，通过关系来表示与其他配置对象的关联。

项目数据库(中心存储库)：存储所有相关配置对象



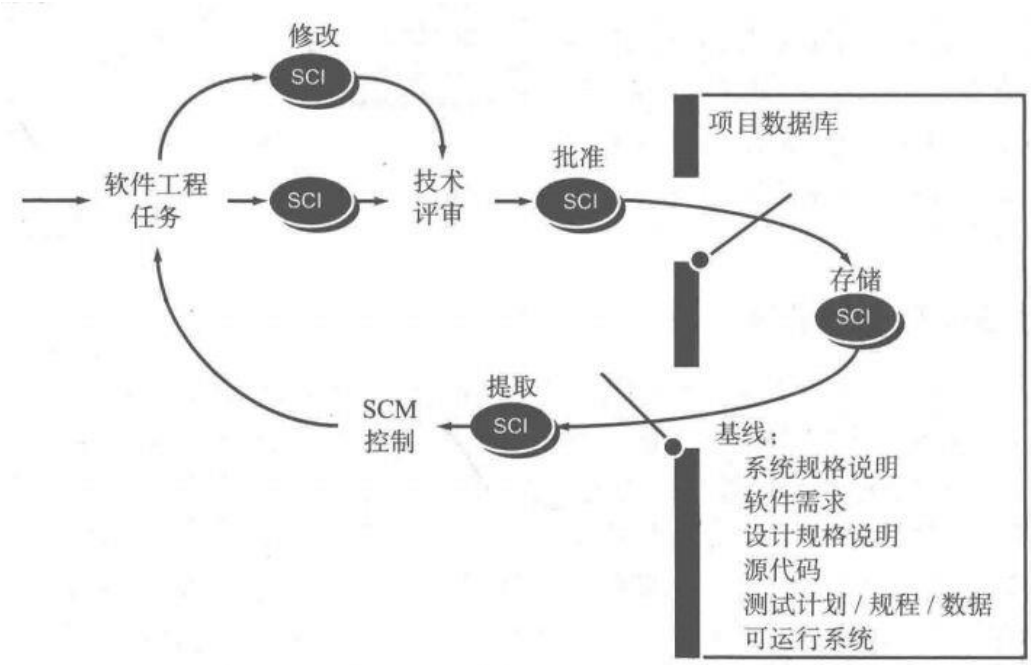


图 21-1　基线化的 SCI 和项目数据库

23. 简述选择软件团队结构时应考虑的 7 个因素（第 22 章）（P510）

- 待解决问题的难度
- 开发程序的规模 ，以代码行或者功能点来度量
- 团队成员需要共同工作的时间 (团队生存期)
- 能够对问题做模块化划分的程度
- 待开发系统的质量要求和可靠性要求
- 交付日期的严格程度
- 项目所需要的友好交流的程度

24. 简述软件团队的组织范型（第 22 章）（P510）

- 封闭式范型 ——按照传统的权利层次来组织团队
  1 个高级工程师（主程序员），2-5 个技术人员，1 个后备工程师
- 随机式范型 ——松散地组织团队，团队工作依赖于团队成员个人的主动性
- 开放式范型 ——试图以一种既具有封闭式范型的控制性，又包含随机式范型的创新性的方式来组织团队
- 同步式范型——依赖于问题的自然划分，组织团队成员各自解决问题的一部分，他们之间没有什么主动的交流

25. 简述如何避免“团队毒性”（第 22 章）

- 项目经理应确保团队可获取完成工作所需的所有信息；主要目标一旦确定（除非绝对必要），就不应该修改
- 给予软件团队尽可能多的决策权。
- 理解将要开发的产品和完成工作的人员，以及允许团队选择过程模型。
- 团队本身应该建立自己的责任机制（技术评审），并规定一系列当团队未能

完成任务时的纠正方法。

- 建立基于团队的信息反馈方法和解决问题的技术。

二、综合题题型（30 分左右）

1. 用例图、类图、状态图、活动图（泳道图）
2. 白盒测试（计算环复杂度、列出具体的独立路径，可从代码或流程图画出流图、也可从流图画出流程图）
3. 黑盒测试（用表列出有效等价类和无效等价类，设计测试用例和预期结果）一般与边界法相结合
4. 过程度量与项目度量、软件项目估算（代码行，功能点等）。见 23 章、24 章
5. 综合题亦可改变为简答题

三、填空、选择、判断（40 分左右）（填空来自第 8 版 PPT）

填空 10 分-5 道题，选择 20 分-10 道题，判断 10 分-10 道题