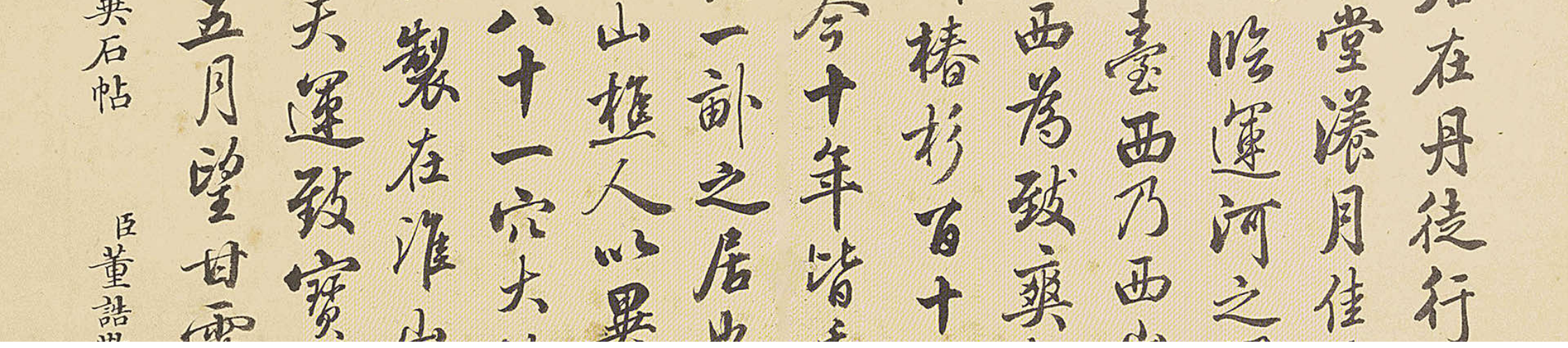


软件工程复习

填空题

1. 三个基础的软件过程模型，他们是什么？

   Waterfall（瀑布模型），Evolutionary development（进化发展），Component-based software engineering（CBSE，基于组件的软件工程）

2. 软件工程区别于其他工程的特点

   软件工程与系统工程的区别

   - System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this process concerned with developing the software infrastructure, control, applications and databases in the system.

     系统工程涉及基于计算机的系统开发的所有方面，包括硬件，软件和过程工程。软件工程是此过程的一部分，涉及开发系统中的软件基础结构，控制，应用程序和数据库。

   软件工程与计算机科学的区别

   - Computer science is concerned with theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.

     计算机科学与理论和基础有关；软件工程与开发和交付有用软件的实用性有关。

3. risk 的两个维度：（发生概率和严重性）

4. 三个 risk planning：

   - 规避策略：Avoidance strategies

     The probability that the risk will arise is reduced

     采用这些策略就会降低风险出现的可能性。

   - 最低风险策略：Minimisation strategies

     The impact of the risk on the project or product will be reduced

     采用这些策略就会减小风险的影响

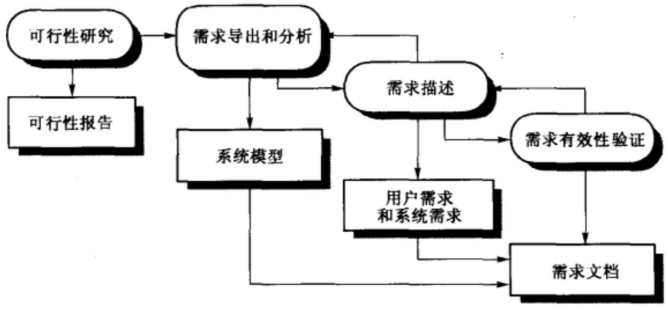
   - 应急计划：Contingency plans

     If the risk arises, contingency plans are plans to deal with that risk

     采用这些策略，就算最坏的事情发生，我们也是有备而来

5. 需求工程过程的活动：

   - Feasibility study 可行性研究
   - Requirements elicitation and analysis 需求导出和分析
   - Requirements specification 需求描述
   - Requirements validation 需求有效性验证
   - Requirements management 需求管理



6. 所有软件产品的重要属性：

   The software should deliver the required functionality and performance to the user and should be maintainable, dependable and acceptable.

   软件应向用户提供所需的功能和性能，并应可维护、可靠和可接受。

   - Maintainability

     Software must evolve to meet changing needs;

   - Dependability

     Software must be trustworthy;

   - Efficiency

     Software should not make wasteful use of system resources;

   - Acceptability

     Software must accepted by the users for which it was designed. This means it must be understandable, usable and compatible with other systems.

7. 软件是什么?

   - Computer programs and associated documentation such as requirements, design models and user manuals.

     计算机程序和相关文档，如要求、设计模型和用户手册。

   - Software products may be developed for a particular customer or may be developed for a general market.

     软件产品可以为特定客户开发，也可以为一般市场开发。

   - Software products may be：

     - Generic – developed to be sold to a range of different customers
       e.g. PC software such as Excel or Word.
     - Bespoke (custom) – developed for a single customer according
       to their specification.

   - New software can be created by developing new programs, configuring generic software systems or reusing existing software.

     可以通过开发新程序，配置通用软件系统或重用现有软件来创建新软件。

8. 测试的两种分类？分别是做什么的?

   - Component testing 组件测试
     - Testing of individual program components;
     - Usually the responsibility of the component developer
       (except sometimes for critical systems);
     - Tests are derived from the developer's experience.
   - System testing 系统测试
     - Testing of groups of components integrated to create a system or sub–system;
     - The responsibility of an independent testing team;
     - Tests are based on a system specification.

9. CASE 工具支持的方法/功能（六七种）

   Graphical editors for system model development

需求描述或软件设计的图形化系统模型开发

Data dictionary to manage design entities

通过数据字典了解一个设计，数据字典包含了关于设计中实体和关系的信息

Graphical UI builder for user interface construction

通过用户交互式生成的图形化界面描述产生用户界面

Debuggers to support program fault finding

通过提供执行程序的信息来调试程序

Automated translators to generate new versions of a program

程序的自动转换，例如，从旧的 COBOL 语言程序转换成一个新的程序

10. 组件接口有哪些?

    - Provides interface 提供接口

      Defines the services that are provided by the component to other components.

      定义组件向其他组件提供的服务。

    - Requires interface 需求接口

      Defines the services that specifies what services must be made available for the component to execute as specified.

      定义服务，该服务指定组件必须提供哪些服务才能按指定执行。

11. 再工程（Re-engineering）是什么?

    - Re-structuring or re-writing part or all of a legacy system without changing its functionality.
    - Applicable where some but not all sub-systems of a larger system require frequent maintenance.
    - Re-engineering involves adding effort to make them easier to maintain. The system may be re-structured and re-documented.

12. 项目管理（deliverable、milestone）

    - 里程碑：milestone

      一个里程碑就是一项软件过程活动的终结。每个里程碑应该有一个正式的可以提交给管理层的输出结果。不确定的里程碑是没有意义的

    - 可交付的文档：deliverable

      可交付的文档是交付给用户的项目成果，可交付的文档也是里程碑，但是里程碑不需要交付。

13. corba 标准

    - An object model for application objects

      - A CORBA object is an encapsulation of state with a well-defined, language-neutral interface defined in an IDL (interface definition language).

      应用对象模型，其中一个 CORBA 对象实现了对状态的一个封装，对象具有语言无关的接口，这些接口用 IDL 语言（接口定义语言）来描述

    - An object request broker that manages requests for object services.

      对象请求代理（ORB），负责对象服务请求的管理。由 ORB 来定位提供服务的对象、准备对象的服务、发送服务请求并向请求服务的对象返回结果。

    - A set of general object services of use to many distributed applications.

      一组通用的对象服务，这些服务是许多分布式应用普遍需要的。服务的例子有目录服务、交易服务和持久服务。

    - A set of common components built on top of these services.

      一组通用组件，这些组件是建立在应用需求的基本服务之上的。这些组件可能是垂直领域的相关组件或水平的适用于许多应用的通用组件。

14. 人的管理，人的三种类型（为生存而工作、为兴趣而工作、为沟通而工作）

    - 关键要素
        1. 一致性
        2. 尊重
        3. 包容
        4. 诚实
    - 工作动力
        1. 满足员工的社会需求，就是给员工提供与同时交往的时间和场所
        2. 满足员工的受尊重需要
        3. 满足员工的自我实现需求
    - 人的三种类型
        - 面向任务型：Task-oriented

          The motivation for doing the work is the work itself

          这类专业人员的动力来自于他们所从事的工作。在软件工程中，他们是技术人员，软件开发智力上的挑战激发了他们的工作热情。
        - 面向自我型：Self-oriented

          The work is a means to an end which is the achievement of individual goals – e.g. to get rich, to play tennis, to travel etc.

          这类人的动力主要来自个人成功和社会认同。他们更乐于把软件开发视为自己达到目的的手段
        - 面向交互型：Interaction-oriented

          The principal motivation is the presence and actions of co-workers. People go to work because they like to go to work

          这类人的动力来自于同事的存在和行动

15. 基于复用/组件的开发是基于什么样的需求而采用的

    - achieve better software, more quickly and at lower cost

      降低软件产品和维护的成本、更快的系统移交以及提高软件质量的要求

16. 进化式开发（evolutionary development）

    - Exploratory development（探索式开发）

      Objective is to work with customers and to evolve a final system from an initial outline specification. Should start with well-understood requirements and add new features as proposed by the customer.

      目标是与客户合作，并从最初的大纲规范发展最终系统。应从良好理解的需求开始，并添加客户提出的新功能。
    - Throw-away prototyping（抛弃式原型）

      Objective is to understand the system requirements. Should start with poorly understood requirements to clarify what is really needed.

      目标是了解系统要求。应从不甚了解的需求开始，以明确真正需要的是什么。

17. （补充资料）传统结构化分析方法的主要模型

    - Data modelling
    - Functional modelling
    - Behavior modelling

18. 面向对象分析方法的主要模型

    - 分为两大类：静态模型和动态模型

- 静态模型

  Static models describe the static structure of the system in terms of object classes and relationships.

  通过系统对象类及其之间的关系来描述系统的静态结构。

- 动态模型

  Dynamic models describe the dynamic interactions between objects

  描述系统的动态结构和系统对象（不是对象类）之间的交互

实例：

子系统模型：

Sub-system models that show logical groupings of objects into coherent subsystems.

说明对象的逻辑分组，每个分组构成一个子系统。使用类的图表格式来表示，每个子系统以一个包的形式存在。子系统是静态模型

序列模型：

Sequence models that show the sequence of object interactions.

说明对象的交互序列。使用 UML 序列图或协作图表示。序列模型是动态模型

状态机模型：

State machine models that show how individual objects change their state in response to events.

说明单个对象如何相应事件来改变他们的状态。使用 UML 的状态图来表示。状态机模型是动态模型

19. 迭代过程模型（区别于 waterfall model 和 evolutionary model）

System requirements ALWAYS evolve in the course of a project so process iteration where earlier stages are reworked is always part of the process for large systems.

在一个项目的过程中，系统需求总是（ALWAYS）在不断变化（evolve）的，所以对于大型系统来说，流程迭代，即对早期阶段进行重做，总是流程的一部分

两种相关的方法

渐进式交付（Incremental delivery）– 螺旋式发展（Spiral development）。

Incremental delivery：

Rather than deliver the system as a single delivery,the development and delivery is broken down into increments with each increment delivering part of the required functionality.

User requirements are prioritised and the highest priority requirements are included in early increments.

Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.

系统的开发和交付不是作为一个单一的交付，而是分成若干个增量（increment），每个增量提供部分所需的功能。
用户需求被优先考虑（prioritised），最高优先级的需求被包含在早期的增量中。
一旦开始开发一个增量，需求就会被冻结，尽管后面的增量的需求可以继续发展。

优势：

- Customer value can be delivered with each increment so system functionality is available earlier.

  客户的价值可以通过每一个增量来实现，所以系统功能可以更早的实现。

- Early increments act as a prototype to help elicit requirements for later increments.

  早期的增量作为一个原型（prototype），有助于为以后的增量引出（elicit）需求。

- Lower risk of overall project failure.

  降低整个项目失败的风险。

- The highest priority system services tend to receive the most testing.

最优先的系统服务往往会得到最多测试。

Spiral development：

- Process is represented as a spiral rather than as a sequence of activities with backtracking.

  流程被表示为一个螺旋，而不是一连串的活动与回溯（backtracking）。

- Each loop in the spiral represents a phase in the process.

  螺旋中的每个循环都代表流程中的一个阶段。

- No fixed phases such as specification or design – loops in the spiral are chosen depending on what is required.

  没有固定的阶段，如规格或设计--螺旋中的循环是根据所需的东西来选择的。

- Risks are explicitly assessed and resolved throughout the process.

  风险在整个过程中被明确地评估和解决。

20.validation 和 verification 的区别

- Validation 有效性验证

  The software should do what the user really requires.

- Verification 检验

  The software should conform to its specification.

21.P-CMM 五级模型

人员管理那章 25 People Capability Maturity Model 人员能力成熟度模型

Five stage model
– Initial. Ad-hoc people management
Repeatable. Policies developed for capability improvement
Defined. Standardised people management across the organisation
Managed. Quantitative goals for people management in place
Optimizing. Continuous focus on improving individual competence and workforce motivation

初始级——特定的、非正规的人员管理实践

可重复级——制定政策，开发员工的能力

已定义级——使得整个机构的成功的人员管理实践标准化

已管理级——制定并引入量化了的人员管理目标

优化级——时刻注意提高个人能力和端正工作动机

判断题

名词解释

1. nonfunctional requirement/functional requirement

   - Functional requirement

     Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.

     包括对系统应该提供的服务、如何对输入做出反应以及系统在特定条件下的行为的描述。在某些情况下，功能需求可能还需声明系统不应该做什么。

     - Describe functionality or system services.

       描述功能或系统服务。

     - Depend on the type of software, expected users and the type of system where the software is used.

       取决于软件类型、预期用户和开发的系统类型。

- Functional user requirements may be high-level statements of what the system should do but functional system requirements should describe the system services in detail.

  功能用户需求可能是系统应该做什么的高级陈述，但功能系统需求应该详细描述系统服务。

- Functional requirement

  Statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations.

  包括对系统应该提供的服务、如何对输入做出反应以及系统在特定条件下的行为的描述。在某些情况下，功能需求可能还需声明系统不应该做什么。

  - Describe functionality or system services.

    描述功能或系统服务。

  - Depend on the type of software, expected users and the type of system where the software is used.

    取决于软件类型、预期用户和开发的系统类型。

  - Functional user requirements may be high-level statements of what the system should do but functional system requirements should describe the system services in detail.

    功能用户需求可能是系统应该做什么的高级陈述，但功能系统需求应该详细描述系统服务。

  - Non-functional requirement

    Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.

    对系统提供的服务或功能给出的约束。包括时间约束、开发过程的约束、标准等。非功能需求常用于整个系统。通常不用在单个系统或服务中。

    - These define system properties and constraints.

      定义了系统的属性和约束

    - Process requirements may also be specified mandating a particular CASE system, programming language or development method.

      过程要求也可以指定为特定的 CASE 系统、编程语言或开发方法。

    - Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless.

      非功能性需求可能比功能性需求更为关键。如果不满足这些条件，系统将毫无用处。

2. 软件产品线（产品线的四种实例化方式）

   - Platform specialisation 平台特化

     Different versions of the application are developed for different platforms.

     为不同的平台开发应用程序的不同版本

   - Environment specialisation 环境特化

     Different versions of the application are created to handle different operating environments e.g. different types of communication equipment.

     创建应用的版本来处理特殊的操作环境和外部设备

   - Functional specialisation 功能特化

     Different versions of the application are created for customers with different requirements.

     为不同需求的客户创建不同的应用程序版本

   - Process specialisation 过程特化

     Different versions of the application are created to support different business processes.

     调整系统使之与特殊的业务过程配套

3. 设计模式（design pattern）

   - A design pattern is a way of reusing abstract knowledge about a problem and its solution.

设计模式是一种重用关于问题及其解决方案的抽象知识的方法。

- A pattern is a description of the problem and the essence of its solution.

  模式是对问题和解决方案的基本内容的描述

- It should be sufficiently abstract to be reused in different settings.

  它应该足够抽象以便在不同的环境中重用。

- Patterns often rely on object characteristics such as inheritance and polymorphism.

  模式通常依赖于对象特性（如继承和多态）来支持它的通性。

- 设计模式的四个主要元素：
  - 名字：是模式的有意义的指代
  - 问题描述：解释什么时候模式可以应用
  - 解决方案描述：描述设计方案的各个部分及其之间的关系和职责。它不是一个具体的设计描述，而是一个设计方案的模板，可以用不同的方式实例化。
  - 结果陈述：说明应用该模式的结果和折衷。

4. incremental development

- 软件描述、设计和实现活动被分散成一系列的增量，然后轮流开发这些增量
  - 增量开发过程中，客户大概地提出系统需要提供的服务，指明哪些服务是最重要的，哪些是最不重要的。
  - 此时，一系列交付增量被确定，每个增量提供系统功能的一个子集。对增量中服务的分配取决于服务的优先次序。最高优先权的服务首先被交付。
  - 开发时，为稍后的增量准备的需求分析不断进行，但对目前增量的需求变更不会被接受

5. 分布对象体系结构（corba）

- CORBA is an international standard for an Object Request Broker – middleware to manage communications between distributed objects.

  CORBA 是对象请求代理中间件的国际标准，用于管理分布式对象之间的通信。

- Middleware for distributed computing is required at 2 levels:

  对支持分布式对象计算的中间件，有两个层次上的需要：

  - At the logical communication level, the middleware allows objects on different computers to exchange data and control information;

    在逻辑通信层，中间件允许不同计算机上的对象交换数据和控制信息；

  - At the component level, the middleware provides a basis for developing compatible components. CORBA component standards have been defined.

    在组件层次上，中间件为开发兼容组件提供了基础。CORBA 组件标准已经定义。

6. C/S 结构

- Distributed system model which shows how data and processing is distributed across a range of components.

  分布式系统模型，显示数据和处理如何在一系列组件中分布。

- Set of stand-alone servers which provide specific services such as printing, data management, etc.

  一组独立的服务器，提供特定的服务，如打印、数据管理等。

- Set of clients which call on these services.

  调用这些服务的客户端集。

- Network which allows clients to access servers.

  允许客户端访问服务器的网络。

- The application is modelled as a set of services that are provided by servers and a set of clients that use these services.

应用程序被建模为一组由服务器提供的服务和一组使用这些服务的客户端。

- Clients know of servers but servers need not know of clients.

  客户端知道服务器，但服务器不需要知道客户端。

- Clients and servers are logical processes.

  服务器和客户机是指逻辑的进程而不是指物理计算机

- The mapping of processors to processes is not necessarily 1 : 1.

  进程和处理机之间没有必要非得一对一地映射

7. 软件审查时的 checklist

   - Checklist of common errors should be used to drive the inspection.
   - Error checklists are programming language dependent and reflect the characteristic errors that are likely to arise in the language.
   - In general, the 'weaker' the type checking, the larger the checklist.

8. UML sequence diagram

   这些显示了用户与系统互动过程中发生的事件的顺序。

   你从上到下阅读它们，可以看到所发生的动作的顺序。

   从 ATM 机上提取现金

   - 验证卡。
   - 处理请求。
   - 完成交易

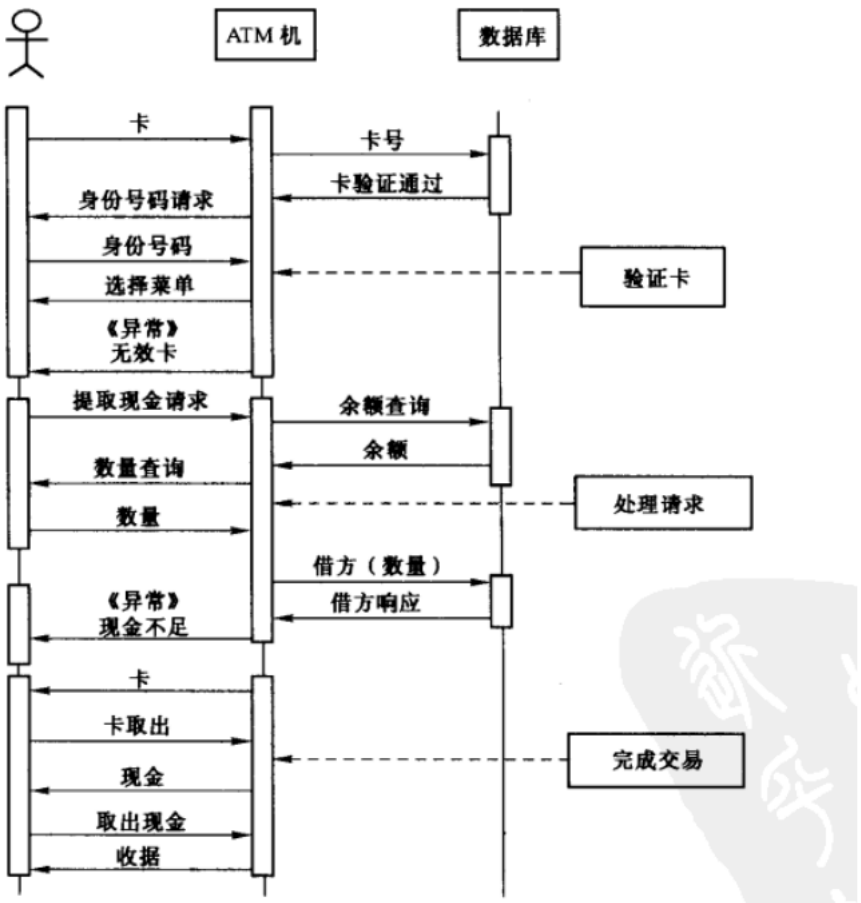


图 6-10　ATM 机上提款的序列图

9. 组件模型中，中间件提供的几种基础服务

   - 中间件为组件集成提供软件支持。对资源分配、事务管理、信息安全及并发的支持。

10. legacy system

    - Socio–technical systems that have been developed using old or obsolete technology.

      使用旧技术或不再使用的技术开发的社会–技术系统。

    - Crucial to the operation of a business and it is often too risky to discard these systems.

      对企业运营至关重要，丢弃这些系统往往风险太大

    - Legacy systems constrain new business processes and consume a high proportion of company budgets.

      遗留系统限制了新的业务流程，并消耗了很高比例的公司预算。

11. equivalence partition（等价类划分）
12. 黑盒测试和白盒测试
13. independent path

    Test cases should be derived so that all of these paths are executed

    应派生测试用例，以便执行所有这些路径

    A dynamic program analyser may be used to check that paths have been executed

    动态程序分析器可用于检查路径是否已执行

14. 对象和对象类的差别

    - Objects are entities in a software system which represent instances of real-world and system entities.

      对象是软件系统中的实体，代表现实世界和系统实体的实例。

    - Object classes are templates for objects. They may be used to create objects.

      对象类是对象的模板。它们可用于创建对象。

    - Object classes may inherit attributes and services from other object classes.

      对象类可以从其他对象类继承属性和服务。

15. re-engineering

    - Re-structuring or re-writing part or all of a legacy system without changing its functionality.
    - Applicable where some but not all sub-systems of a larger system require frequent maintenance.
    - Re-engineering involves adding effort to make them easier to maintain. The system may be re-structured and re-documented.

16. MVC model

    - MVC model
      - （书 P97）体系结构模式
      - 名词解释：M：模型、V：视图、C：控制器

名字	MVC（模型－视图－控制器）
描述	将表示和交互从系统数据中分离出来。系统被设计成由 3 个彼此交互的逻辑组件组成：模型组件管理系统数据和在数据上的操作，视图组件定义和管理如何显示数据给用户，控制器组件管理用户的交互（例如，键按下，鼠标点击等），并传递这些交互给视图和模型。参见图 6-3
实例	图 6-4 说明了采用 MVC 模式的基于 Web 的应用系统的体系结构
使用时机	在数据有多个显示和交互方式的时候使用。也可在对未来数据的交互和表示需求不明朗的时候使用
优点	允许数据独立地改变，不影响表示，反之亦然。支持对相同数据的多种不同方式的表达，对某种表示的变更会传递到所有其他的表示
缺点	可能需要额外的代码，当数据模型和交互很简单时代码的复杂度相对较高

        图 6-2　模型－视图－控制器（MVC）模式

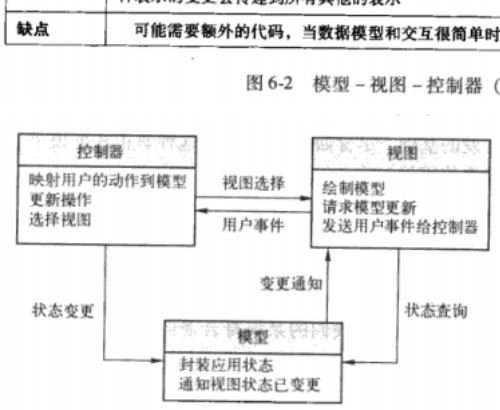
        

        图 6-3　MVC 模式的组成

      - MVC 模式分离了系统的组成元素，允许他们独立变更
      - 模式是系统的概念支撑
      - MVC 模式具有分离性和独立性，分离性和独立性的概念是体系结构设计的基础，因为分离性和独立性使得变更得到局部化

设计题

第八章

考设计图、状态图 画两个图

书上的微波炉状态图、有可能考个洗衣机

问答题

1. 瀑布模型的优势和劣势

   优势：它在每个阶段都生成文档，而且它与其他工程过程模型相一致。

   劣势：它将项目生硬地分解成这些确切的阶段。委托事项一定要在过程的早期阶段清晰给出，而这意味着它难以相应用户需求的变更。

2. 传统结构化设计方法四要素以及至今仍在使用的原因

3. 螺旋式迭代模型（理解该模型嵌入其他模型的过程）

   - Process is represented as a spiral rather than as a sequence of activities with backtracking

     过程表示为螺旋，而不是回溯的一系列活动。

   - Each loop in the spiral represents a phase in the process.

     螺旋中的每个循环代表过程中的一个阶段。

   - No fixed phases such as specification or design – loops in the spiral are chosen depending on what is required.

     根据需要，不会选择固定的阶段，例如规格或螺旋中的设计循环。

   - 每个螺旋线中每个回路被分成四个部分：

     - Objective setting 目标设定

       Specific objectives for the phase are identified.

       确定该阶段的具体目标。

     - Risk assessment and reduction 风险评估和规避

       Risks are assessed and activities put in place to reduce the key risks.

       对风险进行评估，并开展活动以减少关键风险。

     - Development and validation 开发和有效性验证

       A development model for the system is chosen which can be any of the generic models.

       在风险预估以后，就可以为系统选择开发模型。

       举例来说，如果用户界面风险时主要的，一个适当的开发模型可能是建立进化式原型。如果安全风险时主要的，则基于形式化转换的开发可能就是适当的。如果主要风险在于子系统集成，那么瀑布模型可能是最适当的。

     - Planning 规划

       The project is reviewed and the next phase of the spiral is planned.

       对项目进行评审以确定是否需要进入螺旋线的下一个回路。如果决定继续，则要做出项目的下个阶段计划。

4. 对软件复用、基于组件式的开发的理解（十八、十九章）

   - In most engineering disciplines, systems are designed by composing existing components that have been used in other systems.

     在大多数工程学科中，系统是通过组合已在其他系统中使用的现有组件来设计的。

   - Software engineering has been more focused on original development but it is now recognised that to achieve better software, more quickly and at lower cost, we need to adopt a design process that is based on systematic software reuse.

     软件工程更注重原始开发，但现在人们认识到，为了更快、更低地实现更好的软件，我们需要采用基于系统软件重用的设计过程。

5. clean-room 方法（没讲过，自己在书上找）（22 章）

净室过程是最著名的形式化开发过程。净室过程中，每一个软件增量都要形式化描述，然后此描述经过变换得以实现。软件的正确性通过形式化方法证明。在开发过程中不存在缺陷测试，而系统测试的重心在于评估系统的可靠性。

净室方法和另外一个基于 B 方法的形式化开发方法特别适合对安全性、可靠性或信息安全性需求极高的系统开发。形式化方法简化了安全或信息安全性案例的开发，对于这样的系统，采用别的开发方法就需要向客户或认证机构提供充分的素材来说明系统确实是符合安全或信息安全需求的。

软件开发的净室方法基于 5 个关键策略：

- 形式化描述
- 增量式开发
- 结构化程序设计
- 静态检验
- 系统的统计性测试

6. 提出一个过程模型以应对 emergency change

   进化式开发

7. 紧急变更会带来工程上什么样的问题

   - 增加成本
   - 延迟系统交付

8. 中间件在分布式体系结构里面提供哪些支持和服务

   - Transaction processing monitors;
   - Data converters;
   - Communication controllers.

9. 变更管理过程中，软件 CASE 工具提供哪六个方面的支持

   - Form editor to support processing the change request forms;
   - Workflow system to define who does what and to automate information transfer;
   - Change database that manages change proposals and is linked to a VM system;
   - Change reporting system that generates management reports on the status of change requests

10. 软件复用的问题（好处、问题）

    好处：

    - Increased dependability 增加可靠性

      Reused software, that has been tried and tested in working systems, should be more dependable than new software. The initial use of the software reveals any design and implementation faults. These are then fixed, thus reducing the number of failures when the software is reused.

      在运行着的系统中重复使用的组件要比一个新组件的可靠性高。它们可能已经得到了各种环境的实际使用和测试。在最初的组件使用中，组件在设计和实现上的缺陷都已经暴露出来了，并得到了修正，因而组件在复用时的失败次数下降。

    - Reduced process risk 降低的过程风险

      If software exists, there is less uncertainty in the costs of reusing that software than in the costs of development. This is an important factor for project management as it reduces the margin of error in project cost estimation. This is particularly true when relatively large software components such as sub–systems are reused.

      如果组件已经存在，则复用组件的成本不确定性较之开发组件的成本不确定性减少，这是项目管理当中的一项重要因素，以为这降低了项目成本估计当中的不确定性。相对来讲，大规模的组件如子系统复用时这一点更明显。

    - Effective use of specialists 专家的有效使用

      Instead of application specialists doing the same work on different projects, these specialists can develop reusable software that encapsulate their knowledge.

不同的专家在不同的项目中做重复的工作，而是让他们开发可复用的组件，用这些组件来封装他们的知识。

- Standards compliance 与标准的兼容

  Some standards, such as user interface standards, can be implemented as a set of standard reusable components. For example, if menus in a user interfaces are implemented using reusable components, all applications present the same menu formats to users. The use of standard user interfaces improves dependability as users are less likely to make mistakes when presented with a familiar interface.

  有些标准，如用户界面标准，可以实现为一组标准组件。例如，可以将用户界面中的菜单实现为一个可复用组件。所有的应用在用户面前呈现相同的菜单格式。使用标准用户界面增进了可靠性，因为用户面对相同的用户界面，出现错误的可能性较低。

- Accelerated development 加快开发速度

  Bringing a system to market as early as possible is o ften more important than overall development costs. Reusing software can speed up system production because both development and validation time should be reduced.

  尽快让系统走向市场要比总开发成本更重要。复用组件加速了系统产品的形成，因为无论开发还是验证时间都缩短了。

问题

- Increased maintenance costs 增加的维护成本

  If the source code of a reused software system or component is not available then maintenance costs may be increased as the reused elements of the system may become increasingly incompatible with system changes.

  如果组件的源代码是不可得的，那么维护成本就要增加，因为随着系统的变更，系统中复用部分的不兼容性在增加。

- Lack of tool support 缺乏工具支持

  CASE toolsets may not support development with reuse. It may be difficult or impossible to integrate these tools with a component library system. The software process assumed by these tools may not take reuse into account.

  CASE 工具集没有对复用的支持，将这些工具与组件库系统集成是困难甚至不可能的。由这些工具承担的软件过程没有将复用考虑在内。

- Not-invented-here syndrome 孤芳自赏

  Some software engineers sometimes prefer to re-write components as they believe that they can improve on the reusable component. This is partly to do with trust and partly to do with the fact that writing original software is s een as more challenging than reusing other people's software.

  有些软件人员时常更愿意重新写一个组件，因为他们相信自己能改善可复用组件。这一方面是源于信任因素，另一方面重写软件更具挑战性。

- Creating and maintaining a component library 维护一个组件库

  Populating a reusable component library and ensuring the software developers can use this library can be expensive. Our current techniques for classifying, cataloguing and retrieving software components are immature.

  填充组件库并保证软件人员能用上这个库是很费钱的。我们当前对组件的分类、编写目录和恢复技术尚不成熟。

- Finding, understanding and adapting reusable components 查找和改编可复用的组件

  Software components have to be discovered in a library, understood and, sometimes, adapted to work in a n ew environment. Engineers must be reasonably confident of finding a component in the library before they will make routinely include a component search as part of their normal development process.

  软件组件必须从库中找出来加以理解，有时还需要做些改编的工作使之适应当前的环境。软件工程人员对在库中找到组件应该有合理的自信，然后才能将所找组件纳入正常开发过程中。

11. 面向对象设计过程的五个关键活动

- Define the context and modes of use of the system;

  了解并定义上下文和系统的使用模式

- Design the system architecture;

  设计系统体系结构

- Identify the principal system objects;

  识别出系统中的主要对象

- Develop design models;

  开发设计模型

- Specify object interfaces.

  描述对象接口

12. 再工程的主要活动

- Source code translation 源代码转换

  Convert code to a new language.

  从旧的程序设计语言转换到相同语言的一个比较新的版本或另一种语言。

- Reverse engineering 反向工程

  Analyse the program to understand it;

  对程序进行分析并从中抽取信息来记录它的结构和功能；

- Program structure improvement 程序结构改善

  Restructure automatically for understandability;

  对程序的控制结构进行分析和修改，使它更容易阅读和理解；

- Program modularisation 程序模块化

  Reorganise the program structure;

  程序的相关部分被收集在一起，在一定程度上消除冗余。

- Data re-engineering 数据再工程

  Clean-up and restructure system data.

  对程序处理的数据作改变以反映程序变更。

13. 如何利用等价类划分构造测试用例