第5章

面向对象的软件开发过程

5.1 软件开发过程

软件开发过程

指开发软件产品的一整套活动，其中主要包括软件描述、软件开发、软件有效性验证和软件演化。不同的软件开发商，针对不同的开发项目可能会采用不同的方式组织上述4项活动的实施。

软件开发模型

软件开发模型是对软件开发的全过程、活动和任务的抽象描述。选择合适的软件开发模型将有利于提高软件开发的效率、软件产品的质量，以及日后的软件维护能力。

5.1.1 软件开发面临的主要问题

软件开发主要面临以下几个迫切需要解决的问题：

1．软件可靠性

软件可靠性是指软件系统能否在既定环境下运行并达到预期的结果。尽管通过对软件进行调试和测试可以排除大约40%的错误，任何人也不能保证任何一个软件产品没有错误。

2．软件生产率

计算机硬件的迅猛发展，带动了人们对软件需求的急剧增长。与计算机硬件的发展速度相比，软件的生产效率极其低下。

3．软件重用性

不同的应用领域要开发不同的应用软件，既使相同的应用目的也会由于需求上的微小差别，导致重新开发整个应用软件。

4．软件维护性

多么优秀的软件开发队伍也无法保证软件产品在使用过程中不会出现任何错误，因此，日后的维护工作将显得格外重要，而修改和完善软件产品在使用过程中显现出来的错误和不足之处是软件维护阶段的主要任务。

5.1.2 软件的生命周期

软件工程将按照工程化的方法组织和管理软件的开发过程，具体地说，它将软件开发过程划分成若干个阶段，每个阶段按照制定的规范标准完成相应的任务。软件的生命周期是指从某个软件的需求被提出并开始着手开发到这个软件被最终废弃的整个过程。通常在这个过程中，应该包括制定计划、需求分析，系统设计、程序编码、系统测试、系统运行及维护阶段。

1．制定计划

在正式开始开发软件项目之前，充分地研究、分析待开发项目的最终目标，整理出其功能、性能、可靠性及接口等方面的需求，计算出所需人力、物力的资源开销，推测日后可能获取的经济效益，提供支持该项目的技术能力以及给出开发该项目的工作计划。

2. 需求分析

这个阶段的任务需要系统分析员与用户共同完成。这是正式进入软件开发的标志性阶段。其主要任务是对待开发软件项目的需求进行仔细地分析，并给出准确、详细的定义。在此基础上，划清系统边界，明确哪些需求由软件系统完成，哪些需求不属于软件系统的功能范畴等。

3．系统设计

系统设计是整个软件项目开发的核心阶段，软件设计人员需要根据软件需求规格说明书，设计出系统的总体结构，划分好模块，并确定各模块之间的相互关系以及每个模块所应该完成的任务。

4．程序编码

利用某一种程序设计语言将系统设计阶段描述的所有内容用计算机可以接受的程序形式表达出来，并将其组装起来加以调试是这个阶段的主要任务。

5. 系统测试

找出程序中存在的错误，利用设计的测试用例从不同角度检测软件的各个组成部分。测试主要包括单元测试、集成测试、确认测试和系统测试，使用的测试方式主要有白盒测试和黑盒测试。

6. 系统运行及维护

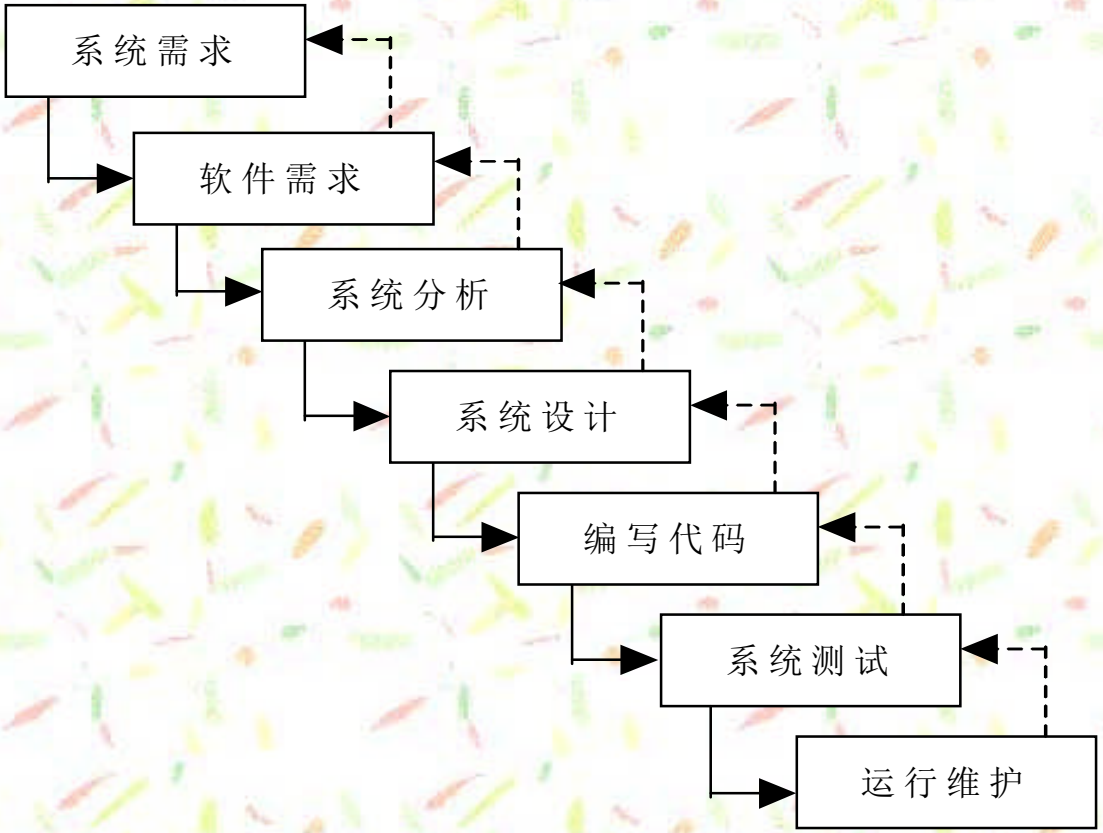
系统运行期间可能会出现各种意想不到的异常现象，需要软件维护人员及时找出问题所在，并给予修正。除此之外，由于软件运行环境的改变，可能需要对原有软件系统进行适当地调整。

5.1.3 软件开发模型

软件开发模型是指软件开发全过程、活动和任务的结构框架，它能够清楚、直观地表达软件开发的全过程，明确各阶段所需要完成的具体任务，并对开发过程起到指导和规范化的作用。

瀑布模型

瀑布模型将软件开发过程划分为7个阶段，前一个阶段的成果将作为后一个阶段的输入。整个开发过程形如瀑布流水。



面向对象程序设计

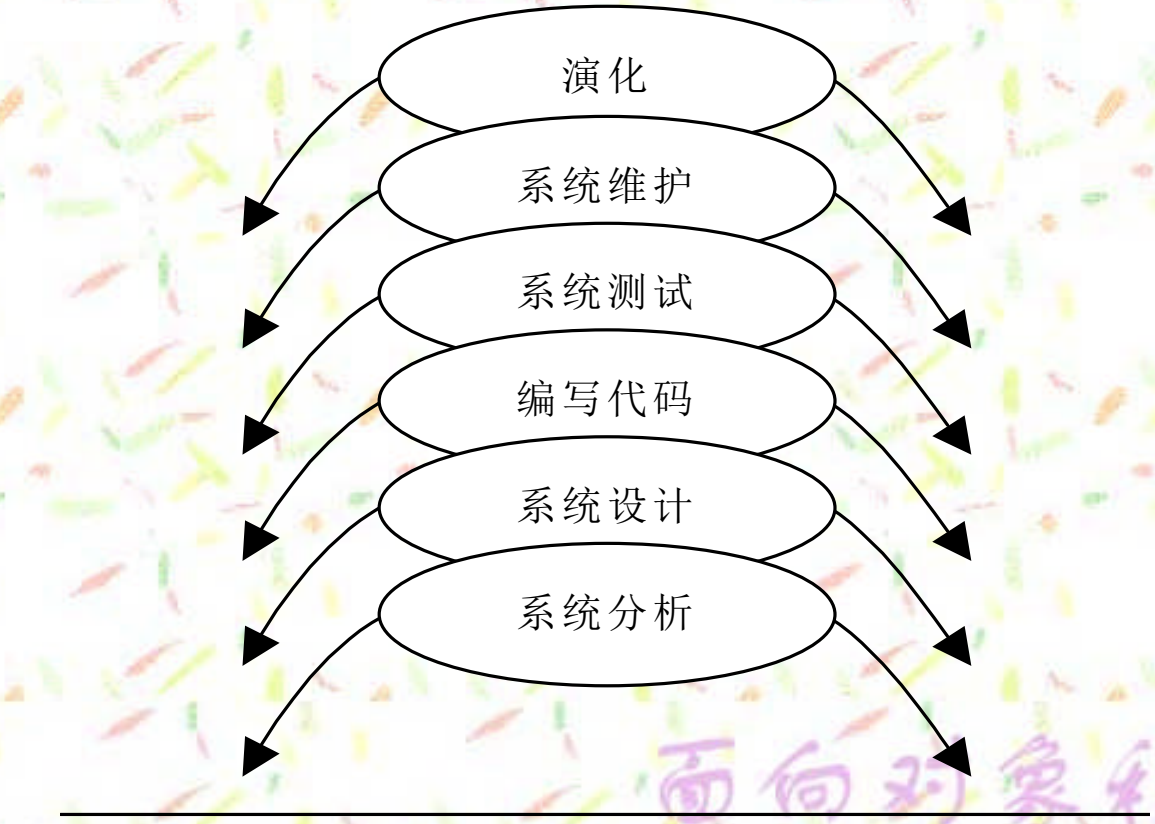
演化模型

不要求用户在开发系统之前，必须将全部的需求提交出来，而是只提出系统的核心需求，开发者最初只实现核心需求，并交给用户试用，以便得到及时、有效的反馈意见，细化、增强系统功能的补充需求说明，软件开发人员再根据用户的反馈，对先前的系统进行二次开发，即迭代一次。与初次开发一样，同样需要经过需求分析、系统设计、编写代码、系统测试等一系列过程。如果用户试用之后还不满意，就继续进行第三次开发，每一次重新开发的结果都会更加逼近用户的最终需求。

面向对象程序设计

喷泉模型

喷泉模型将软件开发过程的各个阶段描述为相互重叠和多次反复的过程，就好像泉水由泉眼喷出后又回落的场景，这种开发模型主要用于支持面向对象的开发过程。

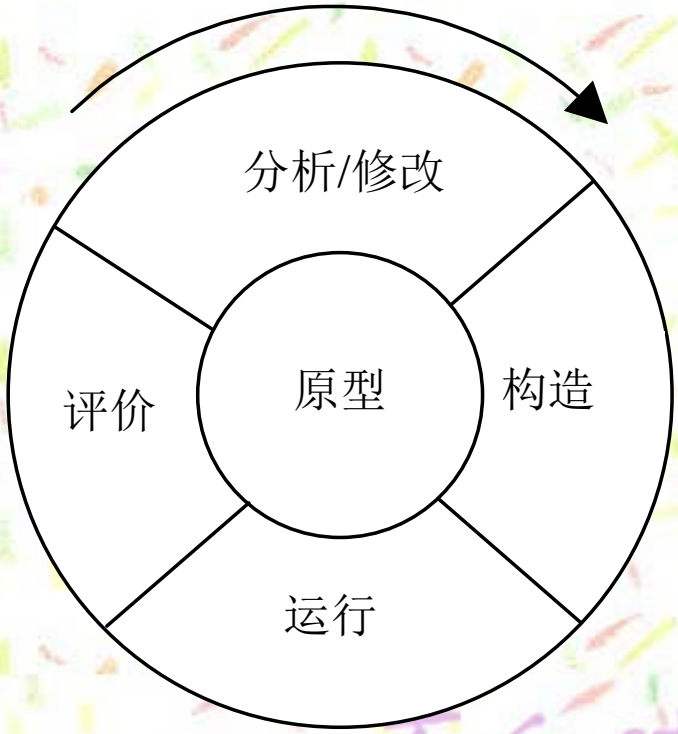


螺旋模型

螺旋模型是在瀑布模型和演化模型的基础上，加入风险分析所形成的一种软件开发模型。螺旋模型将软件开发过程刻画为多次螺旋上升的过程，每向前走一步都要为下一步做出风险预测，一旦发现风险过大，开发者和用户无法承受，就应该尽早终止开发，以便将损失降到最低程度。

原型开发模型

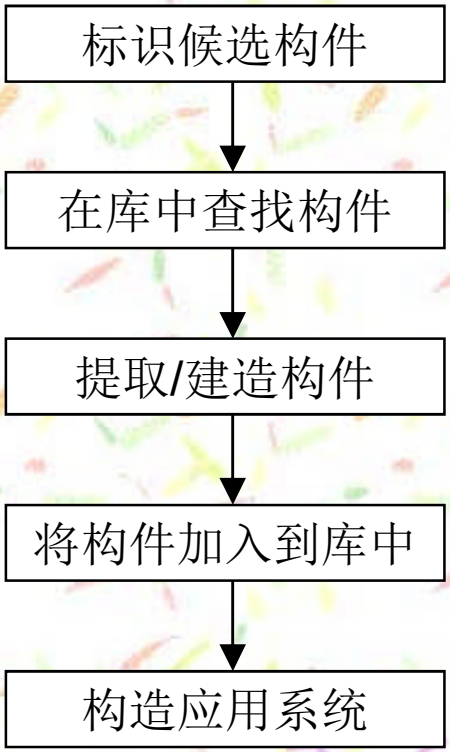
原型开发模型将软件开发分为需求分析、构造原型、运行原型、评价原型和修改原型几个阶段，并不断重复这个过程，直到用户满意为止。



面向对象程序设计

基于构件的软件开发模型

该模型融合了螺旋模型的许多特征，强调软件开发过程的迭代性并利用预先包装好的软件构件来构造应用系统。



5.2 面向对象的软件开发过程

5.2.1 面向对象技术

对象:面向对象技术的核心是对象，对象可以被用来描述现实世界中的任何客体，现实世界中的任何客体都可以通过建模转换成对象。在软件系统中，对象由一组属性和行为组成，属性是对客体属性状态的静态描述，行为是对属性状态的有效操作，其中主要包括获取状态值，修改状态值等。

类:类与对象是两个完全不同的概念，但又存在着密不可分的关系。类是对一组属性和行为相同的对象的描述；对象是类的实例，在程序运行后被创建，对象之间通过发送消息相互操作。我们将利用这种机制开发的软件系统称为面向对象软件系统。

面向对象软件系统应该具备的特征

1. 通过抽象机制，将问题域中各个成员表达成对象。
2. 通过类实现封装。将抽象出来的对象属性和行为绑定在一起就形成了类。
3. 对象之间通过传递消息相互驱动。系统中各个对象不是　孤立存在的，而是需要相互作用，共同完成既定的任务。
4. 对象生命周期。是指对象存在的时间段。
5. 类层次结构。类层次结构可以刻画不同类之间的关联关系。
6. 多态性。多态性是面向对象系统最终表现出来的基本特征。

5.2.2 面向对象分析

面向对象分析（object-oriented analysis，缩写OOA）是软件开发的初始阶段，参与这个阶段工作的人员既有客户，又有开发技术人员，他们的中心任务是实现需求分析和系统分析，即分析问题域的特征，确定问题的解决方案，并为目标系统寻找对象，明确对象的属性和行为以及对象之间的各种关系，以便为最终的软件系统建立一个分析模型，该模型将从不同的侧面描述系统的基本特征。

OOA应该经历的主要过程

分析问题域，明确用户需求

在系统开发人员接收开发任务的最初阶段，对项目的具体需求并不完全明确，用户给予的需求说明也往往含糊不清，这就需要开发人员与未来的用户共同合作，帮助开发人员尽快地理解业务领域的相关知识，取得对问题域的一致认识，明确用户对未来系统的需求，定义系统的职责范围和边界，探讨问题的初步解决方案。

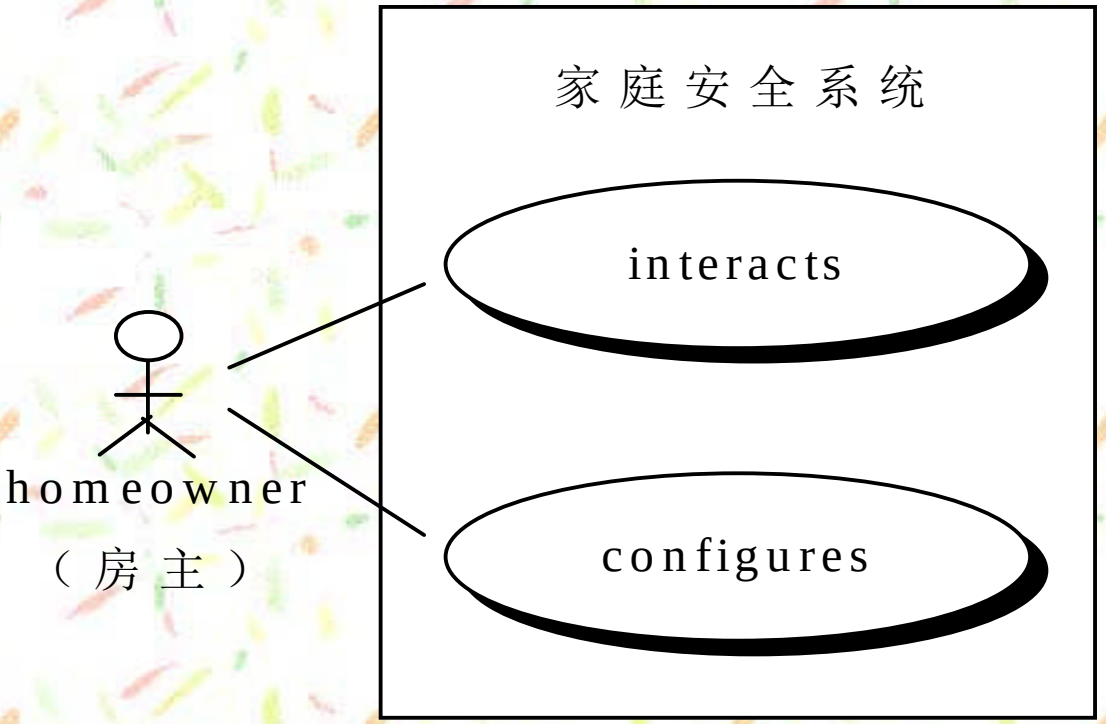
标识用况（Use Case）

用况定义了在业务活动中的业务规则和任务，反映了系统应该如何支持这些业务活动的基本过程，它是构造分析模型的起点，是确定功能需求及未来系统解决方案的重要依据。分析用况的主要目的是通过描述外部活动者与系统的交互，定义未来系统的功能需求。

在UML中，提供了一种被称为use-case图的表示法，专门用来描述活动者、用况以及它们之间的关系，由此可以体现整个系统提供的基本功能。

面向对象程序设计

UML用例图



在这个use-case图中，标识了两个用况，表明该系统主要有两个基本功能，一个是interacts（交互），另一个是configures（配置）。

识别对象并将其抽象成类

哪些能够成为最终系统的对象，需要参考下面6项特征：

（1）这个对象的信息需要被记忆，否则系统无法正常地工作；

（2）这个对象应该具有一组确定的操作，通过它们实现对对象属性的修改；

（3）这个对象应该具有多于一个属性；

（4）能够定义一组适用于所有实例的公共属性；

（5）能够定义一组适用于所有实例的公共操作；

（6）属于基本需求内容。

标识对象的属性和行为

目标系统中选定的每个类都要有一定的职责。每个类的职责将通过属性和行为体现，其中属性标识了类的静态特征，即类需要记录的信息；行为标识了类应该具有的操作能力。明确每个类职责的过程就是寻找类的属性和方法的过程，属性可以从问题的陈述中抽取，或通过对类性质的理解加以辨认；行为可以从对系统的处理叙述中获得，即可以将动词标识的动作作为候选行为。

确定类职责应该遵循下列5条原则：

（1）系统包含的所有职责应该均匀地分担到每个类中，避免出现某些类的职责过大，而另一些类的职责过小的现象。

（2）每个职责应该尽可能地被概括描述，即一般性的职责应该被放置在较高层次的类中。

（3）属性记录的信息与相关的行为应该位于同一个类中，它体现了面向对象的封装性。

（4）反映一个事物的信息应该被放置在一个类中，而不要分散到多个类中。

（5）在相关类之间共享职责。

定义类之间的关系

每个类主要以两种方式完成它的职责：一是应用类自身的行为操作自己的属性；二是与其他类协作共同完成某项职责。通常协作是指客户向服务器发出了某项请求，以便完成客户的某项职责。如果为了完成某项职责，一个对象需要向另外一个对象发送消息，则我们说在这两个对象之间就产生了协作。

用户界面需求

用户界面是系统的重要组成部分，在对系统进行功能需求分析的同时，不要遗忘人机交互部分的探察。对使用系统的人员进行分析，可以有的放矢地设计出适合这些人员特点的交互方式和界面表现形式；对人和机器交互过程进行分析，可以明确操作人员如何向系统发出命令，以及系统如何提交操作结果和反馈信息。

5.2.3 面向对象设计

面向对象设计（OOD）的主要任务是确定系统的体系结构和完成对象的设计。包含下列主要步骤：

（1）系统分解与分层

（2）确定任务管理策略和控制驱动机制

（3）设计人机交互界面

（4）确定实现数据管理的策略

（5）对象设计

（6）评审设计模型，在必要的时候可以对此过程给予迭代

系统分解与分层

系统分解是在系统设计阶段使用的重要手段，通常使用将分析模型划分为子系统的方法对系统进行分解。子系统是对OOA模型中类的逻辑分组，每个分组构成一个子系统，它是描述实现用户需求的组件，每个子系统可看成一个高层次的模块。

确定任务管理策略和控制驱动机制

在OOD中由任务管理部分负责设计管理并发行为的策略和控制驱动机制。任务管理设计首先需要根据对象的动态模型分析、定义系统的并发性，并通过识别和建立控制流程（包括进程或线程）来设计面向对象系统的任务，选择软件实现的控制方法。

设计人机交互界面

人机交互界面是软件系统与用户直接接触的部分，它给予用户的影响和感受最为明显，所以人机交互界面的质量对于软件系统能否成功具有至关重要的作用。界面的设计应该遵循下列基本过程和策略：

（1）选择界面支持系统

（2）选择界面元素

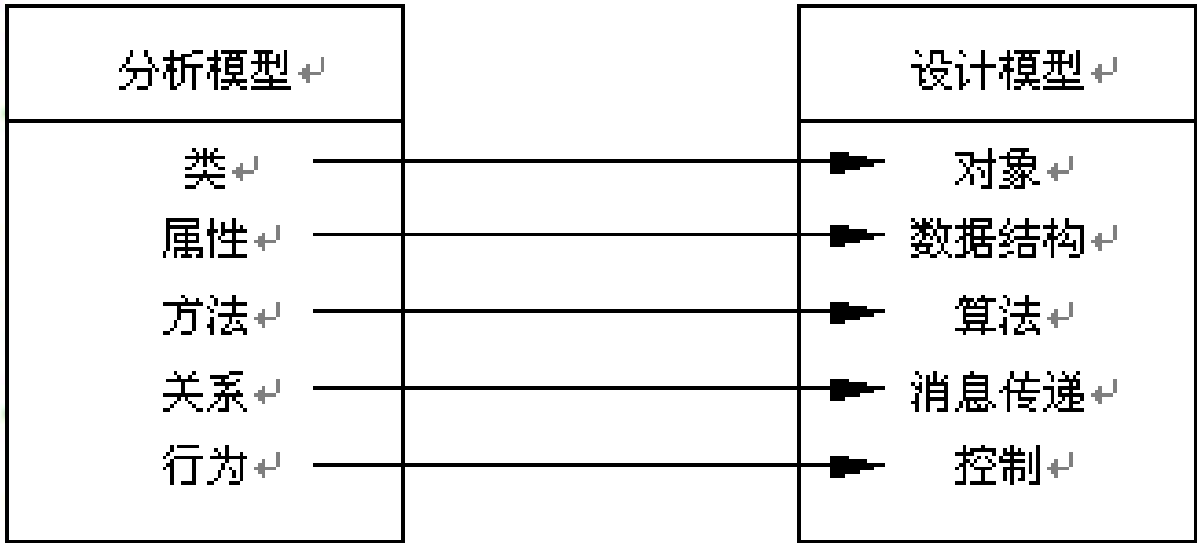
（3）用00概念表示界面元素

确定实现数据管理的策略

数据管理部分提供了数据在数据管理系统中存储和检索对象的基本结构，它是建立在某种数据存储管理系统之上的系统存储或检索对象的基本设施。它的作用是将目标系统中依赖开发平台的数据存取部分与其他功能分离开，使数据存取可以通过一般的数据管理系统（不管是文件系统、关系型数据库、面向对象数据库或其它方式）来实现。

对象设计

对象设计强调从问题域概念到计算机域概念的转换，重点在于为每个类的属性和行为做出详细的设计，主要包括确定每个属性的数据结构和行为操作的实现算法。



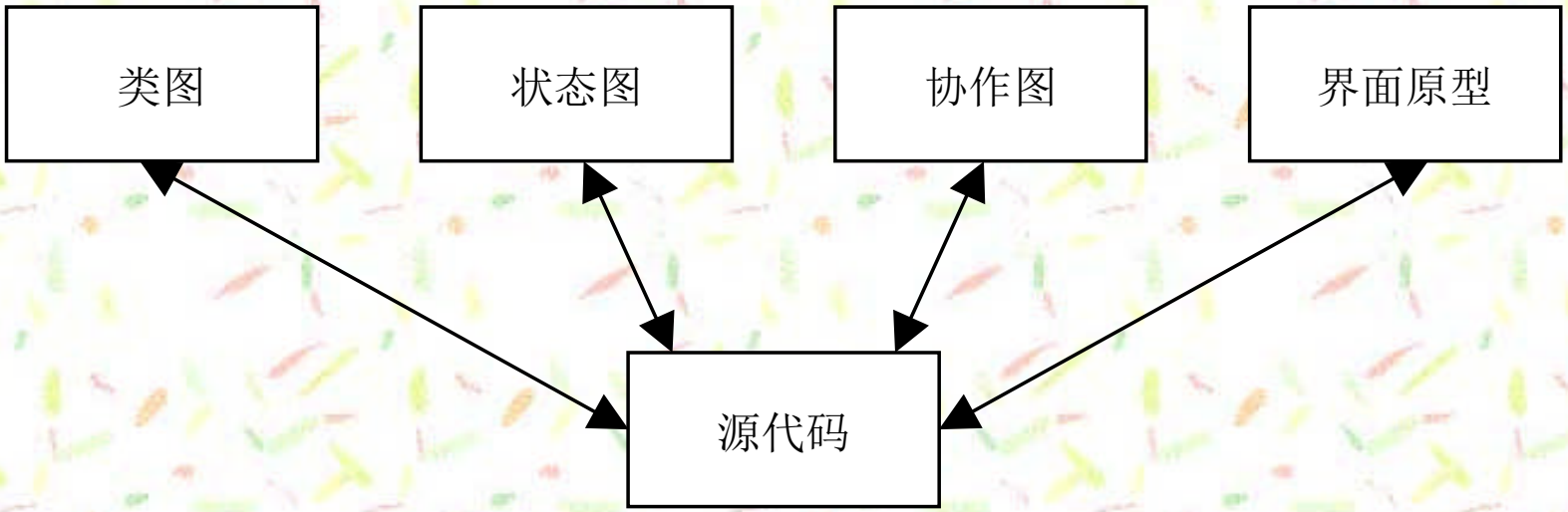
分析模型转换为设计模型

5.2.4 面向对象程序设计

面向对象程序设计（object-oriented programming，缩写OOP）的目标是根据分析阶段和设计阶段的成果，选用一种支持面向对象程序设计思想的程序设计语言，编写应用系统的程序代码，最终完成一个可供测试的应用系统源代码。

设计模型与编写源代码的关系

设计与编写源代码是一个高度相关的迭代过程，编码依赖于设计模型。



面向对象程序设计

5.2.5 面向对象测试

面向对象测试（object-oriented testing，缩写OOT）的整体目标与传统的结构化测试的目标是一致的，即发现尽可能多的软件错误，保证软件的质量。

面向对象软件的测试必须注意三个问题：

- 扩大测试的视角

  测试必须包括OOA测试、OOD测试和OOP测试

- 单元测试和集成测试的策略必须有很大改变

  面向对象软件中最小的可测试单位是封装的类或对象

- 测试用例的设计必须考虑面向对象软件的特征

  测试需要了解对象的详细状态

面向对象程序设计