# 第02讲 面向对象方法

徐峰磊

QQ:474017512

# 主要内容

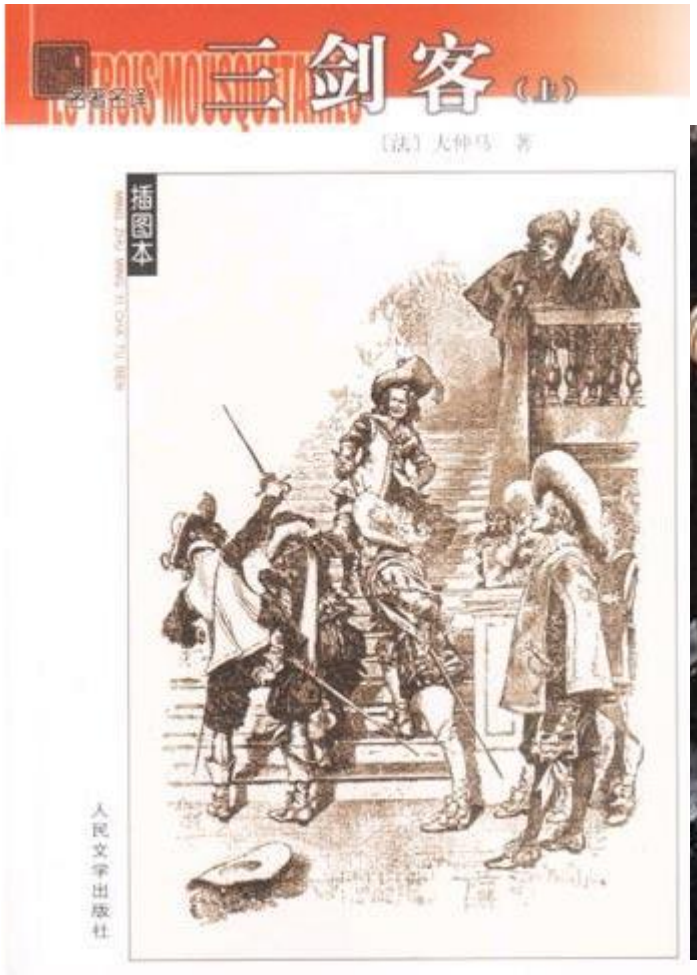
1. 产生原因 (Reason)
2. 基本概念与特征 (Principle)
3. 开发过程 (Progress)
4. 下一步发展方向 (Direction)

# 2.1 了解面向对象产生的原因史

- 俗话说：
- “天下大势，分久必合，合久必分。”


三國演義
THREE KINGDOMS
孙权
曹操
刘备
三大枭雄登场

Grady Booch Ivar Jacobson Rumbaugh


三剑客（上）
[法] 大仲马 著
插图本
人民文学出版社

- 起初软件是手工作坊的生产方式，没有标准化的过程、工具和技术，从而导致了大量软件错误。之后计算机专家们提出了各种语言和方法，但还是不能避免错误的发生。
- 小型的软件（5000行代码以下的软件）基本能正确的生产出来，但大中型软件（50000行代码以上的软件是大型软件，其间的为中型软件）项目就很难保证。

例:Windows95有1000万行代码
Windows2000有5000万行代码

Exchange2000和 Windows2000开发人员结构

| | Exchange2000 | Windows2000 |
|---|---|---|
| 项目经理 | 25人 | 约250人 |
| 开发人员 | 140人 | 约1700人 |
| 测试人员 | 350人 | 约3200人 |

# IE项目

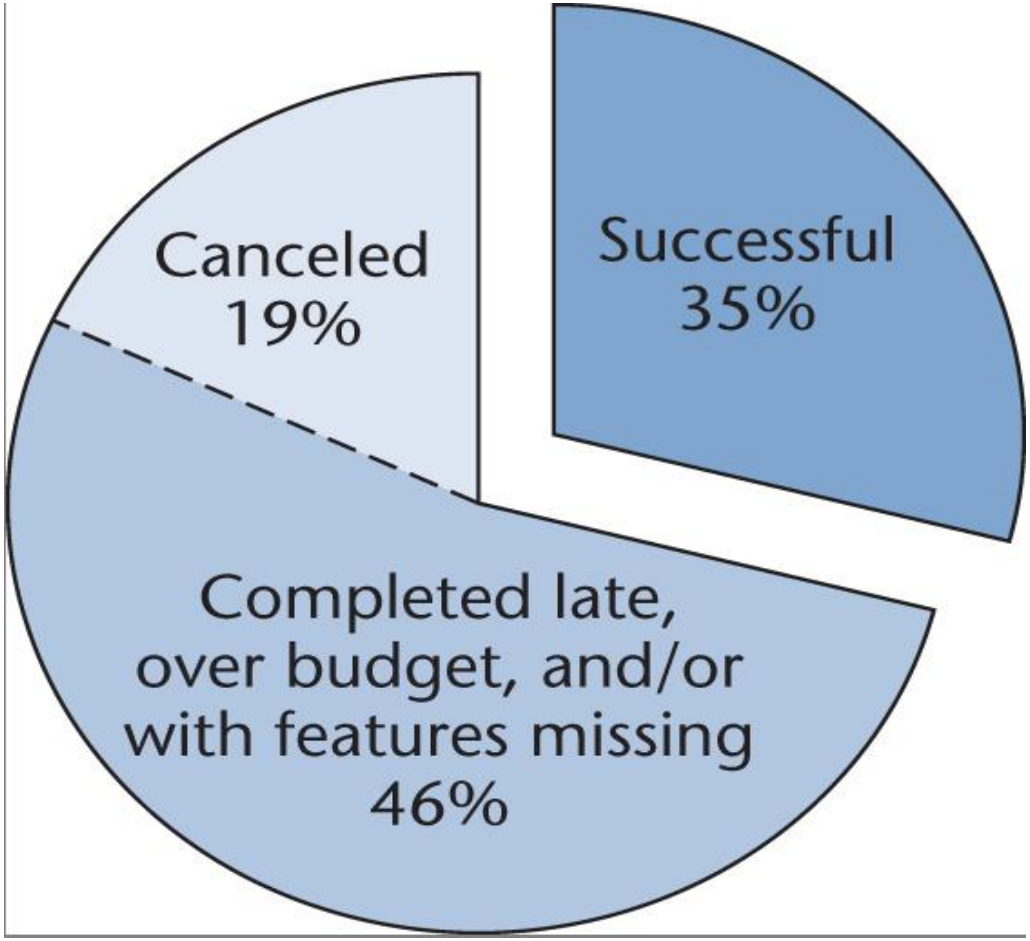
- IE各版本

| 发布时间 | 版本 | 开发人员数目（约） |
| --- | --- | --- |
| 1994 年冬季 | IE 1.0 | 7 |
| 1995 年 11 月 | IE 2.0 | 30 |
| 1996 年 8 月 | IE 3.0 | 70 |
| 1997 年 6 月 | IE 4.0 | 300 |
| 1999 年 8 月 | IE 5.0 | 500 |
| 2000 年 7 月 | IE 5.5 | 300 |
| 2001 年 8 月 | IE 6.0 | 100 |
| 2002 年 10 月 | IE 6.0 QFE（快速漏洞修补） | 5 |

- 传统的软件工程方法学曾经给软件产业带来了巨大进步，部分地缓解了软件危机，使用这种方法学开发的许多中、小规模软件项目都获得了成功。但是，当把这种方法学应用于大型软件产品的开发时，却很少取得成功。

# Standish Group Data

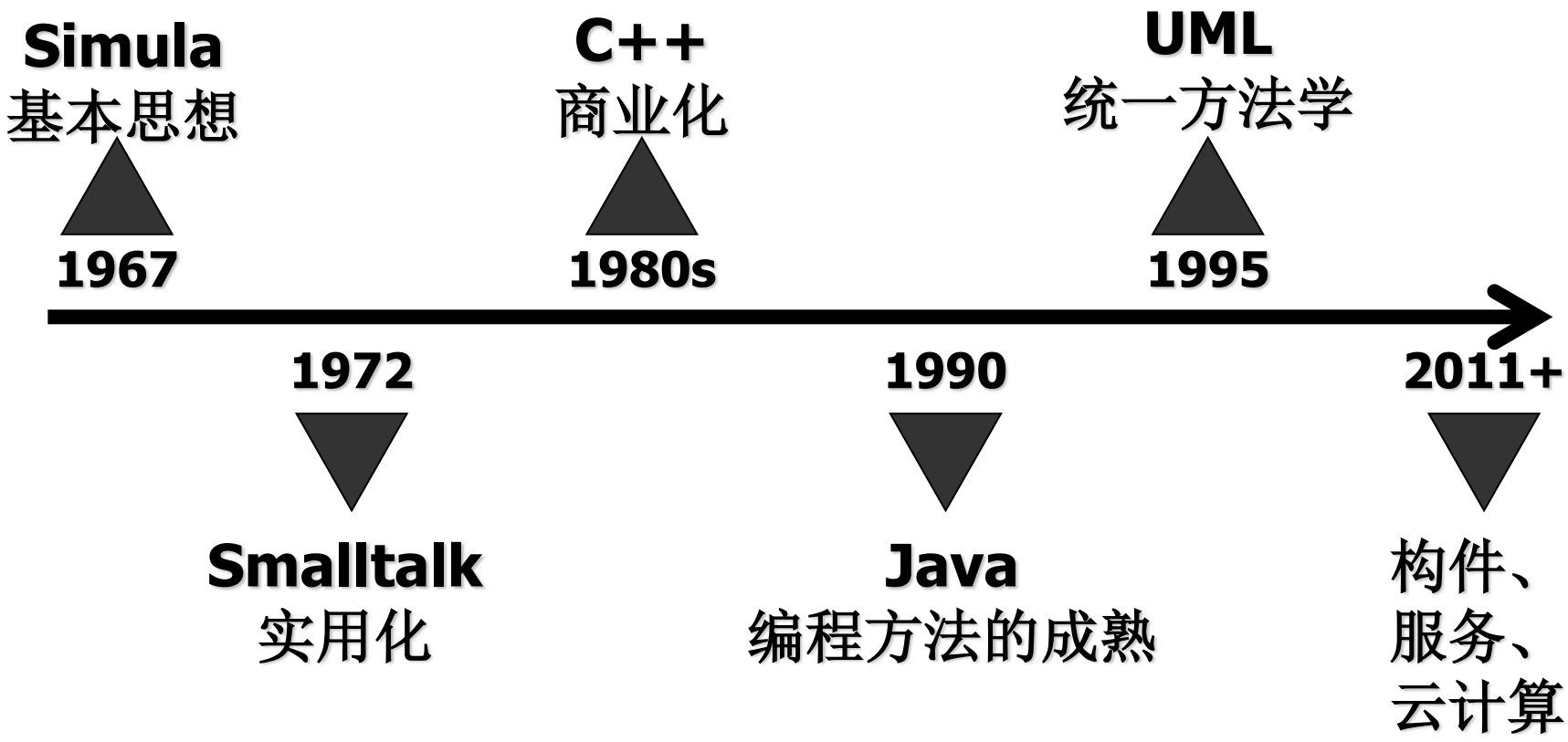
- **Data on projects completed in 2006**




Just over one in three projects was successful

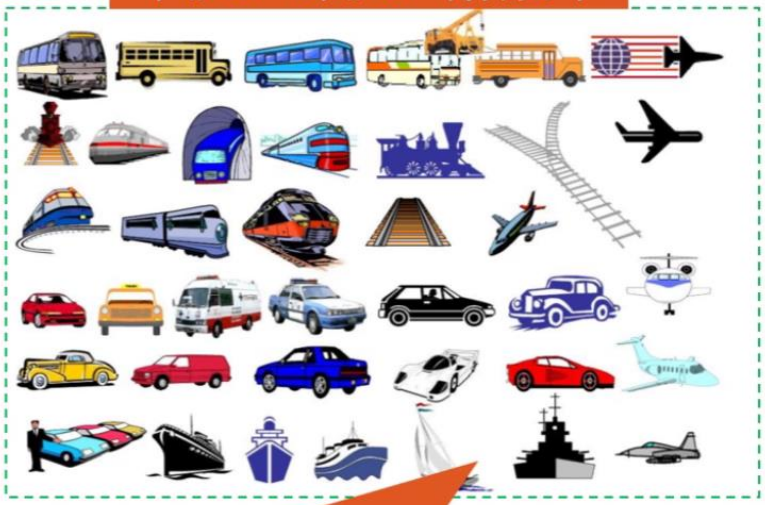
- 在20世纪60年代后期出现的面向对象编程语言Simula-67中首次引入了类和对象的概念，自20世纪80年代中期起，人们开始注重对面向对象分析和设计的研究，从而逐步形成了面向对象方法学。到了20世纪90年代，面向对象方法学已经成为人们在开发软件时首选的范型。可以说，面向对象技术是当前最好的软件开发技术。

# 对象技术的发展历史


Simula
基本思想
1967
C++
商业化
1980s
UML
统一方法学
1995
1972
Smalltalk
实用化
1990
Java
编程方法的成熟
2011+
构件、
服务、
云计算

# 2.2 面向对象方法基本概念与特征


类：
军队、部队、解放军



多个类：
汽车、火车、飞机、交通工具

类：
学生、小学生、少先队员

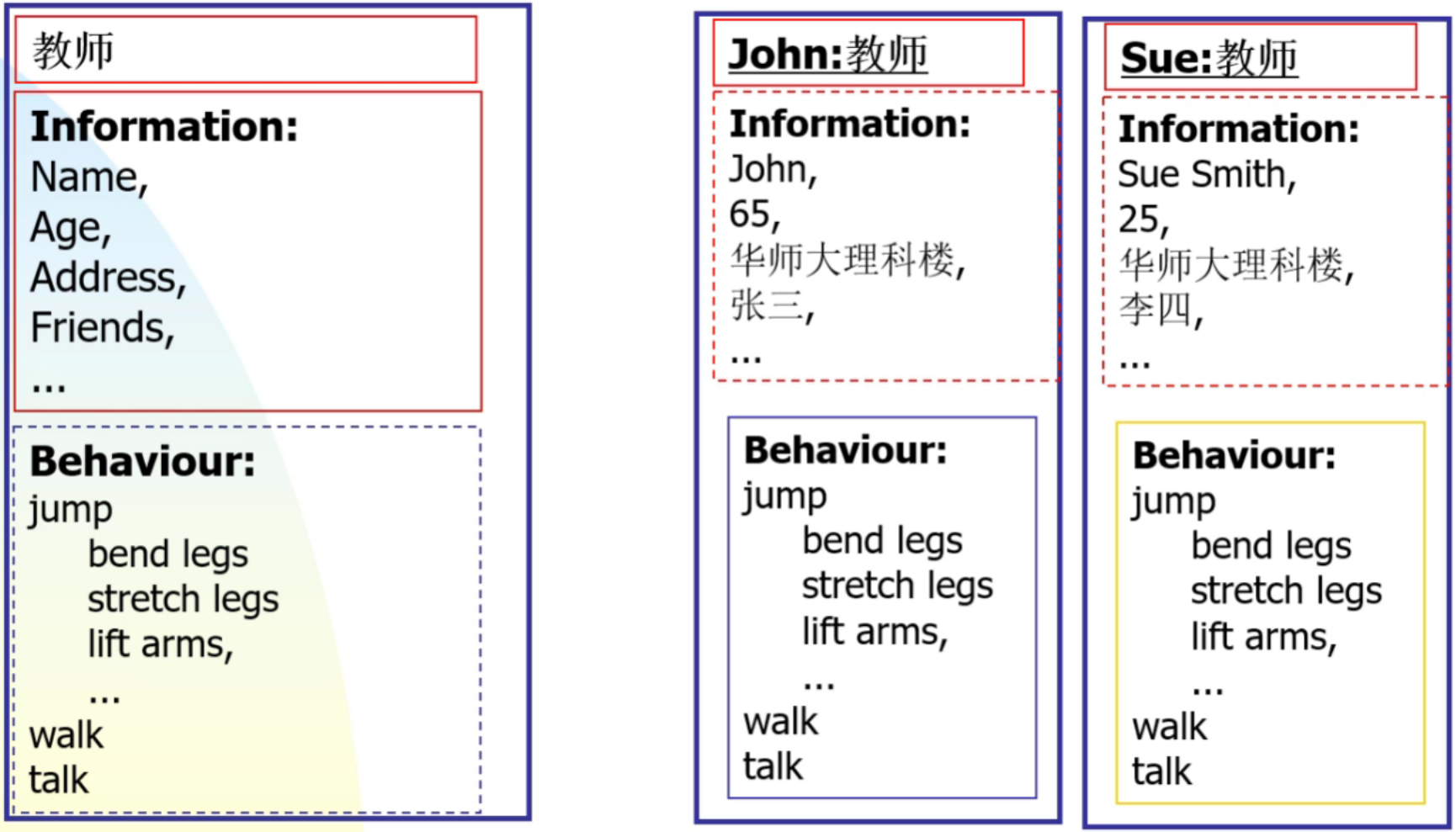
# 2.2 面向对象方法基本概念与特征

对象：
这是三班的陈刚
这是二排的王健康
这是一连长吴刚


对象：
这是张三的红色小汽车
这是南航的7892号飞机

对象：
这是张建伟同学、这是王海三同学
这是吴丽霞同学

# 2.2 面向对象方法基本概念与特征



# 2.2 面向对象方法基本概念与特征

- 注意：概念之间的互用
  - 属性Attribute == 数据Data == 状态state == 信息information
  - 操作operation == 方法Method ==行为behaviour = = 职责responsibility
  - 对象object == 实例instance

# 2.2 面向对象方法基本概念与特征

- 对象：对象是要研究的任何事物。从一本书到图书馆，单个整数到庞大的数据库、极其复杂的自动化工厂、航天飞机都可看作对象，它不仅能表示有形的实体，也能表示无形的（抽象的）规则、计划或事件。对象由数据（描述事物的属性）和作用于数据的操作（体现事物的行为）构成一独立整体。
- 类：类是对象的模板。即类是对一组有相同数据和相同操作的对象的定义，一个类所包含的方法和数据描述一组对象的共同属性和行为。类是在对象之上的抽象，对象则是类的具体化，是类的实例。类可有其子类，也可有其父类，形成类层次结构。
- 消息：消息是对象之间进行通信的一种规格说明。一般它由三部分组成：接收消息的对象、消息名及实际变元。

# 2.2 面向对象方法基本概念与特征

软件功能如何实现的？

- 类
  - 定义了对象群体的逻辑结构，包括属性和操作
  - 系统运行时，类作为产生对象的模板，在物理层面是不存在的
- 对象
  - 系统运行时必须为每一个需要的对象分配内存、保存数据
  - 对象存在于物理层面，每个对象都有自己的数据空间（内存）
  - 所有的对象共享同一块代码空间
- 消息
  - 对象之间的一种交流手段
    - 就像我们日常工作中的各种交流手段
- 所有相关对象之间相互协作完成软件功能

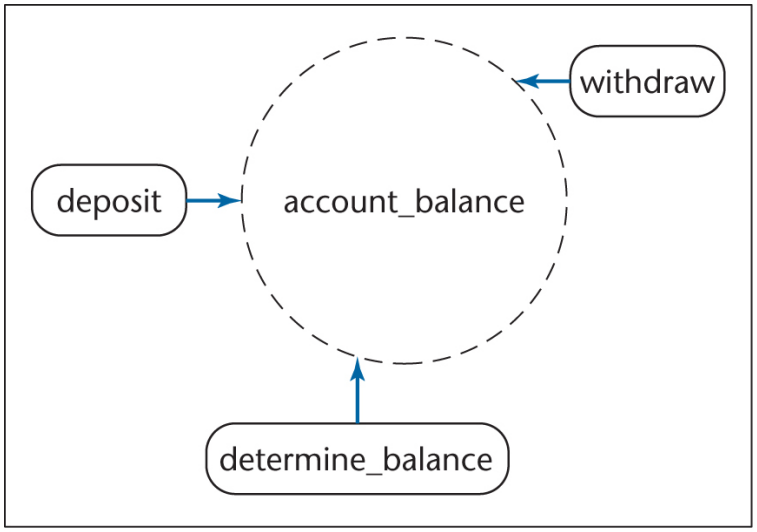
# 2.2 面向对象方法基本概念与特征

- **Coad和Yourdon将面向对象概念概括为以下方程：**

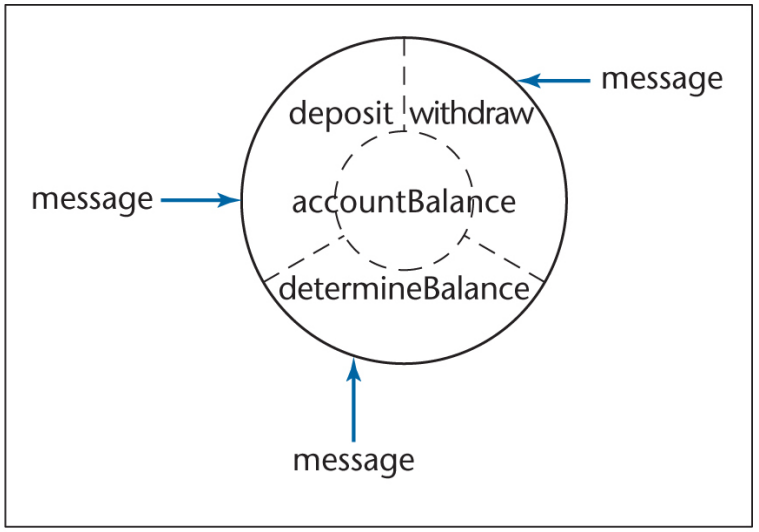
  面向对象 = 对象 + 类 + 继承 + 通信

即：面向对象就是既使用对象又使用类和继承等机制，而且对象之间仅能通过传递消息实现彼此通信。

- 封装性：封装是一种信息隐蔽技术，它体现于类的说明，使数据更安全，是对象的重要特性。封装使数据和加工该数据的方法（函数）封装为一个整体，以实现独立性很强的模块，使得用户只能见到对象的外部特性（对象能接受哪些消息，具有那些处理能力），而对象的内部特性（保存内部状态的私有数据和实现加工能力的算法）对用户是隐蔽的。

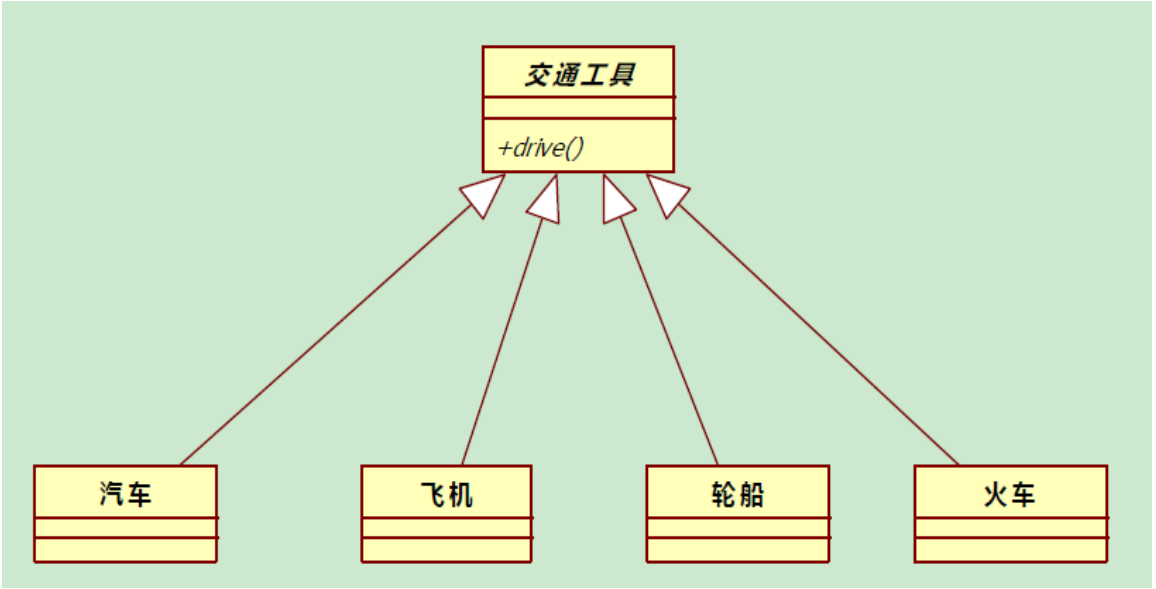



(a)

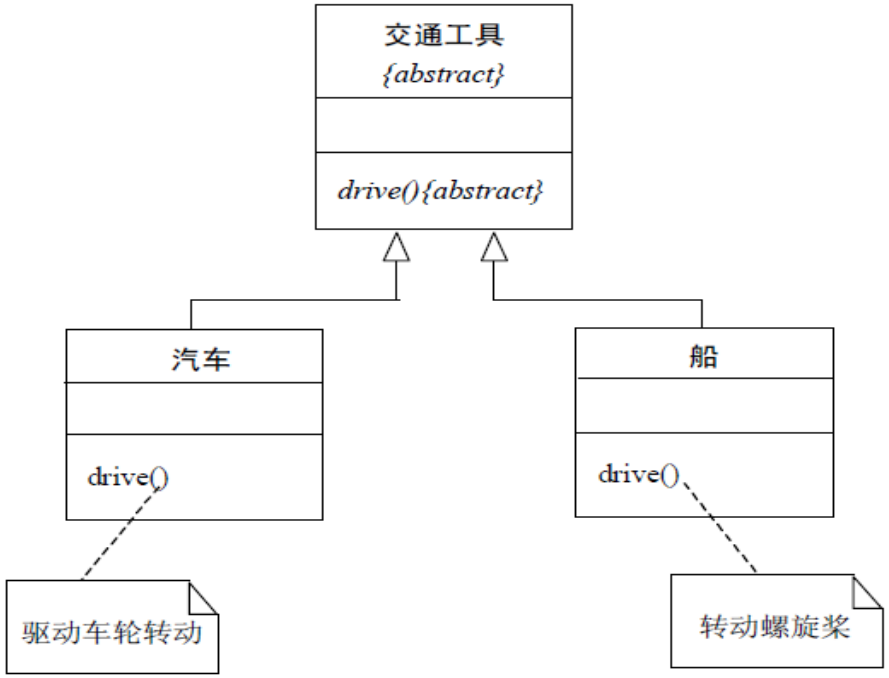



(b)

- 继承性：继承性是子类自动共享父类数据和方法的机制。它由类的派生功能体现。一个类直接继承其父类的全部描述，同时可修改和扩充。



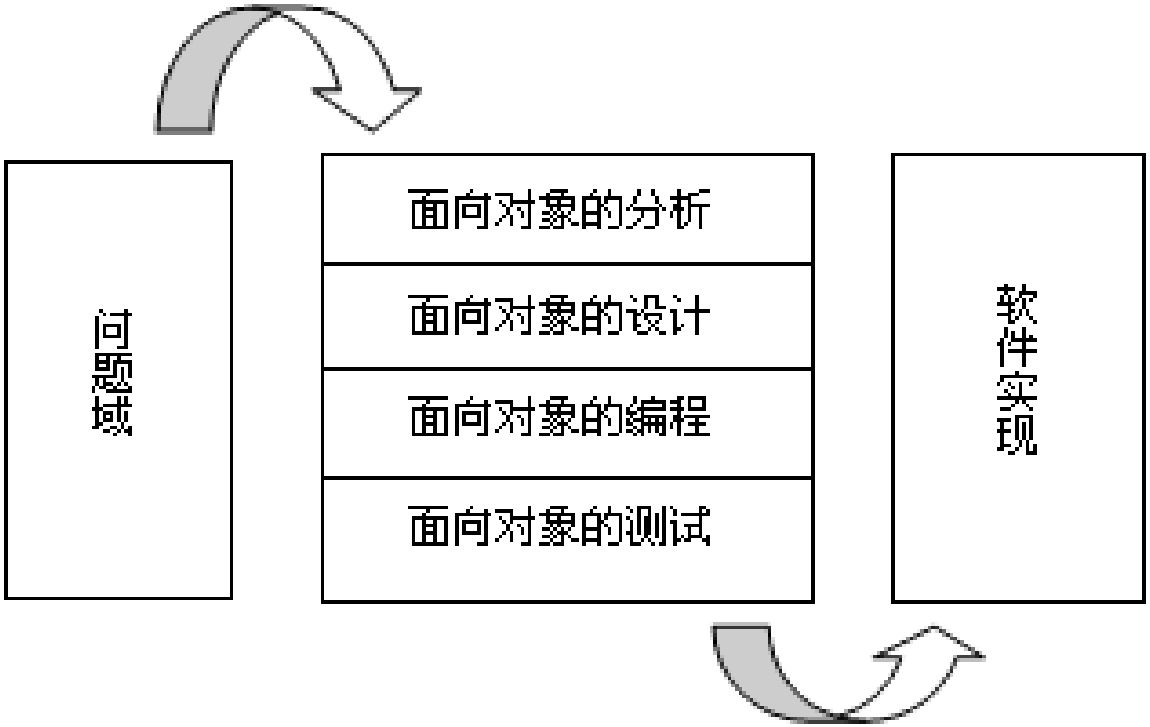
- 多态性：对象根据所接收的消息而做出动作。同一消息为不同的对象接受时可产生完全不同的行动，这种现象称为多态性。利用多态性用户可发送一个通用的信息，而将所有的实现细节都留给接受消息的对象自行决定，因此，同一消息即可调用不同的方法实现。

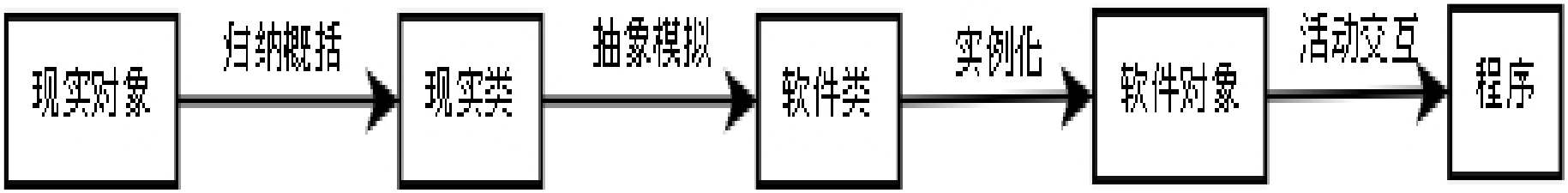


# 2.2 面向对象方法基本概念与特征

- Everything is an object.
- A program is a bunch of objects telling each other what to do by sending messages.
- Each object has its own memory made up of other objects.
- Every object has a type.
  - each object is an instance of a class, in which "class" is synonymous with "type."
- All objects of a particular type can receive the same messages.
  - an object of type "circle" is also an object of type "shape," a circle is guaranteed to accept shape messages. This means you can write code that talks to shapes and automatically handle anything that fits the description of a shape.

# 2.3 面向对象方法学开发过程



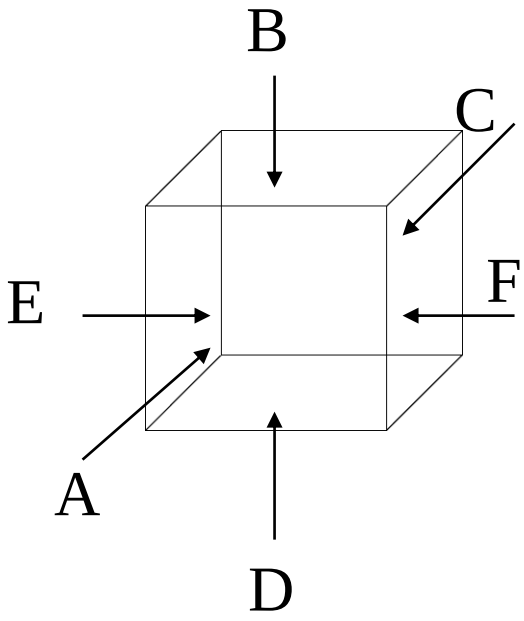



- 现实世界中存在现实对象，然后人们根据自己的观察角度和要求将现实对象抽象成现实类，然后软件设计人员基于现实类模拟出软件类，最后在程序中将软件类实例化成软件对象，最终的程序就是软件对象的活动和交互。


学生实例
李华
陈丽
张涛
刘霞
...
抽象
学生类
属性：
学号
姓名
专业
班级
成绩
...
行为：
吃饭
睡觉
上课
锻炼
...
创建
学生对象
Li Hua
Chen Li
Zhang Tao
Liu Xia
...

# 用对象观点认识事物

- A.这里面有什么东西？
  类与对象
- B.每个东西看上去是什么样的？
  类的属性
- C.每个东西能做点什么用？
  类的操作
- D.这些东西都呆在什么地方？
  类的行为、状态、部署
- E.这些东西之间有什么关系？
  类间的关联
- F.这些东西是怎么成事的？
  类间的协作(用例实现)



- 首先要进行面向对象的分析，其任务是了解问题域所涉及的对象、对象间的关系和作用（即操作），然后构造问题的对象模型，力争该模型能真实地反映出所要解决的"实质问题"。在这一过程中，抽象是最本质、最重要的方法。针对不同的问题性质选择不同的抽象层次，过简或过繁都会影响到对问题的本质属性的了解和解决。

- 其次就是进行面向对象的设计，即设计软件的对象模型。根据所应用的面向对象软件开发环境的功能强弱不等，在对问题的对象模型的分析基础上，可能要对它进行一定的改造，但应以最少改变原问题域的对象模型为原则。然后就在软件系统内设计各个对象、对象间的关系（如层次关系、继承关系等）、对象间的通信方式（如消息模式）等，总之是设计各个对象应做些什么"。

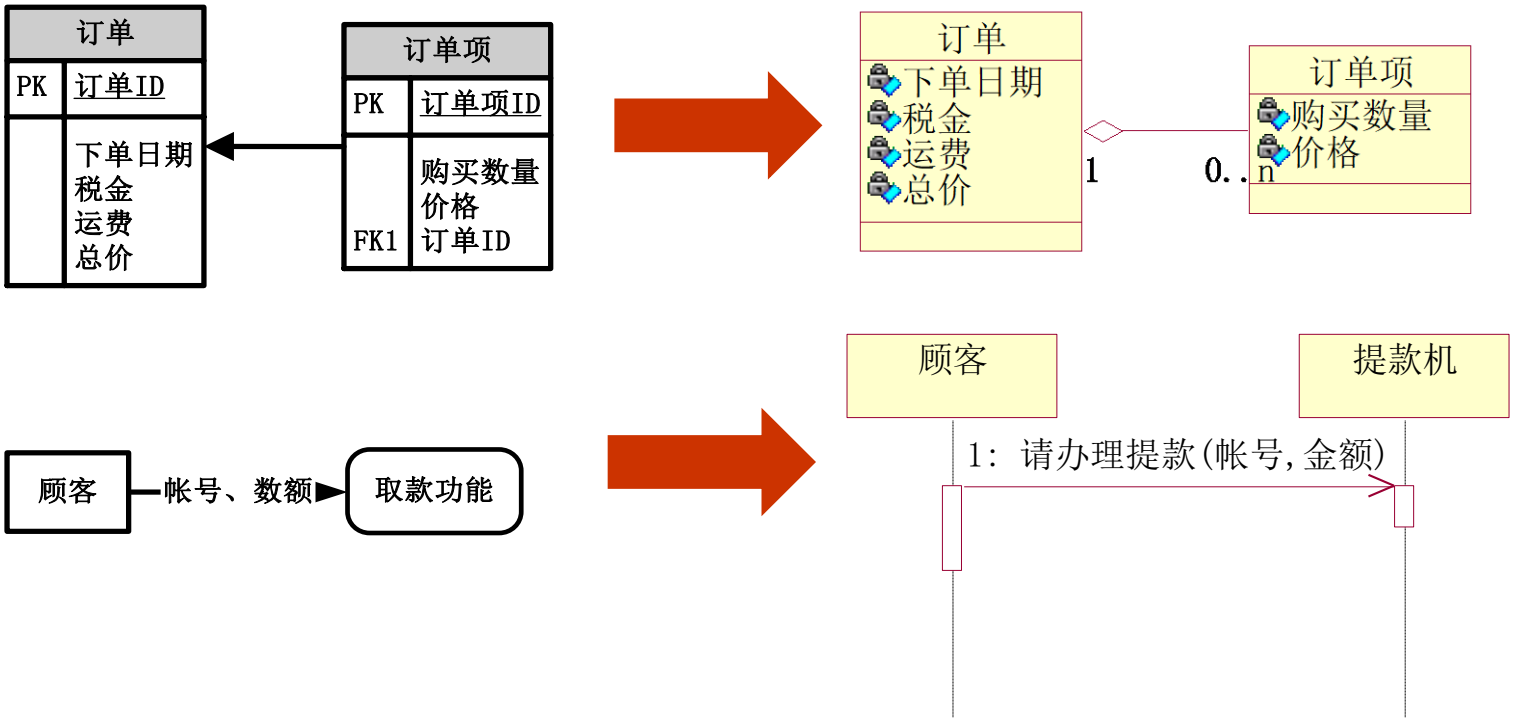
- 最后阶段是面向对象的实现,即指软件功能的编码实现，它包括：每个对象的内部功能的实现；确立对象哪一些处理能力应在哪些类中进行描述；确定并实现系统的界面、输出的形式及其它控制机理等，总之是实现在设计阶段所规定的各个对象所应完成的任务。

# 总结：结构化VS面向对象

- 结构化思维用过程刻画数据间关系
- 对象思维直接用类表达数据间关系
- 结构化中，数据是死的，全部依赖算法操作
- 对象思维中，数据是活的，“她”知道自己的信息（属性），并能完成自己的工作（操作）
- 结构化思维更像是一个人在解决所有问题
- 对象思维更像是一个团队的分工协作

# 面向对象 VS 结构化-1

- 扬弃，不是否定



# 面向对象 VS 结构化-2

- (程序)实现角度

| | |
|---|---|
| 数据结构+算法＝程序设计 | 以对象为中心组织数据与操作 |
| 数据 | 对象属性 |
| 操作 | 对象的行为 |
| 类型与变量 | 类与对象实例 |
| 函数（过程）调用 | 消息传递 |
| 类型与子类型 | 一般类与特殊类，继承 |
| 构造类型 | 整体－部分结构，聚合 |
| 指针 | 关联 |

# 面向对象 VS 结构化-3

| | 传统结构化方法 | 面向对象方法(UML） |
|---|---|---|
| 需求模型 | 输入I、处理P、输出O的视角,<br>面向功能的文档(用户需求规格说明书)需求变化，其功能变化，所以系统的基础不稳固 | 从用户和整体角度出发<br>使用系统抽象出用例图，获取需求；如需求变化，对象的性质相对功能稳定，系统基础稳定 |
| 分析模型 | 面向过程的数据流图DFD、 实体—关系图ERD、数据字典DD表示分析模型<br>功能分解,数据和功能/过程分开 | 把问题作为一组相互作用的实体，显式表示实体间的关系<br>数据模型和功能模型一致<br>类、对象图表示分析模型，状态、顺序、协作、活动图细化说明 |
| 设计模型 | 功能模块(SC图)，模块之间的连接/调用是模块的附属形式 | 类和对象实现, 类/对象的关联、聚集、继承等连接、连接规范和约束作为显式定义 |
| 实施模型 | 体系结构设计 | 构件图，部署图 |
| 测试模型 | 根据文档进行单元测试，集成测试,确认测试 | 单元测试采用类图，集成测试用实现图和交互图，确认测试采用用例图 |

# 2.4 下一步发展方向

- (1) 组件化
- 软件生产如果象硬件生产一样能够快速高效的生产出来，而且质量有保证，那软件生产将会彻底摆脱软件危机的困扰。那软件的目标之一就是软件的组件化，希望软件系统的模块象组件一样可按需要生产、组装、调试、维护。

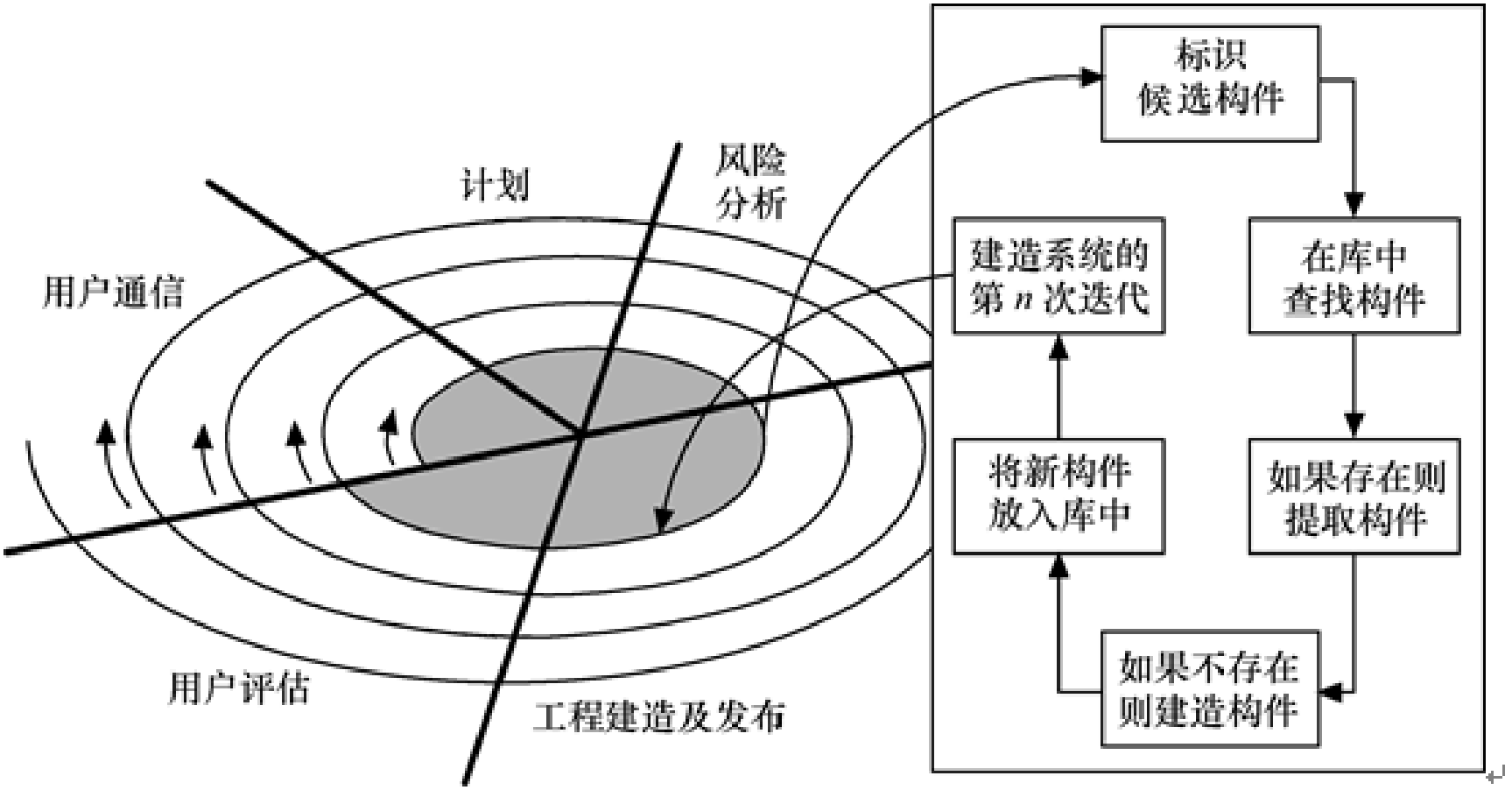
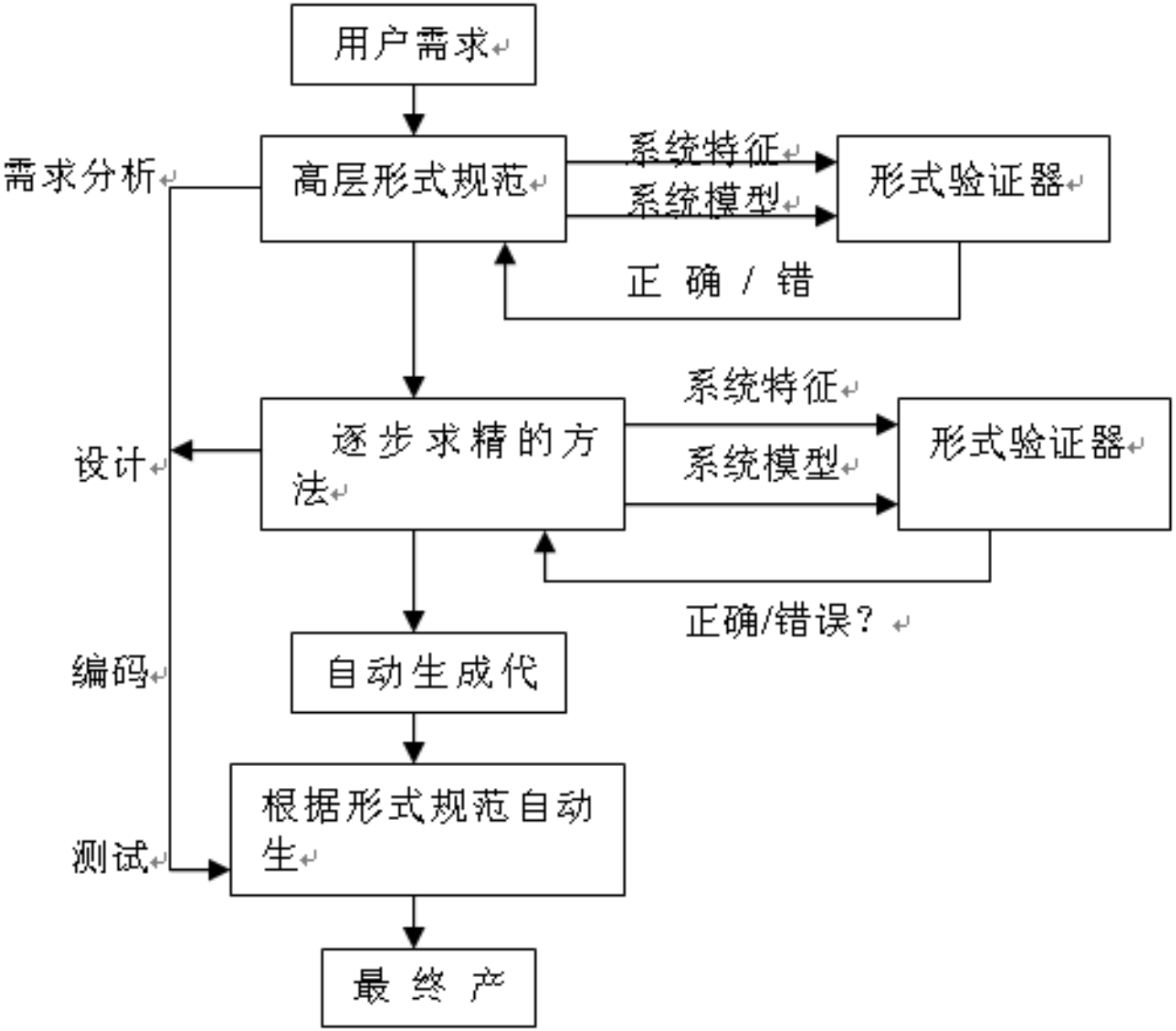

计划
风险
分析
用户通信
用户评估
工程建造及发布
标识
候选构件
在库中
查找构件
如果存在则
提取构件
如果不存在
则建造构件
将新构件
放入库中
建造系统的
第 n 次迭代

图 16.1　构件组装模型

- (2) 形式化
- 软件工程学科中引入数学的概念和定律，软件生产将会更加科学化，后面的高级课题中将进一步说明在软件开发中应用形式化方法的可行性和一般步骤。形式化方法有希望定量刻画用户的需求，并自动化的生产出所需要的软件。


用户需求
需求分析
高层形式规范
系统特征
系统模型
形式验证器
正 确 / 错
系统特征
设计
逐步求精的方法
系统模型
形式验证器
正确/错误？
编码
自动生成代
根据形式规范自动生
测试
最 终 产

- (3) 智能化
- 二十世纪是计算机科学的年代，计算机硬件结构和功能已经得到很大的发展，软件的需求还层出不穷，而且存在大量重复生产的状况，如果能够生产出一个软件产品，而且该产品能够适用于一类系统中，如图书馆系统如果生产一个而且能通用于所有单位的图书馆系统，那软件的智能化水平将更高。
- 关于智能化需要向生物学中的大脑智能学习，而二十一世纪是生命科学的年代，我们还需要向生物大脑中学习大脑智能的机理，并运用于软件工程中来，这可能还有很长的路要走。

Thank you !