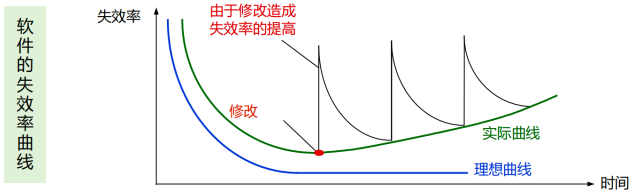
重点

软件的本质特性

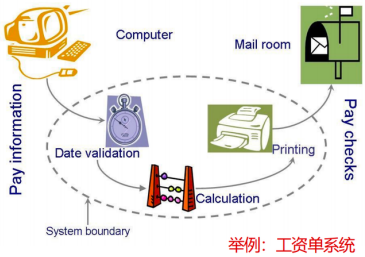
软件具有**复杂度、一致性、可变性和不可见性**等固有的内在特性，这是造成软件开发困难的根本原因。

软件的本质特性：演化性

- 人们总是认为软件是容易修改的，但忽视了修改所带来的副作用
- 不断的修改最终导致软件的退化，从而结束其生命周期



系统的本质

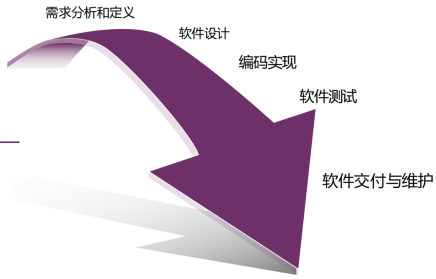


举例：工资单系统

系统的特征：

- 系统是相互联系的一组元素的集合
- 系统是具有特定功能的有机整体
- 系统是有边界的
- 系统需要与其他系统交互
- 一个系统可能包含另一个系统
- 系统是逐渐演变形成的

工程的方法



工程的特征

- 平衡与决策
- 度量与验证
- 训练有素的过程
- 团队协作与角色分工
- 系统地运用工具
- 工程原则、标准和实践
- 重用设计和设计制品

注重系统的构建过程

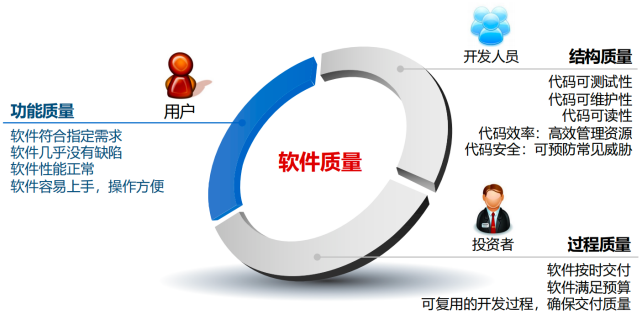
什么是软件工程

软件工程是 ① 将系统性的、规范化的、可定量的方法应用于软件的开发、运行和维护，即工程化应用到软件上；② 对①中所述方法的研究。

软件工程的基本目标：

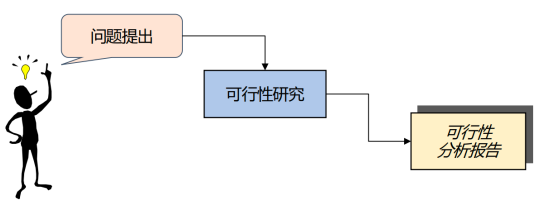
- 较低的开发成本
- 按时完成开发任务并及时交付
- 实现客户要求的功能
- 所开发软件具有良好的性能
- 较高的可靠性、可扩展性、可移植性
- 软件维护费用低

什么是好的软件



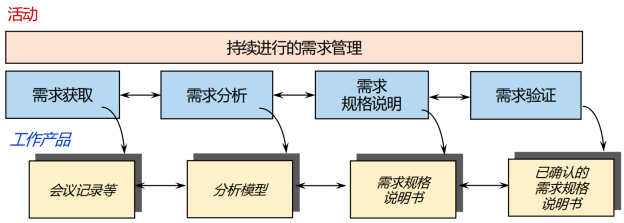
软件开发活动

问题定义：人们通过开展技术探索和市场调查等活动，研究系统的可行性和可能的解决方案，确定待开发系统的总体目标和范围。



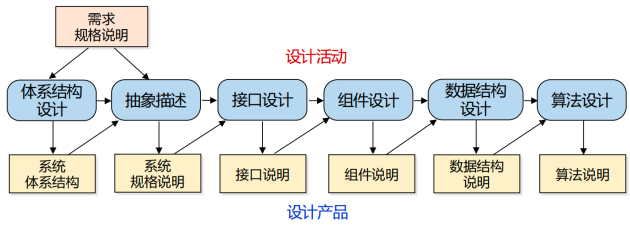
软件开发活动

需求开发：在可行性研究之后，分析、整理和提炼所收集到的客户需求，建立完整的需求分析模型，编写软件需求规格说明。



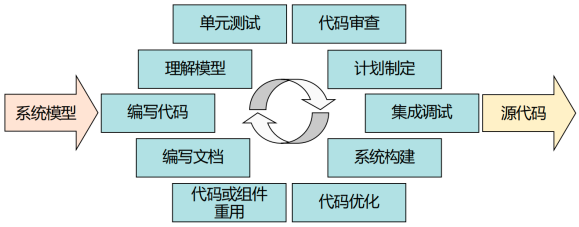
软件开发活动

软件设计：根据需求规格说明，确定软件体系结构，进一步设计每个系统部件的实现算法、数据结构及其接口等。



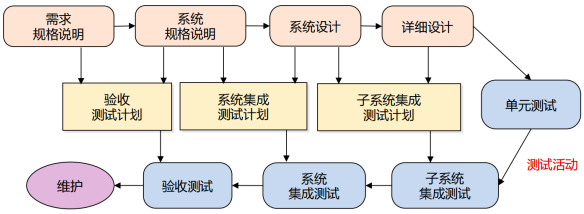
软件开发活动

软件实现：概括地说是将软件设计转换成程序代码，这是一个复杂而迭代的过程，要求根据设计模型进行程序设计以及正确而高效地编写和测试代码。



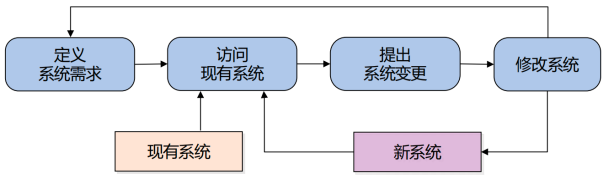
软件开发活动

软件测试：检查和验证所开发的系统是否符合客户期望，包括单元测试、子系统测试、集成测试和验收测试等。



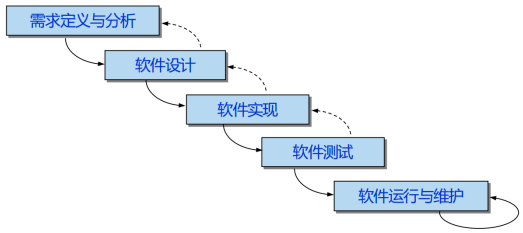
软件开发活动

软件演化：系统投入使用后对其进行改进，以适应不断变化的需求。完全从头开发的系统很少，将软件系统的开发和维护看成是一个连续过程更有意义。



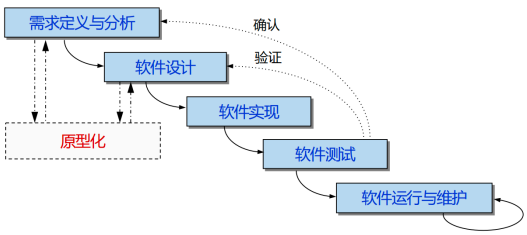
瀑布模型

瀑布模型的开发阶段严格按照线性方式进行，每一个阶段具有相关的里程碑和交付产品，且需要确认和验证。



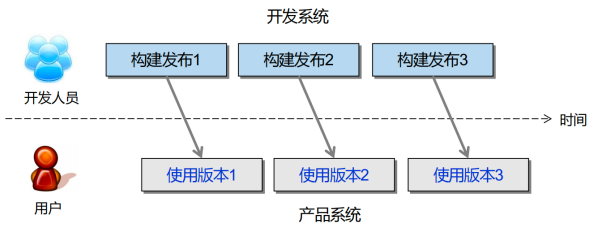
原型化模型

原型化模型需要迅速建造一个可运行的软件原型，它使用户和开发人员对系统的相关方面进行检查，以决定是否合适和恰当。



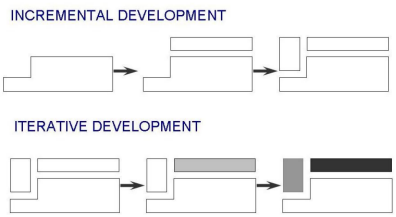
阶段化开发

今天的商业环境需要快速地推出新产品，阶段化开发使得软件系统能够一部分一部分地交付，从而缩短软件开发周期。



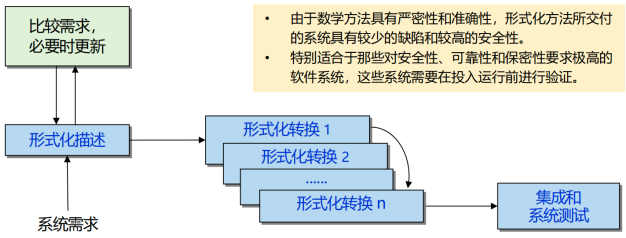
阶段化开发

增量模型：在每一个新的发布中逐步增加功能直到构造全部功能。
迭代模型：一开始提交一个完整系统，在后续发布中补充完善各子系统功能。



可转换模型

可转换模型是采用形式化的数学方法描述系统，并利用一系列转换将形式化的需求规格说明变为可交付使用的系统。



- 由于数学方法具有严密性和准确性，形式化方法所交付的系统具有较少的缺陷和较高的安全性。
- 特别适合于那些对安全性、可靠性和保密性要求极高的软件系统，这些系统需要在投入运行前进行验证。

到底什么是软件需求？

- 软件需求是利益相关方对目标软件系统的要求和期望，细分为：
  - **功能需求**：控制月球车移动并到达指定地点；
  - **性能需求**： MCS必须在1秒内完成最优路径规划；
  - **可靠性需求**： MCS平均无故障工作时间必须大于24小时；
  - **约束性需求**：MCS必须在10个月内通过验收测试；

需求工程活动

- 需求抽取（Elicitation）
- 需求分析（Analysis）
- 需求规约（Specification）
- 需求管理（Management）
- 需求验证（Validation）

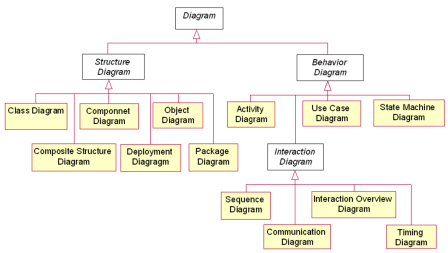
序号	敏捷方法	传统方法
1	增量价值与风险管理	在开始时尝试了解一切的分阶段方法
2	拥抱变化	防止变化
3	尽早交付，尽早失败	在结束时交付价值，在结束时失败
4	透明度	详细规划，僵化控制
5	检查与适应	带有严格控制程序和最终答案的元解决方案
6	自我管理	指挥与控制
7	持续学习	学习次于交付压力

What is the UML?

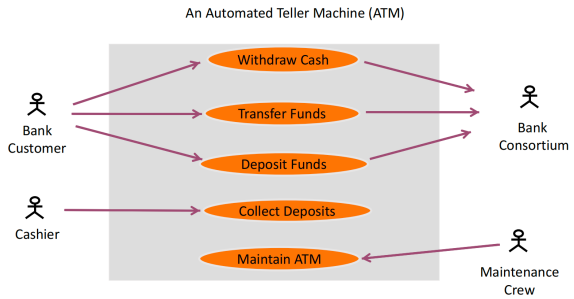
- UML stands for **Unified Modeling Language**.
- The UML is a language for
  - visualizing （可视化）
  - specifying （详述）
  - constructing （构造）
  - documenting （文档化）

UML is the de facto standard notation for software design and analysis...

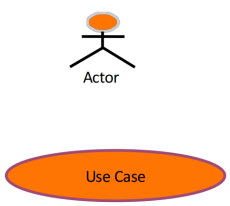
Classification of Diagrams in the UML 2.0



Use-Case Diagram



Major Use-Case Modeling Elements



Actor

Someone/something outside the system, acting in a role that interacts with the system

Use case

Represents something of value that the system does for its actors

有箭头的线条，表示角色与系统交互的过程中，数据的流向，如果箭头指向用例，就说明角色需要往系统输入数据，如果箭头指向角色，说明系统往角色输出数据。 而没有箭头的线条，则没有明确表示数据的流向，一般情况下不需要明确表示数据的流向，只需要画无箭头的线条就可以了。

选择题

1. 为了提高模块的独立性，模块内部最好是（ C ）。//外部耦合，功能内聚

A．逻辑内聚　　B．时间内聚　　C．功能内聚　　D．通信内聚

1. 模块的内聚性最高的是（ D ）。

A．逻辑内聚　　B．时间内聚　C．　偶然内聚　　D．功能内聚

1. 下面哪些测试方法属于白盒测试（ A E ）。//BCD是黑盒测试

A、基本路径测试　B、等价类划分　C、边界值分析　D、错误推测　E、逻辑覆盖测试

1. UML是软件开发中的一个重要工具，它主要应用于哪种软件开发方法(C)

A、基于瀑布模型的结构化方法　B、基于需求动态定义的原型化方法

C、基于对象的面向对象的方法　D、基于数据的数据流开发方法

1. 可行性研究要进行一次（D）需求分析。

A．深入的 B．详尽的 C．彻底的 D．简化的、压缩了的

填空题

27. 传统软件维护一般分为 4 大类，分别是**纠错性维护**、**适应性维护**、**完善性维护和预防性维护**。

28. 在 V 测试模型中，编码结束后，首先作**单元测试**，然后是**集成测试**、**系统测试和** 验收测试 。

31. 用例之间的关系主要有三种：**包含（include）、扩展（extend）和继承。**

简答题

1. 简述软件设计的过程。

软件设计是把许多事物和问题抽象起来，并且抽象它们不同层次和角度，是将需求转变为软件陈述的过程，是迭代的过程。

1. 简述结构化程序设计方法的基本特点。

1）尽可能少用goto语句的程序设计方法。

2）每个代码块都单入单出。

3）使用控制结构：顺序、选择和循环。

4） 自顶向下逐步求精。

1. 什么是黑盒测试法?

黑盒测试法把程序看成一个黑盒子，完全不考虑程序的内部结构和处理过程（2分）。

它只检查程序功能是否能按照规格说明书的规定正常使用（1分），

程序是否能适当地接收输入数据（1分），

产生正确地输出信息（1分）。

1. 简述面向对象的4个要点的含义。

面向对象的 4 个要点是：对象分解、数据专有、继承、封装性。（1 分）

1）对象分解：认为客观世界是由各种对象组成的。 （1 分）

2）数据专有，方法共享：把所有对象都划分成各种对象类(简称为类，class)，每个对象类都定义了一组数据和一组方法。 （1 分）

3）继承：按照子类(或称为派生类)与父类(或称为基类)的关系，把若干个对象类组成一个层次结构的系统(也称为类等级)。（1 分）

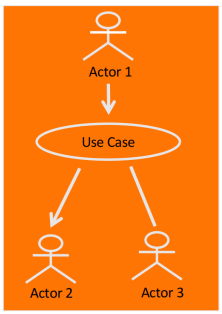
4）封装性：对象彼此之间仅能通过传递消息互相联系。（1 分）

需求工程中的一些问题，比如针对特定的情境如何获取用户需求，比如问卷法、访谈法等。

SRS目录结构?

UML图，针对拍卖管理系统的功能模型、类图，状态变迁图

DFD，最短完成时间甘特图



- A channel of communication between an actor and a use case.
- A line is used to represent a communicates-association.
  - An arrowhead indicates who initiates each interaction.
  - No arrowhead indicates either end **can** initiate each interaction.

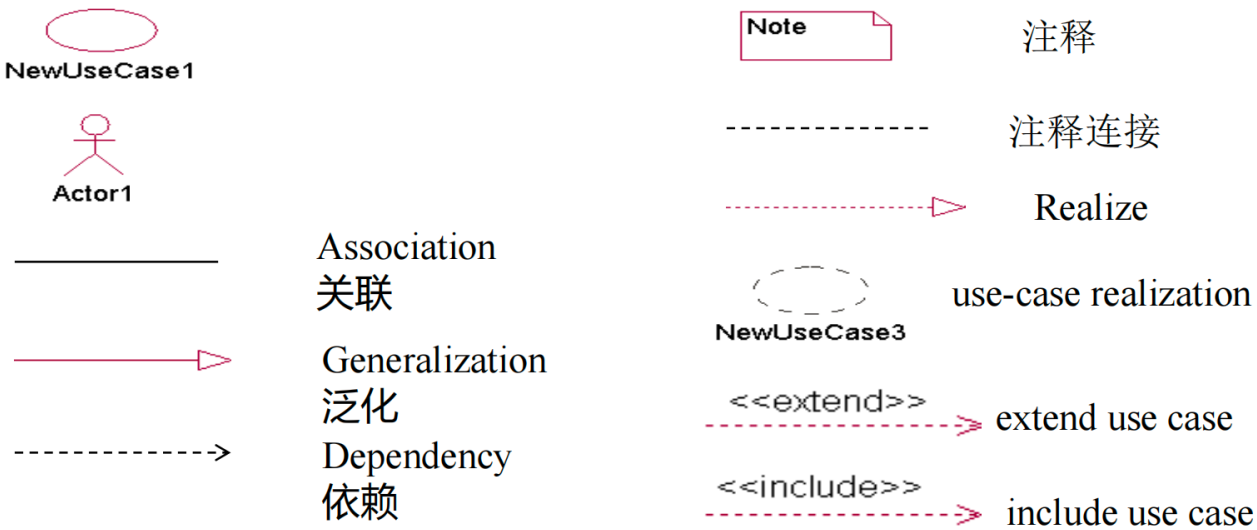
How Should I Name a Use Case?

- Indicate the value or goal of the actor.
- Use the active form; begin with a verb.
- Imagine a to-do list.
- Examples of variations
  - Register for Courses
  - Registering for Courses
  - Acknowledge Registration
  - Course Registration

Which variations show the value to the actor? Which do not? Which would you choose as the use-case name? Why?

用例图

用例图中的一些主要图标



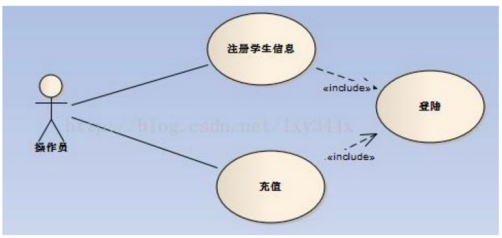
说明：UML中不使用颜色来作为图形语义的区分标记。

Relationships between Use Case

- Use Case除了和参与者有**关联(association)**外，Use Case之间也存在着一定的关系(relationship)。包括：**泛化(generalization)关系**、**包含(include)关系**、**扩展(extend)关系**等。
- 也可以利用UML的扩展机制自定义Use Case间的关系。
- **relationship(关系)**, **association(关联)**, **generalization(泛化)**, **dependency(依赖)**的区别。
  - association, generalization, dependency都属于relationship。
  - include，extend 属于 dependency。

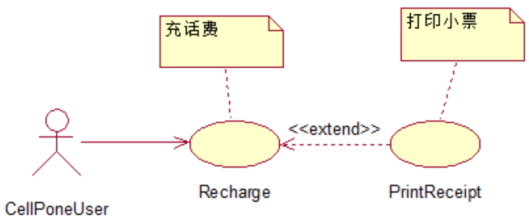
When to use Includes？何时使用包含关系？

- You have a piece of behavior that is similar across many use cases （多个用例有共享行为）
- Break this out as a separate use-case and let the other ones "include" it （为共享行为单独创建用例）
- Examples include
  - Login (登录)



二、扩展关系（extend）

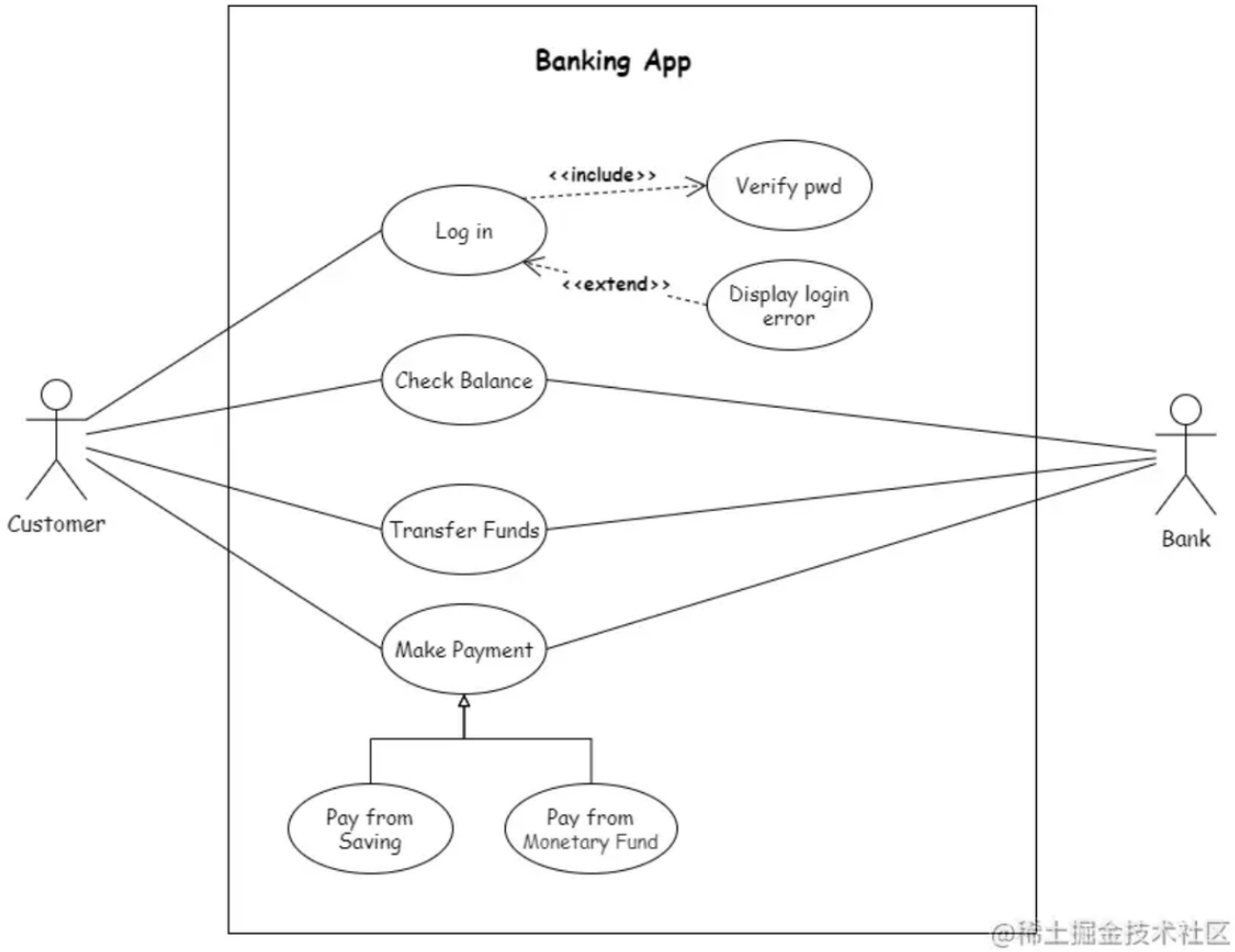
扩展关系用一个虚箭头外加版型《extend》表示，由扩展用例指向被扩展用例



扩展关系可以基于以下理由：

1、表明用例的某一部分是可选的系统行为，这样就可以将用例图中的可选行为和必选行为分开。
2、表明只在特定条件下才执行的特定分支用例
3、表明多个基本用例中都有可能触发的某个可选用例

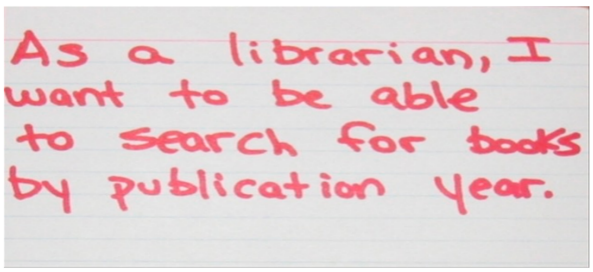
extend关系和include关系最明显的区别就是：扩展用例是可选的，包含用例是必选的，如上图所示：手机用户在用自动缴费机充值之后，可以打印小票，也可以不打印，这完全取决于用户的意愿，并不是必须要执行的。



实线白箭头：泛化==继承

User Story Typical Format

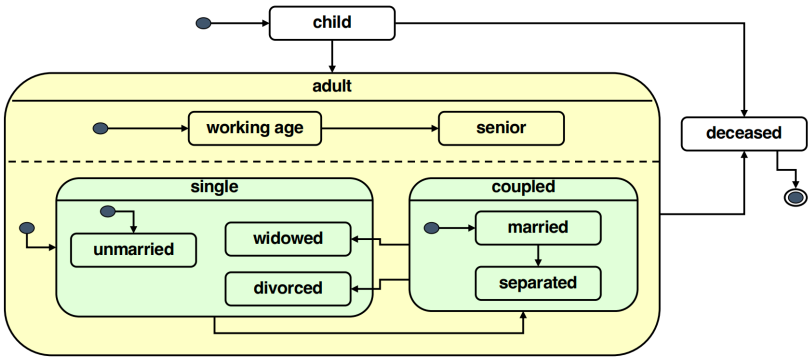
As a <type of user>, I want <some goal> so that <some reason>.



3C's

- A well written user story will describe what the desired functionality is, who it is for, and why it is useful.
- There are 3 parts to a fully fleshed out user story. If you like marketing-speak, then you can call them the "3 C's":
  - The Card
  - The Conversation
  - The Confirmation

A more detailed example



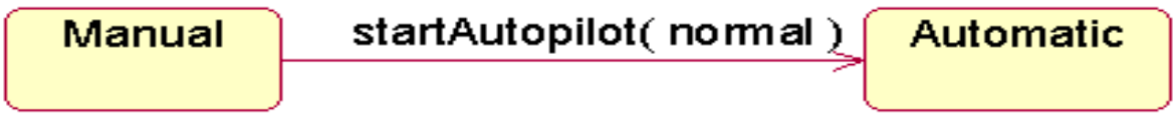
(1) Call event: The event of receiving a call for an operation that is implemented by actions on state machine transitions.

- Call event的语法格式如下：

  事件名（[逗号分隔的参数列表]）

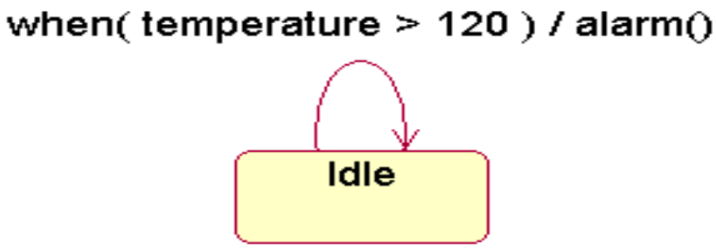
其中参数列表中的参数格式为：

例：



(2) Change event: The event of a Boolean expression becoming satisfied because of a change to one or more of the values it references.

- Change event用关键字when表示。例

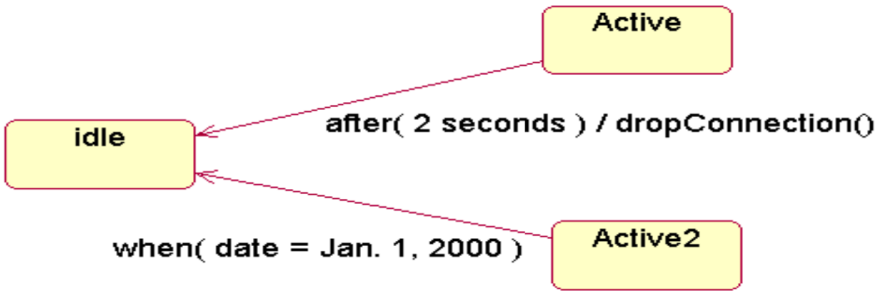


- Change event和警戒条件(guard condition)的区别：
  - 警戒条件只在所相关的事件出现后计算一次，如果值为false，则不进行状态转移。

(3) Time event: An event that denotes the satisfaction of a time expression, such as the occurrence of an absolute time or the passage of a given amount of time after an object enters a state.

- Time event用关键字after或when表示。

例:

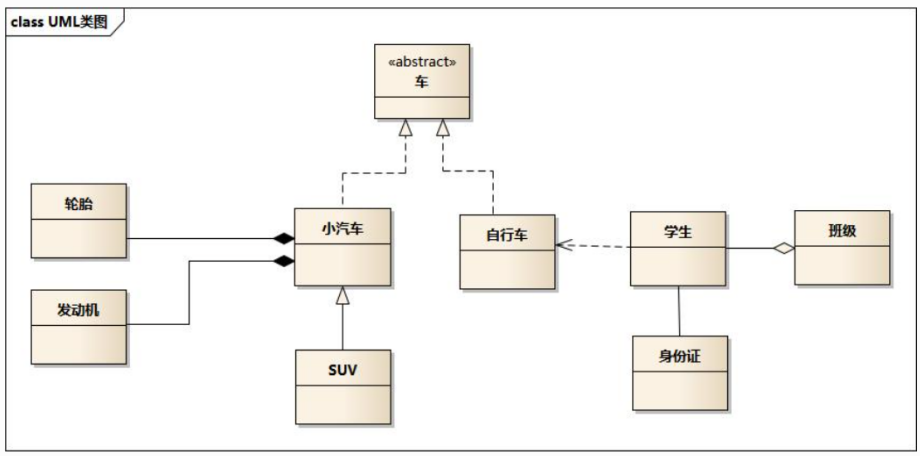


(4) Signal event: An event that is the receipt by an object of a signal sent to it, which may trigger a transition in its state machine.

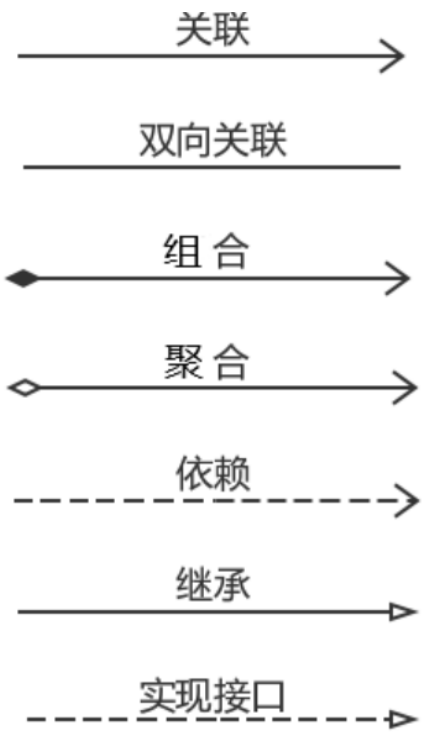
- Signal event的语法格式和Call event一样。

- 信号事件是一个异步事件，调用事件一般是一个同步事件。

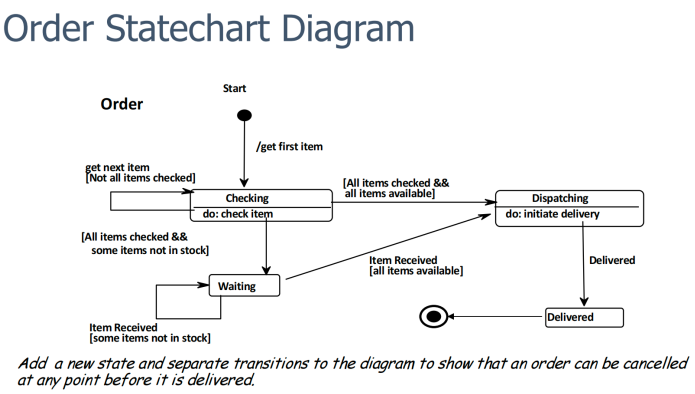
类图

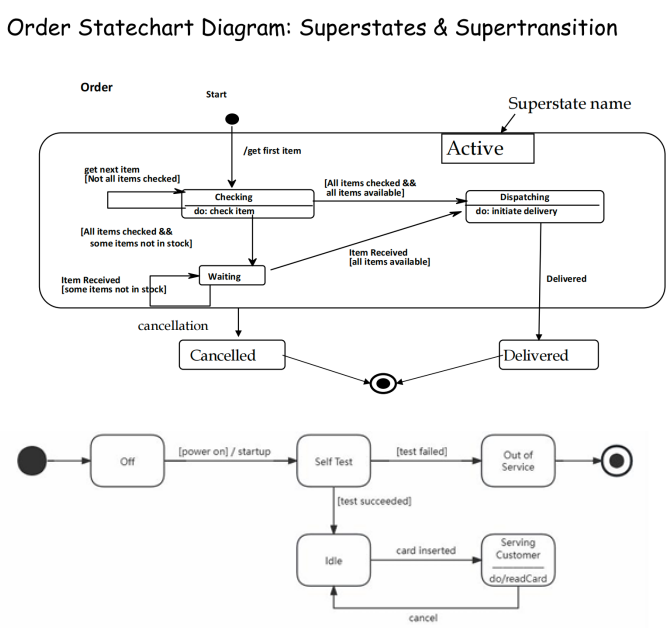


- 车的类图结构为<<abstract>>，表示车是一个抽象类；
- 它有两个继承类：小汽车和自行车；它们之间的关系为实现关系，使用带空心箭头的虚线表示；
- 小汽车为与SUV之间也是继承关系，它们之间的关系为泛化关系，使用带空心箭头的实线表示；
- 小汽车与发动机之间是组合关系，使用带实心箭头的实线表示；
- 学生与班级之间是聚合关系，使用带空心箭头的实线表示；
- 学生与身份证之间为关联关系，使用一根实线表示；
- 学生上学需要用到自行车，与自行车是一种依赖关系，使用带箭头的虚线表示；



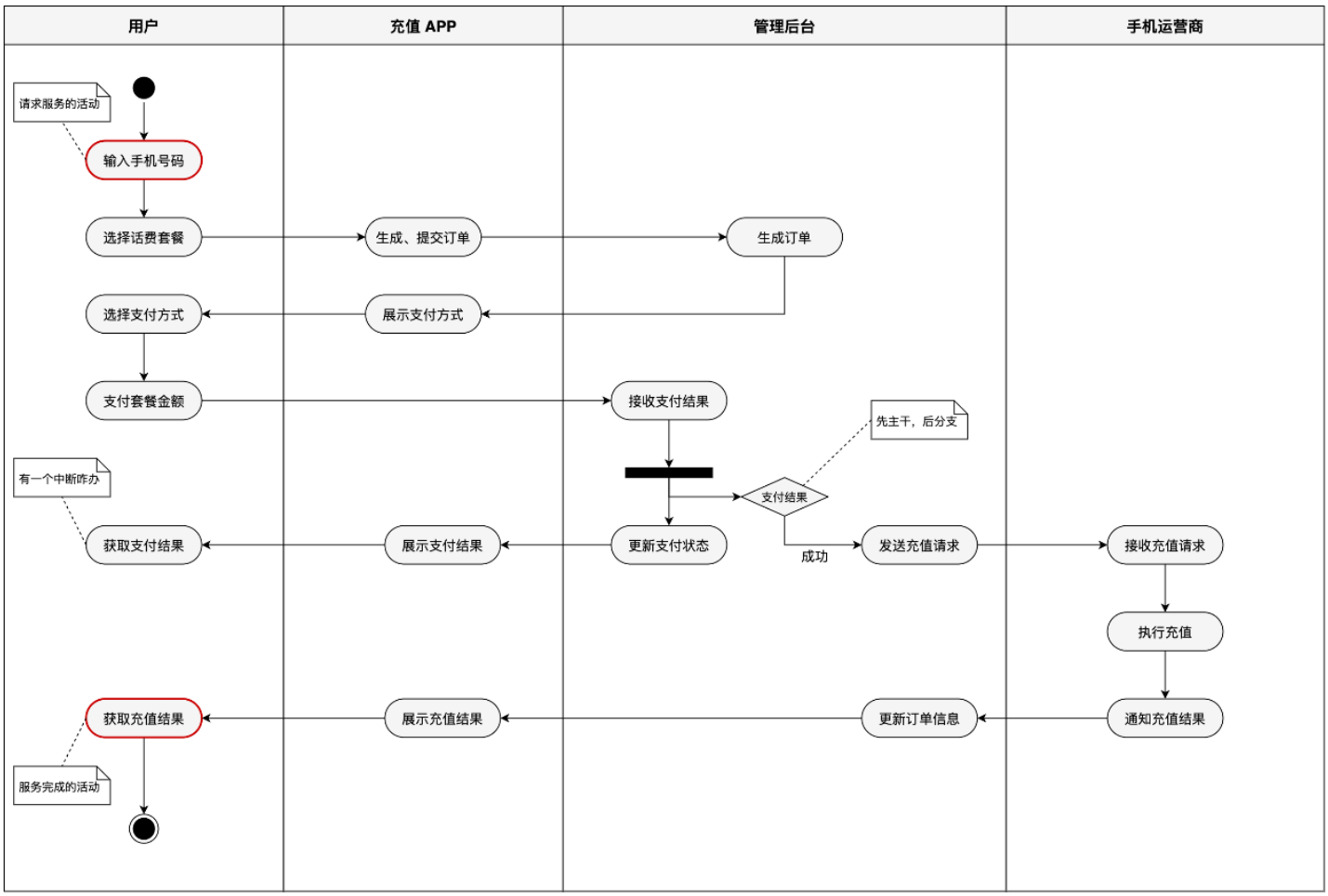
状态图


Order Statechart Diagram
Order
Start
/get first item
get next item
[Not all items checked]
Checking
do: check item
[All items checked &&
all items available]
Dispatching
do: initiate delivery
[All items checked &&
some items not in stock]
Waiting
Item Received
[all items available]
Delivered
Item Received
[some items not in stock]
Delivered
Add a new state and separate transitions to the diagram to show that an order can be cancelled at any point before it is delivered.


Order Statechart Diagram: Superstates & Supertransition
Order
Start
Superstate name
Active
/get first item
get next item
[Not all items checked]
Checking
do: check item
[All items checked &&
all items available]
Dispatching
do: initiate delivery
[All items checked &&
some items not in stock]
Item Received
[all items available]
Waiting
Item Received
[some items not in stock]
Delivered
cancellation
Cancelled
Delivered

Off
[power on] / startup
Self Test
[test failed]
Out of Service
[test succeeded]
Idle
card inserted
Serving Customer
do/readCard
cancel

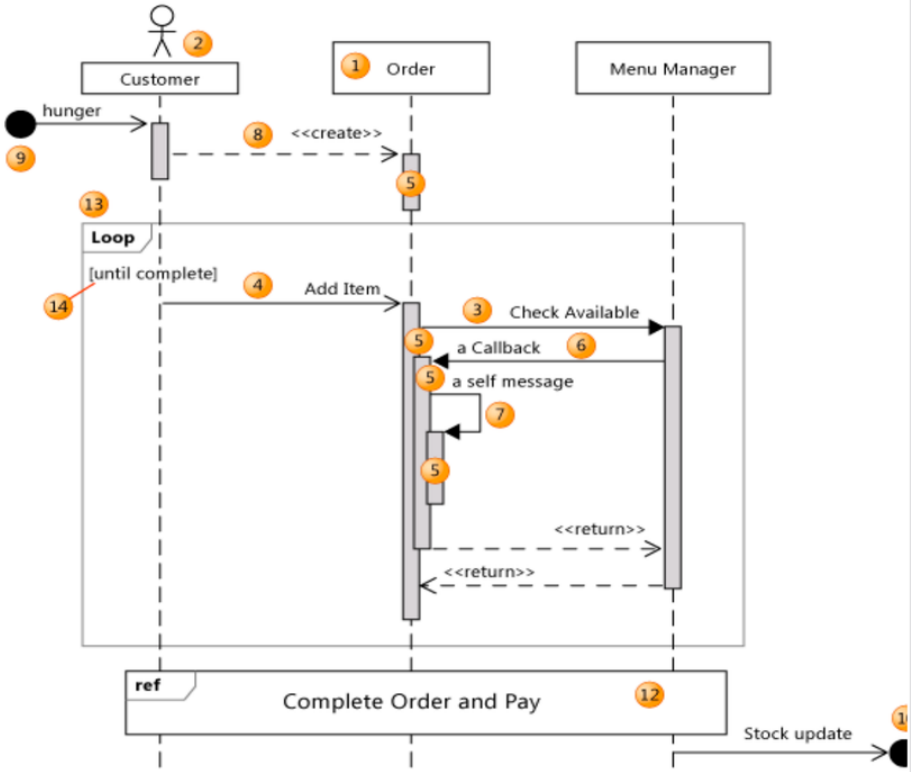
活动图


用户
充值 APP
管理后台
手机运营商
请求服务的活动
输入手机号码
选择话费套餐
生成、提交订单
生成订单
选择支付方式
展示支付方式
支付套餐金额
接收支付结果
先主干，后分支
有一个中断咋办
支付结果
获取支付结果
展示支付结果
更新支付状态
成功
发送充值请求
接收充值请求
执行充值
获取充值结果
展示充值结果
更新订单信息
通知充值结果
服务完成的活动

顺序图

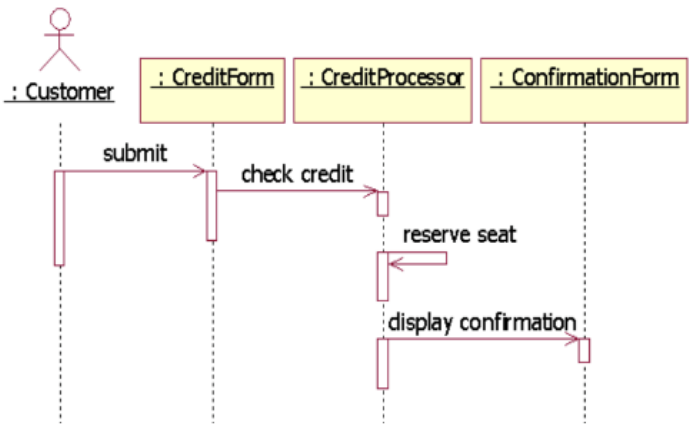
Key parts of a SD

- **participant**: an object or entity that acts in the sequence diagram
  - sequence diagram starts with an unattached "found message" arrow
- **message**: communication between participant objects
- the axes in a sequence diagram:
  - **horizontal**: which object/participant is acting
  - **vertical**: time (down -> forward in time)



选择C

2. 关于下面顺序图正确的说法是（　　）。
   A. 类 CreditProcessor 是控制类，因此它向外发出很多消息。
   B. 类 CreditProcessor 必须实现 checkCredit()，reserveSeat() 和 displayConfirmation() 方法。
   C. CreditForm 为边界类。
   D. ConfirmationForm 为实体类。



- OOA offers five kinds of concepts: objects, attributes, structures, services and subjects.

Sequence Diagram and Use Cases顺序图与用例的关系 II

- A Sequence Diagram can be seen as an expansion of a use case to the lowest possible level of detail.

  顺序图可帮助分析人员对用例图进行扩展、细化和补遗
- Sequence diagrams can be drawn at different levels of details and also to meet different purposes at several stages in the development life-cycle.

  顺序图可用于开发周期的不同阶段，服务于不同目的，描述不同粒度的行为
- Analysis Sequence Diagrams normally **do not**
  - include design objects 分析阶段的顺序图不包含设计对象
  - Specify message signatures in any detail 分析阶段的顺序图不关注消息参数

Drawing Sequence Diagram 绘制顺序图

- The name of the class and the name of the instance of that class (object), if required, at the top of the column that represents it.

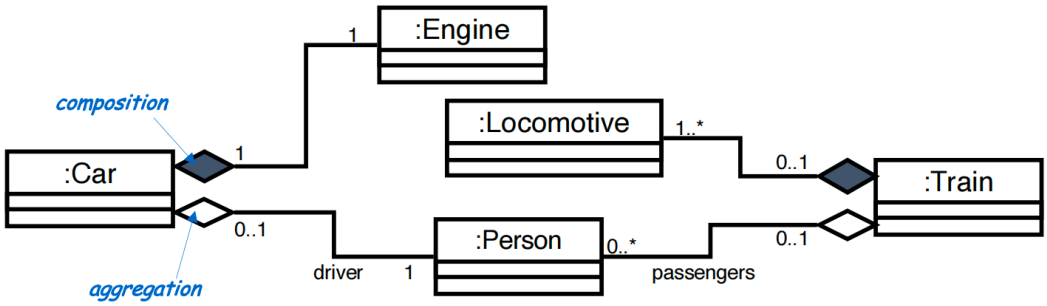
  在顺序图顶端写出类名和实例（对象）名
- Including the arrows marked Return makes it easier to follow the flow of control from object to object.

  在图中包含表示返回信息的箭头便于跟踪对象间的控制流
- The vertical line for each object represents its lifeline.

  每个对象下面的竖线表示该对象的生命线。
- The thick bar represents its activation. 生命线上的矩形框表示对象的激活期
  - i.e. when it is doing something 即正在执行某项操作

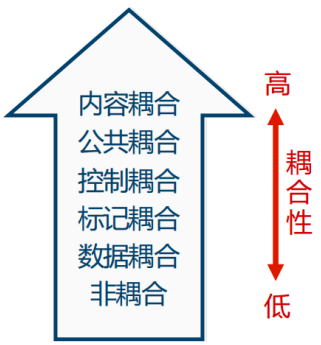
Aggregation and Composition

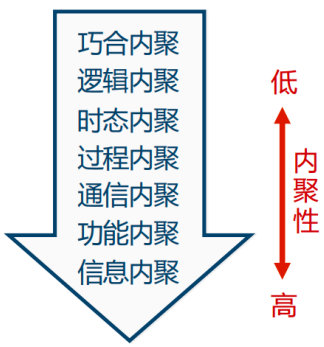
- Aggregation
  - This is the "Has-a" or "Whole/part" relationship
- Composition
  - Strong form of aggregation that implies ownership:
    - if the whole is removed from the model, so is the part.
    - the whole is responsible for the disposition of its parts



软件设计原则

模块化（也称关注点分离）是将系统中各不相关的部分进行分离的原则，以便于各部分能够独立研究。





solid原则：

1. **单一职责原则（Single Responsibility Principle, SRP）**

   定义： 一个类应该只有一个引起变化的原因（即一个类只负责一个职责）。

2. **开放-封闭原则（Open/Closed Principle, OCP）**

   定义： 软件实体（类、模块、函数）应该对扩展开放，对修改封闭。

3. **里氏替换原则（Liskov Substitution Principle, LSP）**

   定义： 子类应该可以替换父类，并且行为保持一致，不破坏程序的正确性。

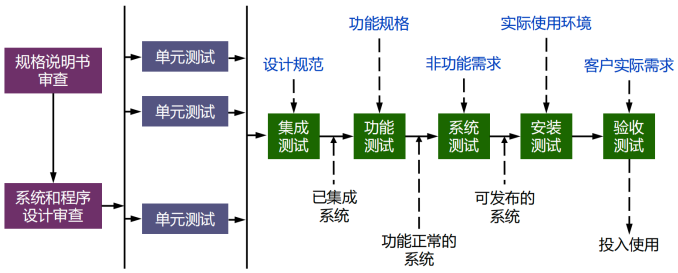
4. **接口隔离原则（Interface Segregation Principle, ISP）**

   定义： 一个类不应该被强迫实现它用不到的接口（即多个特定接口比一个通用接口更好）。

5. **依赖倒置原则（Dependency Inversion Principle, DIP）**

   定义： 高层模块不应该依赖于低层模块，两者都应该依赖于抽象（接口或抽象类）。

软件测试阶段



集成测试

集成测试（Integration Testing）是在单元测试的基础上，将所有模块按照总体设计的要求组装成为子系统或系统进行的测试。

- 一次性集成方式：分别测试每个单元，再一次性将所有单元组装在一起进行测试。
- 渐增式集成方式：先对某几个单元进行测试，然后将这些单元逐步组装成较大的系统，在组装过程中边连接边测试。

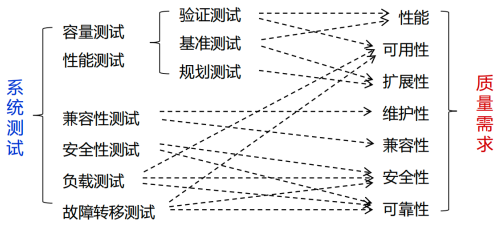
集成测试的对象是模块间的接口，其目的是找出在模块接口上，包括系统体系结构上的问题。

功能测试

功能测试（Functional Testing）是在已知产品所应具有的功能基础上，从用户角度来进行功能验证，以确认每个功能是否都能正常使用。

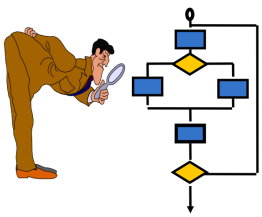
系统测试

系统测试（System Testing）是在实际运行环境或模拟实际运行环境下，针对系统的非功能特性所进行的测试，包括负载测试、性能测试、恢复测试、安全测试和可靠性测试等。



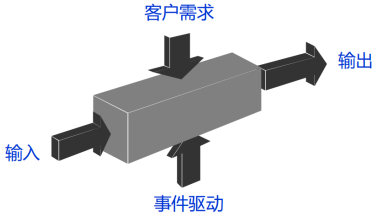
白盒测试

白盒测试（White Box Testing）：又称结构测试，它把测试对象看做一个透明的盒子，允许测试人员利用程序内部的逻辑结构及有关信息，设计或选择测试用例，对程序所有逻辑路径进行测试。



黑盒测试

黑盒测试（Black Box Testing）：又称功能测试，它将测试对象看做一个黑盒子，完全不考虑程序内部的逻辑结构和内部特性，只依据程序的需求规格说明书，检查程序的功能是否符合它的功能说明。



白盒测试方法

白盒测试需要测试人员阅读和理解被测对象的实现代码及其结构，适当地选择一些执行路径，并且观察所选择的路径是否产生了预期的结果。

黑盒测试方法

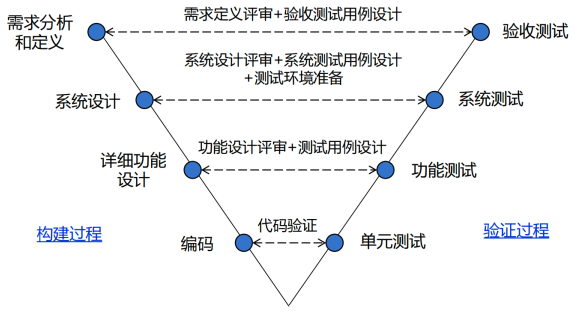
黑盒测试并不是白盒测试的替代品，而是用于辅助白盒测试发现其他类型错误。通常由独立测试人员根据用户需求文档来进行，但不一定要求用户参与。

静态测试与动态测试

静态测试：通过人工分析或程序正确性证明的方式来确认程序正确性。

动态测试：通过动态分析和程序测试等方法来检查程序执行状态，以确认程序是否有问题。

V 模型



举例：判断三角形类型

	规则	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
条件桩	C1：a、b、c构成三角形？	Y	Y	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N	N	N
	C2：a=b?	Y	Y	Y	Y	N	N	N	N	Y	Y	Y	Y	N	N	N	N
	C3：a=c?	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N
	C4：b=c?	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N	Y	N
动作桩	A1：非三角形									Y	Y	Y	Y	Y	Y	Y	Y
	A2：不规则三角形								Y								
	A3：等腰三角形				Y		Y	Y									
	A4：等边三角形	Y															
	A5：不符合逻辑		Y	Y		Y											

举例：ATM取款

测试用例设计

序号	场景	PIN	账号	取款金额	账面金额	ATM现金	预期结果
1	场景1：成功提款	V	V	V	V	V	成功提款
2	场景2：ATM里没有现金	V	V	V	V	X	提款选项不可用，用例结束
3	场景3：ATM里现金不足	V	V	V	V	X	警告信息，返回基本流相应步骤，重新输入金额
4	场景4：PIN有误（还有不止一次机会）	X	V	n/a	V	V	警告信息，返回基本流相应步骤，重新输入PIN
5	场景4：PIN有误（还有一次机会）	X	V	n/a	V	V	警告信息，返回基本流相应步骤，重新输入PIN
6	场景5：PIN有误（不再有输入机会）	X	V	n/a	V	V	警告信息，卡被没收，用例结束
	.....						

面向对象设计原则

- 单一职责原则
- 开放封闭原则
- Liskov替换原则
- 接口分离原则
- 依赖倒置原则

软件能力成熟度模型（CMM）基本概念：

CMM等级

能力等级	特点	关键过程
第一级 初始级（最低级）	软件工程管理制度缺乏，过程缺乏定义、混乱无序。成功依靠的是个人的才能和经验，经常由于缺乏管理和计划导致时间、费用超支。管理方式属于反应式，主要用来应付危机。过程不可预测，难以重复。	
第二级 可重复级	基于类似项目中的经验，建立了基本的项目管理制度，采取了一定的措施控制费用和时间。管理人员可及时发现问题，采取措施。一定程度上可重复类似项目的软件开发。	需求管理,项目计划,项目跟踪和监控,软件子合同管理,软件配置管理,软件质量保障
第三级 已定义级	已将软件过程文档化、标准化，可按需要改进开发过程，采用评审方法保证软件质量。可借助CASE工具提高质量和效率。	组织过程定义,组织过程焦点,培训大纲,软件集成管理,软件产品工程,组织协调,专家审评
第四级 已管理级	针对制定质量、效率目标，并收集、测量相应指标。利用统计工具分析并采取改进措施。对软件过程和产品质量有定量的理解和控制。	定量的软件过程管理和产品质量管理
第五级 优化级（最高级）	基于统计质量和过程控制工具，持续改进软件过程。质量和效率稳步改进。	缺陷预防,过程变更管理和技术变更管理

CMM能力成熟度各级特点和关键过程。 [3]

敏捷开发