



# 软件架构概述

lliao@seu.edu.cn

# 目 录

- 软件架构概论
- 软件架构建模方法
- 软件架构风格
- 基于架构的软件开发
- 基于架构的评估



# 第I章: 概 论

- (一) 软件危机
- (二) 软件架构的兴起和发展
- (三) 软件重用
- (三) 面向对象与重用
- (四) 构件与重用

# 一、软件危机

- 第一次软件危机（ **1945 to 1968**）

- 危机表现

- 软件生产效率低下,软件成本高: 软件开发成本和进度的估计常常很不准确。
- 软件产品质量差: 用户对“已完成的”软件系统不满意的现象经常发生。
- 软件产品难以(不能)维护: 软件通常没有适当的文档资料

# 一、软件危机

- 第一次软件危机（ **1945 to 1968** ）
  - 危机根源
    - 人员知识结构单一
    - 手工作坊： **code and fix**
    - 理论体系不成熟[软件开发]: 语言、工具、过程、方法、质量保证.....
    - 软件开发管理不完善： 规划、预算、可行性分析、风险预测与控制

# 一、软件危机

- 第二次软件危机（ **1968 to 2001** ）
  - 危机表现
    - The project will produce the wrong product[有错的产品].
    - The project will produce a product of inferior quality[低质量的产品].
    - The project will be late[延迟].
    - We'll have to work 80 hour weeks[每周工作80小时].
    - We'll have to break commitments[违约].
    - We won't be having fun[没有乐趣]

# 一、软件危机

- 第二次软件危机（ **1968 to 2001** ）
  - 危机**根源**
    - The assumption is that requirements can be fully understood prior to development
    - Interaction with the customer occurs only at the beginning(requirements) and end (after delivery)
    - Unfortunately the assumption almost never holds

# 一、软件危机

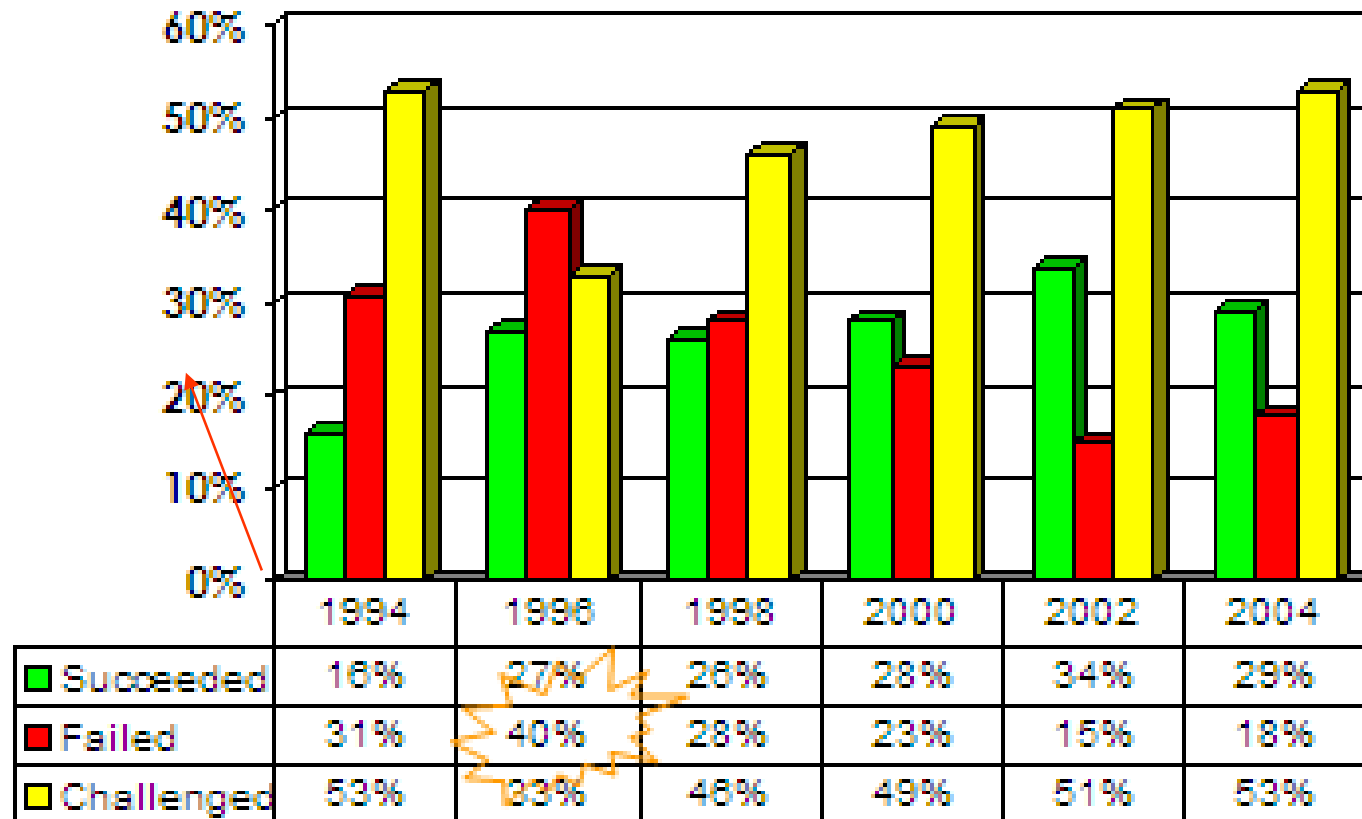
- 第三次软件危机（**2001**年至今）
  - 危机表现
    - 变更/演化/重用（质量、效率、成本）
    - 知识产权问题、安全问题
  - 危机根源
    - 大量遗产系统的挖掘和重用
    - 大量开源软件/代码的使用
    - 普遍存在的动态/不确定需求



# 软件危机数据——项目拖延或取消的事例



## *CHAOS Research* STUDIES



# 软件危机数据

## MODERN RESOLUTION FOR ALL PROJECTS

	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

*The Modern Resolution (OnTime, OnBudget, with a satisfactory result) of all software projects from FY2011-2015 within the new CHAOS database. Please note that for the rest of this report CHAOS Resolution will refer to the Modern Resolution definition not the Traditional Resolution definition.*

## CHAOS RESOLUTION BY PROJECT SIZE

	SUCCESSFUL	CHALLENGED	FAILED
Grand	2%	7%	17%
Large	6%	17%	24%
Medium	9%	26%	31%
Moderate	21%	32%	17%
Small	62%	16%	11%
TOTAL	100%	100%	100%

*The resolution of all software projects by size from FY2011-2015 within the new CHAOS database.*



## 二、软件架构的兴起和发展

- 1.背景分析
- 2.软件架构定义 - 百家之言
- 3.软件架构的意义
- 4.软件架构发展史

## 二、软件架构的兴起和发展

- I. 背景分析
  - 小规模软件开发：算法 + 数据结构 **is enough!**
  - 大规模软件开发：总体的系统设计和规格说明 **must be more important!**  
⇒ 软件架构 ( **Software architecture** )

## 二、软件架构的兴起和发展

### ❁ 2. 软件架构定义 - 百家之言

#### (1) Dewayne Perry和Alex Wolf的定义 (1992)

软件架构是具有某种特定形式的结构化元素(即构件)的集合, 包括处理构件、数据构件和连接构件。处理构件负责对数据进行加工, 数据构件是被加工的信息, 连接构件把架构的不同部分组合连接起来。(A set of architectural (or, if you will, design) elements that have a particular form. Perry and Wolf distinguish between processing elements, data elements, and connecting elements.)

## 二、软件架构的兴起和发展

### ❁ 2. 软件架构定义 - 百家之言

#### (2) Mary Shaw 和 David Garlan 的定义 (1993)

软件架构是软件设计过程中的一个层次，这一层次超越计算过程中的算法设计和数据结构设计。架构问题包括总体组织和全局控制、通信协议、同步、数据存取，给设计元素分配特定功能，设计元素的组织、规模和性能，在各设计方案间进行选择等。软件架构处理算法与数据结构关于整体系统结构设计和描述方面的一些问题，如全局组织和全局控制结构，关于通信、同步与数据存取的协议，设计构件功能定义，物理分布与合成，设计方案的选择、评估与实现等。

## 二、软件架构的兴起和发展

### ❁ 2. 软件架构定义 - 百家之言

#### (3) Hayes Roth的定义 (1994)

软件架构是一个抽象的系统规约，主要包括用其行为来描述的功能构件和构件之间的相互连接、接口和关系。

( ...an abstract system specification consisting primarily of functional components described in terms of their behaviors and interfaces and component-component interconnections. )



## 二、软件架构的兴起和发展

### ❁ 2. 软件架构定义 - 百家之言

(4) Booch, Rumbaugh, and Jacobson (1999)

An architecture is the set of **significant decisions** about the organization of a software system, **the selection** of the structural elements and their interfaces by which the system is composed, **together with their behavior** as specified in the collaborations among those elements, **the composition** of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization---these elements and their interfaces, their collaborations, and their composition (The UML Modeling Language User Guide, Addison-Wesley, 1999).

## 二、软件架构的兴起和发展

### ❁ 2. 软件架构定义 - 百家之言

( 5 ) Bass, Clements, Kazman(from "Software architecture in practice" (second edition))(2003)

The software architecture of a program or computing system is the structure or structures of the system, which comprise **software elements**, the **externally visible properties** of those elements, and **the relationships** among them.

"Externally visible" properties refers to those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.

## 二、软件架构的兴起和发展

### ❁ 2. 软件架构定义 - 百家之言

(6) 综上所述，可以认为软件架构由3种基本元素组成，它们是：

- 构件（component）
- 连接件（connector）
- 配置（configuration）/规约

## 二、软件架构的兴起和发展

### ❁ 2. 软件架构定义

#### ❁ 构件（component）

- ❁ 构件是计算或数据存储的单位，大致对应于常规编程语言的编译单元和其他用户级对象。
- ❁ 它可以是简单型或复合型，例如一个系统就可以看成是复核型构件。
- ❁ 构件作为一个封装的实体，只能通过其接口与外部交互，构件的接口由一组端口组成，每个端口表示了构件与外部的交互点。

## 二、软件架构的兴起和发展

### ❁ 2. 软件架构定义

#### ● 连接件（connector）

- ❁ 连接件是构件之间的交互及控制这些交互的规则。
- ❁ 它们有时是简单交互，如过程调用、共享变量访问；有的则是复杂的和语义丰富的交互，如C/S协议等。
- ❁ 连接件通常不能与编译单元各自相对应。
- ❁ 它们的表现形式可以是表项、缓冲区、动态数据结构、调用、初始化参数等。

## 二、软件架构的兴起和发展

### ❁ 2. 软件架构定义

- 配置（configuration）
  - 也可以看做是“约束/拓扑结构”
  - 软件架构的拓扑结构反映了该结构的基本准则

## 二、软件架构的兴起和发展

### ❁ 3. 软件架构的意义

(1) 架构是风险承担者进行交流的手段

—SA代表了系统的公共的高层次的抽象。

(2) 架构是早期设计决策的体现

a. 软件架构明确了对系统实现的约束条件——架构设计者不必是算法设计者或编程语言，只需重点考虑系统的总体权衡问题，而构件的开发人员在架构给定的约束下进行开发。



## 二、软件架构的兴起和发展

### ❁ 3. 软件架构的意义

#### (2) 架构是早期设计决策的体现

- b. 软件架构决定了开发和维护组织的组织结构——SA决定组织结构
- c. 软件架构制约着系统的质量属性——好的SA是必要条件
- d. 通过研究软件架构可能预测软件的质量——评价SA预测质量
- e. 软件架构使推理和控制更改更简单——SA需易更改
- f. 软件架构有助于顺序渐进的原型设计——按可执行模型来构造原型
- g. 软件架构可以作为培训的基础



## 二、软件架构的兴起和发展

### ❁ 3. 软件架构的意义

#### (3) 软件架构是可传递和可重用的模型

a. SA体现了一个相对来说比较小又可理解的模型；

b. SA级的重用意味着SA的决策能在具有相似需求的多个系统中发生影响，这比代码级的重用要有更大的好处。

## 二、软件架构的兴起和发展

### ❁ 4. 软件架构发展史

- ❁ 20世纪70年代以前：汇编语言为主 - 规模小 - 不考虑系统结构；
- ❁ 70年代中后期：结构化方法（概要设计 - 详细设计） - 数据流设计和控制流设计 - 出现软件结构；
- ❁ 20世纪80年代初到90年代中期：面向对象方法（数据和基于数据之上操作的封装） - 对象建模（统一数据流设计和控制流设计） - UML从功能模型、静态模型、配置模型描述应用系统的结构。

## 二、软件架构的兴起和发展

### ❁ 4. 软件架构发展史

- ❁ 20世纪90年代以后：基于构件的软件开发阶段 - 以过程为核心，强调软件开发采用构件化技术和架构技术，要求开发出的软件具备很强的自适应性、互操作性、可扩展性和可重用性等。

## 二、软件架构的兴起和发展

### ❁ 4. 软件架构发展史

❁ 软件架构发展的四个阶段：

- (1) “无架构”设计阶段：以汇编语言进行小规模应用程序开发为特征。
- (2) 萌芽阶段：出现了程序结构设计主题，以控制流图和数据流图构成软件结构为特征。
- (3) 初期阶段：出现了从不同侧面描述系统的结构模型，以UML为典型代表。
- (4) 高级阶段：以描述系统的高层抽象为中心，不关心具体的建模细节，划分了架构模型与传统软件结构的界限，该阶段以Kruchten提出的“4 + 1”模型为标志。由于概念尚不统一，描述规范也不能达成一致认识，在软件开发实践中软件架构尚不能发挥重要作用。

### 三、软件重用

- 新的需求: “**faster, better, and cheaper**”

1) **Faster**: The software has to meet a market window. Competitive organizations set that time.

2) **Better**: The software has to serve the requirements of the process it is to support and later, when serving its process, it has to do so with few failure.

3) **Cheaper**: The software has to be less expensive to produce and maintain.

### 三、软件重用

- **Two paths to these goals:**

- 1) Increase the effectiveness of the organization that produces the software.

Such as software engineering.

- 2) Software reuse (软件重用)

---from “Software Reuse”

Ivar Jacobson et al.

## 三、软件重用

### ■ 定义

软件重用是将已有的软件及其有效成分用于构造新的软件或系统。它不仅是对软件程序的重用，还包括对软件生产过程中其它劳动成果的重用，如项目计划书、可行性报告、需求分析、概要设计、详细设计、编码(源程序)、测试用例、文档与使用手册等等。因此，软件重用包括软件产品重用和软件过程重用两部分的内容。



## 三、软件重用

- 重用和移植

软件移植是指对软件进行修改和扩充，使之在保留原有功能、适应原有平台的基础上，可以运行于新的软硬件平台。而重用则指在多个系统中，尤其是在新系统中使用已有的软件成分。



## 三、软件重用

- 白盒重用与黑盒重用

黑盒重用指对已有产品或构件不需作任何修改，直接进行重用，这是理想的重用方式。它主要基于二进制代码的重用，包括可执行程序的重用和基于库（包括动态链接库和静态库）的重用。

白盒重用指根据用户需求对已有产品进行适应性修改后才可使用。白盒重用一般为源代码一级的重用，以及相应的测试用例、文档等的重用。

## 三、软件重用

- 软件重用的三个基本原则
  - a. 必须有可以重用的对象。
  - b. 所重用的对象必须是有用的。
  - c. 重用者需要知道如何去使用被重用的对象。

# 20180306 复习

- 什么是软件架构?
- 架构有何作用?
- 什么是软件重用?
- 为何需要重用?
- 重用需要遵从什么原则?
- 思考题:  
“面向对象” 与 “基于构件”  
模型

## 四、面向对象技术与重用

- 支持软件重用是人们对面向对象方法寄托的主要希望之一，也是这种方法受到广泛重视的主要原因之一。
- 1.面向对象对重用的支持
- 2.重用对面向对象的支持

## 四、面向对象技术与重用

- 1. 面向对象对重用的支持
  - 与其它软件工程方法相比，面向对象方法的一个重要优点是，它可以在整个软件生命周期达到概念、原则、术语及表示法的高度一致。这种一致性使得各个系统成分尽管在不同的开发与演化阶段有不同的形态，但可具有贯穿整个软件生命周期的良好映射。

## 四、面向对象技术与重用

- 1.面向对象对重用的支持
- (1) OOA模型

OOA方法建立的系统模型分为基本模型（如类图）和补充模型（如交互图），强调在OOA基本模型中只表示最重要的系统建模信息，较为细节的信息则在详细说明中给出。这种表示策略使OOA基本模型体现了更高的抽象，更容易成为一个可重用的系统架构。当这个架构在不同的应用系统中重用，在很多情况下可通过不同的详细说明体现系统之间的差异，因此对系统构件的改动较少。

## 四、面向对象技术与重用

- 1.面向对象对重用的支持

### (2) OOA与OOD分工

OOA只注重与问题域及系统责任有关的信息，OOD考虑与实现条件有关的因素。这种分工使**OOA**模型独立于具体的实现条件，从而使分析结果可以在问题域及系统责任相同而实现条件互异的多个系统中重用，并为从同一领域的多个系统的分析模型提炼领域模型创造了有利条件。



## 四、面向对象技术与重用

- 1.面向对象对重用的支持

### (3) 对象的表示

所有的对象都用类作为其抽象描述。对象的一般信息，包括对象的属性、行为及其对外关系等等都是通过对象类来表示的。类作为一种可重用构件，在运用于不同系统时，不会出现因该类对象实例不同而使系统模型有所不同的情况。

### (4) 一般—特殊结构

引入对一般-特殊结构中多态性的表示法，从而增强了类的可重用性。通过对多态性的表示，使一个类可以在需求相似而未必完全相同的系统中被重用。



## 四、面向对象技术与重用

- 1.面向对象对重用的支持
- (5) 整体—部分结构

把部分类作为可重用构件在整个类中使用，这种策略的原理与在特殊类中使用一般类是一致的，但在某些情况下，对问题域的映射比通过继承实现重用显得更为自然。另外还可通过整体-部分结构支持领域重用的策略---从整体对象中分离出一组可在领域范围内重用的属性与服务，定义为部分对象，使之成为领域重用构件。

## 四、面向对象技术与重用

- 1.面向对象对重用的支持

- (6) 实例连接

建议用简单的二元关系表示各种复杂关系和多元关系。这一策略使构成系统的基本成分（对象类）以及它们之间的关系在表示形式和实现技术上都是规范和一致的，这种规范性和一致性对于可重用构件的组织、管理和使用，都是很有益的。

## 四、面向对象技术与重用

- 1.面向对象对重用的支持  
(7) 类描述模板

作为OOA详细说明主要成分的类描述模板，对于对象之间关系的描述注意到使用者与被使用者的区别，仅在使用者一端给出类之间关系的描述信息。这说明可重用构件之间的依赖关系不是对等的。因此，在继承、聚合、实例连接及消息连接等关系的使用者一端描述这些关系，有利于这些关系信息和由它们指出的被依赖成份的同时重用。在被用者一端不描述这些关系，则避免了因重用场合的不同所引起的修改。

## 四、面向对象技术与重用

- 1.面向对象对重用的支持

### (8) USE CASE

由于USE CASE是对用户需求的一种规范化描述，因此它比普通形式的需求文档具有更强的可重用性。每个use case 是对一个活动者使用系统的一项功能时的交互活动所进行描述，它具有完整性和一定的独立性，因此很适于作为可重用构件。

## 四、面向对象技术与重用

- 2.软件重用支持面向对象方法  
(1) 类库

在面向对象的软件开发中，类库是实现对象类重用的基本条件。人们已经开发了许多基于各种OOPL的编程类库，有力地支持了源程序级的软件重用，但要在更高的级别上实现软件重用，仅有编程类库是不够的。实现OOA结果和OOD结果的重用，必须有分析类库和设计类库的支持。为了更好地支持多个级别的软件重用，可以在OOA类库、OOD类库和OOP类库之间建立各个类在不同开发阶段的对应与演化关系。即建立一种线索，表明每个OOA的类对应着哪个（或哪些）OOD类，以及每个OOD类对应着各种OO编程语言类库中的哪个OOP类。



# 四、面向对象技术与重用

- 2.软件重用支持面向对象方法

## (2) 构件库

类库可以看作一种特殊的可重用构件库，它为在面向对象的软件开发中实现软件重用提供了一种基本的支持。但类库只能存储和管理以类为单位的可重用构件，不能保存其它形式的构件；但是它可以更多地保持类构件之间的结构与连接关系。构件库中的可重用构件，既可以是类，也可以是其它系统单位；其组织方式，可以不考虑对象类特有的各种关系，只按一般的构件描述、分类及检索方法进行组织。在面向对象的软件开发中，可以提炼比对象类粒度更大的可重用构件，例如把某些结构或某些主题作为可重用构件；也可以提炼其它形式的构件，例如use case或交互图。这些构件库中，构件的形式及内容比类库更丰富，可为面向对象的软件开发提供更强的支持。

## 四、面向对象技术与重用

- 2.软件重用支持面向对象方法

### (3) 架构库

如果在某个应用领域中已经运用OOA技术建立过一个或几个系统的OOA模型，则每个OOA模型都应该保存起来，为该领域新系统的开发提供参考。当一个领域已有多个OOA模型时，可以通过进一步抽象而产生一个可重用的软件架构。形成这种可重用软件架构的更正规的途径是开展领域分析。通过正规的领域分析获得的软件架构将更准确地反映一个领域中各个应用系统的共性，具有更强的可重用价值。



## 四、面向对象技术与重用

- 2.软件重用支持面向对象方法

### (4) 工具


有效的实行软件重用需要有一些支持重用的软件工具，包括类库或构件/架构库的管理、维护与浏览工具，构件提取及描述工具，以及构件检索工具等等。以重用支持为背景的OOA工具和OOD工具在设计上也有相应的要求，工具对OOA/OOD过程的支持功能应包括：从类库或构件/架构库中寻找可重用构件；对构件进行修改，并加入当前的系统模型；把当前系统开发中新定义的类（或其它构件）提交到类库（或构件库）。

## 四、面向对象技术与重用

- 2.软件重用支持面向对象方法

### (5) OOA过程

在重用技术支持下的OOA过程，可以按两种策略进行组织：第一种策略是，基本保持某种OOA方法所建议的OOA过程原貌，在此基础上对其中的各个活动引入重用技术的支持；另一种策略是重新组织OOA过程。



第一种策略是在原有的OOA过程基础上增加重用技术的支持，应补充说明的一点是，重用技术支持下的OOA过程应增加一个提交新构件的活动。即在一个具体应用系统的开发中，如果定义了一些有希望被其它系统重用的构件，则应该把它提交到可重用构件库中。

第二种策略的前提是：在对一个系统进行面向对象的分析之前，已经用面向对象方法对该系统所属的领域进行过领域分析，得到了一个用面向对象方法表示的领域架构和一批类构件，并且具有构件/架构库、类库及相应工具的支持。在这种条件下，重新考虑OOA过程中各个活动的内容及活动之间的关系，力求以组装的方式产生OOA模型，将使OOA过程更为合理，并达到更高的开发效率。

## 四、构件与重用

- 1.什么是软件构件
- 2.构件模型及实现
- 3.构件获取
- 4.构件管理
- 5.构件重用
- 6.软件重用的层次

# 五、构件与重用

- 1、什么是软件构件

## 1) 对象与构件

构件的特征: (1) A component is a unit of independent deployment. (2) A component is a unit of third-party composition. (3) A component has no persistent state.

对象的特征: (1) An object is a unit of instantiation; it has a unique identity. (2) An object has state; this state can be a persistent state. (3) An object encapsulates its state and behavior.

## 五、构件与重用

- 1.什么是软件构件
- 2)软件构件定义

A software Component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

Next



## 五、构件与重用

[Back](#)

- a. **Interface (接口)**: A component will have **multiple interfaces corresponding to different access points**. Each access point may provide a different service, catering for different client needs. Emphasizing the **contractual nature** of the interface specification is important: as the component and its clients are developed in mutual ignorance, it is the contract that forms a common ground for successful interaction.



## 五、构件与重用

[Back](#)

### b. Explicit context dependencies (清晰的上下文依赖):

Besides the specification of provided interfaces, the above definition of components also requires components to specify their needs. In other words, the definitions requires specification of what the deployment environment will need to provide, such that the component can function. **These needs are called context dependencies**, referring to the context of composition and deployment.

# 五、构件与重用

- 1、什么是软件构件

## 3) 面向对象与构件

a. 面向对象技术已达到了类级重用（代码重用），它以类为封装的单位。这样的重用粒度太小，不足以解决异构互操作和效率更高的重用。

b. 构件将抽象的程度提高到一个更高的层次，它是对一组类的组合进行封装，并代表完成一个或多个功能的特定服务，也为用户提供了多个接口。

c. 整个构件隐藏了具体的实现，只用接口对外提供服务。

# 五、构件与重用

- 2. 构件模型与实现

构件模型（Component Model）是对构件本质特征的抽象描述。

目前国际上流行的构件模型：

- a. 参考模型，如3C模型；
- b. 描述模型，如RESOLVE模型和REBOOT模型；
- c. 实现模型，如CORBA，EJB，DCOM。

# 五、构件与重用

- 2.构件模型与实现

实现模型将构件的接口与实现进行了有效的分离，提供了构件交互的能力，从而增加了重用的机会，并适应了目前网络环境下大型软件系统的需要。

**a.外部接口：**构件的外部接口包含了向其重用者提供的基本信息，如构件名称、功能描述、对外功能接口、所需的构件、参数化属性等。

**b.内部结构：**构件的内部结构包含了两方面的内容：内部成员及其之间的关系。其中内部成员包括具体成员与虚拟成员，而成员关系包括了内部成员之间的互联，以及内部成员与外部接口之间的互联。

# 构件模型与实现...

基于构件的软件开发通常包括构件获取、构件分类和检索、构件评估、适应性修改以及将现有构件在新的语境下组装成新的系统。

# 五、构件与重用

## 3. 构件获取

a. 从现有构件中获得符合要求的构件，直接使用或做适应性（flexibility）修改，得到可重用的构件。

b. 通过遗产工程（legacy engineering），将具有潜在重用价值的构件提取出来，得到可重用的构件。

c. 从市场上购买现成的商业Components，即COTS（Commercial Off-The-Shelf）构件。

d. 开发新的符合要求的构件

# 五、构件与重用

## 4.构件管理

对大量的构件进行有效的管理，以方便构件的存储、检索和提取，是成功地重用构件的必要保证。构件管理内容包括构件描述、构件分类、构件库组织、人员及权限管理和用户意见反馈等。



# 五、构件与重用

## 4. 构件管理

**a. 构件描述：**包括实现方式、实现体、注释、生产者、生产日期、大小、价格、版本和关联构件等信息，它们与构件模型一起组成了对构件的完整描述。

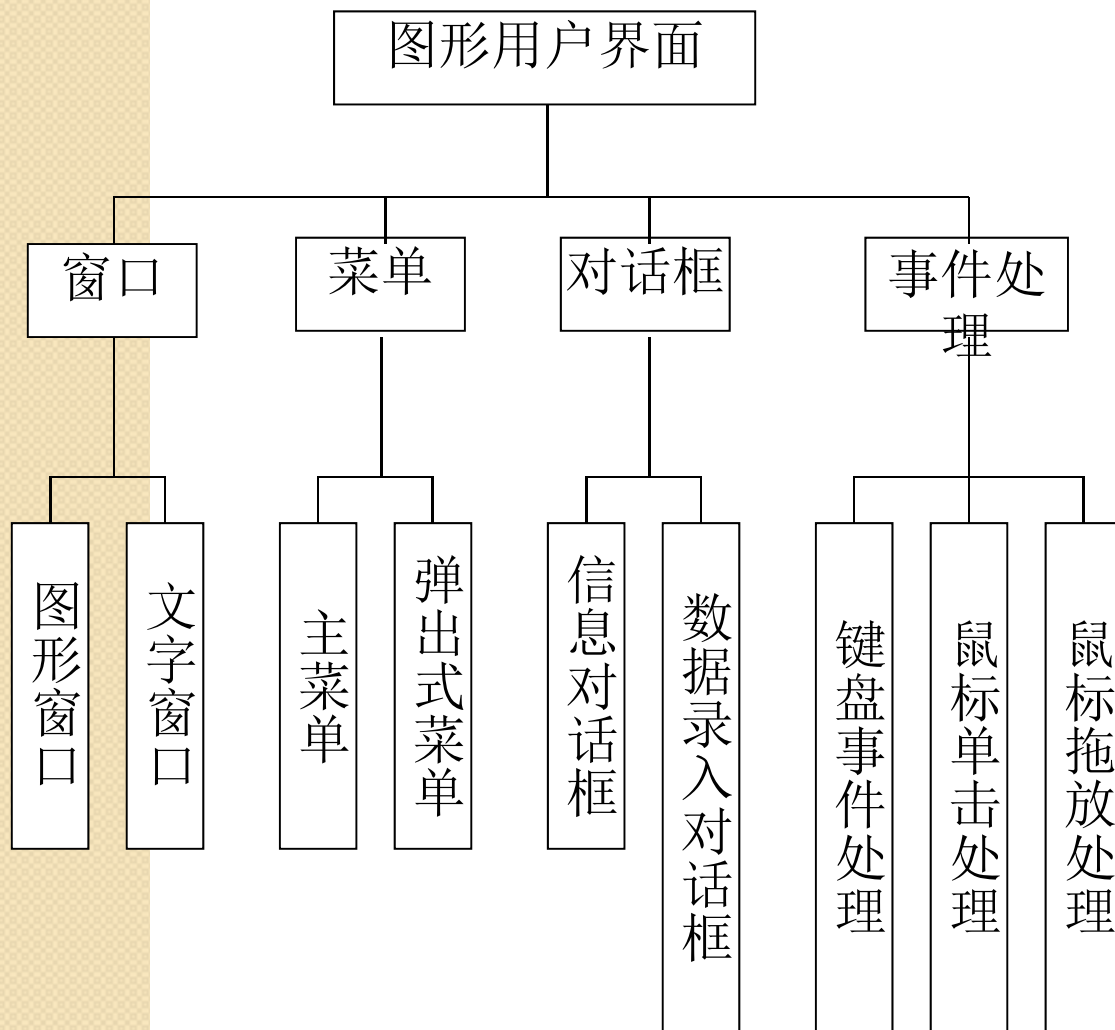
**b. 构件分类与组织：**为了给使用者在查询构件时提供方便，同时也为了更好地重用构件，我们必须对收集和开发的构件进行分类并置于构件库的适当位置。

# 构件管理...

三类构件分类方法：

(1) 关键字分类法(keyword classification)

**基本思想：**根据领域分析的结果将应用领域的概念按照从抽象到具体的循序逐次分解为树状或有向无回路图结构。每个概念用一个描述性的关键字表示。



左边是构件库的关键字  
分类结构示例，  
它支持图形用户界面设计。

当加入构件时，库管理员必须对构件的功能或行为进行分析，在浏览上述关键字分类结构的同时将构件置于最合适的原子级关键字之下。

# 五、构件与重用

## 4.构件管理

### (2) 刻面分类法(faceted classification) ;

在刻面分类机制中，定义若干用于刻画构件特征的“刻面(facet)”，每个刻面包含若干概念，这些概念表述构件在刻面上的特征。刻面可以描述构件执行的功能、被操作的数据、构件应用的语境或其他特征。描述构件的刻面的集合称为刻面描述符（facet descriptor）。当描述符中出现空的特征值时，表示该构件没有相应的刻面。

## 五、构件与重用

### 4. 构件管理

例如，考虑使用下列构件描述符的模式：

**function, object type, system type}**

刻面描述符中每个刻面含有一个或多个值，这些值一般是描述性关键词。例如，如果功能是某构件的刻面，赋给此刻面的典型值可能是：

**function=(copy from) or (copy, replace, all)**

青鸟构件刻面：使用环境，应用领域，功能，层次，表示方法等。

## 五、构件与重用

### 4.构件管理

(3) 超文本组织法(hypertext classification)。

**主要思想：**所有构件必须辅以详尽的功能(function)或性能(performance)说明文档；说明中出现的重要概念或构件以网状链接方式相互连接；检查者在阅读文档的过程中可按照人类的联想思维方式任意跳转到包含相关概念或构件的文档；全文检索系统将用户给出的关键字与说明文档中文字进行匹配，实现构件的浏览检索。Windows环境下的联机帮组系统就是一种典型的超文本系统。



# 五、构件与重用

## 4.构件管理

### 商业化构件分类:

- (1) 用户界面类、数据库类
- (2) 商务应用类
- (3) 工具类、网络通讯类
- (4) 核心技术类

### 从构件的外部形态分类:

- (1) 独立而成熟的构件
- (2) 有限制的构件
- (3) 适应性构件
- (4) 装配的构件
- (5) 可修改的构件



# 五、构件与重用

## 5.构件重用

构件开发的目的是重用，为了让构件在新的软件项目中发挥作用，构件库的使用者必须完成以下工作：检索与提取构件、理解与评价构件、修改构件，最后将构件组装到新的软件产品中。

- 检索与提取构件

- (1) 基于关键字的检索
- (2) 刻面检索法
- (3) 超文本检索法
- (4) 其他检索法

# 五、构件与重用

## 5.构件重用

- 理解与评价构件

(1) 设计信息对于理解构件的必要性和构件用户逆向发掘设计信息的困难性，必须要求构件的开发过程遵循公共软件工程专业规范，并且在构件库的文档中全面、准确地说明以下内容：构件的功能与性能；相关的领域知识；可适应性约束条件与例外情形；可预见的修改部分及修改方法。

(2) 对构件重用的评价，是通过收集并分析构件的用户在实际重用该构件的历史过程中的各种反馈信息来完成的。这些信息包括：重用成功的次数、对构件的修改、构件的健壮性度量、性能度量等。

# 五、构件与重用

## 5.构件重用

### ■ 修改构件

修改构件是指对构件进行或多或少的修改，以适应新的需求。

### ■ 构件组装

构件组装是指将库中构件经适当修改后相互连接，或者将它们与当前开发项目中的软件元素相连接，最终构成新的目标软件。

# 五、构件与重用

## 5.构件重用

### ■ (1) 基于功能的组装技术

基于功能的组装技术采用子程序调用和参数传递的方式将构件组装起来。它要求库中构件以子程序/过程/函数的形式出现，并且接口说明必须清晰。当使用这种组装技术进行软件开发时，开发人员首先应对目标软件系统进行功能分解，将系统分解为强内聚、松耦合的功能模块。然后根据各模块的功能需求提取构件，对它进行适应性修改后再挂接在上述功能分解框架中。

# 五、构件与重用

## 5.构件重用

### (2) 基于数据的组装技术

基于数据的组装技术首先根据当前软件问题的核心数据结构设计出一个框架，然后根据框架中各节点的需求提取构件并进行适应性修改，再将构件逐个分配到框架中的适当位置。此后，构件的组装方式仍然是传统的子程序调用与参数传递。这种组装技术也要求库中构件以子程序形式出现，但它所依赖的软件设计方法不再是功能分解，而是面向数据的设计方法，例如 Jackson 系统开发方法。



# 五、构件与重用

## 5.构件重用

### (3) 面向对象的组装技术

由于封装和继承特性，面向对象方法比其他软件开发方法更适合支持软件重用。在OO软件开发方法中，如果从类库中检索出来的基类能够完全满足新软件项目的需求，则可以直接应用。否则，必须以类库中的基类为父类采用构造法或子类法生成子类。

## 构件重用...

```
//定义基类
class Person{
public:
    Person(char *name, int age);
    ~Person();
protected:
    char *name;
    int age;
};
```

```
//基类构造函数
Person::Person(char *name, int age)
{
    Person::name=new char[strlen(name)+1];
    strcpy(Person::name, name);
    Person::age=age;
    cout<<"Construct Person"<<name<<","<<age<<".\n";
    return;
}
```

```
//基类析构函数
Person::~~Person()
{
    cout<<"Construct Person"<<name<<","<<age<<".\n";
    delete name;
    return;
}
```



# 构件重用...

**a. 构造法:** 为了在子类中使用库中基类的属性和方法, 可以考虑在子类中引进基类的对象作为子类的成员变量, 然后在子类中通过成员变量重用基类的属性与方法。

**b. 子类法:** 与构造法完全不同, 子类法将子类直接说明为库中基类的子类, 通过继承和修改基类的属性与行为完成新子类的定义。

```
//下面采用构造法生成 Teacher 类
class Teacher{
public:
    Teacher(char *name, int age, char *teaching);
    ~Teacher();
protected:
    Tperson *Person;
    char *course;
};

//Teacher 类的实现
Teacher::Teacher(char *name, int age, char *teaching)
{
    //重用基类的方法 Person()
    Tperson=new Person(name, age);
    strcpy(course, teaching);
    return;
}
Teacher::~~Teacher()
{
    delete Tperson;
}
```

# 五、构件与重用

## 6. 构件重用的层次

- 代码重用

包括目标代码和源代码的重用。其中目标代码的重用级别最低，历史也最久，当前大部分编程语言的运行支持系统都提供了连接（**Link**）、绑定(**Binding**)等功能来支持这种重用。源代码的重用级别略高于目标代码的重用，程序员在编程时把一些想重用的代码段复制到自己的程序中，但这样往往会产生一些新旧代码不匹配的错误。想大规模的实现源程序的重用只有依靠含有大量可重用构件的构件库。如“对象链接及嵌入”（**OLE**）技术，既支持在源程序级定义构件并用以构造新的系统，又使这些构件在目标代码的级别上仍然是一些独立的可重用构件，能够在运行时被灵活地组合为各种不同的应用。

# 五、构件与重用

## 6. 构件重用的层次

- 设计重用

设计结果比源程序的抽象级别更高，因此它的重用受实现环境的影响较少，从而使可重用构件被重用的机会更多，并且所需的修改更少。这种重用有三种途径：**(1)**从现有系统的设计结果中提取一些可重用的设计构件，并把这些构件应用于新系统的设计；**(2)**把一个现有系统的全部设计文档在新的软硬件平台上重新实现，也就是把一个设计运用于多个具体的实现；**(3)**独立于任何具体的应用，有计划地开发一些可重用的设计构件。

# 五、构件与重用

## 6. 构件重用的层次

- 需求重用

这是比设计结果更高级别的重用，可重用的分析构件是针对问题域的某些事物或某些问题的抽象程度更高的解法，受设计技术及实现条件的影响很少，所以可重用的机会更大。重用的途径也有三种：**(1)**即从现有系统的分析结果中提取可重用构件用于新系统的分析；**(2)**用一份完整的分析文档作输入产生针对不同软硬件平台和其它实现条件的多项设计；**(3)**独立于具体应用，专门开发一些可重用的分析构件。

例如：事务型系统中：工作流程抽象