

# 第 5 章 软件实现

---

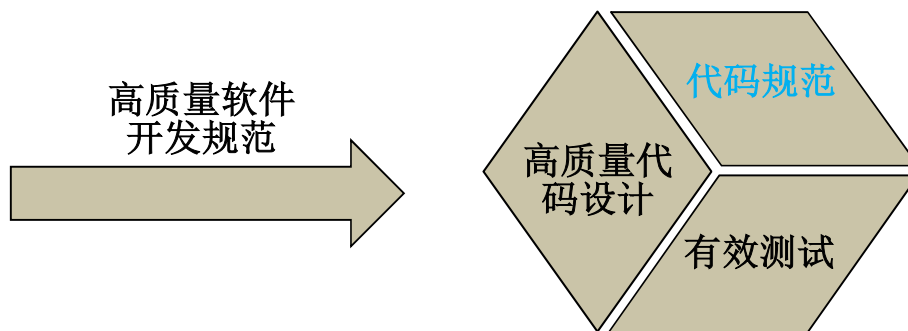
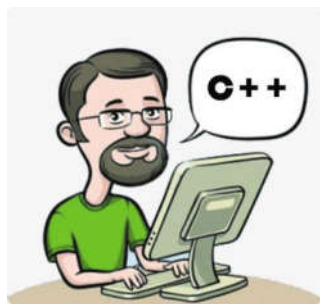
- 程序设计语言
- 程序设计风格
- 代码复用
- 代码审查

# 程序设计语言

软件实现包括编码、测试、调测、优化等一系列工作。
















本章从提高软件的质量和可维护性的角度，讨论在编码阶段面临的主要问题：

- 程序设计语言的特性及选择的原则
- 程序设计风格
- 软件代码审查



# 程序设计语言

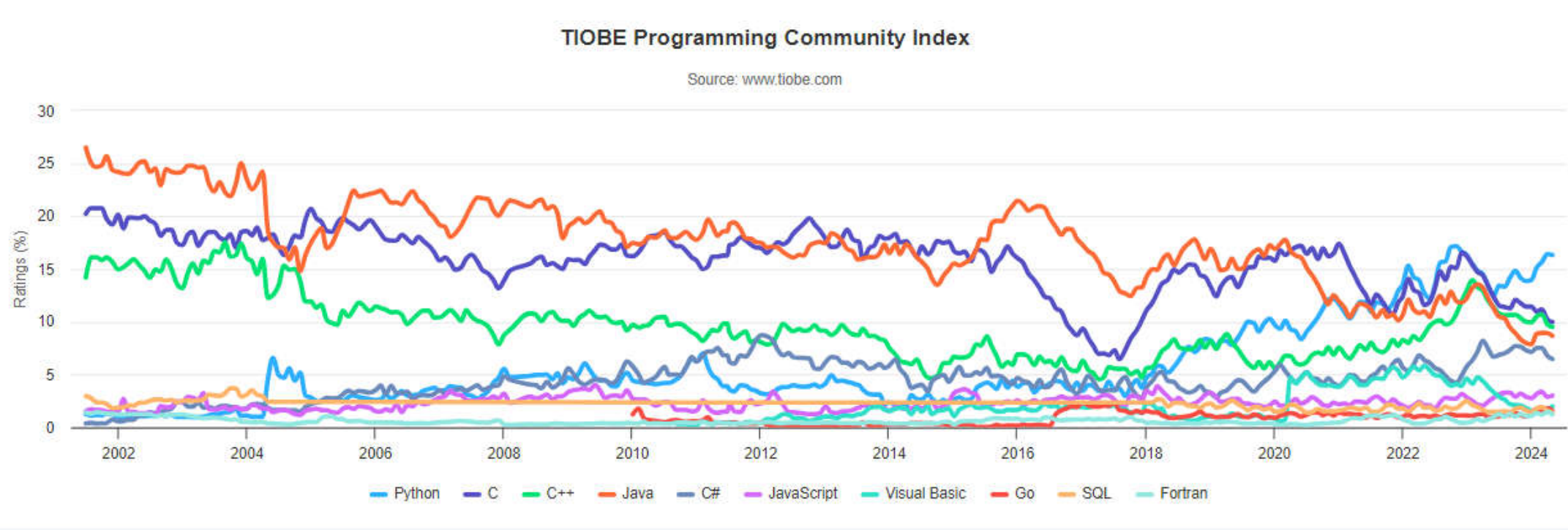
2024年5月“TIOBE世界编程语言排行榜”统计，前15种常用编程语言的使用概率进行排名，可以了解目前常用编程语言的使用情况，并展现全球范围内编程语言的应用趋势。

May 2024	May 2023	Change	Programming Language		Ratings	Change
1	1		 Python		16.33%	+2.88%
2	2		 C		9.98%	-3.37%
3	4	▲	 C++		9.53%	-2.43%
4	3	▼	 Java		8.69%	-3.53%
5	5		 C#		6.49%	-0.94%
6	7	▲	 JavaScript		3.01%	+0.57%
7	6	▼	 Visual Basic		2.01%	-1.83%
8	12	▲▲	 Go		1.60%	+0.61%
9	9		 SQL		1.44%	-0.03%
10	19	▲▲	 Fortran		1.24%	+0.46%
11	11		 Delphi/Object Pascal		1.24%	+0.23%
12	10	▼	 Assembly language		1.07%	-0.13%
13	18	▲▲	 Ruby		1.06%	+0.26%
14	15	▲	 MATLAB		1.06%	+0.18%
15	14	▼	 Swift		1.01%	+0.09%

<https://www.tiobe.com/tiobe-index/>

# 程序设计语言

## TOP 10 编程语言指数走势 (2002~2024)



# 程序设计语言

不同的程序语言机制，对设计的支持不尽相同，目前被广泛采用的是结构化程序设计语言和面向对象语言。

结构化程序设计语言机制以功能为切入点，因而首要考虑：

- 数据结构的表示
- 算法的模块化分解
- 控制结构

# 程序设计语言

对于类似“百度”的搜索引擎，如何进行搜索接口设计？搜索结果呢？



百度一下

标准搜索模式

高级搜索

搜索结果：包含以下全部的关键词

包含以下的完整关键词：

包含以下任意一个关键词

不包括以下关键词

时间：限定要搜索的网页的时间是

文档格式：搜索网页格式是

关键词位置：查询关键词位于 ☒ 网页的任何地方 ☐ 仅网页的标题中 ☐ 仅在网页的URL中

站内搜索：限定要搜索指定的网站是

高级搜索

高级搜索模式

# 程序设计语言

不同的程序语言机制，对设计的支持不尽相同，目前被广泛采用的是结构化程序设计语言和面向对象语言。

面向对象程序设计语言机制需考虑到：

- 类：考虑局部化设计原则。
- 继承性：继承性是使得类自动具有其它类的属性（数据结构）和方法（功能）的机制。
- 多态性：多态性是指相同的模块接口定义，却有完全不同的实现过程。
- 消息：消息是实现多态性的重要机制之一。

# 程序设计语言

## 消息机制实现——图书借阅系统中的消息机制

图书借阅系统中要处理的基本信息是书，因而设计类**BookItem**来描述“书”，它包括书名与该书的数量。所有的书构成“图书集”，用类**BookItemList**来描述。定义图书馆**Libray**类，实现对“图书集”的管理。读者类**Reader**实现借书、还书的功能。

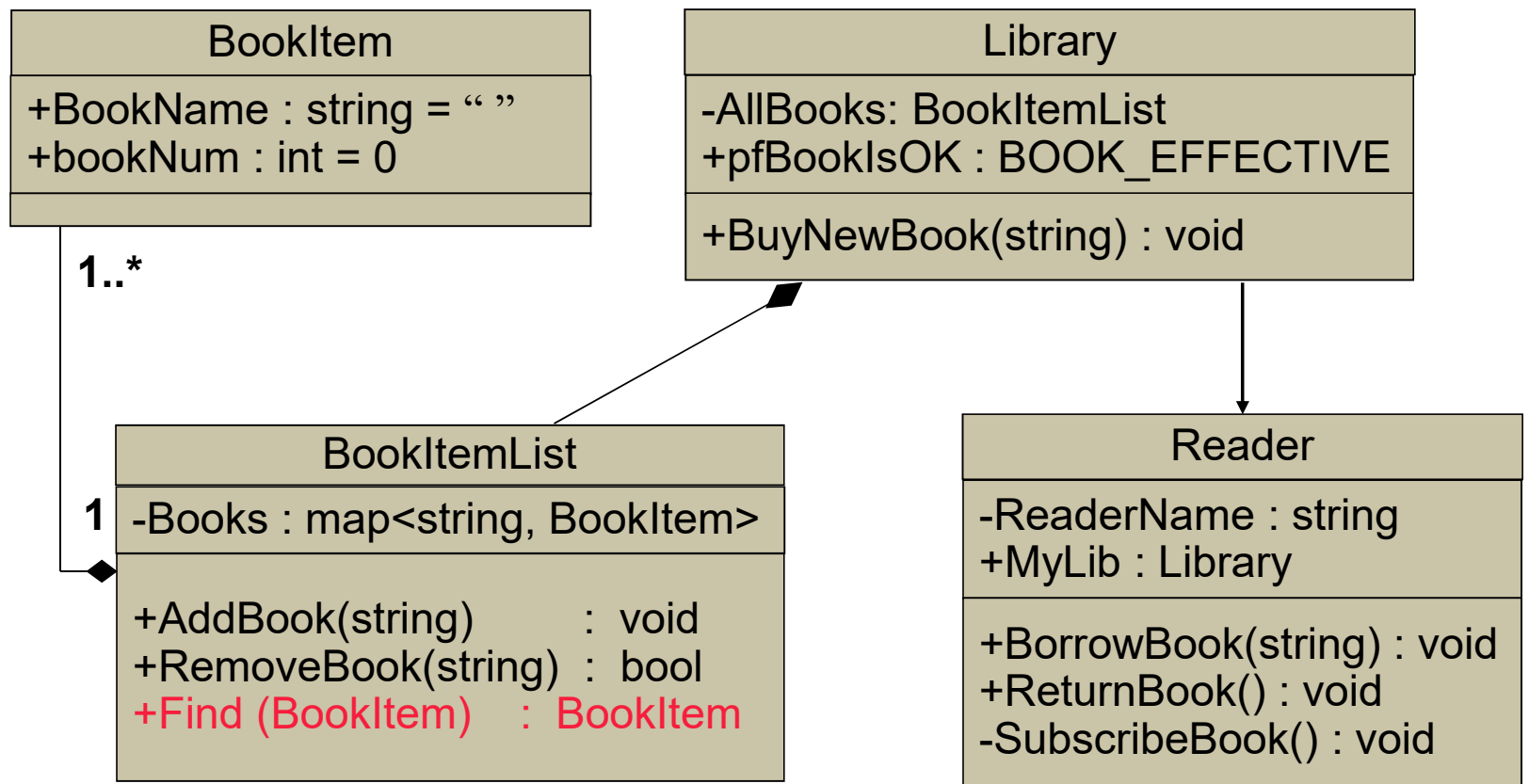
消息机制：如果所借阅的图书暂时没有，则登记订阅该书。当有其他读者归还该书时，图书馆会 **自动**给订阅该书的读者发送通知。



# 程序设计语言

## 消息机制实现——图书借阅系统中的消息机制

类图



# 程序设计风格

---

讨论程序设计风格，力图从编码原则的角度来探讨提高程序的可读性、改善程序质量的方法和途径。

- 代码文件
- 数据说明
- 语句结构的处理
- 输入输出设计准则

# 程序设计风格

```
/* ****  
* 实验：定义并实现字符串类CMyString，完成常见的字符串操作。  
* *****/
```

```
#include <iostream>  
using namespace std;
```

→ 代码文件头部，包括程序运行所需的外部文件。注意文件路径。

```
// 定义字符串类  
class CMyString {  
public:
```

→ 类的定义及简要说明

```
    CMyString(const char* = NULL);           // 可构造无参的字符串  
    CMyString(const CMyString&);  
    ~CMyString();  
    int Find(char, int = 0);                 // 从指定位置开始查找字符  
    int Find(const CMyString&);              // 查找字符串  
    int Length();                            // 获取字符串长度  
    CMyString Mid(int, int);                 // 获取子串  
    char operator[] (int index);             // 获取指定位置的字符  
    CMyString& operator=(const CMyString&);  
    CMyString& operator+=(const CMyString&); // 重载加运算符对应的 += 运算符  
    friend CMyString operator+(const CMyString&, const CMyString&);  
    friend ostream& operator<<(ostream&, const CMyString&);
```

```
protected:
```

```
    char* m_pString; // 保存的字符串  
    int m_iSize;     // 字符串长度
```

→ 类的成员属性定义及简要说明

```
};
```

类的成员函数接口定义及简要说明

# 程序设计风格

```
/*  
* 实现：获取指定起始位置和长度的子串  
* 参数：startPos：指定子串的起始位置；Length：子串的长度  
* 返回：返回获取子串的对象  
*/  
CMyString CMyString::Mid(int startPos, int Length) {  
    // 确保参数的有效性  
    if (startPos < 0 || startPos >= m_iSize || Length <= 0) return CMyString("");  
    if (Length > m_iSize - startPos) Length = m_iSize - startPos;  
    char* pTemp = new char[Length + 1]; // 保存获取的子串  
    int i = 0;  
    for (i = 0; i < Length; i++)  
        pTemp[i] = m_pString[startPos++];  
    pTemp[i] = '\0';  
  
    // 不要直接写成：return MyString(pTemp);  
    // 否则将无法释放pTemp的内存空间，造成内存泄漏。  
    CMyString my(pTemp);  
    delete[] pTemp;  
    return my;  
}
```

成员函数接口参数说明

成员函数过程说明

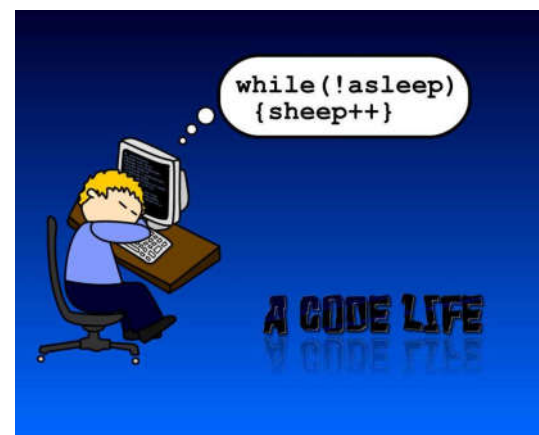
成员函数内部变量说明

成员函数中其它必要的说明

# 程序设计风格

只编写够用的注释，重质量而非重数量

- 好的注释解释为什么，而不是怎么样；
  - 不要在注释中重复代码；
  - 写完注释之后，需要在代码的上下文中回顾一下；
  - 当修改代码时，切记维护代码周围的注释；
  - 注释不要产生歧义，更不能误导。
- 
- 注释不是目的，最好的、可理解的程序就是代码文件自身；
  - 要编写可阅读的代码，代码文件自身就简单易懂。



# 代码评审

---

代码编写风格参考

IE\_Coding\_Standard.htm

<https://github.com/google/styleguide>

# 代码评审

代码评审，也称为代码复查，是指在软件开发过程中，通过阅读源代码和相关设计文件，对源代码编码风格、编码标准以及代码质量等活动进行系统性检查的过程。



代码评审主要分为**正式评审**和**轻量级评审**。

# 代码评审

## 代码评审——正式评审

正式评审是针对代码编写完成之后召开的评审会议。评审会议由评审小组组织完成，成员包括组长、评审员、质量过程管理员和程序员，评审秉承沟通、协作、互助、学习的态度。



### 评审内容包括：

- 功能设计：按照软件需求，设计是否恰当、良好；
- 复杂性：代码功能描述和性能描述；
- 规范：代码编写标准、接口定义规范；
- 文档：源代码及相关文档。

```
/*  
 * 实现：获取指定起始位置和长度的子串  
 * 参数：startPos：指定子串的起始位置；Length：子串的长度  
 * 返回：返回获取子串的对象  
 */  
CString CString::Mid(int startPos, int Length) {  
    // 确保参数的有效性  
    if (startPos < 0 || startPos >= m_iSize || Length <= 0) return CString("");  
    if (Length > m_iSize - startPos) Length = m_iSize - startPos;  
    char* pTemp = new char[Length + 1]; // 保存获取的子串  
    int i = 0;  
    for (i = 0; i < Length; i++)  
        pTemp[i] = m_pString[startPos++];  
    pTemp[i] = '\0';  
  
    // 不要直接写成：return CString(pTemp);  
    // 否则将无法释放pTemp的内存空间，造成内存泄漏。  
    CString my(pTemp);  
    delete[] pTemp;  
    return my;  
}
```



# 代码评审

## 代码评审——轻量级评审

轻量级评审主要采取非正式的代码走查方式，不需要组织正式会议，因此它成本小、灵活性强。

轻量级评审方式包括：

- 程序员向评审者提供需评审的代码；
- 程序员可以借助代码自动评审工具；
- 程序员可以与评审员同步进行软件设计和开发；
- 定期召开小型的评审会，共同探讨代码编写过程中遇到的各类问题。



# 代码评审

## 代码评审——轻量级中的9项实践方式

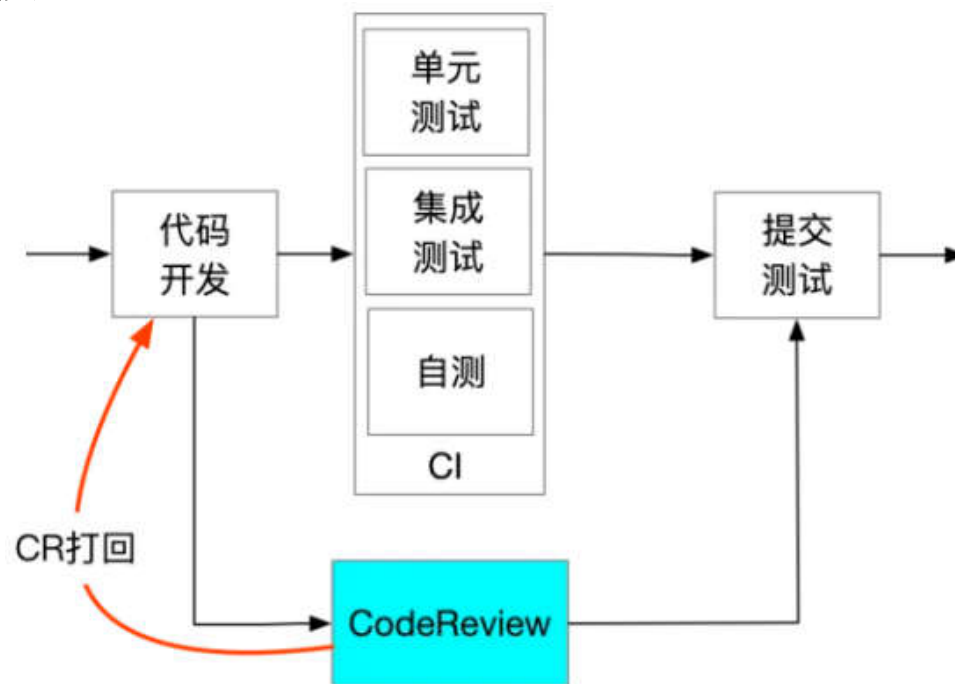
- 1、一次评审少于 200 ~ 400 行的代码。
- 2、目标为每小时低于 300 ~ 500 LOC 的检查速率。
- 3、花足够的时间进行正确缓慢的评审，但不要超过60 ~ 90 分钟。
- 4、确定开发者在评审开始之前就已注释了源代码。
- 5、为代码评审和获取制度建立可量化的目标，以利于改进流程。
- 6、使用检查列表。
- 7、确认缺陷确实得到修复。
- 8、培养良好的代码评审文化氛围，维护开发团队的团结协作。
- 9、采用轻量级，能用工具支持的代码评审，例如CodeStriker、IDE。



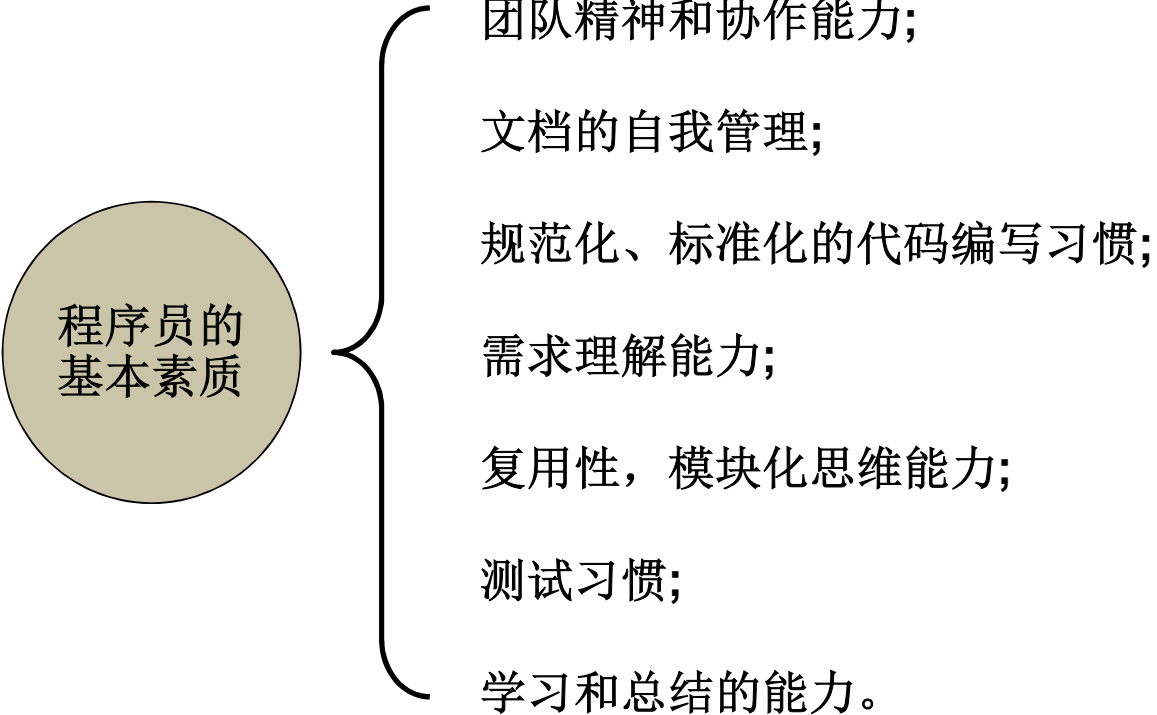
# 代码评审

## 代码评审——所处阶段

企业项目开发过程中的代码评审阶段



# 代码评审



程序员的基本素质

团队精神和协作能力;

文档的自我管理;

规范化、标准化的代码编写习惯;

需求理解能力;

复用性，模块化思维能力;

测试习惯;

学习和总结的能力。