软件工程与实践

授课教师：许毅　　电子邮箱：　，　微信电话:

- 5.1 软件实现的概念和工作
- 5.2 实现语言

1. 5.3高质量编码

第五章 实现

介绍软件实现的语言、编程是的习惯和注意事项、如果编写高效率的程序，如何管理不同版本的代码以及代码的更改。

5.1 软件实现的概念和工作

- 软件实现的概念
- 软件实现的工作
- 软件实现的生产性和创作性
- 软件实现与软件设计的关系
- 软件实现与软件测试的关系
- 软件实现的原则

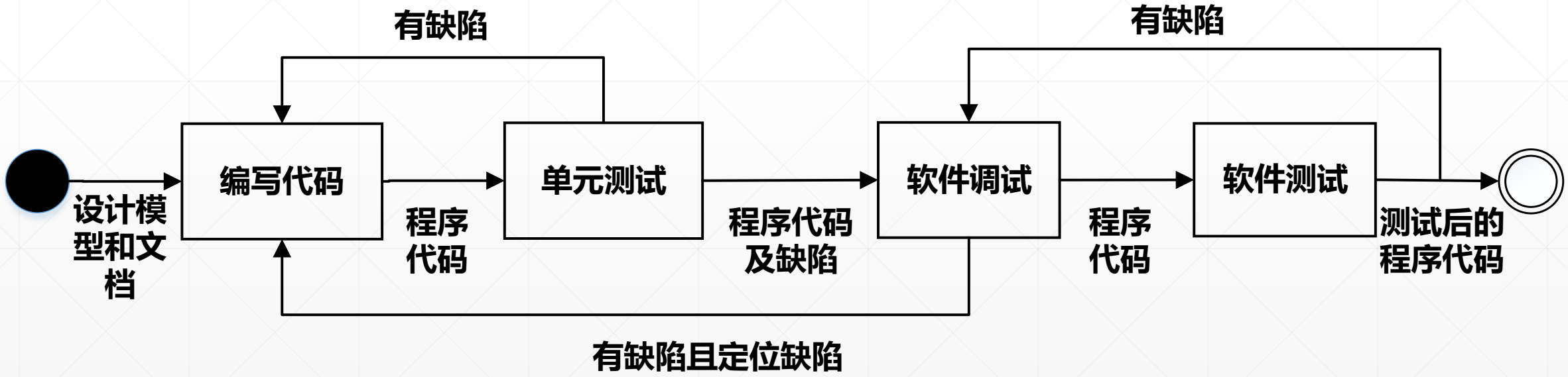
5.1.1软件实现概念

- 根据软件**设计模型**，编写出目标软件系统的程序**代码**，并对代码进行必要的**测试**，以发现和纠正代码存在中的**缺陷**，并将可运行的目标代码部署到目标计算机上运行

- 软件实现不仅要编写出程序代码，还要确保代码的质量，因此软件实现涉及多方面的开发工作，如**编码、测试、调试**等

5.1.2 软件实现的工作

➢输入和输出



编码+单元测试+调试

- 这三项工作均由程序员负责完成
- **编码**
  - 基于软件设计模型和文档，采用选定的程序设计语言，编写出目标软件系统的程序代码
- **单元测试**
  - 对自己编写的各个基本模块进行单元测试，以发现模块单元中存在的缺陷和问题
- **调试**
  - 发现产生缺陷原因，定位缺陷位置，进而对代码缺陷进行修复

软件实现的其他工作

- 软件架构搭建
  - 软件框架：软件模块组织框架
  - 经过严格验证的通用软件框架是软件实现的集成
  - Spring Boot、Vue.js、React、Django、.NET
- 使用辅助工具
  - 版本管理工具
  - 编辑、编译、部署工具
  - 测试工具、调试工具
  - 集成开发环境
- 软件集成，构建软件系统

5.1.3 软件实现兼具创作和生产

- 生产性活动
  - 需要根据软件设计规格说明书和软件设计模型，生产出与之相符合的软件制品，即程序代码
  - 遵循设计文档和模型来编写程序，而且还要求程序员遵循编码原则和风格来编写出高质量的程序代码，并通过单元测试、集成测试、确认测试等一系列的软件测试活动来保证代码质量
- 创作性活动
  - 发挥软件开发工程师的智慧和主观能动性，创作出目标软件系统的程序代码。这一过程高度依赖于程序员的编程经验、程序设计技能和素养，以及软件测试工程师的软件测试水平

5.1.4 软件实现与软件设计的关系

- 基于软件设计来开展软件实现
  - 照软件设计模型和文档来进行编码

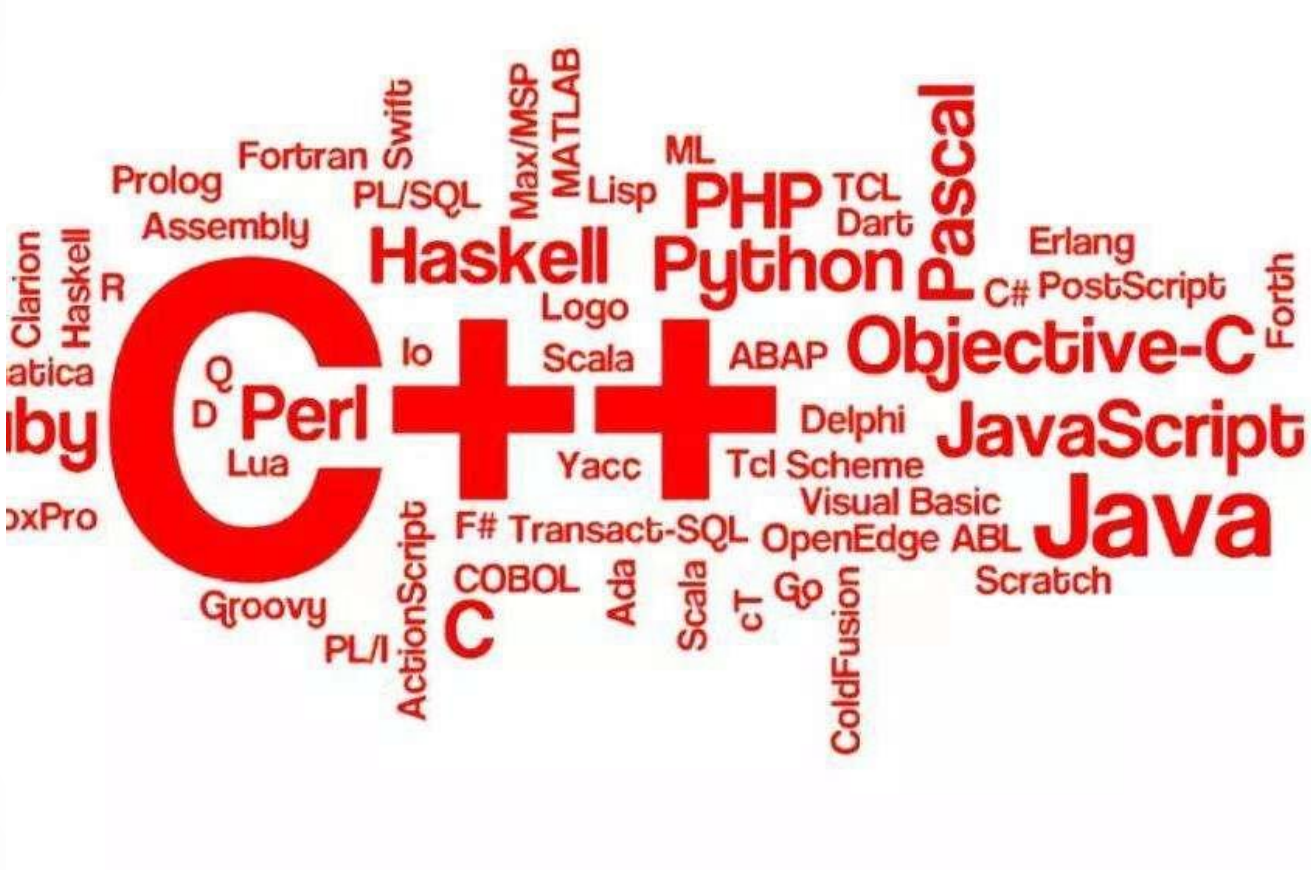
- 根据实现中发现的问题来纠正和完善软件设计
  - 设计不够详细，程序员需要进行进一步的软件设计和程序设计，才能编写出程序代码
  - 设计考虑不周全，软件设计时没有认真考虑编码实现的具体情况（如程序设计语言和目标运行环境的选择），导致有些软件设计不能通过程序设计语言加以实现

5.1.5 软件实现与软件测试的关系

- 独立的测试团队有利于保证公正性（利益分割，思维独立，独立汇报）
- 测试的依据仍然是软件详细设计、概要设计和需求描述文档。程序代码是测试对象而不是依据。
- 程序员和测试工程师需要相互协作。工作的界限并不是明确的。
- 单元测试多以程序员为主。
- 软件集成是实现人员的工作，而集成测试由测试团队负责。
- 系统部署由程序和设计师共同完成，而系统测试和确认测试以测试团队为主。

5.1.6 软件实现要遵循的原则

- 基于设计来编码
  - 切忌“拍脑袋”写程序
- 质量保证贯穿全过程
  - 要有非常强的“质量”意识
  - 既要重视外部质量，也要重视内部质量



5.2 程序实现语言

- 引言
- 概念
- 特点
- 分类
- 选择要素

程序设计语言提供的支持

- 提供了语法、语义和语用三方面的要素
- 支持程序员来编写程序代码
- 人们提出了二千多种的程序设计语言，不同的语言适合于不同的应用开发

2.1 程序设计语言的类别（1/3）

❑ 机器语言

- ✓ 由“0”、“1”所组成的机器指令
- ✓ 极为繁琐、费时费力的工作；软件开发效率非常低，而且程序代码可读性非常差，极容易出错，不易于维护、移植性差；但程序代码的执行效率会非常高

❑ 汇编语言

- ✓ 一种**低级语言**，用助记符代替机器指令的操作码，用地址符号或标号代替指令或操作数的地址
- ✓ 较为低级和复杂，程序可读性差，代码编写的效率低，对代码进行维护非常困难，程序调试也不容易，代码兼容性差
- ✓ 程序代码占用存储空间少、运行速度快、执行效率高

程序设计语言的类别（2/3）

❑结构化程序设计语言

- ✓以**过程或函数**作为基本的编程单元，采用三类控制结构（顺序、条件和循环）来刻画模块的处理过程和流程
- ✓属于**高级程序设计语言**，程序可读性、可理解性、可维护性等有了明显的提升；配套CASE工具较为完善，有结构化程序设计方法学的指导
- ✓不足：以过程和函数作为基本模块，模块的粒度小，可重用性差；程序代码抽象层次低，无法对问题域及其求解进行自然抽象
- ✓如C、Fortran、Pascal等

程序设计语言的类别（3/3）

- **面向对象程序设计语言**
  - ✓以类作为基本的模块单元，借助于**面向对象的一组概念和机制**来进行程序设计，
  - ✓有系统的**方法学指导**，建立起可直观反映问题域、模块粒度更大、可重用性更好的程序代码，已经成为计算机领域的主流编程语言
  - ✓如Java、C++等
- **描述性程序设计语言**
  - ✓描述程序需要**解决什么样的问题**，无需在程序中显式地定义如何来解决问题
  - ✓如Prolog、Lisp、ML等

2.2 程序设计语言的表达能力

编程语言的类别	平均代码量	编程语言	平均代码量
机器语言	320	**C**	128
汇编语言	107	**Fortran**	107
高级语言	80	**C++/Java**	53

一个功能点用不同的语言来实现所需的代码量是不一样的

2.3 程序设计语言的选择（1/3）

❑软件的应用领域

- ✓不同应用领域的软件通常会选择不同的程序设计语言来加以实现
- ✓科学和工程计算领域选用Fortran、C等程序设计语言，数据库应用软件开发会选用Delphi、Visual Basic、SQL等程序设计语言，机器人等嵌入式应用选用C、C++等程序设计语言，AI多用Python，互联网应用开发选用Java、ASP等程序设计语言

❑与遗留软件系统的交互

- ✓考虑待开发软件系统是否需要与遗留软件系统存在交互。如果有该方面的实际需要，那么程序员需要解决二个系统之间的互操作问题

程序设计语言的选择（2/3）

- **软件的特殊功能及需求**
  - ✓是否需要与底层的硬件系统进行交互，如果需要，可以考虑采用诸如C、汇编语言
  - ✓是否需要丰富的软件库来支持功能的实现，如果需要，可以考虑具有丰富软件库的编程语言，如Python、Java等
  - ✓是否需要对相关的知识进行表示和推理，如果需要，可以考虑选用描述性的程序设计语言，如Prolog、Lisp等

程序设计语言的选择（3/3）

❑软件的目标平台

- ✓如果目标软件系统需要运行在特定的软件开发框架、软件中间件、基础设施之上，那么程序员还需要考虑目标平台对程序设计语言的支持，并依此来选定所需的编程语言
- ✓如果目标软件系统需要部署在J2EE架构之上，那么就需要选择Java编程语言；如果需要借助于ROS来开发机器人软件，那么建议选择C、C++和Python等编程语言

❑程序员的编程经验

- ✓应该选择对于自己而言较为熟悉的语言，尽量避免选择没有使用过的程序设计语言

Java语言

- 纯粹的面向对象编程语言
- 简单性
- 分布性
- 兼具编译和解释性以及可移植性
- 强类型语言和健壮性
- 安全性，没有指针，使用字节码验证策略，防止恶意代码

5.3高质量编码

- 原则
- 要素

3.1 编写代码的原则（1/3）

- **易读，一看就懂**
  - ✓能够理解代码的语义和内涵，了解相关语句和代码的实现意图，方便修改和维护代码
  - ✓采用**缩进**的方法来组织代码的显示，用**括号**来表示不同语句的优先级，对关键语句、语句块、方法等要加以**注释**
- **易改，便于维护**
  - ✓或者在适当的位置**增加新的代码**以完善代码功能，或者对某些代码进行**修改**以便纠正代码中的缺陷和错误
  - ✓对将来可能需要进行修改和维护的代码（包括常元、变量、方法等）进行**单独的抽象、参数化和封装**，以便将来对其修改时不会影响其他部分的代码

编写代码的原则（2/3）

- **降低代码的复杂度**
  - ✓将一个类代码组织为一个**文件**，并用统一的**命名规则**来命名文件
  - ✓在代码中适当的**增加注释**以加强对代码的理解，不用“goto”语句，**慎用嵌套**或者减少嵌套的层数，尽量选用**简单**的实现算法
- **尽可能地开展软件重用和编写可重用的程序代码**
  - ✓尽可能地**重用**已有的软件制品，如函数库、类库、软构件、开源软件、甚至代码片段等等
  - ✓在编码时要考虑所编写代码的**可重用性**，使得所编写的代码能为他人或者在其它软件系统开发中被再次使用

编写代码的原则（3/3）

- **要有处理异常和提高代码的容错性**
  - ✓编写必要的**异常定义和处理代码**，使得程序能够对异常情况进行必要的处理，防止由于异常而导致的程序终止或崩溃
  - ✓编写程序代码以支持**故障检测、恢复和修复**，确保程序在出现严重错误时仍然能够正常运行，或者当崩溃时能尽快恢复执行
- **代码要与模型和文档相一致**
  - ✓程序员在编写代码的同时要同步修改和完善相应的软件设计模型和文档，确保**代码、模型和文档三者之间保持一致**

3.2 遵循编码风格（1/4）

- **格式化代码的布局，尽可能使其清晰、明了**
  - ✓充分利用水平和垂直两个方向的编程空间来**组织程序代码**，便于读者阅读代码
  - ✓适当地**插入括号“{ }”**，使语句的层次性、表达式运算次序等更为清晰直观
  - ✓有效地**使用空格符**，以显式地区别程序代码的不同部分（如程序与其注释）

```
package net.micode.notes.data;


import ...


public class NotesProvider extends ContentProvider {
    private static final UriMatcher mMatcher;

    private NotesDatabaseHelper mHelper;

    private static final String TAG = "NotesProvider";

    private static final int URI_NOTE            = 1;
    private static final int URI_NOTE_ITEM       = 2;
    private static final int URI_DATA            = 3;
    private static final int URI_DATA_ITEM       = 4;

    private static final int URI_SEARCH          = 5;
    private static final int URI_SEARCH_SUGGEST  = 6;

    static {
        mMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        mMatcher.addURI(Notes.AUTHORITY, "note", URI_NOTE);
        mMatcher.addURI(Notes.AUTHORITY, "note/#", URI_NOTE_ITEM);
```

遵循编码风格（2/4）

- **尽可能提供简洁的代码，不要人为地增加代码的复杂度**
  - ✓ 使用**简单的数据结构**，避免使用难以理解和难以维护的数据结构（如多维数组、指针等）
  - ✓ 采用**简单而非复杂的实现算法**
  - ✓ **简化**程序中的算术和逻辑表达式
  - ✓ 不要引入**不必要的变元和动作**
  - ✓ **防止变量名重载**
  - ✓ 避免模块的冗余和重复

```
package net.micode.notes.data;


import ...


public class NotesProvider extends ContentProvider {
    private static final UriMatcher mMatcher;

    private NotesDatabaseHelper mHelper;

    private static final String TAG = "NotesProvider";

    private static final int URI_NOTE            = 1;
    private static final int URI_NOTE_ITEM       = 2;
    private static final int URI_DATA            = 3;
    private static final int URI_DATA_ITEM       = 4;

    private static final int URI_SEARCH          = 5;
    private static final int URI_SEARCH_SUGGEST  = 6;

    static {
        mMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        mMatcher.addURI(Notes.AUTHORITY, "note", URI_NOTE);
        mMatcher.addURI(Notes.AUTHORITY, "note/#", URI_NOTE_ITEM);
```

遵循编码风格（3/4）

- 对代码辅之以**适当的文档，以加强程序的理解**
  - ✓有效、必要、简洁的**代码注释**
  - ✓代码注释的**可理解性、准确性和无二义性**
  - ✓确保代码与设计模型和文档的**一致性**

```
package net.micode.notes.data;


import ...


public class NotesProvider extends ContentProvider {
    private static final UriMatcher mMatcher;

    private NotesDatabaseHelper mHelper;

    private static final String TAG = "NotesProvider";

    private static final int URI_NOTE            = 1;
    private static final int URI_NOTE_ITEM       = 2;
    private static final int URI_DATA            = 3;
    private static final int URI_DATA_ITEM       = 4;

    private static final int URI_SEARCH          = 5;
    private static final int URI_SEARCH_SUGGEST  = 6;

    static {
        mMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        mMatcher.addURI(Notes.AUTHORITY, "note", URI_NOTE);
        mMatcher.addURI(Notes.AUTHORITY, "note/#", URI_NOTE_ITEM);
```

遵循编码风格（4/4）

- **加强程序代码的结构化组织，提高代码的可读性**
  - ✓按一定的**次序**来说明数据
  - ✓按**字母顺序**说明对象名
  - ✓避免使用嵌套循环结构和嵌套分支结构
  - ✓使用统一的**缩进规则**
  - ✓确保每个模块内部的代码**单入口、单出口**

```
package net.micode.notes.data;


import ...


public class NotesProvider extends ContentProvider {
    private static final UriMatcher mMatcher;

    private NotesDatabaseHelper mHelper;

    private static final String TAG = "NotesProvider";

    private static final int URI_NOTE            = 1;
    private static final int URI_NOTE_ITEM       = 2;
    private static final int URI_DATA            = 3;
    private static final int URI_DATA_ITEM       = 4;

    private static final int URI_SEARCH          = 5;
    private static final int URI_SEARCH_SUGGEST  = 6;

    static {
        mMatcher = new UriMatcher(UriMatcher.NO_MATCH);
        mMatcher.addURI(Notes.AUTHORITY, "note", URI_NOTE);
        mMatcher.addURI(Notes.AUTHORITY, "note/#", URI_NOTE_ITEM);
```

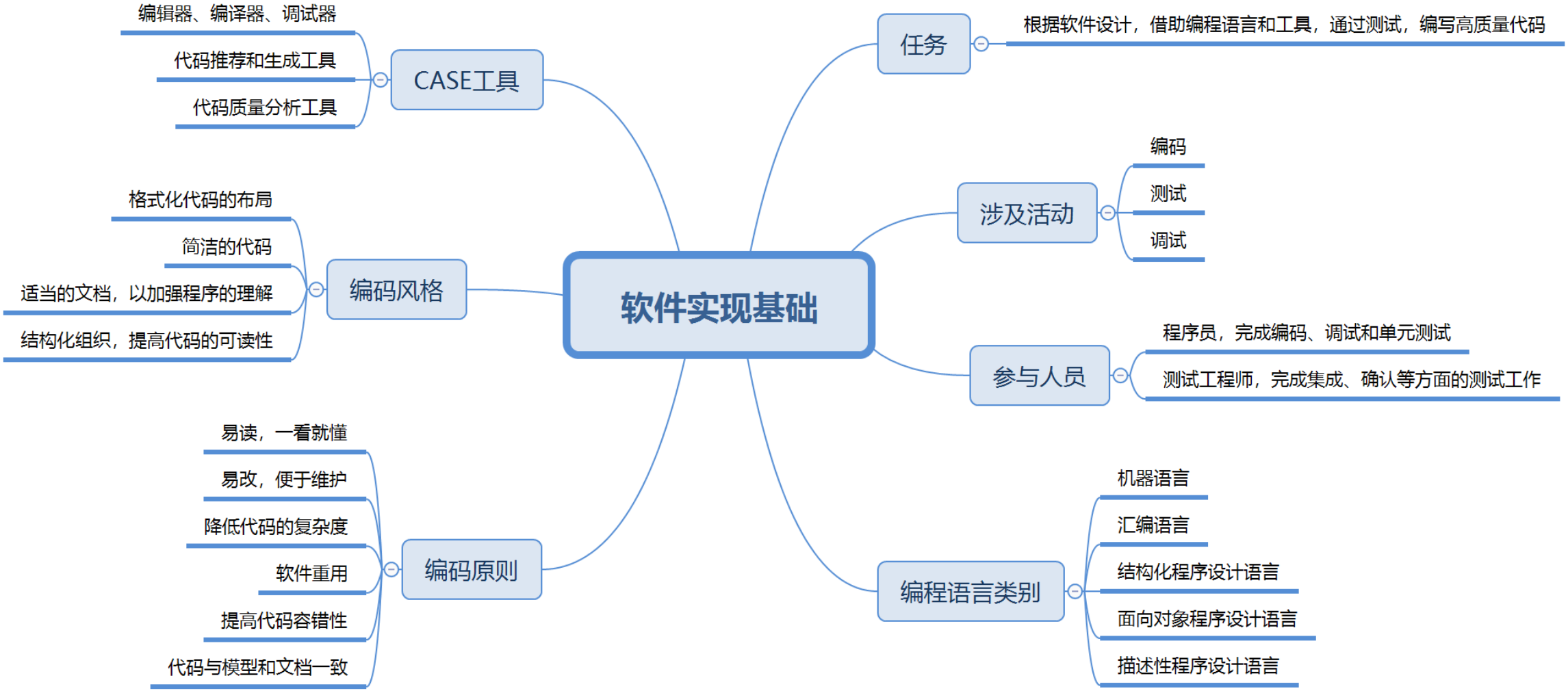
3.3 支持软件实现的CASE工具

- 编辑器
- 编译器
- 调试器
- 测试工具
- 代码推荐工具
- 基于大模型的生成工具
- 代码静态分析工具
- 集成环境

3.4 软件实现的输出

- 源程序代码
- 部署在不同计算节点上的可执行程序代码
- 软件测试报告等

本章知识图谱


软件实现基础
任务
根据软件设计，借助编程语言和工具，通过测试，编写高质量代码
涉及活动
编码
测试
调试
参与人员
程序员，完成编码、调试和单元测试
测试工程师，完成集成、确认等方面的测试工作
编程语言类别
机器语言
汇编语言
结构化程序设计语言
面向对象程序设计语言
描述性程序设计语言
CASE工具
编辑器、编译器、调试器
代码推荐和生成工具
代码质量分析工具
编码风格
格式化代码的布局
简洁的代码
适当的文档，以加强程序的理解
结构化组织，提高代码的可读性
编码原则
易读，一看就懂
易改，便于维护
降低代码的复杂度
软件重用
提高代码容错性
代码与模型和文档一致

小结

- **软件实现**
  - ✓ 软件实现包括**编码、测试、调试、部署**等一系列的活动
  - ✓ 基于**软件设计模型**，编写出目标软件系统的程序代码，并对代码进行必要的**测试**，以发现和纠正代码存在中的缺陷，并将目标代码**部署**到计算机上运行
- **编程语言的选择**
  - ✓ 要根据软件所属的应用领域、与遗留软件系统的交互、程序员的经验等多个方面，考虑选择什么样的程序设计语言来进行编程
- **程序员遵循编码的原则和规范来编写出高质量的程序代码**