《软件工程》

2023年春

第七讲 软件实现与测试

内容

一．软件实现基础

二．编写代码

三．软件测试

一、软件实现基础

内容

1. 软件实现概述

✓软件实现的任务、过程与原则

2. 软件实现语言

✓编程语言的类别和选择

3. 高质量编码

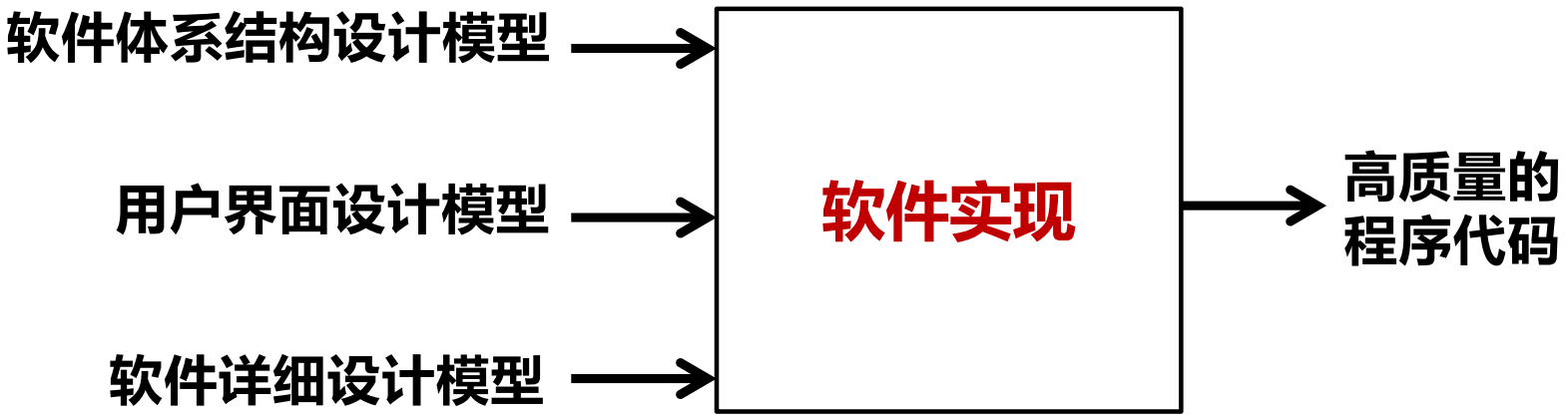
✓编码的原则和要求

1.1 何为软件实现

- 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**

软件实现的任务



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**

1.2 软件实现需考虑多方面的因素

❑与多类不同的人员相关

- ✓包括程序员、软件测试工程师等

❑程序员要考虑的因素

- ✓不仅要对照设计来编写代码，还需要通过遵循编码规范、程序设计原则等来提高代码的质量

❑软件测试工程师需要考虑的因素

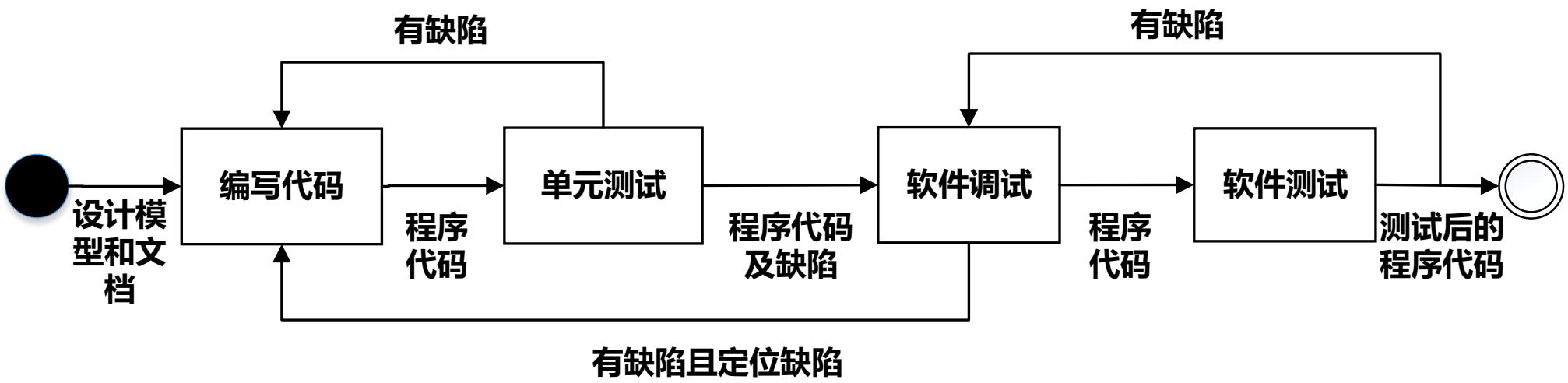
- ✓针对代码开展测试，不仅要发现代码中存在的功能性缺陷，如代码功能实现不正确，还要发现代码中存在的非功能性缺陷

软件实现与软件设计之间的关系

- **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

1.3 软件实现的过程



软件实现包含编码、测试、调试等一系列的开发活动

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

软件测试

- 包括多项的软件测试工作
  - ✓集成测试、确认测试、系统测试等
- 这项工作由软件测试工程师来完成

1.4 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

程序员的职责和能力要求

- 自我学习能力
- 独立解决问题的能力
- 良好的编程习惯
- 质量意识
- 学会软件测试
- 阅读和学习他人的代码
- 善于利用CASE工具
- 团队合作和沟通

内容

1. 软件实现概述
   - ✓软件实现的任务、过程与原则

2. 软件实现语言
   - ✓编程语言的类别和选择

3. 高质量编码
   - ✓编码的原则和要求

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平均代码量

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++、Python等程序设计语言，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

2.4 流行的程序设计语言

❑2021年7月份程序设计语言的使用排行榜

排名	语言名称	使用占比
1	C	11.62%
2	Java	11.17%
3	Python	10.95%
4	C++	8.01%
5	C#	4.83%
6	Visual Basic	4.5%
7	Java Script	2.71%
8	PHP	2.58%
9	汇编语言	2.4%
10	SQL	1.53%

C/C++语言

- 面向过程、通用的结构化（面向对象）程序设计语言
- 简洁的语言
- 结构化的控制语句
- 丰富的数据类型和运算符
- 良好的可移植性
- 目标代码执行效率高

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

Python语言

- 面向对象、解释型、通用的脚本编程语言
- 功能强大，Python 类库极其丰富
- 语法简单，入门容易
- 开源和免费，可移植性
- 解释性
- 混合型语言，既支持面向过程的编程也支持面向对象的编程

内容

1. 软件实现概述
   - ✓软件实现的任务、过程与原则

2. 软件实现语言
   - ✓编程语言的类别和选择

3. 高质量编码
   - ✓编写代码的原则和要求

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);
```

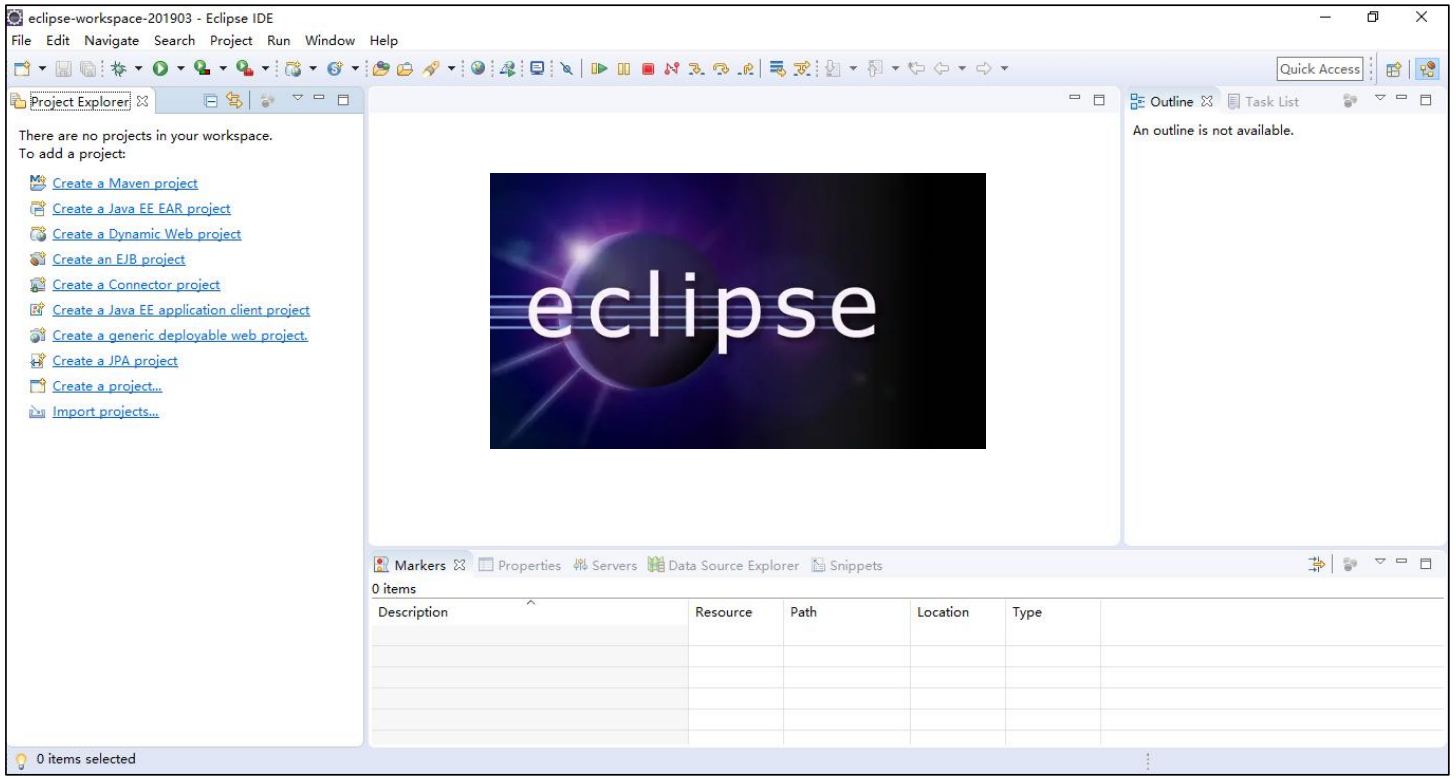
遵循编码风格（4/4）

- 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
- ❑集成开发环境



Eclipse集合开发环境

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

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**

思考和讨论

编程实现面临的主要挑战是什么？它对程序员和测试工程师提出什么样的要求？

问题和讨论