1-3

- 区别：内部质量：关注程序的内部结构、代码的可读性、可维护性、可扩展性等，是从开发者角度衡量程序的质量，不直接面向用户使用体验。外部质量：关注程序在运行时呈现给用户的特性，如功能正确性、性能（速度、响应时间等）、易用性、可靠性等，是用户能直接感知到的程序质量。

计算两数和的Python代码

```
def add(a,b):
    return a+b
```

内部质量:代码简洁,数名 add 清晰表达功能，参数命名 a 、b虽简单但在这个简单场景下也能理解，具有较好的可读性和可维护性，内部质量不错。外部质量：功能正确，能准确计算两数之和，运行速度快（因为逻辑简单），外部质量也较好。

1-4

- 内部质量不高的情况：代码难以理解和维护，其他开发者接手时需要花费大量时间去梳理逻辑。代码可扩展性差，当需要添加新功能时，需要对大量现有代码进行修改，容易引入新的错误。可能存在隐藏的逻辑错误，在特定情况下才会暴露，难以调试。
- 外部质量较差的问题：功能存在缺陷，无法正确完成用户期望的任务，导致用户无法正常使用程序。性能差，运行缓慢、响应时间长，用户体验差，可能会使用户失去耐心而放弃使用。易用性差，界面不友好、操作复杂，用户难以掌握如何使用程序。
- 高质量程序的特征：内部质量方面：代码结构清晰、可读性好、注释恰当；具有良好的可维护性和可扩展性；逻辑严谨，错误处理完善。外部质量方面：功能正确、完整；性能优异，运行高效；界面友好、操作简便；稳定可靠，不易崩溃。

1-5

- 编写清晰的代码和注释：清晰的代码结构和恰当的注释能让开发者更容易理解代码逻辑，便于维护和后续扩展，提升内部质量。
- 进行充分的测试：包括单元测试、集成测试等，能提前发现功能错误、性能问题等，确保程序在各种情况下都能正确运行，提升外部质量。
- 遵循编码规范：如统一的命名规则、代码格式等，使代码风格一致，增强可读性和可维护性，提升内部质量。
- 注重代码的模块化设计：将程序划分为多个功能独立的模块，降低代码的耦合性，提高可维护性和可扩展性，提升内部质量，同时也有助于功能的清晰实现，间接提升外部质量。

1-13

- 区别：程序：是为实现特定功能，用编程语言编写的指令集合，通常是代码的直接体现，更侧重于代码层面的逻辑。比如一段简单的Python代码 print("Hello World")，这就是一个程序。软件：是程序、相关文档以及数据的集合。它不只是代码，还包括使用说明、设计文档、帮助文档等，以及程序运行所需的数据。例如我们常用的办公软件Microsoft Office，不仅有实现各种功能（如文字处理、表格制作等）的程序代码，还有详细的使用手册、帮助文档，以及用户创建的文档数据等，这些共同构成了软件。
- 关系：程序是软件的核心部分，软件包含程序，同时还包含支持程序运行和使用的文档与数据等。没有程序，软件就失去了功能实现的基础；而仅有程序，没有相关文档和数据，也不能构成完整可用的软件。

1-18

以12306软件系统为例：

- 正确性：能准确实现购票、退票、改签等核心功能，用户按照操作流程，输入正确信息，就能得到预期的票务处理结果，比如成功预订车票、查询到准确的车次余票等。
- 可靠性：在大量用户同时访问、购票高峰等情况下，系统能稳定运行，不易崩溃或出现大规模故障，保障用户正常使用票务服务。
- 安全性：采用多种安全措施，保护用户的个人信息（如身份证号、手机号等）以及支付安全，防止用户信息泄露和资金损失。
- 私密性：对用户的个人购票记录、出行信息等隐私数据进行严格保护，仅在必要且合法的情况下才会使用或披露这些信息。
- 可维护性：当系统需要更新功能（如新增支付方式、优化界面等）或修复漏洞时，能够高效地进行维护操作，且维护过程对用户正常使用的影响较小。

1-22

软件系统复杂性体现

- 需求复杂：用户需求多样且不断变化，不同用户对软件功能、使用体验等有不同期望，还可能存在模糊、矛盾的需求。
- 结构复杂：大型软件系统通常由众多模块、组件组成，模块间存在复杂的依赖、交互关系，代码逻辑也会很繁琐。
- 技术复杂：涉及多种技术栈，如不同的编程语言、框架、数据库等，技术选型和整合难度大，还需考虑技术的兼容性与性能。

主要挑战

- 需求管理：准确捕捉、分析和管理不断变化的用户需求，确保开发出的软件符合用户期望。
- 项目管理：合理规划项目进度、资源（人力、物力等），协调团队成员工作，应对开发过程中的各种风险，保证项目按时、按质完成。
- 质量保证：在开发过程中保证软件的正确性、可靠性、安全性等质量属性，需要进行全面的测试、代码审查等工作。

组织开发有规模和复杂度软件系统的困难和问题

- 团队协作：团队成员众多，沟通成本高，可能出现信息传递不畅、理解偏差，导致开发工作不协调。

- 技术难题：在开发过程中会遇到各种技术瓶颈，如系统性能不足、高并发处理困难、与其他系统集成问题等，解决这些难题需要大量时间和技术投入。
- 进度与成本控制：规模大、复杂的软件系统开发周期长，容易出现进度延迟，且开发成本（人力、设备、时间等）也难以精准控制，可能超出预算。
- 需求变更应对：开发过程中用户需求变更频繁，若处理不当，会导致开发工作反复，影响项目进度和软件质量。

2-7

- “系统的”：软件工程的方法是成体系的，涵盖软件开发从需求分析、设计、编码、测试到运维的全生命周期，各阶段的方法相互关联、衔接，形成一个完整的流程框架，能对软件开发和运维进行全面、系统的指导。
- “可量化的”：软件工程的方法可以将软件开发和运维过程中的各种要素（如工作量、进度、质量等）用具体的量化指标来衡量。例如，通过计算代码行数、缺陷数量、测试覆盖率等量化数据，来评估项目的进展和质量状况，便于进行管理和优化。
- “规范化的”：软件工程制定了一系列标准、规范和流程，要求在软件开发和运维过程中遵循统一的规则。比如编码规范，规定了代码的命名、格式、注释等标准；文档规范，明确了不同阶段文档的内容和格式要求，以保证开发过程的一致性和可重复性。

2-8

软件工程三要素关系

软件工程的三要素（方法、工具、过程）相互依存、相互支持。方法是核心，提供了软件开发和运维的技术手段；工具是方法的辅助，能自动化或半自动化地实现方法，提高开发效率；过程则是将方法和工具结合起来，规定了如何按照一定的顺序和步骤使用方法和工具，以完成软件开发和运维任务。

面向对象软件工程三个构成要素具体内涵

- 方法：以面向对象思想为核心，包括面向对象的分析（OOA）、设计（OOD）、编程（OOP）等方法，通过识别对象、类，建立对象间的关系（如继承、关联、聚合等）来进行软件的分析和设计，使软件更符合人类对现实世界的认知和建模方式。
- 工具：支持面向对象软件开发的各类工具，如UML建模工具（用于绘制类图、对象图等）、面向对象的集成开发环境（IDE，提供代码编写、调试、编译等支持）、版本控制工具（管理面向对象代码的版本）等，这些工具能辅助开发者更高效地进行面向对象的软件开发。
- 过程：结合面向对象方法的软件开发过程，例如统一过程（UP）等，定义了面向对象软件开发各阶段（如初始、细化、构造、移交等）的任务、活动和里程碑，确保面向对象的方法和工具能有序、有效地应用到软件开发中。

2-9

共性

都追求在一定的约束条件下（如时间、成本等），交付高质量的产品（软件产品或一般工程产品），满足用户的需求，并且都需要对项目进行管理，包括资源调配、进度控制、质量保证等。

差异性

- 产品形态不同：软件工程的产品是软件，是逻辑实体，具有无形性、易变性等特点；一般工程的产品多为物理实体，具有固定的形态和物理属性。
- 生产方式不同：软件开发主要是脑力劳动，基于知识和逻辑进行构建；一般工程生产更多依赖物理加工、制造等过程，涉及更多的物理操作和材料处理。
- 质量衡量标准不同：软件工程除了功能、性能等质量属性外，还特别关注软件的可维护性、可扩展性、可靠性（在复杂逻辑和运行环境下的稳定程度）等；一般工程产品的质量更多关注物理性能（如强度、硬度、耐用性等）、尺寸精度等。

2-10

- 抽象原则：面向对象程序设计通过类来对现实世界中的事物进行抽象，提取其共同的属性和行为。例如，定义“汽车”类，抽象出汽车的属性（如颜色、型号、速度等）和行为（如启动、加速、刹车等），将具体的汽车对象的共性进行抽象封装，体现了软件工程的抽象原则，便于对复杂的现实世界进行建模和处理。
- 模块化原则：面向对象的类本身就是一种模块化的体现，每个类都封装了特定的属性和方法，形成相对独立的模块。例如，在一个电商系统中，“用户”类、“商品”类、“订单”类等都是独立的模块，各自负责相关的功能，模块间通过定义好的接口（如方法调用）进行交互，降低了系统的复杂性，提高了可维护性，体现了软件工程的模块化原则。
- 信息隐藏原则：面向对象通过访问控制（如public、private、protected）来实现信息隐藏，将类的内部实现细节隐藏起来，只对外提供公开的接口。例如，一个“银行账户”类，将账户余额等敏感信息设置为private，只提供public的“存款”“取款”“查询余额”等方法，外部代码只能通过这些方法来操作账户，而无法直接访问和修改余额，保证了数据的安全性和类的封装性，体现了信息隐藏原则。