1. 现代软件工程如何处理软件开发所具有的本质困难？

1. 复杂性

挑战：随着软件应用的深入，系统的复杂性增加，导致管理和维护的难度上升，甚至可能导致项目失败。

应对策略：

微服务架构：通过将大系统拆分为多个小而独立的服务，减少每个服务的复杂性，便于理解和维护。同时，每个微服务可以独立开发、测试和部署，降低了单个系统的复杂性。

领域驱动设计（DDD）：通过聚焦于业务领域，识别核心领域和子域，将复杂的业务逻辑拆分为更小的部分，进而简化系统设计。

架构模式：采用事件驱动架构、消息队列等技术来降低模块间的耦合，提高系统的灵活性。

2. 隐匿性

挑战：软件开发过程中的盲区和沟通障碍，导致各参与者对项目的理解不一致。

应对策略：

持续交付与持续部署：缩短开发周期，频繁交付小规模的功能，帮助团队快速获得反馈，减少隐匿期，确保每个阶段的可见性。

DevOps文化：促进开发与运维之间的协作，增强团队的沟通和信息共享，确保所有成员对软件的状态有清晰的了解。

文档与可视化工具：使用工具（如需求管理工具、可视化仪表板等）来提高项目的透明度，使所有相关人员能够实时了解项目进展。

3. 配合性

挑战：当软件开发规模扩大时，协作成本显著增加，尤其是在进行新增或修改时。

应对策略：

服务划分原则：微服务架构通过将系统划分为多个独立服务，减少了内部配合的复杂性。每个服务可以独立开发和测试，从而降低了跨服务的协作需求。

API管理：通过清晰的API设计和管理，使各个微服务之间的交互标准化，降低配合的复杂性。

团队自治：鼓励小团队拥有独立决策权，减少依赖性，使得各团队能够更灵活、高效地响应变化。

4. 易变性

挑战：软件应用的环境变化迅速，传统架构难以快速适应新需求。

应对策略：

敏捷开发：采用迭代与增量的方法，通过小步快跑的方式快速响应需求变化。

微服务独立性：每个微服务独立负责特定功能，团队可以快速调整和发布功能，以适应不断变化的用户需求。

反馈机制：建立快速反馈机制，利用用户反馈和数据分析，持续改进和迭代软件功能，确保软件能够适应变化。

1. 需求工程包含哪些主要的活动？如何理解“描述”是需求工程的核心？

主要的活动：

1.需求获取：通过访谈、调查和观察等方式收集用户需求。

2.需求分析：对收集到的需求进行分类、优先级排序和可行性评估。

3.需求规格说明：将需求以文档或模型形式明确记录，确保所有利益相关者的理解一致。

4.需求验证：检查需求的完整性和一致性，确保其满足项目目标。

5.需求管理：持续跟踪需求的变更，确保在开发过程中及时更新和沟通。

“描述”是需求工程的核心，因为清晰的需求描述能够有效传达用户的期望，避免误解和错误，从而为后续的设计和开发提供明确的基础。通过精确的描述，团队可以确保开发出的软件真正满足用户需求，降低风险和成本。

1. 结合本节讲义，完整的描述什么是需求。

在软件需求工程中，需求是指用户或利益相关者对软件系统应实现的功能、性能、约束和其他特性的期望和要求。需求可以分为以下几类：

1.功能需求：描述系统必须执行的特定功能或行为。例如，用户登录、数据输入、报表生成等。

2.非功能需求：涉及系统的性能、可用性、可靠性、安全性等方面的要求。例如，系统必须在5秒内响应用户请求，支持1000个并发用户等。

3.约束：指在软件开发过程中必须遵循的限制条件，可能包括技术、环境、法律或政策等要求。例如，必须使用特定的技术栈或符合某些行业标准。

4.业务需求：描述系统在商业或业务层面需要满足的需求，通常由业务目标驱动。例如，提升用户满意度、降低运营成本等。

需求的准确描述对于软件开发至关重要，因为它确保了所有利益相关者对系统的期望一致，减少了开发过程中的误解和返工，提高了最终产品的质量和用户满意度。需求描述不仅应具体、清晰，还应可验证和可追踪，以便在开发和测试过程中进行有效管理。