Lecture02_Software Process

Created	@September 18, 2024 12:56 PM
Class	CPT 203
Type	Lecture
Materials	Week 2 - Software Processes.pdf
Reviewed	☐

Software Process

▼ Four fundamental activities to software engineering: (include sub-activities and supporting process)

- Software specification
- Software design and implementation: comprises of programming/codding
- Software validation
- Software evolution

Types of Software Process

- For a critical systems, a very structured development process is required
- Fore non-critical systems, with rapidly changing requirements, a less formal, flexible process is likely to be more effective

Categories of Software Process

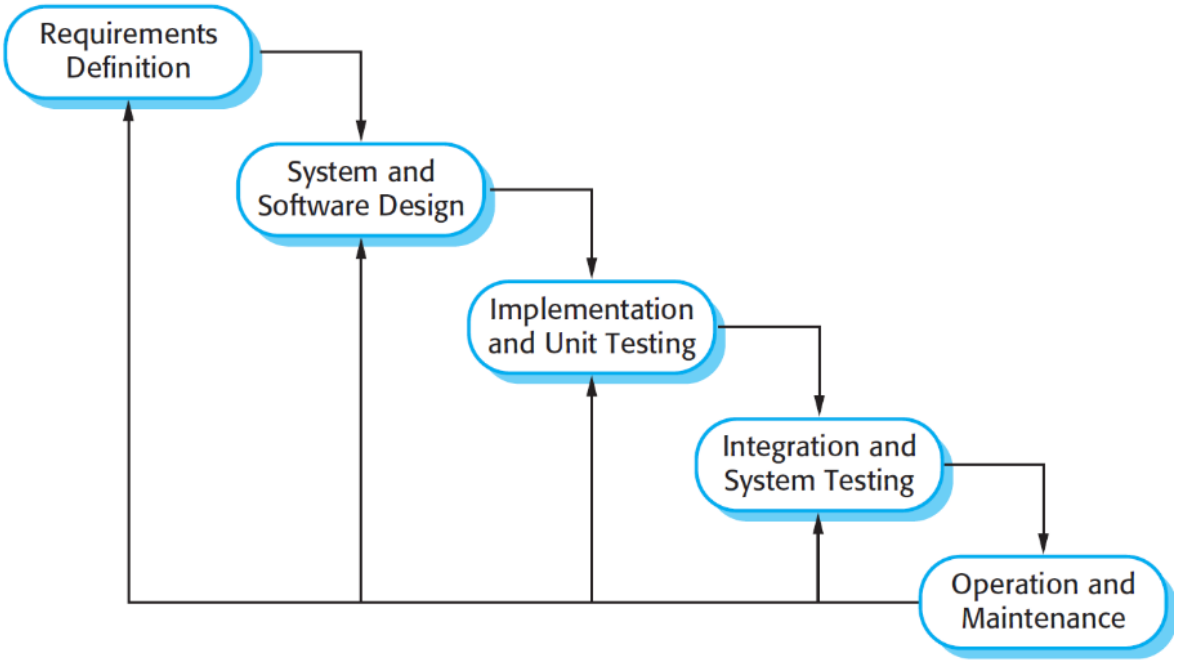
- **Linear and Sequential Models:** Waterfall, V-Model
- **Iterative and Incremental Models:** Incremental, Iterative, Spiral
- **Agile Models:** Scrum, Kanban, Extreme Programming (XP)
- **Prototyping Models:** Throwaway Prototyping, Evolutionary Prototyping
- **Component-Based Models:** Component-Based Development (CBD)
- **Formal Methods:** Formal Specification, Model Checking
- **Hybrid Models:** Agile-Waterfall Hybrid, DevOps
- **Rapid Application Development (RAD) Models:** RAD Model, DSDM
- **Lean Models:** Lean Software Development

Software Process Models

- When the software process is execution, the software process model provides the structures framework that guides this execution
- The model ensures that best practices are followed, risks are managed, and quality is maintained, leading to more predictable, efficient, and successful software development project.

The Waterfall Model

- Definition: The waterfall Model is a linear and sequential approach to software development. It processes through distinct phases, each with specific deliverables and review processes.



▼ Requirements Definition

- **目标**：收集并记录所有功能性和非功能性需求。
- **交付物**：需求规范文档。
- **活动**：利益相关者访谈、调查、需求研讨会。

▼ System and Software Design

- **目标**：基于需求创建详细的设计。
- **交付物**：系统架构文档、设计规范。
- **活动**：架构设计、数据库设计、用户界面设计。

▼ Implementation and Unit Testing

- **目标**：将设计文档转化为实际代码。
- **交付物**：源代码、代码文档。
- **活动**：编码、单元测试、代码评审。

▼ Integration and System Testing

- **目标**：集成所有模块并测试整个系统。
- **交付物**：测试计划、测试用例、测试报告。
- **活动**：集成测试、系统测试、用户验收测试。

▼ Operation and Maintenance

- **目标**：将系统部署到生产环境，并提供持续的支持和改进。
- **交付物**：部署计划、用户手册、维护报告、更新补丁。
- **活动**：安装、配置、用户培训、错误修复、性能改进、功能更新。

Characteristics of the Waterfall Model

- **Linear and Sequential**: Each phase must be completed before the next one begins.
- **Documentation-Driven**: Extensive documentation is produced at each stage.
- **Phase-Specific Deliverables**: Each phase has specific deliverables and milestones.
- **Review and Approval**: Each phase requires review and approval before proceeding to the next.

▼ Characteristics of the Water Model In practice

- 理论上，每个阶段的结果是一个或多个获得批准的文档，下一阶段在前一个阶段完成前不应开始。
- 实际上，这些阶段常常重叠，相互反馈。
- 软件过程并不是一个简单的线性模型，而是一个涉及各阶段之间反馈的过程。
- 产生的文档可能需要修改以反映所做的更改。

▼ Limitation and Challenge

- **高昂的迭代成本**：文档的生成和批准成本高，使得迭代非常昂贵，涉及大量返工。
- **冻结开发任务**：通常在少量迭代后冻结部分开发任务，如需求说明，以继续后续开发阶段。这可能导致问题被忽略或留待以后解决。
- **维护阶段的挑战**：在最终生命周期阶段（操作和维护）期间，软件被投入使用时，可能会发现原始软件需求中的错误和遗漏，系统必须不断演化以保持实用性。

▼ 适用场景和局限性

- 瀑布模型的主要问题是项目被不灵活地分为不同的阶段，使得在过程早期做出的承诺难以随着客户需求的变化而调整。
- 因此，瀑布模型只适用于需求明确且在系统开发过程中不太可能发生显著变化的项目。

▼ Strengths and Benefits

- **简单易用(Simplicity and Ease of Use)**：
  - **清晰结构(Clear Structure)**：瀑布模型结构简单易懂，便于实施和管理。

    - **明确定义的阶段(Well-Defined Phases)**：每个阶段都有明确的开始和结束点，帮助理解项目进度和管理工作流。
- **可预测性和规划(Predictability and Planning)**：
    - **详细的计划(Detailed Planning)**：可以在项目初期进行详细的规划，精确估算成本、时间和资源。
    - **可预测的结果(Predictable Outcomes)**：明确定义的阶段和交付物，使结果更具可预测性，有利于利益相关者和项目经理。
- **文档记录(Documentation)**：
    - **全面的文档(Comprehensive Documentation)**：每个阶段需要详细的文档记录，有助于未来的维护和知识转移。
    - **可追溯性(Traceability)**：文档提供决策和变更的清晰追溯性，便于跟踪项目历史和理解设计选择的理由。
- **纪律性和控制(Discipline and Control)**：
    - **结构化方法(Structure approach)**：线性和顺序性确保了开发过程的纪律性。
    - **阶段完成(Phase Completion)**：每个阶段必须完成并经过审查批准才能继续，确保项目的各个方面得到充分审查。
- **资源管理(Resource Management)**：
    - **资源分配(Resource Allocation)**：明确的阶段有助于更好地规划和分配资源。
    - **技能利用(Skill Utilization)**：团队可以根据每个阶段所需的具体技能进行组织。
- **里程碑和交付物(Milestones and Deliverables)**：
    - **明确的里程碑(Clear Milestones)**：每个阶段有明确的里程碑和交付物，便于跟踪进度和管理项目时间表。
    - **客户批准(Client Approval)**：每个阶段可以得到客户的审查和批准，确保项目保持与客户期望的一致性。
- **风险管理(Risk Management)**：
    - **控制变更(Controlled Changes)**：变更在受控的情况下进行，所有变更都经过彻底审查和记录。

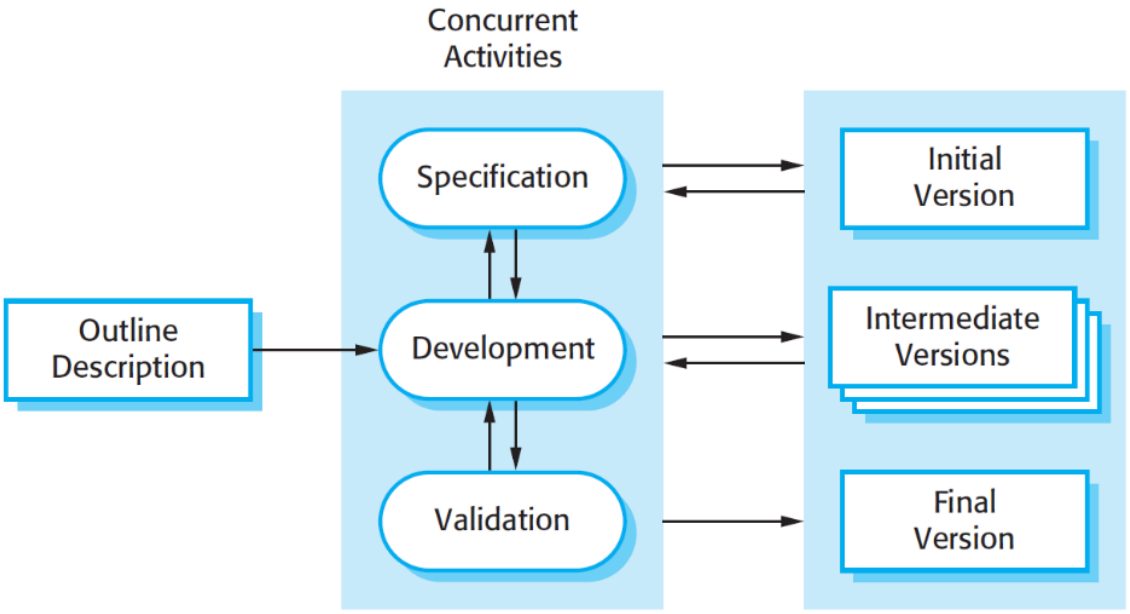
▼ Weaknesses and Limitations

- **缺乏灵活性(Inflexibility)**：
  - **刚性结构(Rigid Structure)**：难以在一个阶段完成后再进行变更。
  - **顺序进展(Sequential Progression)**：每个阶段必须完成后才能进入下一个阶段，导致一个阶段的延误或问题会影响整个项目时间表。
- **测试滞后(Late Testing)**：
  - **问题检测延迟(Late Problem Detection)**：测试通常在开发过程的后期进行，导致缺陷和问题被发现得较晚，增加了修复成本。
  - **有限的迭代反馈(Limited Iterative Feedback)**：由于测试在结束时进行，开发过程中提供反馈和改进的机会有限。
- **风险管理不足(Risk Management)**：
  - **高风险(High Risk)**：瀑布模型不包括风险管理活动，不适合不确定性高的项目。
- **客户参与度低(Customer Involvement)**：
  - **有限的反馈(Limited Feedback)**：客户通常在开发周期结束时才能看到产品，可能导致最终产品不完全符合他们的需求或期望。
  - **完美需求的假设(Assumption of Perfect Requirements)**：模型假设所有需求在一开始就能明确，这在现实项目中通常不成立。
- **适应性差(Adaptability)**：
  - **适应性差(Poor Adaptability)**：不适用于需求变化频繁或迭代开发有利的项目。
  - **范围蔓延(Scope Creep)**：范围的变化难以管理和整合，可能导致延误和成本增加。
- **开销和文档管理(Overhead and Documentation)**：
  - **文档繁重(Heavy Documentation)**：过多的文档可能耗费时间，导致延迟。
  - **管理开销(Administrative Overhead)**：管理文档和阶段转换增加了行政负担，不适合小型项目。
- **资源利用率低(Resource Utilization)**：
  - **资源闲置(Idle Resources)**：某些阶段资源可能未充分利用。
  - **顺序资源分配(Sequential Resource Allocation)**：不易进行并行工作流，导致资源利用效率低下。

- **项目时间线延长(Longer Project Timelines)**：
  - **时间线延长(Extend Timelines)**：每个阶段必须完成后才能开始下一个阶段，导致时间线延长。
  - **延迟交付(Delayed Delivery)**：最终产品只能在项目结束时交付，延迟了价值的实现。

Incremental Model

- Definition: The Incremental Model is an iterative approach to software development where the system is built incrementally. Each increment adds functional process to the system until the complete product is delivered.
- 允许更多的灵活性和迭代来解决Waterfull模型的局限



▼ Outline Description and Planning

- **目标**：定义项目整体范围和高层次需求。
- **交付物**：项目计划、高层次需求文档。
- **活动**：利益相关者访谈、需求研讨会、项目规划。

▼ Specification and Planning

- **目标**：为每个增量进行详细规划，包括具体需求、设计和任务。
- **交付物**：增量计划、详细需求。

- **活动**：需求分析、任务分解、资源分配。

▼ Design and Development

- **目标**：为特定增量设计并开发功能。
- **交付物**：设计文档、源代码、单元测试。
- **活动**：架构设计、编码、单元测试、代码评审。

▼ Integration and Validation

- **目标**：将新的增量与现有系统集成并进行测试。
- **交付物**：集成系统、测试用例、测试报告。
- **活动**：集成测试、系统测试、用户验收测试。

▼ Deployment / Increment

- **目标**：将增量部署到生产环境。
- **交付物**：已部署的增量、用户手册。
- **活动**：安装、配置、用户培训。

▼ Review and Feedback

- **目标**：从用户和利益相关者收集反馈。
- **交付物**：反馈报告、更新后的需求。
- **活动**：用户反馈会、需求更新。

▼ Next Increment Planning

- **目标**：根据反馈和更新后的需求规划下一个增量。
- **交付物**：更新后的项目计划、详细需求。
- **活动**：需求分析、任务分解、资源分配。

Characteristics of the Incremental Model

- **Iterative Development**: The project is divided into smaller, manageable increments.
- **Early Delivery**: Functional pieces of the system are delivered early and frequently.
- **User Feedback**: Regular feedback from users is incorporated into each increment.
- **Flexibility**: Easier to accommodate changes in requirements and feedback.
- **Risk Management**: Early identification and mitigation of risks through iterative development.

- **迭代开发**：通过初始实现逐步暴露给用户进行反馈，并通过多个版本的改进不断演变系统，直到开发出足够完善的系统。
- **规范、开发和验证交错进行**：这些活动不是分离的，而是紧密交织的，能够在开发过程中快速获得反馈。
- **与敏捷开发的联系**：增量开发是敏捷方法的重要组成部分，尤其适用于大多数商业、电子商务和个人系统开发。
- **便于变更**：在开发过程中对软件进行变更更加容易且成本较低。
- **早期版本提供核心功能**：系统的早期增量通常包括最重要或最紧急的功能，用户可以在早期阶段验证系统是否符合需求。
- **灵活性强**：如果功能不符合要求，可以仅修改当前增量，未来增量可以定义新的功能。

▼ Case of Incremental Model

- **Project**: Development of an E-commerce Platform
- **Phases**:
  - **Initial Planning**: Defined high-level requirements and project scope.
  - **Increment Planning**: Planned the first increment to include basic product listing and user registration features.
  - **Design and Development**: Designed and developed the first increment, performed unit testing.
  - **Integration and Testing**: Integrated the first increment with the existing system, performed system testing.
  - **Deployment**: Deployed the first increment to the production environment.
  - **Review and Feedback**: Gathered feedback from users and stakeholders, updated requirements.
  - **Next Increment Planning**: Planned the next increment to include shopping cart and checkout features.
  - **Iterative Process**: Repeated the cycle for subsequent increments, adding features like payment processing, order tracking, and user reviews.

- **初步规划**：定义高层需求和项目范围。

- **增量规划**：规划首个增量，包含基本的产品列表和用户注册功能。
- **设计和开发**：设计并开发第一个增量，并进行单元测试。
- **集成与测试**：将第一个增量与现有系统集成，并进行系统测试。
- **部署**：将第一个增量部署到生产环境。
- **反馈与评审**：收集用户和利益相关者的反馈，更新需求。
- **下一个增量规划**：规划下一增量，加入购物车和结账功能。

▼ Strengths and Benefits

- **早期交付与反馈 (Early Delivery and Feedback)**：
  - **早期功能交付 (Early Functional Delivery)**：增量模型允许系统的功能部分在开发过程中早期交付。
  - **用户反馈 (User Feedback)**：早期增量可以让用户进行审查，提供有价值的反馈，用于后续增量的优化。
- **灵活性与适应性 (Flexibility and Adaptability)**：
  - **适应变更 (Accommodates Changes)**：模型具有灵活性，可以更轻松地适应需求的变化。
  - **迭代开发 (Iterative Development)**：增量模型的迭代特性允许系统持续改进和优化。
- **风险管理 (Risk Management)**：
  - **降低风险 (Reduced Risk)**：通过将项目分解为较小的增量，减少了开发过程中的风险。
  - **早期问题检测 (Early Problem Detection)**：在每个增量结束时进行测试，有助于早期发现并解决缺陷和问题。
- **资源利用优化 (Improved Resource Utilization)**：
  - **并行开发 (Parallel Development)**：不同的增量可以并行开发，优化资源利用率，减少开发时间。
  - **专注开发 (Focused Efforts)**：团队可以专注于开发和优化特定功能，提高最终输出的质量。
- **更好的规划与估算 (Better Planning and Estimation)**：
  - **短期规划周期 (Shorter Planning Cycles)**：增量开发使得规划更具可控性，频繁进行，有助于提高估算的准确性。

    - **里程碑与交付物 (Milestones and Deliverables)**：每个增量提供明确的里程碑和交付物，便于跟踪进度和管理项目。
- **客户满意度 (Customer and Stakeholder Satisfaction)**：
    - **定期交付 (Regular Deliveries)**：定期交付功能性增量，使客户和利益相关者保持参与感，并及时获得反馈。
    - **符合预期 (Alignment with Expectations)**：持续的用户反馈确保产品符合客户的期望和需求。
- **提高质量 (Enhanced Quality)**：
    - **持续测试 (Continuous Testing)**：每个增量经过测试，确保早期发现并解决缺陷，有助于提高最终产品质量。
    - **增量集成 (Incremental Integration)**：逐步集成减少了系统集成的复杂性，有助于保持系统稳定性。
- **可扩展性 (Scalability)**：
    - **可扩展的方法 (Scalable Approach)**：增量模型具有可扩展性，适用于不同规模和复杂度的项目，尤其适合大规模项目。

▼ Weakness and Limitations

- **集成复杂性 (Complexity in Integration)**：
    - **集成挑战 (Integration Challenges)**：随着多个增量的开发和集成，集成增量的复杂性可能增加，确保所有增量无缝工作可能具有挑战性。
    - **依赖性管理 (Dependency Management)**：管理增量之间的依赖关系可能很困难，尤其是一个增量的变更可能影响到其他增量。
- **早期系统不完整 (Incomplete Systems Early On)**：
    - **部分功能性 (Partial Functionality)**：早期增量可能仅交付部分功能，无法立即为所有利益相关者提供完整的价值。
    - **客户不满 (Customer Dissatisfaction)**：如果早期增量不符合用户期望或提供的功能有限，可能导致客户不满和挫败感。
- **资源分配问题 (Resource Allocation Issues)**：
    - **资源限制 (Resource Constraints)**：在多个增量之间有效分配资源可能具有挑战性，团队需要在维护之前的增量同时开发新的增量。
    - **技能要求 (Skill Requirements)**：不同的增量可能需要不同的技能组合，可能需要一个多才多艺的团队或高效地分配专业资源。

- **规划与协调 (Planning and Coordination)**：
  - **详细规划需求 (Detailed Planning Required)**：虽然增量模型具有灵活性，但仍然需要详细的规划和协调，以确保每个增量与整体项目目标保持一致。
  - **复杂的时间表 (Complex Scheduling)**：协调多个增量的开发、测试和集成可能会产生复杂的时间安排挑战。
- **范围蔓延风险 (Risk of Scope Creep)**：
  - **无法控制的变更 (Uncontrolled Changes)**：在每个增量中引入变更的灵活性可能导致范围蔓延，如果不加以管理和控制，可能会影响项目进度。
  - **项目超支 (Project Overruns)**：如果变更频繁且影响重大，项目可能面临成本和进度超支。
- **增量交付的局限 (Incremental Delivery Limitations)**：
  - **相互依赖的增量 (Interdependent Increments)**：相互依赖性较高的增量可能需要更多的努力来确保一个增量的变更不会对其他增量产生不利影响。
  - **延迟的完全功能 (Delayed Full Functionality)**：用户可能无法在所有增量交付完成之前体验到系统的完整功能，可能延迟项目整体效益的实现。
- **管理复杂性 (Management Complexity)**：
  - **项目管理 (Project Management)**：管理一个增量项目比管理一个线性项目更复杂，可能需要精细的规划、协调和监控，以确保所有增量与整体项目目标保持一致。
  - **利益相关者管理 (Stakeholder Management)**：在增量开发过程中，保持利益相关者的参与和信息通畅可能具有挑战性，特别是在早期增量未满足预期的情况下。

Incremental Model vs Waterfall Model

▼ Different

- **需求稳定性 (Requirement Stability)**：
  - **增量模型 (Incremental Model)**：适合需求不断演变和灵活的项目，允许基于反馈进行迭代优化。
  - **瀑布模型 (Waterfall Model)**：更适合需求稳定且明确的项目，变化可能性小。

- **项目规模和复杂性 (Project Size and Complexity)**：
  - **增量模型 (Incremental Model)**：适合大型、复杂系统，可以分解为更小的、易于管理的增量。
  - **瀑布模型 (Waterfall Model)**：更适合规模较小、复杂度较低或需求明确的项目。
- **早期功能交付 (Early Delivery of Functionality)**：
  - **增量模型 (Incremental Model)**：允许系统的功能部分早期交付，能更早为利益相关者提供价值。
  - **瀑布模型 (Waterfall Model)**：完整的功能通常只有在整个开发周期结束时才能交付。
- **风险管理 (Risk Management)**：
  - **增量模型 (Incremental Model)**：适合高风险项目，通过迭代开发识别和减少风险。
  - **瀑布模型 (Waterfall Model)**：更适合低风险项目或风险明确的项目。
- **利益相关者参与 (Stakeholder Involvement)**：
  - **增量模型 (Incremental Model)**：需要积极和持续的利益相关者参与和反馈，贯穿整个开发过程。
  - **瀑布模型 (Waterfall Model)**：利益相关者的参与通常在项目开始和结束时较为有限。
- **文档和审计 (Documentation and Audits)**：
  - **增量模型 (Incremental Model)**：每个增量持续进行文档记录和合规检查，确保项目始终与需求保持一致。
  - **瀑布模型 (Waterfall Model)**：文档主要在项目开始和结束时创建，较难适应变更。
- **灵活性和适应性 (Flexibility and Adaptability)**：
  - **增量模型 (Incremental Model)**：高灵活性，适应需求变化和反馈，适合动态环境。
  - **瀑布模型 (Waterfall Model)**：灵活性较低，一旦阶段完成后难以进行变更。
- **资源可用性 (Resource Availability)**：

    - **增量模型 (Incremental Model)**：适合资源有限、按需求分配资源的项目。
    - **瀑布模型 (Waterfall Model)**：更适合资源稳定可用的项目。
- **维护和增强 (Maintenance and Enhancement)**：
    - **增量模型 (Incremental Model)**：适合需要持续维护和增强的项目，每个增量可解决新的需求。
    - **瀑布模型 (Waterfall Model)**：更适合需求变化较少的项目，主要在交付后进行维护。
- **上市时间 (Time-to-Market)**：
    - **增量模型 (Incremental Model)**：适合需要快速上市核心功能的项目，允许用户早期采用。
    - **瀑布模型 (Waterfall Model)**：适合时间要求不太紧迫的项目。
- **法规和合规要求 (Regulatory and Compliance Requirements)**：
    - **增量模型 (Incremental Model)**：适合在受监管行业中的项目，能持续进行合规验证和调整。
    - **瀑布模型 (Waterfall Model)**：更适合需求明确且变化不大的合规项目。
- **并行开发 (Parallel Development)**：
    - **增量模型 (Incremental Model)**：支持多个增量或模块的并行开发，加快整体进度。
    - **瀑布模型 (Waterfall Model)**：强调顺序开发，极少并行活动。
- **用户中心开发 (User-Centric Development)**：
    - **增量模型 (Incremental Model)**：适合需要用户反馈来优化需求和功能的项目。
    - **瀑布模型 (Waterfall Model)**：更适合用户需求在初期明确的项目。
- **集成需求 (Integration Requirements)**：
    - **增量模型 (Incremental Model)**：适合频繁需要集成新功能和组件的项目。
    - **瀑布模型 (Waterfall Model)**：更适合在项目后期进行较少集成需求的项目。
- **市场驱动开发 (Market-Driven Development)**：

- **增量模型 (Incremental Model)**：适合竞争激烈的市场，定期需要更新和功能改进。
- **瀑布模型 (Waterfall Model)**：更适合需求更新较少的稳定市场。

Real-world Example of Software Projects(考过)

▼ Waterfall Model

- **航空航天和国防系统**
  - **示例**：飞机上的航电系统软件。
  - **原因**：这些项目需要严格的前期规划、严格的合规要求以及详细的文档记录。瀑布模型的结构化方法确保所有要求在进入下一阶段前被满足。
- **医疗设备软件**
  - **示例**：MRI机器或患者监控系统的软件。
  - **原因**：医疗设备软件必须符合严格的法规标准，并通过严格的测试和验证。瀑布模型的顺序阶段支持全面的文档和合规检查。

▼ Incremental Model

- **电子商务平台**
  - **示例**：亚马逊、eBay。
  - **原因**：电子商务平台需要不断进化以应对市场需求和用户期望的变化。增量模型允许定期更新和新功能的发布。
- **社交媒体应用**
  - **示例**：Facebook、Twitter。
  - **原因**：社交媒体平台需要频繁更新和引入新功能来保持用户参与度。增量模型支持迭代开发和持续的用户反馈。
- **政府项目**
  - **示例**：国家税务管理系统或公共安全系统。
  - **原因**：政府项目通常有固定的要求和预算，并需要详细的文档和合规标准。瀑布模型的线性方法符合这些需求。
- **教育软件**
  - **示例**：在线学习平台如Coursera。

- **原因**：教育软件需要不断添加新课程和功能，并根据用户反馈进行改进。增量模型允许定期更新和增强功能。

Reuse-oriented Software Engineering(面向复用的软件工程)

- 在大多数软件项目中存在某种形式的非正式软件复用，且不论所使用的开发过程如何。
- 面向复用的开发方法依赖于大量可复用的软件组件，以及用于集成这些组件的框架

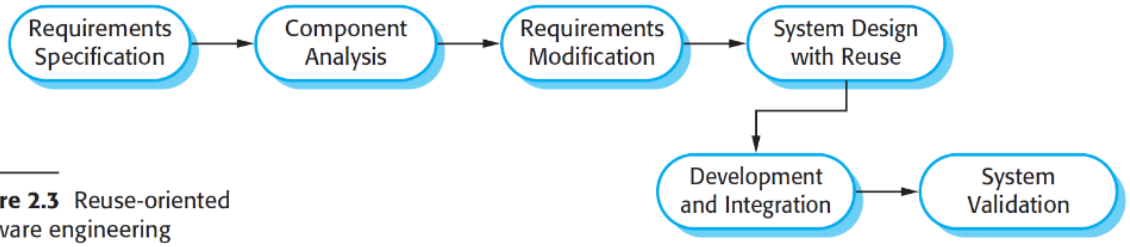


Figure 2.3 Reuse-oriented software engineering

复用过程中的组件类型：

- **Web服务(Web services)**：根据服务标准开发，可用于远程调用。
- **对象集合(Collections of objects)**：作为一个包开发，集成到组件框架中（如.NET或J2EE）。
- **独立软件系统(Stand-alone software systems)**：配置为在特定环境中使用。

面向复用的软件工程的优缺点：

- 优点：
  - 减少开发的软件量，降低成本和风险。
  - 通常能够加快软件的交付速度。
- 缺点：
  - 需求妥协是不可避免的，这可能导致系统不完全满足用户的需求。
  - 系统演化的部分控制权丧失，因为复用组件的新版本并不受使用该系统的组织控制。

Software Process Activities

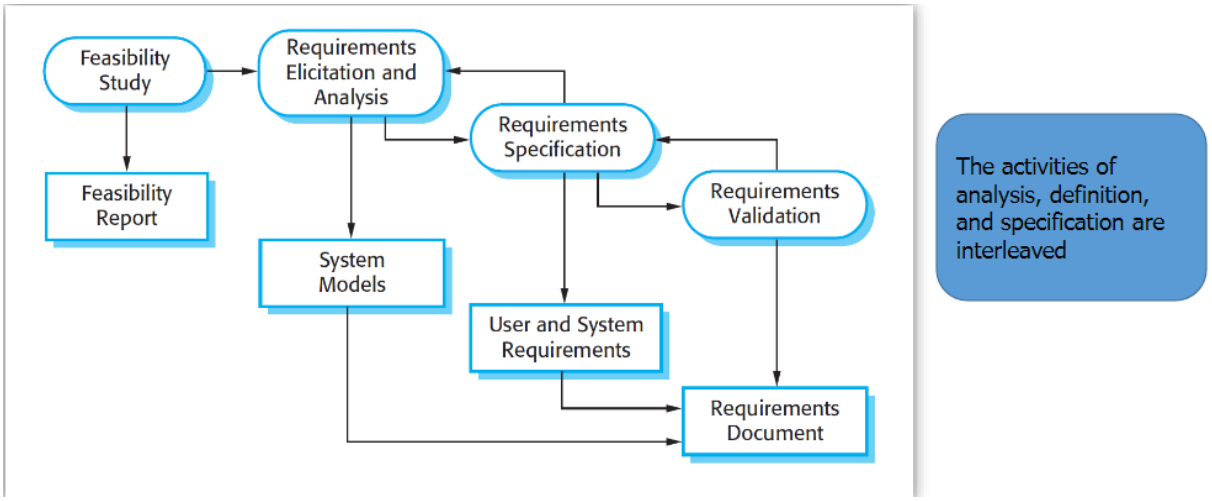
▼ Four fundamental activities to software engineering: (include sub-activities and supporting process)

- Software specification
- Software design and implementation: comprises of programming/codding
- Software validation
- Software evolution

Software Specification

- 软件规格或需求工程是理解和定义系统需要提供的服务以及系统的操作和开发所面临的限制的过程。
- 需求工程是软件开发过程中一个非常关键的阶段，因为这一阶段的错误往往会导致后续系统设计和实现中的问题。

Software Specification workflow



Software specification 目标是完成一个文档来满足股东的需求，需求通常分为两层：

- 客户和终端用户需要高层次的需求说明；
- 开发人员需要更详细的系统规格说明。

▼ Main activities in the requirements

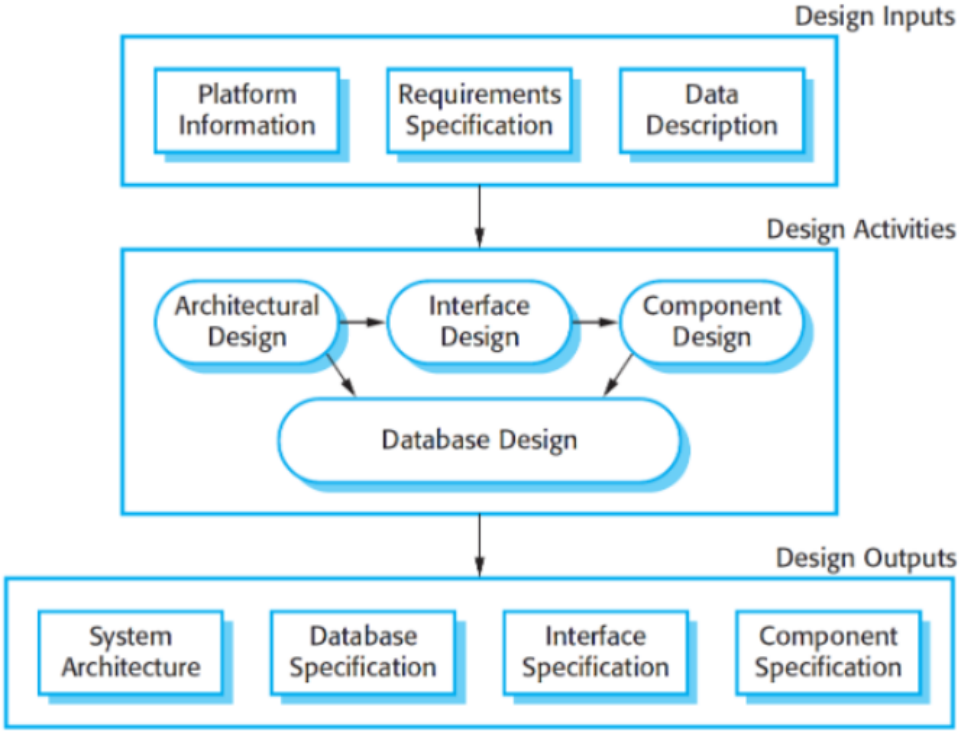
1. Feasibility study—可行性研究
2. Requirements elicitation and analysis—需求获取与分析

3. Requirements specification—需求规格说明
4. Requirements validation—需求验证

Software Design and Implementation

- 该阶段的主要目的是将Sofwtare Specification转换为可执行系统。
- 涉及软件设计和编程的过程，如果使用增量开发方式，可能还会涉及到对软件规格的改进。
- 软件设计通常是对将要实现的结构、数据模型、系统组件接口以及有时使用的算法的描述。

Software Design and Implementation workflow



- 设计并非一次性完成，而是通过迭代进行的，不断完善设计，修正之前的设计。

Design Activities

▼ Architectural design

- 架构设计涉及整个系统的总体结构规划。

- 设计者需要确定系统的主要组成部分或子系统（也称为模块）的划分，以及这些子系统之间的关系和分布方式。
- 这一阶段的设计会影响到系统的整体性能、可扩展性和可维护性。

▼ Interface design

- 接口设计的目的是定义系统组件之间如何进行通信。
- 界面设计要求定义明确、无二义性，以确保各个组件能够无缝合作。
- 良好的接口设计可以提高系统的灵活性，使得各个模块在保持功能独立的同时能够协同工作。

▼ Component design

- 在组件设计阶段，设计者将软件系统中的每个组件逐一设计，以定义其功能和操作方式。
- 具体内容包括：
  - **预期功能的简单描述**：即每个组件应该执行的核心任务。
  - **对可复用组件所需更改的列表**：列出组件需要的修改，使其适应当前项目的需求。
  - **详细设计模型**：通常采用基于模型的设计方法，生成可视化的设计图来表示组件的运作。

▼ Database design

- 这一设计活动专注于系统中数据结构的设计，以及如何将数据表示在数据库中。
- 设计师需要考虑数据库的表结构、字段类型、索引等，以便数据能够高效地存储和检索。

Limitation and Solution

▼ The detail and representation of the design output vary considerably

- **对于关键系统 (Critical Systems)**：
  - 在开发关键系统时，通常需要生成非常详细的设计文档，这些文档要对系统进行精确、准确的描述。
  - 例如，医疗系统、航空控制系统等涉及生命安全的系统，必须确保设计的准确性，以保证系统运行的可靠性和安全性。
- **模型驱动方法 (Model-driven Approach)**：

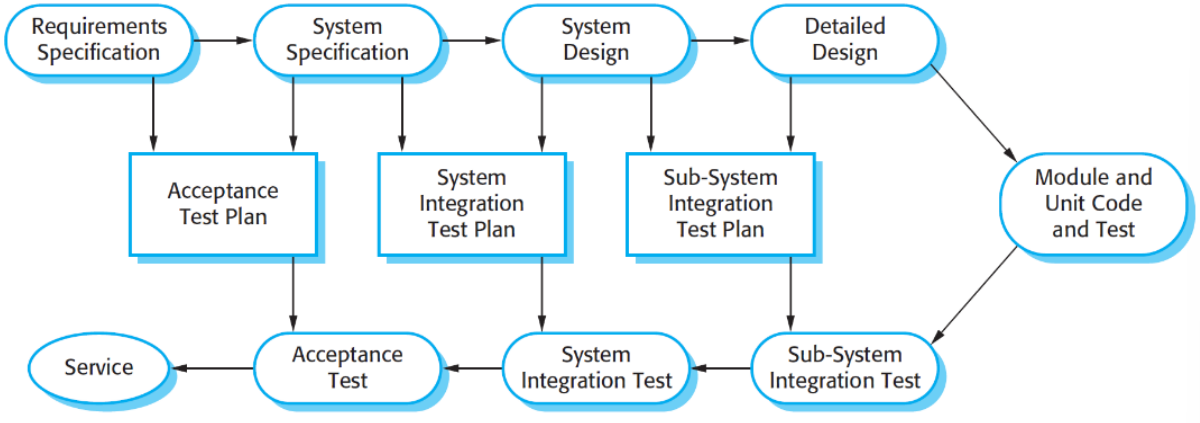
- 如果采用模型驱动的设计方法，设计的输出通常是图表或模型。
- 这种方法将软件系统的设计过程抽象化为模型，并通过这些模型定义系统的架构和细节。这类设计常常用于系统的可视化和结构化设计。

- **敏捷开发方法 (Agile Methods)**：
  - 在使用敏捷开发方法时，设计过程的输出可能并不一定是独立的设计文档。
  - 敏捷方法注重快速迭代开发，设计细节可能直接体现在程序代码中，而不是以传统的文档形式存在。
  - 敏捷开发倾向于减少文档生成的时间，将更多的精力投入到编码和功能实现上。

Software Validation

- **软件验证**，或更广泛的验证与确认（Verification and Validation, V&V），旨在确保系统：
  1. 符合其规格说明书(specification)；
  2. 满足系统客户的期望。
- **验证技术**：
  - **程序测试**：在测试中，系统通过模拟测试数据执行，这是主要的验证技术。
  - 验证过程还可能涉及检查流程，例如每个软件过程阶段的检查和审查，从用户需求定义到程序开发。

Software Validation workflow



Three stage for testing

- 系统组件测试(System components)；
- 集成系统测试(Integrated system)；
- 最终使用客户数据进行系统测试(Customer’s data)。

▼ The stages in the testing process:

- **开发测试(Development testing)**：由开发人员独立地测试每个系统组件。
- **系统测试(System testing)**：将各个系统组件集成在一起进行测试，重点在于检测意外的组件交互问题。
- **接受测试(Acceptance testing)**：这是最终阶段，使用客户提供的数据对系统进行测试，确保其可以正常运营。

- 通常，组件开发和测试是交错进行的，程序员会逐步测试开发的代码。
- 在增量开发中，每个增量都在开发时测试。
- 在计划驱动的软件开发中，测试会基于预先制定的测试计划，由独立的测试团队负责。

Software Evolution(软件演化）

- **硬件与软件的更改**：
  - 一旦决定制造硬件，修改硬件设计会变得非常昂贵。然而，软件可以在系统开发的任何时间或开发完成后进行更改。即使是广泛的更改，软件的修改成本也远远低于系统硬件的相应更改成本。
- **软件开发与进化的区别**：
  - 历史上，软件开发过程和软件进化（即软件维护, software maintenance）的过程一直被分为两个不同的阶段。开发是创建新的系统，而进化是维护与调整已有的软件系统。
- **对开发与维护的认知**：
  - 人们通常认为软件开发是一个创造性的过程，从初始概念到可运行的系统逐步形成。而软件维护有时被视为枯燥且无趣的任务。
- **软件工程的进化性本质**：
  - 与其将开发与维护看作两个独立的过程，更实际的观点是把软件工程视为一个不断演进的过程。软件系统会随着时间的发展、用户需求的变化、环境的改变等持续改进和演化。

Appendix

Fundamental Process Activities

▼ Specification

- **Requirements Elicitation（需求获取）**：通过访谈、调查、研讨会、观察等方式从利益相关者那里收集需求。
- **Requirements Analysis（需求分析）**：分析收集到的需求，确保其完整、清晰、可行，并优先处理需求冲突。
- **Requirements Documentation（需求文档）**：将需求记录在文档中，通常使用需求说明文档、用户故事或用例来表达。
- **Requirements Validation（需求验证）**：确保文档中的需求准确反映了利益相关者的需求，通常通过评审、检查和原型进行验证。
- **Requirements Management（需求管理）**：随着项目的推进，管理需求的变化，确保所有利益相关者了解这些变化及其影响。

▼ Design and Implementation

- **Architectural Design（架构设计）**：定义软件的整体结构，包括主要组件、它们之间的相互作用以及使用的技术。
- **Detailed Design（详细设计）**：指定每个组件的内部细节，包括算法、数据结构和接口。
- **Prototyping（原型设计）**：创建原型以探索设计选项，并与利益相关者一起验证设计决策。
- **Coding（编码）**：编写软件组件的实际源代码。
- **Code Reviews and Inspections（代码评审和检查）**：通过正式或非正式的代码审查，尽早识别和修正开发过程中的问题。
- **Unit Testing（单元测试）**：测试单独的组件或代码单元，确保它们按预期工作。
- **Integration（集成）**：将各个组件组合成一个连贯的系统，确保它们正确协同工作。
- **Build Management（构建管理）**：自动化编译和组装软件的过程。

▼ Validation

- **Test Planning（测试计划）**：定义测试的范围、目标和方法，创建测试计划，概述测试策略、资源、进度和交付成果。
- **Test Case Development（测试用例开发）**：创建详细的测试用例，指定输入、预期结果以及每个测试的执行条件。
- **Test Execution（测试执行）**：运行测试用例并记录结果。
- **Defect Tracking（缺陷跟踪）**：识别、记录并跟踪测试中发现的缺陷，通常使用缺陷跟踪工具。
- **Regression Testing（回归测试）**：重新运行之前执行过的测试，以确保修改或修复不会引入新的缺陷。
- **Acceptance Testing（验收测试）**：进行测试以确保软件满足利益相关者定义的验收标准，并为部署做好准备。

▼ Evolution & Maintenance

- **Corrective Maintenance（纠错性维护）**：在软件部署后修复发现的缺陷。
- **Adaptive Maintenance（适应性维护）**：修改软件以适应环境变化，例如新的操作系统、硬件或第三方软件。
- **Perfective Maintenance（完善性维护）**：增强软件性能、可用性或可维护性，可能涉及重构代码、优化算法或改进用户界面。
- **Preventive Maintenance（预防性维护）**：采取措施防止未来问题，例如更新库、改进安全性或添加日志和监控功能。
- **Release Management（发布管理）**：计划和协调更新和新版本的软件发布。
- **User Support（用户支持）**：为用户提供支持，包括问题排查、答疑和培训。

Supporting Process Activities

▼ Project Management

- **Planning（规划）**：定义项目的范围、目标和所需的资源。创建包含时间表和里程碑的项目计划。
- **Scheduling（时间安排）**：分配时间和资源给不同的任务和活动。
- **Risk Management（风险管理）**：识别、分析并缓解可能影响项目的风险。

- **Monitoring and Control（监控与控制）**：跟踪进度，确保项目按计划进行、在预算内，并根据需要做出调整。

▼ Configuration Management

- **Version Control（版本控制）**：管理软件代码、文档和其他工件的变更。
- **Change Management（变更管理）**：处理软件变更请求，并确保这些变更以受控的方式实施。
- **Build Management（构建管理）**：自动化软件编译和组装的过程。

▼ Deployment

- **Release Management（发布管理）**：规划和协调软件发布给用户。
- **Installation and Configuration（安装与配置）**：确保软件在目标环境中正确安装和配置。
- **User Training and Support（用户培训与支持）**：为用户提供培训和支持，确保他们能够有效使用软件。

▼ Documentation

- **User Documentation（用户文档）**：为最终用户创建手册、指南和帮助系统。
- **Technical Documentation（技术文档）**：为开发者和维护人员记录设计、架构和代码。

▼ Process Improvement

- **Process Assessment（过程评估）**：评估当前的软件开发过程，识别其优势和劣势。
- **Process Improvement Initiatives（过程改进措施）**：实施变更以提高软件开发过程的效率和有效性。

Incremental and Agile

▼ 敏捷方法（Agile Methodologies）：

- 敏捷方法具有高度的灵活性，强调与客户的协作，能够快速适应变化。

▼ 增量模型（Incremental Model）：

- 与敏捷方法相比，增量模型更加结构化，专注于计划好的增量交付功能。

- 敏捷开发专注于交付小的、功能性增量，每个增量在较短的时间框架内（time-boxed iterations）完成，并且过程中持续从客户那里获得反馈。
- 核心是灵活和快速迭代，确保在项目开发过程中根据客户需求做出及时调整。

- 增量模型也允许变更，但适应性较弱，更多地强调文档编写和规划。
- 这种方法依赖于按阶段交付各个部分，逐步构建完整的软件系统。

- **共同目标与差异**：
  - 两种方法的共同目标是通过增量的方式交付功能性软件。
  - 区别在于敏捷方法更注重灵活性、客户的持续参与和反复迭代改进，而增量模型则注重提前规划与文档。

TTL

Week 2 Tutorial - Software Process.pdf

Suggest the most appropriate software process model that could be used as a basis for managing the development of the following systems, justify your suggestion:

▼ a. A system to control anti-lock braking in a car

▼ b. A virtual reality system to support software maintenance

▼ c. A corporate accounting system that replaces an existing system

▼ d. An interactive travel planning system that helps users plan journeys with the lowest environmental impact

Explain why incremental development is the most effective approach for developing business software systems, Why is this model less appropriate for real-time systems engineering?

▼ Answer

Consider the reuse-based process model. Explain why is essential to have two separate requirements engineering activities in the process.

▼ Answer

In waterfall software development process model, we complete one stage and process to the next stage. The engineer may revisit the previous stage if refinement is needed. However, it is important that the number of time the engineer revisit the previous stage should be limited. Using your own statement, explain why the number of revisit should be limited.

▼ Answer

Explain why the requirement engineering is a critical stage in software process.

▼ Answer