CPT203 Final

Lecture 1 Introduction

- Computer software
  - a collection of instructions, data, or computer programs that are used to run machines and carry out particular activities. Software product may be developed for a particular customer or may be developed for a general market
- Software system
  - a system that consists of a number of separate computer software, configuration files, system documentation and user documentation
  - The term "software system" should be distinguished from the terms "computer software". The term computer software generally refers to a set of instructions that perform a specific task. However, software system generally refers to a more encompassing concept with many more components
- Software development
  - Amateur 业余的
    - People in business write spreadsheet programs to simplify their jobs
    - scientists and engineers write program to process their experimental data
    - hobbyists write programs for their own interest and enjoyment
  - Professional
    - developed for specific business purposes, for inclusion in other devices, or as software products such as
      - intended for use by someone apart from its developer
      - is usually developed by teams rather than individuals
      - maintained and changed throughout its life
    - Usually has following properties
      - Strict user requirements
      - Required accuracy and data integrity
      - Higher security standard
      - Stable performance for heavy load
      - Required technical support
      - Maintainability
      - Dependability and security
      - Efficiency
      - Acceptability

- Two kinds of software product
    - Generic software product
        - These are systems that are produced by a development organization and sold on the open market to any customer who is able to buy them
    - Customized software products
        - These are systems that are commissioned by a particular customer. A software contractor develops the software especially for that customer
- Software Deterioration
    - In theory, therefore, the failure rate curve for software should take the form of the "idealized curve"
    - To reduce software changes:
        - work closely with the stakeholder to ensure requirements are correctly defined
        - Improve requirement study approach to achieve better requirements study
    - To reduce side effects after changes
        - The software should be modular so that changes will not have a lot of side effects to other part of the software
        - The software must be maintainable
        - Comprehensive testing should put in place to reduce errors
- Software Engineering Approaches
    - The systematic approach that is used in software engineering is sometimes called a software process
    - A software process is a sequence of activities that leads to the production of a software product
    - There are four fundamental activities that are common to all software processes
        - Software specification
        - Software development
        - Software validation
        - Software evolution
- General issues that affect most software
    - Heterogeneity 异构性
        - Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices
    - Business and social change
        - Business and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need

        to be able to change their existing software and to rapidly develop new software
    - Security and trust
        - As software is intertwined with all aspects of our lives, it is essential that we can trust that software
- Software engineering and the web
    - The web is now a platform for running application and organizations are increasingly developing web-based systems rather than local systems
    - Web services allow application functionality to be accessed over the web
    - Cloud computing is an approach to the provision of computer services where applications run remotely on the 'cloud'
        - Users do not buy software buy pay according to use
- Software Engineering Ethics
    - Ethics, as understood here, addresses any intentional action that impacts negatively or positively the lives and values of others
    - The ethical activity involved in technical decisions should be based on an understanding of the impact of those decisions
- Software Development Risk
    - Many software development projects run into difficulties
        - Does not work as expected
        - Over budget
        - Late delivery
    - Many software projects fail because the software developers build the wrong software. The software development team must:
        - Fully understand requirement
        - Validate requirement

Lecture 2 Software Processes

Intro

- A software process is a set of related activities that leads to the production of a software product. These activities may involve the development:
    - of software from scratch
    - by extending and modifying existing systems
    - by configuring and integrating off-the-shelf software or system components
- A software process refers to the actual set of activities, methods, practices, and transformations that people use to develop and maintain software and its associated products(such as project plans, design documents, code, test cases, and user manuals)
  开发 / 维护软件及其相关产品的实际活动，方法，实践和转换的集合

- It is the real-world implementation of the steps involved in creating software
- Software Process Activities
    - There are many different software processes, but all must include four activities that are fundamental to software engineering
        - Software specification
        - Software design and implementation
        - Software validation
        - Software evolution 软件演进
    - In practice, they are complex activities in themselves and include sub-activities
    - There are also supporting process activities such as documentation and software configuration management 文档化 / 软件配置管理
- Types of Software processes
    - Software processes are complex and, there is no ideal process
    - Most organizations have developed their own software processes that take advantage of the capabilities of the people in an organization and the specific characteristics of the systems that are being developed
      大多数组织都开发了自己的软件过程，这些过程利用了组织内人员的能力以及所开发系统的特定特征
        - For a critical systems, a very structured development process is required
        - For 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
      线形顺序模型
    - Iterative and incremental Models: Incremental, Iterative, Spiral
      迭代/增量模型
    - Agile methods: Scrum, kanban, Extreme Programming(XP)
      敏捷模型
    - Prototyping Models: Throwaway Prototyping, Evolutionary Prototyping
      原型模型：抛弃式原型，演化原型
    - Component-Based Models: Component-Based Development(CBD)
      基于组件的模型
    - Formal Methods: Formal Specification, Model Checking
      形式化方法：形式化规格 / 模型检查
    - Hybrid Models: Agile-Waterfall hybrid, DevOps
      混合模型
    - Rapid Application Development(RAD) models: RAD Model, DSDM
    - Lean Models: Lean Software Development
      精益模型

Software Process Models

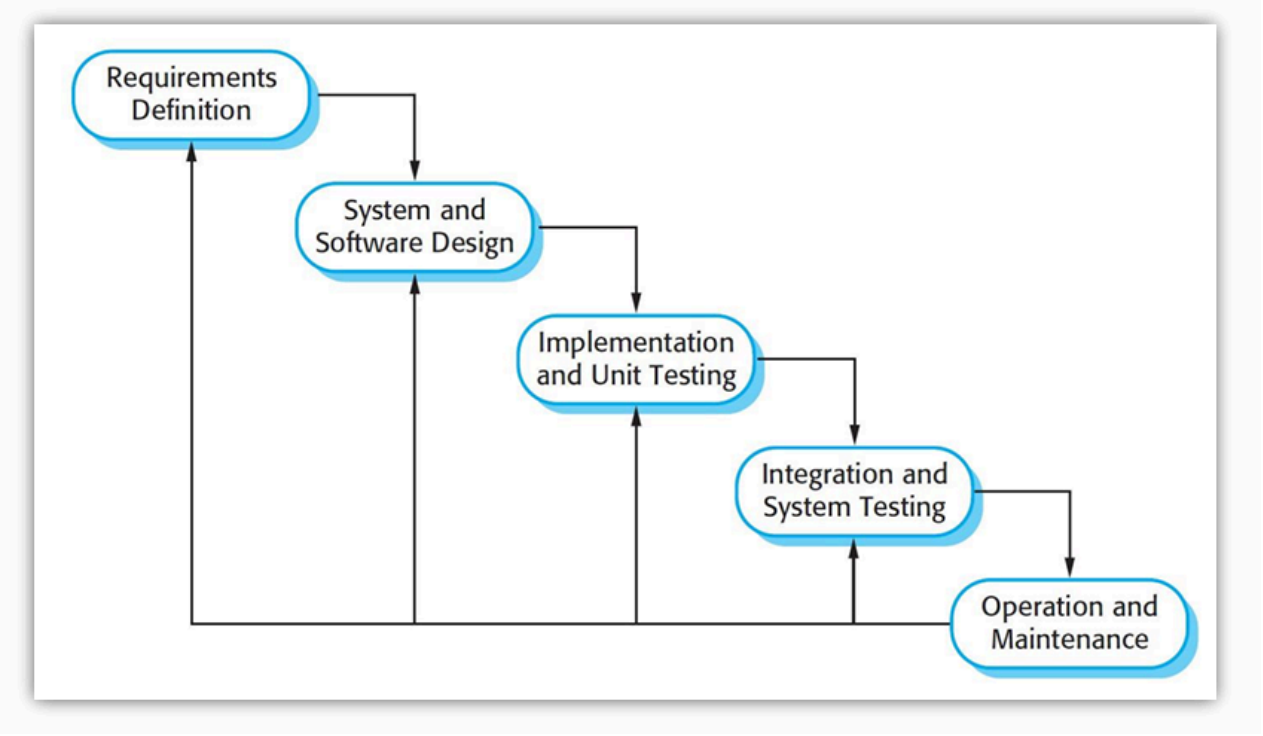
- A software process model is simplified representation of a software process
- These generic models are not definitive descriptions of software process that can be used to explain different approaches to software development
  这些通用模型不是对软件过程的权威描述，他们是过程的抽象，可用于解释不通的软件开发方法
- Three key Software Process Models in our discussion
  - The waterfall model
    - 将开发规格说明，开发，验证和演进这些基本过程活动视为单独的过程阶段，如需求规格说明，软件设计，实现，测试等
  - Incremental development
    - 这种方式交织进行规格说明，开发，验证活动。系统被开发为一系列版本（增量），每个版本都向先前版本添加功能
  - Reuse-oriented software engineering
    - 这种方法基于存在大量可复用组件，系统开发过程侧重将这些组件集成到系统中，而不是从头开发他们

These models are not mutually exclusive and are often used together, especially for large systems development

The Waterfall Model

- Definition: The Waterfall Model is a linear and sequential approach to software development. It progresses through distinct phases, each with specific deliverables and review processes.
  线形顺序的软件开发方法，他通过不同的阶段进展，每个阶段都有特定的交付成果和审查

过程



- Phases of the Waterfall Model
  - The waterfall is an example of a plan-driven process
  - The principal stages of the waterfall model directly reflect the fundamental development activities:
    - Requirements analysis and definition
      - Objective: Gather and document all function and non-functional requirements
      - Deliverables: Requirements Specification Document
      - Activities: Stakeholder(利益相关者) interviews, surveys, and requirement workshops
    - System and software design
      - Objective: Create a detailed design based on the requirements
      - Deliverables: System architecture document, Design specifications
      - Activities: Architectural design, database design, user interface design
    - Implementation and unit testing
      - Objective: Translate design documents into actual code
      - Deliverables: Source code, code documentation
    - Integration and system testing
      - Objective: Integrate all modules and test the complete system
      - Deliverables: Test plans, test cases, test reports
      - Activities: Integration testing, system testing, user acceptance testing

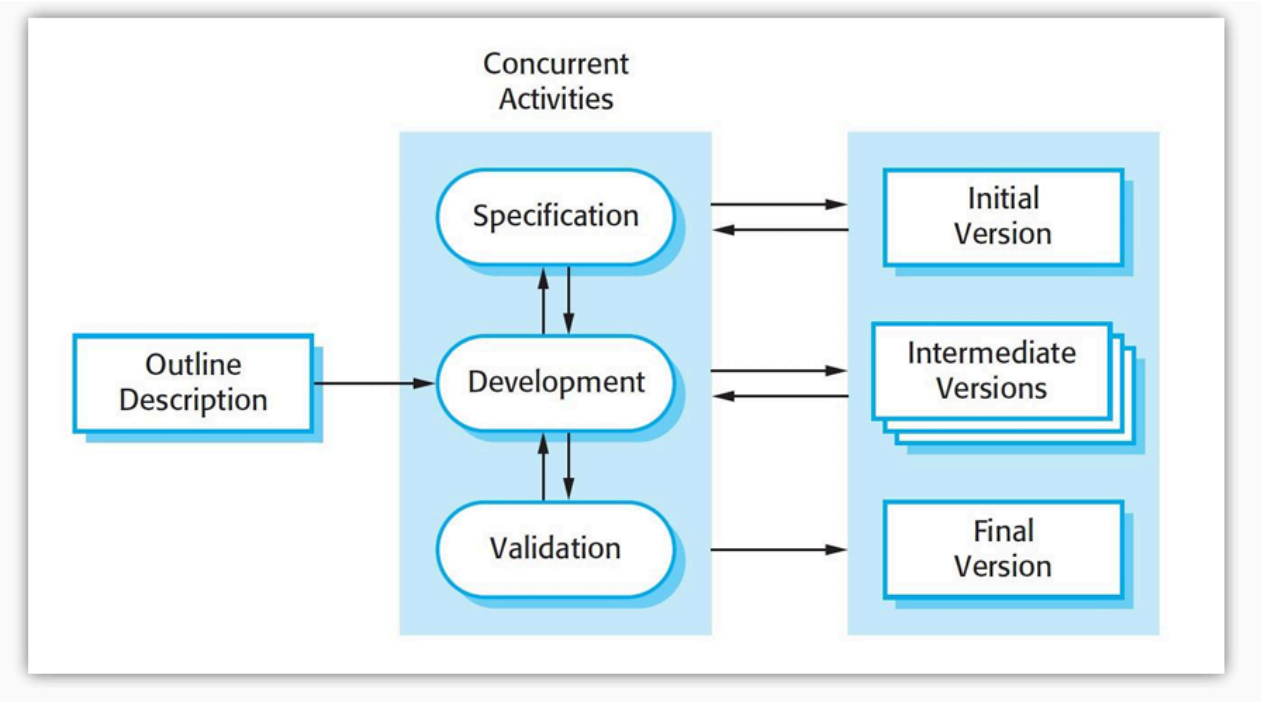
        - Operation and maintenance
            - Objective: Deploy the system to the production environment and provide ongoing support and enhancements
            - Deliverables: Deployment plan, user manuals, maintenance reports, updating patches
            - Activities: Installation, configuration, user training, bug fixes, performance improvements, feature updates
- 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
    - Because of the costs of producing and approving documents, iterations can be costly and involve significant rework. Therefore, after a small number of iterations, it is normal to freeze parts of the development, such as the specification, and to continue with the later development stages. Problems are left for later resolution, ignored, or programmed around
- Strengths of Waterfall Model
    - Simplicity and ease of use 简单易用
        - clear structure
        - well-defined Phases
    - Predictability and Planing 可预测性和规划
        - Detailed planing（可以亲送的计算成本，实践，资源）
        - Predictable Outcomes
    - Documentation
        - Comprehensive Documentation 全面的文档
            - Traceability 可追溯性
    - Discipline and Control
        - Structured Approach
        - Phase completion
    - Resource Management
        - Resource Allocation
        - Skill Utilization
    - Milestones and Deliverables
        - clear milestones

        - client approval
    - Risk Management
        - Controlled changes
- Weaknesses
    - Inflexibility
        - Rigid Structure 刚性结构
        - Sequential Progression 顺序推进（每个阶段必须在进入下一个阶段之前完成，这意味着任何一个阶段的延迟或问题都可能影响整个项目时间线）
    - Late Testing
        - Late Problem Detection 问题发现晚
        - Limited Iterative Feedback 迭代反馈有限
    - Risk Management
        - High risk
    - Customer Involvement
        - Limited Feedback
        - Assumption of Perfect Requirements
    - Adaptability
        - Poor Adaptability: 不太适合需求预期会演变或迭代开发有变的项目
        - Scope Creep
    - Overhead and Documentation
        - Heavy Documentation
        - Administrative Overhead
    - Resource Utilization
        - Idle Resources
        - Sequential Resource Allocation
    - Longer Project Timelines
        - Extended Timelines
        - Delayed Delivery

Incremental Model

- Definition: The incremental model is an iterative approach to software development where the system is built incrementally. Each increment adds functional pieces to the

system until the complete product is delivered



- Phases of the Incremental Model
  - Outline Description and Planning
    - Objective: Define the overall project scope and high-level requirements
    - Deliverable: Project plan, high-level requirements document
    - Activities: Stakeholder interviews, requirement workshops, project planning
  - Specification and Planing
    - Objective: Plan the details for each increment, including specific requirements, design, and tasks
    - Deliverables: Increment plan, detailed requirements for the increment
    - Activities: Requirement analysis, task breakdown, resource allocation
  - Design and Development
    - Objective: Design and develop the functionality for the specific increment
    - Deliverables: Design documents, source code, unit tests
    - Activities: Architectural design, coding, unit testing, code reviews
  - Integration and Validation
    - Objective: Integrate the new increment with the existing system and perform testing
    - Deliverables: Integrated system, testing cases, test reports
    - Activities: Integration testing, system testing, user acceptance testing
  - Deployment / Increment
    - Objective: Delay the increment to the production environment
    - Deliverables: Deployed increment, user manuals
    - Activities: Installation, configuration, user training

    - Review and Feedback
        - Objective: Plan the next increment based on feedback and updated requirements
        - Deliverables: Updated project plan, detailed requirements for the next increment
        - Activities: Requirement analysis, task breakdown, resource allocation
- 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 and feedback
    - Risk Management: Early identification and mitigation of risks through iterative development
    - 规格说明，开发和验证活动是交织在一起的，增量软件开发是agile approach的基本组成部分，对于大多数业务系统，电子商务和个人系统来说，它比瀑布方法更好
- Strengths
    - Early delivery and feedback
        - Early Functional Delivery
        - User feedback
    - Flexibility and Adaptability
        - Accommodates Changes
        - Iterative Development
    - Risk Management
        - Reduced risk
        - Early problem detection
    - Improved Resource Utilization
        - Parallel Development
        - Focused Effort
    - Better Planning and Estimation
        - Shorter Planing Cycles
        - Milestones and Deliverables
    - Customer and Stakeholder Satisfaction
        - Regular Deliveries
        - Alignment with expectation
    - Enhanced Quality
        - Continuous testing

        - Incremental Integration
    - Scalability
        - Scalable Approach
- Weaknesses
    - Complexity in Integration
        - Integration Challenges
        - Dependency Management
    - Incomplete Systems early on
        - Partial functionality
        - Customer Dissatisfaction
    - Resource allocation
        - Resource constraints
        - Skill requirements
    - Planning and Coordination
        - Detailed Planing Required
        - Complex Scheduling
    - Risk of Scope Creep
        - Uncontrolled changes
        - project overruns
    - Incremental Delivery
        - Interdependent increments
        - delayed full functionality
    - Management Complexity
        - Project Management
        - Stakeholder Management

Incremental Model vs Waterfall Model

1. Requirement Stability
    - Incremental Model: Ideal for projects with evolving and flexible requirements. Allows for iterative refinement based on feedback
      适合需求演变的项目
    - Waterfall Model: Best for projects with stable and well-defined requirements that are unlikely to change
      需求稳定
2. Project Size and complexity
    - Incremental Model: Suitable for large, complex systems that can broken down into smaller, manageable increments
      适合大型复杂系统

- Waterfall Model: More suitable for smaller, less complex projects or those with well-understood complexity
  适合复杂度小 完全理解的项目

3. Early delivery and Functionality
   - Incremental Model: enables early delivery of functional parts of the system, providing value to stakeholders sooner
     能够在早期交付，为利益相关者提供价值
   - Waterfall Model: Full functionality is delivered only at the end of the development cycle
     要等到开发周期结束
4. Risk Management
   - Incremental Model: High-risk projects benefit from early identification and mitigation of risks through iterative development
     高风险项目：可以在迭代开发过程中早期识别
   - Waterfall Model: Lower risk projects or those with well-understood risks are more suited to the waterfall approach
     低风险/完全理解
5. Stakeholder Involvement
   - Incremental Model: requires active and continuous stakeholder involvement and feedback throughput the development process
     持续给予反馈
   - Waterfall Model: Limited stakeholder involvement is typically seen, primarily at the beginning and end of the project
     参与有限，通常在开始与结束时间
6. Documentation and Audits
   - Incremental Model: Continuous documentation and compliance checks with each increment ensure ongoing alignment with requirements
     持续文档和合规性检查确保与需求一致
   - Waterfall Model: Extensive documentation is created upfront and at the end of each phase, which can be less adaptable to changes
     广泛的文档在早期或者每个阶段结束创建，难适应变化
7. Flexibility and Adaptability
   - Incremental Model: High flexibility to adapt to changes in requirements and feedback, making it suitable for dynamic environments.
     高度灵活 能够适应需求和反馈变化
   - Waterfall Model: Low flexibility; changes are difficult to incorporate once a phase is completed, making it less adaptable.
     灵活性低 一旦完成，很难纳入变更
8. ResourceAvailability

- Incremental Model: Suitable for projects with limited resources that can be allocated incrementally based on project needs.
  适用于资源有限，可以根据项目增量分配的项目
- Waterfall Model: Best for projects with consistent resource availability throughout the lifecycle.
  整个周期资源可用性一致的项目

9. Maintenance andEnhancement
   - Incremental Model: Ideal for projects requiring ongoing maintenance and enhancements, as each increment can address new needs.
     持续维护和增强的项目
   - Waterfall Model: More suitable for projects with minimal maintenance needs post-deployment.
     部署后维护需求最少的项目
10. Time-to-Market
    - Incremental Model: Suitable for projects needing a quick time-to-market for core functionalities, allowing for early user adoption.
      快速上市
    - Waterfall Model: More suitable for projects where time-to-market is less critical.
      上市时间不那么关键的
11. Regulatory and ComplianceRequirements
    - Incremental Model: Projects in regulated industries benefit from continuous compliance verification and adjustments.
      受监管行业的项目收益与持续的合规性验证和调整
    - Waterfall Model: Suitable for projects with well-understood regulatory requirements that are stable and unlikely to change.
      稳定不太会变化的项目
12. Parallel Development
    - Incremental Model: Supports parallel development of different increments or modules, enabling faster overall progress.
      支持不同增量或模块的并行开发
    - Waterfall Model: Emphasizes sequential development with minimal parallel activities.
      强调顺序开发
13. User-Centric Development
    - Incremental Model: Ideal for projects where user feedback is critical for refining requirements and functionalities.
      适用于用户反馈对于结果重要的项目
    - Waterfall Model: Best for projects with clearly defined user requirements from the outset.
      一开始就有明确的用户需求的项目

14. Integration Requirements
    - Incremental Model: Suitable for projects requiring frequent integration of new features and components.
      需要频繁集成新功能和组件的项目
    - Waterfall Model: More suitable for projects with minimal integration needs until the final stages.
      一开始就有明确定义用户需求的项目
15. Market-Driven Development
    - Incremental Model: Competitive markets requiring regular feature updates and improvements benefit from incremental development.
      定期功能更新和改进的竞争市场
    - Waterfall Model: More suitable for stable markets with less frequent need for updates
      更新需求不频繁的稳定市场

Reused-oriented Software Engineering

- There are 3 types of software component that may be used in a reused-oriented process
    - Web service that are developed according to service standards and which are available for remote invocation
    - Collections of objects that are developed as package to be integrated with a component framework such as .NET or J2EE
    - Stand-alone software systems that are configured for use in particular environment
- Advantages
    - reducing the amount of software to be developed and so reducing cost and risks
    - usually also leas to faster delivery of the software
- However, requirements compromises are inevitable, and this may lead to a system that does not meet the real needs of users
- Furthermore, some control over the system evolution is lost, as new versions of the reusable components are not under the control of the organization using them
  可复用组件的新版崩不由使用它们的组织控制

Software Process Activities

- Real software processes are interleaved sequences of technical, collaborative, and managerial activities with the overall goal of specifying, designing, implementing, and testing a software system
- The four basic process activities of specification, design and implementation, validation and evolution are organized differently in different development process
    - In waterfall, organized in sequence

- In incremental development they are interleaved

Software Specification

- Software specification or called requirements engineering is the process of understanding and defining the followingx
  - what services are required from the system 系统需要提供哪些服务
  - identifying the constraints on the system's operation and development 识别系统运行和开发的约束条件
- Requirements engineering is a particularly critical stage of the software process as errors
- Stage inevitably lead to later problems in the system design and implementation
- The software specification process aims to produce an agreed requirements 旨在生成一份商定的需求翁当，制定一个满足利益相关者的系统
- Requirements are usually presented at 2 levels of detail
  - End-users and customer need high-level statement of the requirement
  - system developers need a more detailed system specification
- The four main activities in the requirements engineering process:
  - Feasibility study 可行性研究
  - Requirements elicitation and analysis 需求获取和分析
  - Requirements specification
  - Requirements validation 需求验证

Software Design and Implementation

- The design and implementation stage of the software process is to convert a system specification into an executable system
- It always involves processes of software design and programming but, if an incremental approach to development is used, may also involve refinement of the software specification.
- A software design is a description of the structure of the software to be implemented, the data models and structures used by the system, the interfaces between system components and, sometimes, the algorithms used
  软件设计是对待实现软件的结构，系统使用的数据模型和结构，系统组件之间的接口，以及有时使用的算法的描述
- Design activities
  - Architectural design, where you identify the overall structure of the system, the principal components (sometimes called sub-systems or modules), their relationships, and how they are distributed
  - Interface design, where you define the interfaces between system components. This interface specification myst be unambiguous

- Component design, where you take each system component and design how will operate
    - a simple statement of the expected functionalities
    - a list of changes to be made to reusable component
    - a detailed design model(model-driven approach)
- Database design, where you design the system data structures and how these are to be represented in a database

Software Validation

- Software validation or, more generally verification and validation(V&V) is intended to show that
    - a system both conforms to its specification
    - it meets the expectations of the system customer
- Validation techniques
    - Programming testing, where the system is executed using simulated data, is the principle validation technique
    - Validation may also involve checking process from user requirements definition to program development
- The 3 stages in the testing process are:
    - Development testing - the components making up the system are tested by the people developing the system. Each component is tested independently, without other system components
    - System testing - System components are integrated to create a complete system. This process is concerned with finding errors that result from unanticipated interactions between components and component interface problems
    - Acceptance testing - This is the final stage in the testing process before the system is accepted for operational use. The system is tested with data supplied by the system customer rather than simulated test data

Software Evolution

- Historically, there has always been a split between the process of software development and the process of software evolution (software maintenance).
- Rather than two separate processes, it is more realistic to think of software engineering as an evolutionary process

Lecture 3 Requirement Engineering

- RE Roadmap
    - Elicitation 获取

    - Analysis
    - Specification
    - Validation
- Case Study: University timetable app
    - build an app for scheduling classes, rooms, and alerts. Stakeholders: Students, lecturers, admins

Step1 Elicitation

- Goal: Gather requirements from stakeholders via various techniques. e.g. interviews and scenarios, e.g. interviews and scenarios
- Software engineers work with stakeholders to find out about:
    - the application domain
    - what services the system should provide 系统应提供的服务
    - the required performance of the system 系统所需的功能
    - hardware constraints, and so on
- Stakeholders could be
    - end users who will interact with the system
    - anyone else in an organization who will be affected by it 组织中受其影响的任何其他人
    - engineers who are developing or maintaining other related systems 正在开发或维护其他相关系统的工程师
    - business managers
    - domain experts
    - trade union representatives 工会代表
- Sources of information during the elicitation phase include documentation, system stakeholders, and specifications of similar systems
    - interviews
    - observation
    - scenario 场景
        - Users can understand and criticize a scenario of how they might interact with a software system
        - Use the information gained from the discussion based on a scenario to formulate the actual system requirements 利用基于场景讨论中获得的信息来制定实际的系统需求
        - A scenario is the descriptions of example interaction sessions. Each scenario usually covers one or a small number of possible interactions 场景是对示例交互会话的描述，每个场景通常涵盖一个或少量可能的交互
        - A scenario starts with an outline of the interaction. During the elicitation process, details are added to create a complete description of that

interaction 场景从交互概述开始，在获取过程中，添加细节以创建对该交互的完整描述

- prototypes 原型

- Finalize the elicitation
  - Refine these scenarios: Add details from interviews
  - Conduct additional interviews if missing information is identified
  - You may need to repeat the process until the requirements are complete and satisfactory

Step2 Analysis

- Goal: Organize requirements, resolve conflicts, and prioritize 组织需求，解决冲突并确定优先级
- Organize requirements: Group requirements into functional(services) and non-functional(constraints)
  - Functional / Non-Functional
    - Functional requirements: These are statements of services the system should provide. How the services should react and behave in certain condition这些是系统应提供服务的陈述，说明了在特定条件下服务应如何反应和表现（特定功能，操作）
    - Non-Functional: These are constraints on the services or functions offered by the system. Non-functional requirements often apply to the system as a whole, rather than individual system features or services. 这是对系统所提供服务或功能的约束，通常适用于整个系统，而非单个系统特性或服务（性能，安全性，可访问性）
      - often more critical than individual function requirements
      - failing to meet a non-functional requirement can mean that the whole system is unusable
      - The implementation of non-functional requirement may be intricately dispersed throughout the system
- Resolve conflicts: Identify conflicts
- Prioritize: Use MoSCoW(Must/Should/Could/Won't) to prioritize
  - Must Have: Critical requirements without which the system is unusable or fails to meet core objectives
  - Should Have: Important but not critical; can be deferred if necessary
  - Could Have: Desirable but low priority; implemented if time / resources allow
  - Won't Have: Out of scope for this release, possibly considered later

Step3 Specification

- User Requirements vs System Requirements

  - High level user requirements, which focus on what the system should do from the stakeholders' perspective 侧重于利益相关者的角度看系统应该做什么
  - These requirements are abstract, user-oriented, and serve as a bridge between stakeholder needs and system design
  - Low-level system requirements, on the other hand, dive into the technical details of how the system will implement those needs 低级别的系统需求则深入探讨系统将如何实现这些需求的技术细节
  - Low-level system requirements should be written after high-level user requirements are validated and approved
- The process of writing down the user and system requirements in a requirements document, which should be clear, unambiguous, easy to understand, complete, and consistent
- Stakeholders interpret the requirements in different ways and there are often inherent conflicts and inconsistencies in the requirements 利益相关者以不同的方式解释需求，并且需求中常常存在固有的冲突
- specify only the external behavior of the system, should not include details of the system architecture or design 只应该指定系统的外部行为，不应包括系统架构或设计的细节，使用自然语言，简单的表格，表单和直观的图表

Step 4 Validation

- Requirements validation is the process of checking that requirements correctly define the system that the customer really wants
- It is important because errors in a requirements document can lead to extensive rework costs when these problems are discovered during development or after the system is in service
- Different types of checks
  - Validity checks:利益相关者提出的功能应与系统需要执行的功能保持一致
  - Consistency checks: 文档中的需求不应该冲突
  - Completeness checks: 需求文档应包含定义系统预期的所有功能和约束的需求
  - Realism checks: 利用现有技术知识，检查需求以确保它们能够现实地实现
  - Verifiability: 为了减少客户与承包商之间的潜在争议，系统需求应始终以可验证的方式编写
- requirements validation techniques
  - Requirements reviews 需求评审
  - Prototyping 原型法
  - Test-case generation

Challenges

- Stakeholder Conflicts 利益相关者的冲突

    - Difficult stakeholders often have conflicting priorities, making consensus difficult
    - Delays in agreement can stall the process
- Ambiguities in Requirements 需求中的歧义
    - Vague or unclear terms(e.g., "efficiently","timely")lead to misinterpretation by developers or stakeholders 模糊不清的术语
    - Ambiguities can cause rework
- Incomplete or Evolving Requirements 不完整或不断变化的需求
    - Requirements may be missing or change over time as stakeholders gain clarity
    - Incomplete requirements risk functionality gaps, while evolving needs require iterative RE, challenging fixed timelines 不完整的需求存在功能缺口的风险，而不断变化的需求需要迭代化的需求工程
- Resource Constraints 资源限制
    - Limited time, budget, or personnel restrict the depth of elicitation, analysis, and validation
    - Constraints mirror real-world startups, potentially compromising quality if not managed
- Validation Difficulties 验证困难
    - Ensuring requirements are testable, realistic, and meet all scenarios is challenging, especially for non-functional requirements
    - Validation failures, can lead to critical errors if not thoroughly addressed
- Communication gaps 沟通鸿沟
    - Misalignment between technical teams and non-technical stakeholders can lead to misunderstandings
    - Poor communication, common in large-scale projects, can lead to requirements misalignment

Lecture 5-6 System modeling

- UML diagram types
    - Activity diagrams, which show the activities involved in a process or in data processing
    - Use case diagrams, which show the interactions between a system and its environment
    - Sequence diagrams, which show interactions between actors and the system and between system components
    - Class diagrams, which show the object classes in the system and the associations between these classes 显示系统中的对象类以及这些类之间的关联
    - State Machine diagrams, which show how the system reacts to internal and external events

Context model

- are used to illustrate the operational context of a system - they show what lies outside the system boundaries 用于说明系统运行上下文，它们显示系统边界之外有什么
- Social and organizational concerns may affect the decision on where to position system boundaries
- System boundaries are established to define what is inside and what is outside the system
    - They show other systems that are used or depend on the system being developed
- Context models simply show the other systems in the environment, not how the system being developed is used in that environment 只用于显示环境中的其他系统，而不显示正在开发的系统如何在该环境中使用
- UML activity diagrams may be used to define business process models

Interaction model

- to user interaction: identify user requirements
- to system interaction: highlights the communication problems that may arise
- to component interaction: helps us understand if a proposed system structure is likely to deliver the required system performance and dependability
- Use case diagrams and sequence diagrams may be used for interaction modeling

Structural models

- Structural models of software display the organization of a system in terms of the components that make up that system and their relationships 根据系统组件和关系来显示系统的组织
- Structural models maybe static models, which show the structure of the system design, or dynamic models, which show the organization of the system when it is executing

Behavioral models

- Behavioral models are models of the dynamic behavior of a system as it is executing. They show what happens or what is supported to happen when a system responds to a stimulus from its environment 显示当系统响应来自其环境的刺激时，会发生什么或应该发生什么
- You can think of these stimuli as being of 2 types:
    - data -Some data arrives that has to be processed by the system
    - Events -Some event happens that triggers system processing. Events may have associated data, although this is not always the case
- data-driven modeling

    - Many business systems are data-processing systems that are primarily driven by data. They are controlled by the data input to the system, with relatively little external event processing
    - Data driven models show the sequence of actions involved in processing input data and generating an associated output
    - They are particularly useful during the analysis of requirements as they can be used to show end-to-end processing in a system
    - Activity diagram and sequence diagram are used in data-driven modeling
- Event-driven modeling
    - Real-time systems are often event-driven, with minimal data processing. For example, a landline phone switching system responds to events such as 'receiver off hook' by generating a dial tone
    - Event-driven modeling shows how a system responds to external and internal events
    - It is based on the assumption that a system has a finite number of states and that events may cause a transition from one state to another

Lecture 8 Design Concept 设计概念

Design in the software engineering(SE) context

- Software Design includes the set of **principles / concepts / practices** that lead to the development of a high-quality system or product
- The goal of design is to produce a model or representation that exhibits
    - Firmness(no bugs)
    - Commodity(useful or valuable) 实用性
    - Delight(pleasurable experience) 愉悦性
- from requirement to design
    - component-level design 构建级设计: transforms structural elements of the software architecture into a procedural description of software components 将软件体系结构的结构元素转换为软件构件的过程性描述
    - Interface design: describes how the software communicates with systems that interoperate with it and with humans who uses it 描述软件如何与其交互操作以及使用的人类通信
    - Architectural design: defines the relationship between major structural elements of the software, the architectural styles and patterns that can be used to achieve the requirements defined for the system, and the constraints that affect the way in which architecture can be implemented
    - Data design / class design

Quality control in the design process

- 3 goals
    - The design should **implement all of the explicit requirements** contained in the requirements model, and it must accommodate all of the implicit requirements desired by stakeholders 满足需求模型中的明确需求，同时也要满足所有利益相关者期望的所有银行需求
    - The design should be **readable, understandable guide** for those who generate code and for those who test and subsequently support the software 可读 移动
    - The design should provide **a complete picture of the software**, addressing the data, functional, and behavioral domains from an implementation perspective
- Importance of software design - Quality
    - Defect Prevention 预防缺陷
    - Cost Efficiency
    - Improved User Satisfaction
    - AND: Enhanced Reliability, Compliance and Standards Adherence, Documentation and Traceability and so on
- 5 quality attributes(FURPS)
    - Functionality 通过评估程序的功能集和能力，说交付功能的通用性以及整个系统的安全性来评估
    - Usability 通过考虑人为因素，整体美观性，一致性和文档来评估
    - Reliability 通过测量故障的频率和严重性，输出结果的准确性，平均无故障时间，从故障中恢复的能力以及程序的可预测性来评估
    - Performance 使用处理速度 响应时间 资源消耗 吞吐量 和效率来衡量
    - Supportability 结合了可扩展性适应性和可服务性
- 8 technical criteria for good design: 架构 / 组件 / 测试 / 模块化 / 数据结构
    - A design should exhibit an architecture that
        - has been created using recognizable architectural styles or patterns 使用了可是别的架构风格或者模式
        - is composed of components that exhibit good design characteristics 由展现出良好设计特征的组件构成
        - can be implemented in an evolutionary fashion, thereby facilitating implementation and testing 能够以演进的方式实现，从而以便于实现和测试
    - A design should be modular; that is the software should be logically partitioned into elements or subsystems
    - A design should contain distinct representations of data, architecture, interfaces, and components
    - A design should lead to data structures that are appropriate for the classes to be implemented and are drawn from recognizable data patterns

- A design should to components that exhibit independent functional characteristics 设计产生出独立功能特性的组件
- A design should lead to interfaces that reduce the complexity of connections between components and with the external environment
- A design should be derived using a repeatable method that is driven by information obtained during software requirements analysis
- A design should be represented using a notation that effectively communicates its meaning

Design Concepts

- General design concepts
  - Abstraction
  - Modularity
  - Functional Independence
    - Coupling
    - Cohesion
  - Object-oriented design

Abstraction

- Abstraction is a design concept that focuses on simplifying complex systems by highlighting essential features while hiding unnecessary details. 突出基本特征隐藏不必要的细节
- The higher the level, the less the detail.
- At the highest level of abstraction, a concept / approach / solution is stated in broad terms using the language of the problem environment.
- At lower levels of abstraction, a more detailed description of the concept/approach/solution is provided
- From high to low: A car $\rightarrow$ a Xiaomi Su7 $\rightarrow$ a Xiaomi Su7 Ultra
- Data abstraction
  A data abstraction is named collection of data that describes a data object 数据抽象是描述数据对象的命名数据集合
- Procedural Abstraction
  A procedural abstraction refers to a sequence of instructions that have a specific and limited function. The name of a procedural abstraction implies these functions, but specific details are omitted 一系列具有特定且有限的指令序列 省略具体细节

Modularity

- Modular design helps us to better organize complex system. 帮助我们更好的组织复杂系统

- It basically clusters similar or relative functions together, sets up boundaries and provides interfaces for communication
- mobile phone as an complex system that is modularized
- modularity increases manufacture efficiency and save time
- Modularity allows the possibility of the development of system parts to be carried out independently of each other, therefore reducing development time. 允许各部分独立开发从而减少开发时间
- However, too many modules in a system increase the complexity of modules integration

Functional independence

指软件模块或组件多大程度上独立于系统内的其他模块或组件运行
Low coupling / High Cohesion

- Coupling
  - the degree of interdependence between different modules or components 不同模块或组件相互依赖的程度
  - 低耦合意味着模块在很大程度上相互独立，并且通常通过定义良好的接口进行交互

```
class Author {
    String name;
    String skypeID;
    public String getSkypeID() {
        return skypeID;
    }
}
```

```
class Editor{
    public void clearEditingDoubts(Author author) {
        setUpCall(author.skypeID);
        converse(author);
    }
    void setUpCall(String skypeID) { /* */}
    void converse(Author author) {/* */}
}
```

Tight coupling; nonpublic variable skypeID is referred to outside its class Author.

    - if a programmer changes the name of the variable `skypeID` in class `Author` to `skypeName`
      solution:

```
class Author {
    String name;
    String skypeName;
    public String getSkypeID() {
        return skypeName;
    }
}
class Editor{
    public void clearEditingDoubts(Author author) {
        setUpCall(author.getSkypeID());
        converse(author);
    }
    void setUpCall(String skypeID) { /* */}
    void converse(Author author) {/* */}
}
```

Change in instance variable name won't affect classes that access this method

Loose coupling; public method getSkypeID() accesses Author's skypeName.

- Cohesion
    - how closely related and focused the responsibilities of a single module or component are 单个模块或组件职责的紧密相关和专注程度 高内聚意味着模块内的元素协同工作以字行单一任务或功能
    - Method Cohesion: A method that implements a clearly defined function, and all statements in the method contribute to implementing this function 一个方法实现一个明确定义的功能
    - Class Cohesion:A high class cohesion is the class that implements a single concept or abstraction with all elements contributing toward supporting this concept. Highly cohesive classes are usually more easily understood and more stable. 类实现单一概念或抽象 所有元素都共同支持这个概念，高内聚的类通常更容易理解和更稳定

Object-Oriented Design

- Object-Oriented Design is a software design approach that organizes a system as a collection of interacting objects, each representing an instance of a class within a specific domain. 软件设计方法，他将系统组织为一系列相互作用的对象 每个对象代表特定领域中的某个累的实例
- These objects encapsulate both data (attributes) and behavior (methods), promoting modularity, reusability, and scalability.
- Principles
    - Encapsulation 封装
    - Inheritance 继承
    - Polymorphism 多态

Elements in Design Model

Data design elements / Architectural Design elements / Interface Design elements / Component-level design elements / Deployment-level elements

Data Design

Like other software engineering activities, data design (sometimes referred to as data architecting) creates a model of data and/or information that is represented at a high level of abstraction (the customer/user’s view of data) 数据设计创建数据 和/或 信息的模型 在较高的抽象级别表示

Architectural design

- The architectural design for software is the equivalent to the floor plan of a house. The floor plan depicts the overall layout of the rooms; their size, shape, and relationship to

one another; and the doors and windows that allow movement into and out of the rooms. 相当于房屋的平面图 平面图描述了房间的整体布局 尺寸 形状和相互关系

- The floor plan gives us an overall view of the house. Architectural design elements give us an overall view of the software.
- Three sources:
    - Information about the application domain for the software to be built; 关于要构建的软件的应用领域的信息
    - Specific requirements model elements such as use cases or analysis classes, their relationships and collaborations for the problem at hand; 需求模型元素 用例/分析类
    - The availability of architectural styles and pattern 可用体系结构风格和模式的可用性

Interface design

- The interface design for software is similar to a set of detailed drawings (and specifications) for the doors, windows, and external utilities of a house. 类似于房屋门窗和外部设施的一套详细图纸
- The interface design elements for software depict information flows into and out of a system and how it is communicated among the components defined as part of the architecture.
- Three important elements:
    - The user interface (UI)
    - External interfaces to other systems, devices, networks, or other producers or consumers of information 与其他系统设备网络或其他信息生产者或消费者的外部接口
    - Internal interfaces between various design components 各种设计组件之间的内部接口

Component level

- The component-level design for software is the equivalent to a set of detailed drawings (and specifications) for each room in a house. 相当于房屋中每个房间的一套详细图纸
- The component-level design for software fully describes the internal detail of each software component (e.g., an object).
- In software engineering, a component is represented in UML diagrammatic form.
- The design details of a component can be modeled at many different levels of abstraction e.g., Flowchart or box diagram -> detailed procedural flow for a component.

Deployment level

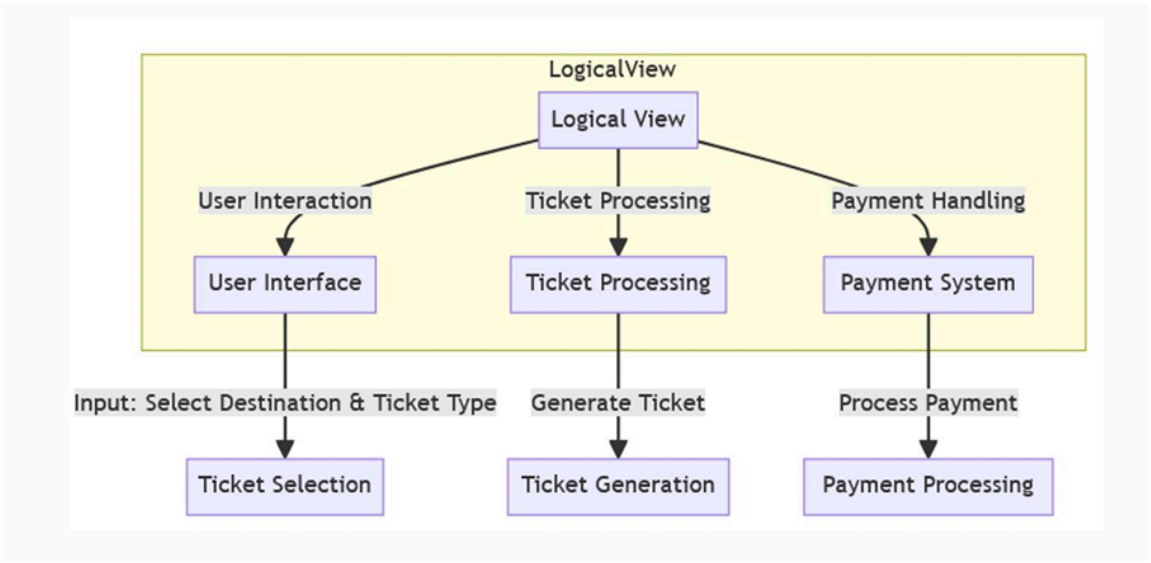
indicate how software functionality and sub-systems will be allocated within the physical computing environment that will support the software 部署级设计要素指示软件功能和子系统将如何在支持的软件的物理计算环境内进行分配

Lecture 8 Software design

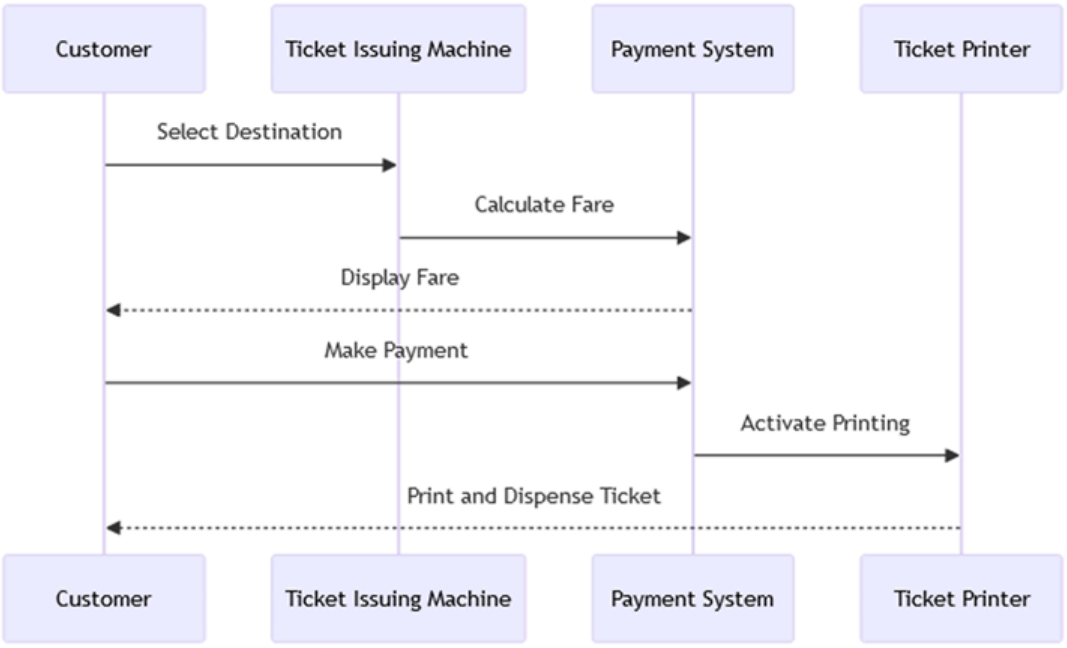
Architecture design

- 关注于理解系统应如何组织并设计该系统的整体结构
- 软件设计过程的第一阶段，是设计和需求工程（RE）的纽带
- 识别系统中的主要结构组件及其之间的关系
- 输出是对软件体系结构的描述
- Two levels of abstraction
    - Architecture in the small is concerned with the architecture of individual programs. At this level, we are concerned with the way that an individual program is decomposed into components. Here is mostly concerned with program architectures. 小规模体系结构关注单个程序的体系结构
    - Architecture in the large is concerned with the architecture of complex enterprise systems that include other systems, programs, and program components. These enterprise systems are distributed over different computers, which may be owned and managed by different companies/organization. 大规模关注复杂企业系统的体系结构 其他系统程序和程序组件 分布在不同的计算机上
- Importance
    - performance robustness distributability and maintainability
    - Individual component 单个组件-实现功能需求
    - System architecture 系统体系结构
    - Stakeholder communication
    - system analysis
    - large-scale reuse 大规模复用
- Different view

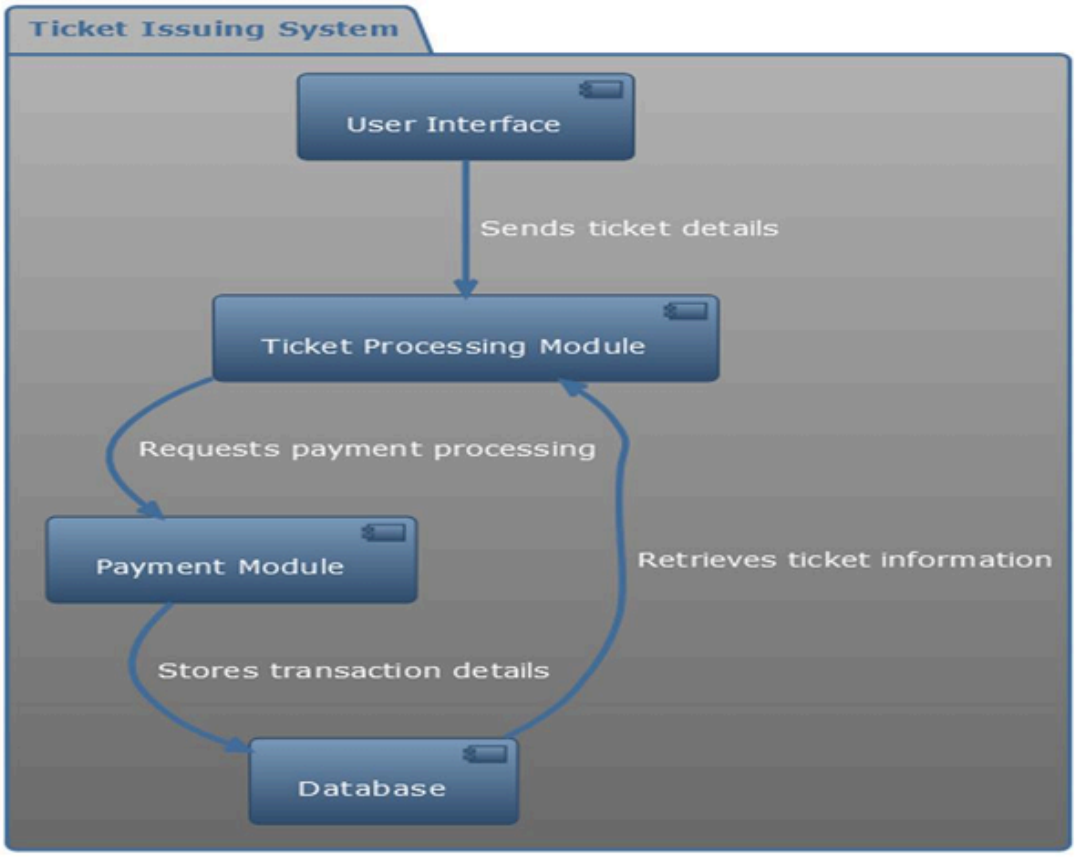
- logical view 显示系统中作为对象/对象类的关键抽象



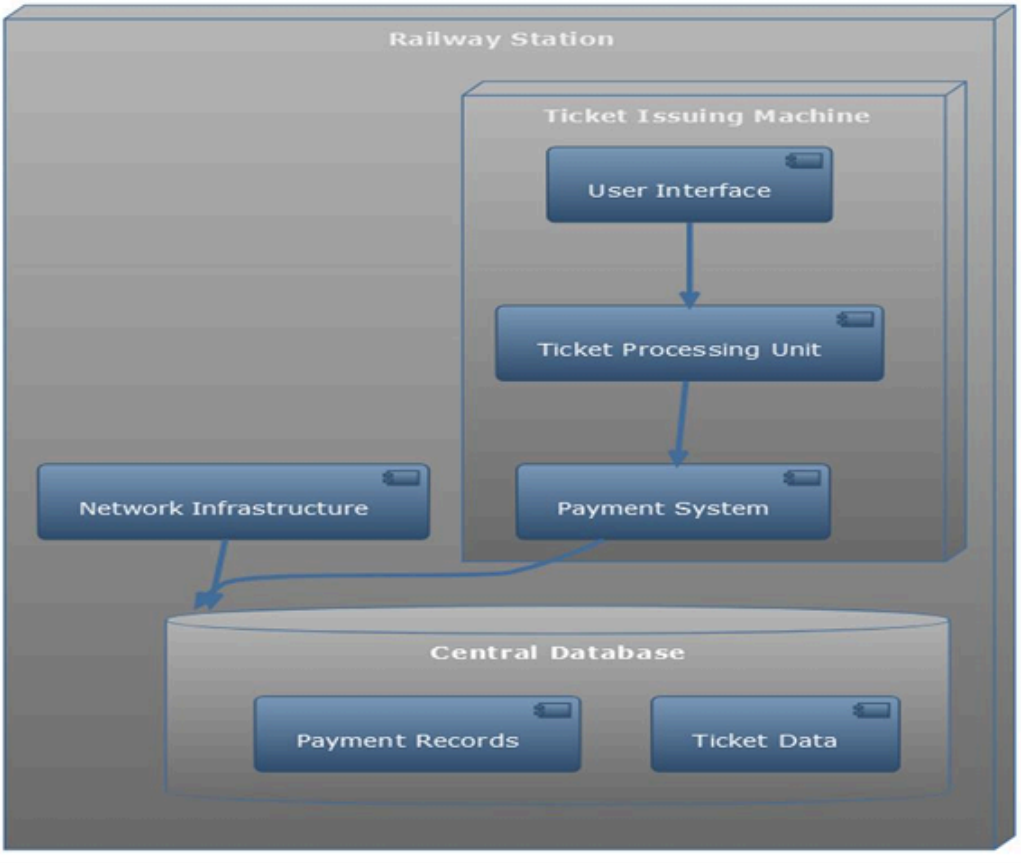
- process view 显示在运行时，系统如何有交互的进程组成



- development view 现实软件如何为开发而分解



- physical view 显示系统硬件以及软件组件如何分布在处理器上



Architecture Patterns

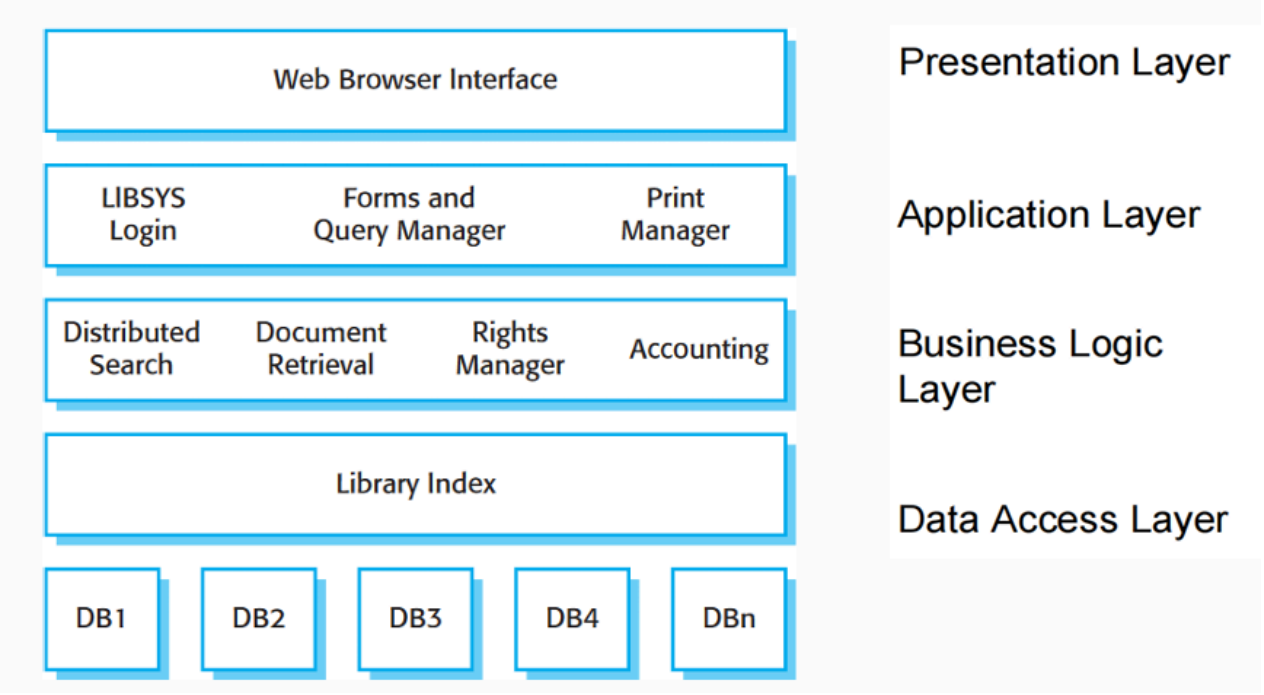
Patterns are means of representing, sharing and reusing knowledge 表示共享和复用知识的手段
一个架构模式是对良好设计实践的程式化描述，已在不同环境中经过尝试和测试
模式应包含关于其何时有用何时无用的信息，可以用表格和图形描述来表示

The MVC pattern

- Model-view-controller(MVC pattern)
- This pattern is the basis of interaction management in many web-based system
- Model: central component of the pattern that directly manages the data, logic and rules of the application 模式的核心组件 直接管理应用程序的数据 逻辑和规则
- View: can be any output representation of information, such as a chart or a diagram. 可以是信息的任何输出表示，例如图表或图形
- Controller: accepts input and converts it to commands for the model or view, enables the interconnection between the views and the model 接受输入并将其转换为对模型或视图的命令，实现视图和模型之间的互联
- Advantages: Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them 允许数据独立于其表示形式进行更改，反之亦然，支持以不同方式呈现相同数据，且在一处表示中所做的更改会体现在所有的表示中
- Disadvantages: Can involve additional code and code complexity when the data model and interactions are simple

The Layered Pattern

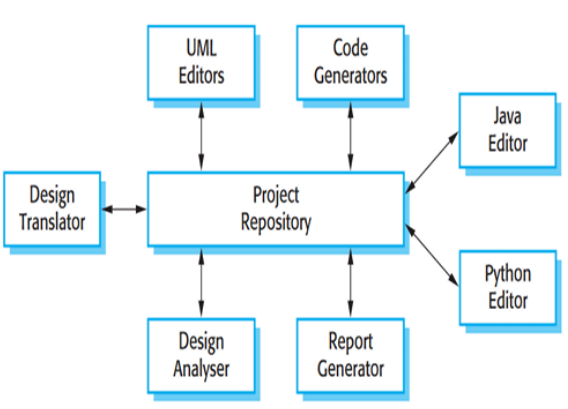
- 系统功能被组织成独立的层，每层只依赖于紧临泣下的一层提供的设施和服务
- 这种分层方法支持系统的增量开发，但开发某一层的时候，该层提供的某些服务可能会对用户可用
- 在高性能应用中表现不佳，应为通过多层来完成业务请求效率不高，对于预算和实践非常紧张的情况下，这是一个不错的选择



- Advantages: Allow replacement of entire layers so long as the interface is maintained. Redundant facilities can be provided in each layer to increase the dependability of the system 只要接口保持不变就可以替换整个层，可以在每一层提供冗余设施以增加系统的可靠性
- Disadvantage: In practice, a clean separation between layers is often difficult 在实践中，通常很难在各层之间提供清晰的分离，高层可能不得不直接与低层交互，而不是紧邻其下的一层

The Repository Pattern

Name	Repository
Description	All data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.
Example	Figure 6.9 is an example of an IDE where the components use a repository of system design information. Each software tool generates information which is then available for use by other tools.
When used	You should use this pattern when you have a system in which large volumes of information are generated that has to be stored for a long time. You may also use it in data-driven systems where the inclusion of data in the repository triggers an action or tool.
Advantages	Components can be independent—they do not need to know of the existence of other components. Changes made by one component can be propagated to all components. All data can be managed consistently (e.g., backups done at the same time) as it is all in one place.
Disadvantages	The repository is a single point of failure so problems in the repository affect the whole system. May be inefficiencies in organizing all communication through the repository. Distributing the repository across several computers may be difficult.

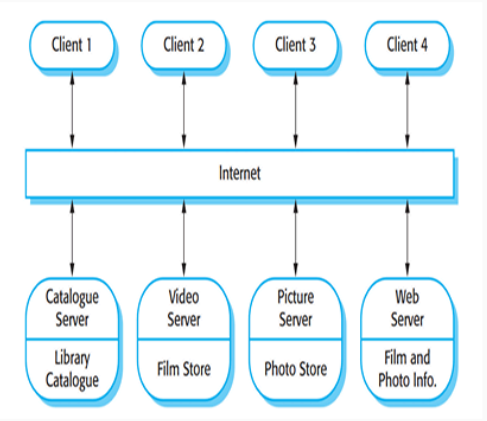


- 系统中的所有数据都在一个所有组件都可访问的中心仓库中管理，组件不直接交互，只通过惭愧进行交互
- 当系统中生成大量信息需要长期存储时，应使用此模式，也可以用于数据驱动的系统，其中数据放入仓库会触发某个操作或工具
- 优点：组件可以独立，它们不需要知道其他组件的存在，一个组件所做的更改可以传播到所有组件，所有数据可以一致性的管理，因为它都位于一个地方

- 缺点：仓库单点故障，由此仓库的问题会影响整个系统，将仓库分布在多台计算机上可能会很困难

The Client-server pattern

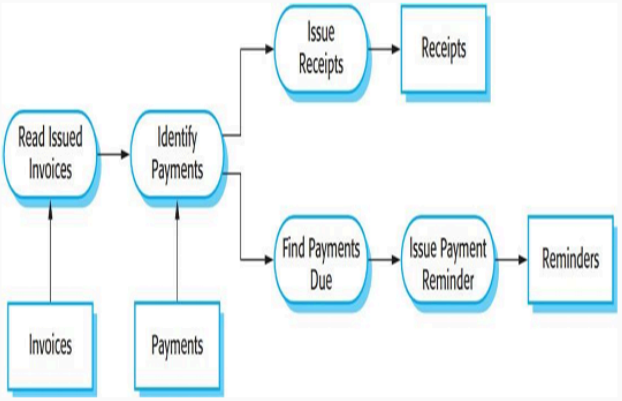
Name	Client–server
Description	In a client–server architecture, the functionality of the system is organized into services, with each service delivered from a separate server. Clients are users of these services and access servers to make use of them.
Example	Figure 6.11 is an example of a film and video/DVD library organized as a client–server system.
When used	Used when data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also be used when the load on a system is variable.
Advantages	The principal advantage of this model is that servers can be distributed across a network. General functionality (e.g., a printing service) can be available to all clients and does not need to be implemented by all services.
Disadvantages	Each service is a single point of failure so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system. May be management problems if servers are owned by different organizations.



- 当需要从多个位置访问共享数据库中的数据时使用，由于服务器可以复制，也可用于系统负载可变的情况
- 优点：服务器可以分布在网络上，通用功能可以对所有用户端可用，无需在所有服务中实现
- 缺点：每个服务都有单点故障的可能，因此容易收到拒绝服务攻击或服务器故障的影响，性能不可预测，因为它依赖于网络和系统，如果服务器由不同组织拥有，可能存在管理问题

The Pipe and Filter Pattern

Name	Pipe and filter
Description	The processing of the data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation. The data flows (as in a pipe) from one component to another for processing.
Example	Figure 6.13 is an example of a pipe and filter system used for processing invoices.
When used	Commonly used in data processing applications (both batch- and transaction-based) where inputs are processed in separate stages to generate related outputs.
Advantages	Easy to understand and supports transformation reuse. Workflow style matches the structure of many business processes. Evolution by adding transformations is straightforward. Can be implemented as either a sequential or concurrent system.
Disadvantages	The format for data transfer has to be agreed upon between communicating transformations. Each transformation must parse its input and unparse its output to the agreed form. This increases system overhead and may mean that it is impossible to reuse functional transformations that use incompatible data structures.



- 常用于数据处理应用（批处理和事务处理），其中输入在独立的阶段进行处理以生成相关的输出
- 优点：易于理解并支持转换的复用，工作流风格符合许多业务流程的结构，通过添加转换来进行演进是直接的，可以为顺序系统或并发系统

- 缺点：必须在通行的转换之间商定数据传输格式，每个转换必须解析其输入并将其输出解构层商定的形式，这增加了系统开销，并可能意味着无法复用或使用不兼容数据结构的函数转换

Component Level Design

- Component-level design defines the data structures, algorithms, interface characteristics, and communication mechanisms allocated to each software component 分配给每个软件构件的数据结构算法接口特征和通信机制
- 3 views of a component
    - An object-oriented view
        - Centers on objects as the fundamental building blocks of software. An object encapsulates both data and behavior
    - A traditional view
        - Emphasizes a top-down approach to software design, focusing on functions or procedures and the flow of data between them 侧重于函数或过程以及它们之间的数据流
        - 在软件发展史的早期阶段更为普遍，适用于线性和不太复杂的应用程序
    - A process view
        - 关注软件构件的运行时行为，它着眼于组件在执行期间如何运行，特别是在进程和线程方面
        - includes process management, inter-process communication, concurrency 并发, synchronization 同步, and resource management
    - 面向对象侧重于将系统建模成一组交互的对象，每个对象封装了与在线购物相关的数据和行为，传统视图将系统结构化为一系列特定的任务（如添加产品，下订单和处理支付）的过程或函数，进程视图从运行时的行为角度审视系统，特别是它如何处理订单和支付的流程，以及管理并发和系统资源

Interface Design

- The golden rules
    - place the user in control
        - define interaction mode in a way that does not force a user into unnecessary or undesired actions 定义交互模式时，不应强迫用户进行不必要的操作
        - Provide for flexible interaction. Because different users have different interaction preferences, choices should be provided 提供灵活的交互，应为不同的用户有不同的交互偏好
        - Allow user interaction to be interruptible and undoable. Interrupt sequence of steps without losing work that had been done 允许用户交互可以中断和撤销

        - Design for direct interaction with object that appear on the screen 设计用于直接与屏幕上的对象进行交互
        - Show the visibility of system status
    - reduce the user's memory
        - Establish meaningful defaults 建立有意义的默认值
        - The visual layout of the surface should be based on a real-world metaphor 页面的视觉设计需要基于现实世界的隐喻
    - make the interface consistent 一致的用户界面
        - 所有视觉信息都根据设计规则进行组织，这些规则在所有屏幕显示中保持一致
- Interface Design Issues
    - Response time: 2 important characteristics: length and variability(与平均响应时间的偏差)
    - Help facilities: 帮助文档 为所有系统功能提供保住
    - Error handling: 用用户能够理解的语言描述问题，提供恢复错误的建设性建议，指出错误的任何负面效果，以便用户可以检查以确保这些后果没有发生，或者如果发生了可以纠正
    - Application accessibility: the disabled 色盲 听力障碍 视力障碍 老年人...
    - Internationalization: The Unicode standard
- User Interface Design Evaluation
  Once you create an operation user interface prototype 原型， it must evaluated to determine whether it meets the needs of the user
    - Effectiveness
        - Measures: Quality of solution, error rates
    - Efficiency
        - Measures: task completion time, learning time, number of clicks
    - Satisfaction
        - Measures: attitude rating scales

Lecture 9 Software Testing I

- Testing is a broader process of software Verification and Validation(V & V)
    - Verification checks that software being developed meets its specification 检查正在开发的软件是否符合其规范说明(static check)
    - Validation checks delivers the functionality expected by the people paying for the software（Dynamically check: testing)
- 2 goals
    - demonstrate to the developer and the customer that the software functionally works as expected 向开发者和客户证明软件在功能上按预期按预期工作

- discover situations in which the behavior of the software is incorrect, undesirable or does not confirm to its specification, for finding defects 发现软件行为的不正确，不理想或不符合规范说明的情况，以找出缺陷

- 3 Stages of testing
  - Development testing 在开发过程中对系统进行测试以发现错误和缺陷
    - From small to large, 3 stages of dev testing
      - Unit testing
        - is the process of testing individual unit in isolation 隔离地测试单个单元的过程
        - Units can be defined at different levels(e.g. a function / method)
        - Complete test coverage of an object class involves
          - Setting and interrogating all objects attributes
          - Testing all operations associated with an object
          - Exercising the object in all possible states
        - Automated Unit Testing
          - whenever possible, unit testing should be automated so that tests are run and checked without manual intervention. J·Unit
          - A setup part 在那里你使用测试用例初始化系统
          - A call part 在那里你调用要测试的对象或方法
          - An assertion part 在那里你将调用的结果与预期结果进行比较
        - Choosing A Unit Testing case
          - Effective tests should verify (1) that the methods/functions/system test behave correctly under normal, expected usage, and (2) should also reveal any defects when abnormal or problematic inputs are used. 在正常预期的使用下行为正确，并且在使用异常或有问题的输入时揭示缺陷
          - Partition Strategy
            - 识别出应被视为等价分区的可能输入组，即这些组中的成员（具体值）应具有共同特征，并应由相同的方式处理
          - Guideline Strategy
            - 这些指南反映了先前经验中程序员在开发构件时常犯的错误类型
      - Component testing

- The functionality of the component is based on these units interacting through interface(通过接口交互的单元)
- Testing interface in the component
  - Unit A,B, and C have been integrated to create a larger component or subsystem. The test case are not applied to the individual units but rather to the interface of the composite component created by combining these components
- Types of Interface errors
  - Interface misuse
  - Interface misunderstanding
  - Timing errors

- System testing
  - checks that components are compatible, interact correctly and transfer the right data at the right time across their interfaces 系统测试构件是否兼容，是否正确交互，并是否在正确的时间通过其接口传输正确的数据
  - Use case-based approach
    - 由于系统测试侧重于交互，基于用力的测试是一种有效的系统测试方法
    - 每个用例由系统中的几个构件或对象实现，为了更好的识别不同用例中设计的交互，我们可以用序列图来识别将要进行的操作，并帮助设计测试
  - System Testing Guidelines
    - Test every major user-visible function 测试每个主要的用户可见的功能
    - Test combinations of functions 测试用户通常一起使用的功能组合
    - Test end-to-end workflows 测试端到端的工作流
    - Test under realistic load and environment conditions:
      - Many users logged in at once
      - slow networks

- Release testing 在发布之前，有一个独立的测试团队对系统的完整版本进行测试
  - Release v.s. system testing
    - 一个完全未参与系统开发的独立团队应负责发布测试
    - 开发团队的系统测试更侧重于发现系统中的技术性错误，发布测试的目标是检查系统是否满足其需求，并且外部使用是否足够好
    - 系统测试重点：整个系统作为一个整体是否功能正常

        - 发布测试重点：系统对于真实用户和真实世界的使用是否可以接受
    - Requirements Based Approach
        - Requirements-based approach involves examining each requirement and developing a test or tests for it 基于需求的方法设计检查每个需求，并为它开发一个或多个测试
    - Scenario Approach
        - 提出典型的使用场景，并使用这些场景为系统开发测试用例
        - 场景是描述系统可能被使用的一种方式的故事
        - 场景应该是现实的，并且真实的系统用户应该能够与之联系起来
    - Testing Performance Approach
        - 发布测试的一部分可能设计测试系统的涌现属性，例如性能和可靠性
        - 性能测试通常设计规划一系列测试，其中负载稳步增加，知道系统性能变得不可接受
        - Use operational Profile
        - Stress Testing
- User testing
    - Alpha testing 软件用户与开发团队合作，在开发者的场所测试软件
    - Beta testing 软件的发布版本被提供给用户，让他们进行试验，并向系统开发者提出他们发现的问题
    - Acceptance testing 客户测试一个系统，以决定是否准备从系统开发者那里接受它并部署在客户环境中，主要用于定制系统
        - Step1 define acceptance criteria 定义验收标准，发生在early in the process
        - Step2 Plan acceptance testing 决定验收测试的资源，时间和预算
        - Step3 Derive acceptance tests 一旦建立了验收标准，就必须设计测试，验收测试旨在测试系统的功能性和非功能性特征
        - Step4 Run acceptance tests 商定的验收测试在系统上执行，理想情况下，在系统将要使用的实际环境中进行
        - Step5 Negotiate test results 协商测试结果 很可能并非所有定义的验收测试都会通过
        - Step6 Reject/accept system

Lecture 10 JUnit Testing

Unit Testing and JUnit

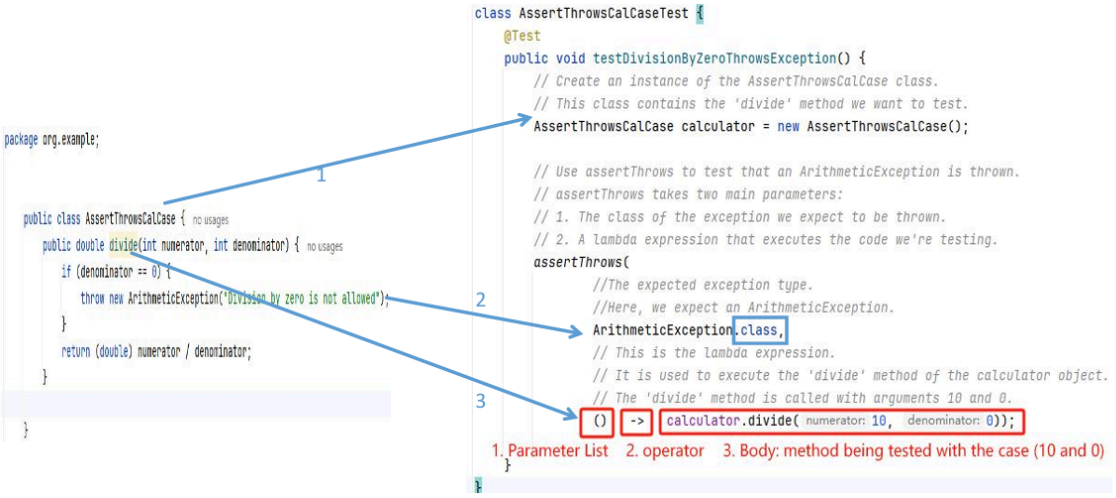
- Testing of an individual software unit 对单个软件单元的测试

    - usually an object class 通常是一个对象类
- Focus on the functions of the unit
    - functionality, correctness, accuracy
- JUnit test structure
    - Non-standard(same - file) test placement is not recommended
    - 标准的布局应该是一个src包下被测试代码放在main包里，junit测试部分放在统一src包的test包里面，简化结构中可以返回在同一个src包下的两个文件中
- JUnit Test Verdicts 测试判定
    - A verdicts is the result of executing a single test case
    - Pass 测试用例执行完成 被测试函数按预期执行
    - Fail 测试用例执行完成，被测试函数未按预期执行
    - Error 测试用例执行未完成，由于意外事件异常或者测试用例设置不当
- Junit Best Practices
    - Tests need failure atomically (ability to know exactly what failed)
        - each test should have a clear, long descriptive name 每个测试都应该有一个清晰长描述性的名称
        - Assertions should always have clear messages to know that failed 断言需要始终有清晰的信息，以了解什么是白了
        - Write many small tests, not one big test
            - each case should have roughly just 1 assertion at its end
        - Choose a descriptive assert method, not always `assertTrue`
        - Choose representative test cases from equivalent input classes
        - Avoid complex logic in test methods if possible

Assertion Methods

- `assertTrue/assertFalse`
    - Assert a Boolean condition is true or false
        - `assertTrue(condition)`
        - `assertFalse(condition)`
    - Optionally, include a failure message
        - `assertTrue(condition, message)`
        - `assertFalse(condition, message)`
- `assertSame/assertNotSame`
    - Assert 2 object references are identical 比较两个对象是否是完全相同的一个
      `assertSame(expected,actual)`
      `assertNotSame(expected,actual)`
    - with a failure message
      `assertSame(expected,actual,(optional)message)`

- `assertEquals/assertNotEquals`
  - Assert two objects are equal to each regarding value/content 断言两个对象在值/内容上是否相等
  - It doesn't matter if expected and actual are the same object or different objects; as long as their content is equal, the test will pass
    `assertEquals(expected, actual, (optional)message)`
- `assertArrayEquals`
  - Assert two arrays are equal
    `assertArrayEquals(expected,actual,(optional)message)`
  - arrays must have same length
  - check for each valid i, comparing values at index 0,1,2,...
- `assertThrows`
  - Used to test that a specific type of exception is thrown during the execution of a block of code 用于测试在执行一段代码期间是否抛出了特定类型的异常
    `assertThrows(expectedExpectionClass,executable)`
    前面的是期望的异常类型，executable 是一个lambda表达式或者方法引用



JUnit Test Cycle and Annotation

- Life cycle
  - Setup: this phase puts the test infrastructure in place. Junit provides class level setup(`@BeforeAll`) and method level setup (`@BeforeEach`). Generally, heavy objects like database connections are created in class level setup while lightweight objects like test objects are reset in the method level setup 此阶段建立测试基础设施。类级别的连接 `@BeforeAll` ， 方法级别的连接 `@BeforeEach` ，重量级对象比如数据库连接在类级别中完成 轻量级对象在方法级别设置中创建
  - Test Execution: In this phase, the test execution and assertion happen. The execution result will signify a success or failure. 在此阶段 发生测试执行和断言，执行结果将表示成功或失败
  - Cleanup: This phase is used to cleanup the test infrastructure setup in the first phase. Just like setup, teardown also happen at class level (`@AfterAll`) and

method level (`@AfterEach`). 在此阶段用于清理第一阶段设置的测试基础设施，就像setup一样，拆卸也反正在类级别和方法级别
    - `@BeforeAll` & `AfterAll` 必须是静态，他为整个测试类运行一次，而不是单个测试运行，静态方法不依赖于对象状态，JUnit强制静态，以避免类级别行为和每个测试行为之间混淆
    - `@BeforeEach` & `AfterEach` 必须是非静态，他们在每个测试方法的前后运行，Junit 为每个测试创建一个新的测试对象，因为这些方法必须在该对象上工作，非静态方法可以访问和重置当前测试对象的字段，而不是全局变量，这对于准备一致的测试条件至关重要
- `@DisplayName // to display meaningful name appear in the test report`
- `@Timeout // Useful for simple performance test`
    - by using `@Timeout(1)`
- `@RepeatedTest // used to mark a test method that should repeat a specified number of times with aconfigurable display name`
    - If we used `@RepeatedTest` , there is no need to use `@Test`

Lecture 11 Project Management

- Criteria for Project Management
    - Deliver the software to the customer at the agreed time 在约定时间将软件交付给客户
    - Keep overall costs within budget 将总成本控制在预算内
    - Deliver software that meets the customer's expectations 交付满足客户期望的软件
    - Maintain a happy and well-functioning development team
- Challenges in Project Management
    - The product is intangible
    - Large software projects are often 'one-off' projects
        - 每个大型系统都反映了独特的业务规则，利益相关者和集成约束
    - Software processes are variable and organization specific
        - 文化法规工具和审批流程的变化是的过程行为不可预测

Risk Management

Risk management involves anticipating risks that might affect the project schedule or the quality of the software being developed, and then taking action to avoid these risks 风险管理设计预测可能影响项目进度或正在开发的软件质量的风险，然后才去行动避免这些风险

- Three major types of risks
    - Project risks 影响项目进度或资源的风险
    - Product risks 影响正在开发的软件质量或性能的风险

    - Business risks 影响开发或采购软件的组织风险（例如 竞争对手的新产品）
- Risk Management Process
    - Risk identification
        - concerned with identifying the risks that could pose a major threat to the software engineering process(project), the software being developed(product), and the development organization(business) 软件工程过程 正在开发的软件 开发组织 构成重大威胁的风险
        - The identification is usually team process, or sometimes the project manager's call
    - Risk analysis
        - Probability
            - Very low($< 10\%$)
            - Low($10 - 25\%$)
            - Moderate($25 - 50\%$)
            - High($50 - 75\%$) or very high($> 75\%$)
        - Seriousness
            - Catastrophic(threaten the survival of the project) 灾难性的
            - Serious(would cause major delays)
            - Tolerable(delays are within allowed contingency)
            - Insignificant
    - Risk planning
        - 考虑每个已识别的关键风险，并制定管理这些风险的策略
        - 思考如果风险中识别的问题发生，可以采取哪些行动来最小化对项目的干扰
        - 思考在监控项目时可能需要收集那些信息，以便预测问题
        - Three strategies
            - Avoidance 改变风险使风险不会发生
            - Minimization 才去措施以减少风险发生时的负面影响
            - Contingency Plan 准备一个备用方案以防万一发生最坏情况
    - Risk monitoring
        - 检查你对产品项目和业务风险的假设是否变化的过程
        - 定期评估每个已识别的风险，以决定改风险的概率和严重性
        - Indicators
            - Technology
            - People
            - Organizational
            - Tools
            - Requirements

- Estimation

Managing People

- 4 Criteria
  - Consistency 一致性
  - Respect
  - Inclusion
  - Honesty

Teamwork in Software Engineering

- Large team should be divided into smaller working groups
- Creating the right group composition is a key managerial responsibility
- members need to agree on:
  - how tasks are shared
  - how decisions are made
  - how communication happens
- Influential Factors
  - Project and organizational context influences team effectiveness 项目和组织环境影响团队效能
  - 3 generic factors
    - The people in the group
    - The group organization
    - Technical and managerial communications

Continuous Improvement

- Characteristics
  - Incremental Progress 渐进式进展 专注于随着时间的推移积累的小而持续的改变
  - Feedback-driven 反馈驱动 依赖于来自测试 用户和监控的反馈循环
  - Goal-oriented 目标导向 针对质量 性能 或团队生产力的可衡量改进
  - Collaborative 协作性
  - Cyclic 周期性
- Strategies for Continuous Improvement
  - Agile
    - Agile focuses on small, frequent improvements, teamwork, and quick responses to change
    - Scrum: 冲刺（sprints）的短开发周期，然后是反思和改进的会议（sprint retrospective）

- CI/CD
  - Continuous Integration and Continuous Delivery(CI/CD)
  - is an automated software engineering pipeline that connects code integration, testing and deployment into a continuous workflow 是一个自动化的软件工程流水线，将代码集成测试和部署连接到一个连续的工作流中
  - Aim: to shorten development cycles, reduce integration problems, and ensure software can be released reliably at any time 缩短开发周期，减少集成问题，并确保软件可以随时可靠的发布
  - Faster feedback / Less manual work / Higher reliability / Always-ready software
  - Upload code $\rightarrow$ CI (build + test) $\rightarrow$ CD(deploy to pre-release environment)
  - CI is a practice where developers frequently integrate their code into a shared repository. Each integration automatically triggers a build and test process 开发人员频繁的将他们的代码集成到共享仓库中，每次集成都会触发构建和测试的过程
    - Importance:
      - Finds problems early
      - Prevent integration conflicts
      - Ensures consistent testing
      - Reduces manual work
  - CD automatically deploys the successfully built code from CI to a testing or pre-release environment 将CI中成功构建的代码部署到测试或预发布环境中
    - Importance
      - Ensures deployable software 确保可部署的软件
      - Reduces release risks 降低发布风险
      - Supports fast delivery 支持快速交付
        - Teams can release updates quickly because the deployment pipeline is already automated
  - Refactoring
  - Monitoring and Feedback
  - Knowledge Sharing
  - A/B Testing
- Matrices for Continuous Improvement
  - For development
    - Time for changes 从代码提交到部署到生产环境的时间
    - cycle time 从开始处理任务到完成所需的时间

    - deployment frequency 新代码部署到生产环境的频率
- For product Quality
    - Defect Density 缺陷密度 每单位代码的缺陷数
    - Mean Time to Recovery 平均恢复时间 从故障中恢复所需的平均时间
    - Escaped Defects 逃逸缺陷 在部署到生产环境后发现缺陷
- For Team Productivity
    - Velocity 团队在一个冲刺期间完成的工作量， 以故事点活任务书衡量
    - Work In progress 当前正在进行的任务书
    - Flow efficiency 任务花费的活跃时间与总时间之比
- For Customer and User
    - Customer Satisfaction 客户对产品或功能满意度的直接衡量
    - Net Promoter Score 衡量客户向他人推荐产品的可能性
    - Feature Adoption Rate 积极使用新发布功能的用户百分比