Lecture04_Software Specification

Created	@October 10, 2024 11:33 AM
Class	CPT 203
Type	Lecture
Materials	To-do - Software Requirements Document.pdf SRS - Contract Management System.pdf Case Study - Contract Management Systems.pdf Week 4 - Requirements Engineering.pdf
Reviewed	☐

Introduction

- **Definition:** 软件规格说明是描述软件系统功能、约束和交互过程的文档。它是软件设计和开发阶段的蓝图。
- **Purpose**: 确保所有股东清楚了解系统的功能和表现.
- **Output:** 软件规格说明会生成一份软件需求文档（Software Requirements Document），它作为开发过程中后续活动的输入。
- **System Requirements**: 软件系统的需求是关于系统应该做什么的描述, 需求应当满足客户的期望.

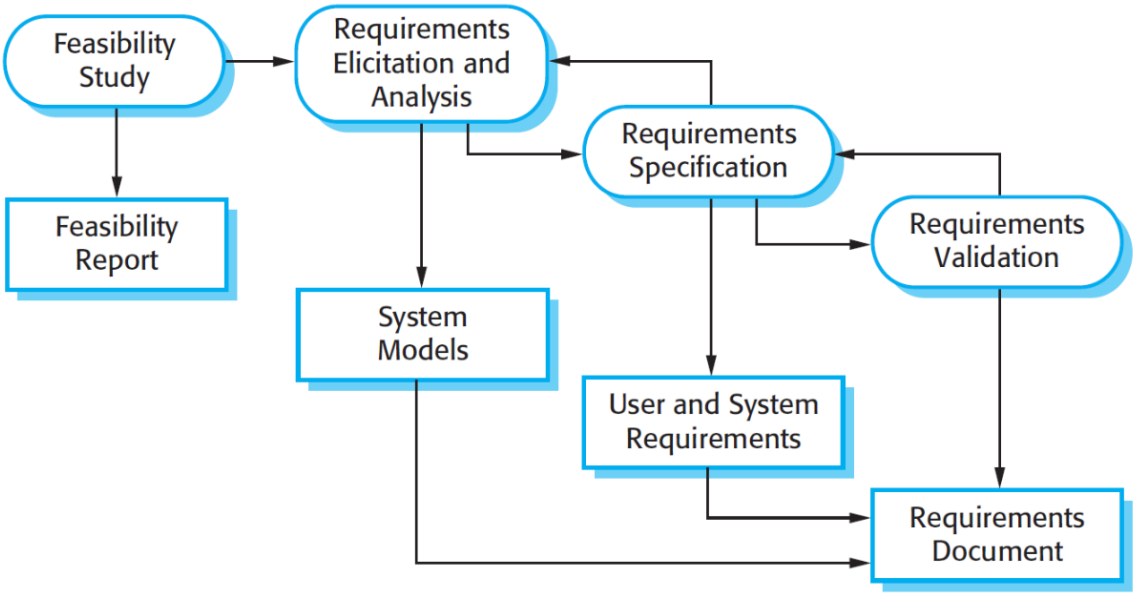
Software Specification

Important of Software Specification

- **清晰性和理解（Clarity and Understanding）**：为所有相关方提供系统需求的清晰描述，避免误解。

- **开发指导（Guidance for Development）**：为开发人员提供一个路线图，确保系统按需求进行构建。
- **测试依据（Basis for Testing）**：为创建测试用例提供依据，确保系统符合需求。
- **降低风险（Risk Reduction）**：在开发过程中，提前识别问题，减少后期修改的成本。

Software Specification Process



▼ **Four high-level activities**

（1）可行性研究（Feasibility Study）

- **目的**：确定拟议的系统在技术上、经济上以及操作上是否可行。
- **输出**：生成一份可行性报告（Feasibility Report），提供是否继续项目的建议。

（2）需求提取与分析（Requirements Elicitation and Analysis）

- **目的**：收集、分析和精炼软件系统的需求。
- **输出**：生成一套经过分析和精炼的需求，以确保其清晰、完整且可行。

（3）需求规格说明（Requirements Specification）

- **目的**：将收集到的需求文档化，形成结构化且清晰的需求说明。
- **输出**：形成一份详细的《软件需求文档》（Software Requirements Document, SRD），其中包括对系统功能和约束的全面描述。

（4）需求验证（Requirements Validation）

- **目的**：确保文档化的需求准确反映利益相关者的需求和期望。
- **输出**：生成经过审查和批准的需求，确保这些需求准确反映了所需系统的特性。

Clarity in Communication

需求可以分为不同层次的抽象描述：

- **高层次描述**：这是一种对系统应提供的服务或约束的抽象陈述，适用于与非技术人员（如最终用户和业务利益相关者）沟通。
- **详细描述**：这是对系统功能的详细、正式定义，适用于与技术人员（如软件开发人员、架构师和技术利益相关者）沟通。

User Requirements and System Requirements

用户需求描述的是用户从系统中需要什么，它们侧重于系统的功能和特性 (functionalities and features)，使用户能够实现他们的目标。

- **用户需求的特征：**
  - **以用户为中心（User-Centric）**：强调终端用户的需求和视角。
  - **功能性（Functional）**：描述系统应该做什么，通常为高层次。
  - **非技术性（Non-Technical）**：通常以用户和利益相关者易于理解的语言编写。

▼ 用户需求的例子

- **电商平台**：
  - 用户应能够搜索产品。
  - 用户应能够将产品添加到购物车。
  - 用户应能够使用各种支付方式完成购买。
- **社交媒体应用**：

    - 用户应能够创建和编辑个人资料。
    - 用户应能够发布更新并分享内容。
    - 用户应能够发送和接收消息。

System Requirements

- 系统需求描述的是系统的技术规范(technical specifications)和约束(constraints)，重点在于通过增加技术细节来优化用户需求。
- **系统需求的特征：**
    - **以系统为中心（System-Centric）**：强调系统的架构、设计和技术细节。
    - **功能性和非功能性（Functional and Non-Functional）**：包括系统应该做什么以及它如何执行的要求。
    - **技术性（Technical）**：以开发人员和技术利益相关者能够理解的语言编写。

▼ Example

- **用户需求定义**：系统应该生成每月的管理报告，显示每个诊所所开药物的费用。
- **系统需求说明**：
    - 每个月的最后一个工作日，生成药物处方的汇总报告，包括费用和开药的诊所。
    - 系统应在每个月的最后一个工作日下午5:30之后自动生成报告。
    - 应为每个诊所生成一份报告，并列出药物名称、处方总数、剂量数量和费用等详细信息。
    - 如果药物有不同的剂量单位，应为每个剂量单位生成单独的报告。
    - 所有报告的访问应限于被授权的用户，并通过管理访问控制列表来管理。

Role of Users in Defining User Requirements

- **主要贡献者(Primary Contributors)**：用户是用户需求的主要贡献者，因为他们最了解自己的需求以及系统如何支持他们的任务。
- **收集用户需求的技术**：
    - **面谈（Interviews）**：与用户进行一对一的面谈以获取详细的见解。
    - **调查和问卷（Surveys and Questionnaires）**：通过调查从更广泛的受众处收集意见。

- **研讨会和焦点小组（Workshops and Focus Groups）**：通过协作讨论收集多样化的意见。
- **观察（Observation）**：在用户的自然环境中观察他们的工作流程和挑战。
- **用户故事和用例（User Stories and Use Cases）**：以用户故事和用例的形式记录需求，捕获用户与系统的交互。

Role of Engineers in Defining System Requirements

- **主要贡献者：**工程师（包括软件开发人员、系统架构师和技术专家）是定义系统需求的主要贡献者。他们具有将用户需求转化为技术规范的技术能力。
- **定义系统需求的技术：**
  - **需求分析（Requirements Analysis）**：分析用户需求，推导出系统需求。
  - **技术研讨会（Technical Workshops）**：与技术利益相关者举行研讨会，定义系统规格说明。
  - **建模与图表（Modeling and Diagrams）**：使用建模技术（例如 UML、ER 图表）来表示系统需求。
  - **原型设计（Prototyping）**：创建原型，以验证系统需求和设计选择。
  - **可追溯性（Traceability）**：确保用户需求和系统需求之间的可追溯性，保持一致性。

▼ Question

- User requirements can be further devised into system requirements.
  - True, 用户需求描述了用户需要的高层次功能, 而系统需求则是根据用户需求推导出的更详细的技术规范.
- Engineers collect system requirements before they discovered the related user requirements.
  - False, 工程师首先从用户那里手机用户需求, 再将这些需求转化为系统需求.
- User requirements are defined by the users, while system requirements are defined by the engineers.
  - True, 用户需求主要来自用户的需求和期望, 而系统需求是工程师根据这些需求转换出的技术规范.

- User requirements are more general, while system requirements are more details and specific.
    - True, 用户需求通常是高层次的, 而系统需求则包含更具体的技术细节.

Functional Requirements

- **定义**：功能性需求描述系统必须执行的特定行为和功能，明确系统应如何满足用户的需求。
- **完整性（Completeness）**：所有用户要求的服务应被定义。
- **一致性（Consistency）**：需求应没有互相矛盾的定义。

在大型、复杂的系统中，实际上很难同时实现需求的一致性和完整性，因为涉及到众多利益相关者且他们的需求可能存在冲突。

- **特征**：功能性需求通常以用户需求的抽象方式描述，更多关注系统的动作和操作。
    - **行为导向（Behavioral）**：专注于系统必须执行的操作。
    - **以用户为中心（User-Centric）**：通常来自用户需求和使用案例。
    - **具体且可测量（Specific and Measurable）**：清晰定义，且可进行正确性测试。

▼ Example

MHC-PMS 系统的需求示例

1. 用户应能够搜索所有诊所的预约列表。
2. 系统应每天为每个诊所生成一份包含当日需预约的患者名单。
3. 每个使用系统的员工应能通过其八位数字的员工编号唯一识别。

▼ Questions

- Functional requirements = user requirements.
    - False, 功能需求和用户需求是相关的, 但不i完全相同. 功能需求可以从用户需求中推导出来, 描述系统需要执行的具体功能.
- Functional requirements specify the services or operations a system should provides to its users.
    - True, 功能需求描述系统应该具备的具体功能, 以满足用户的需求.
- Functional requirements has to be described in general term.

    - False, 功能需求应当具体且可测量, 而不是笼统的描述.

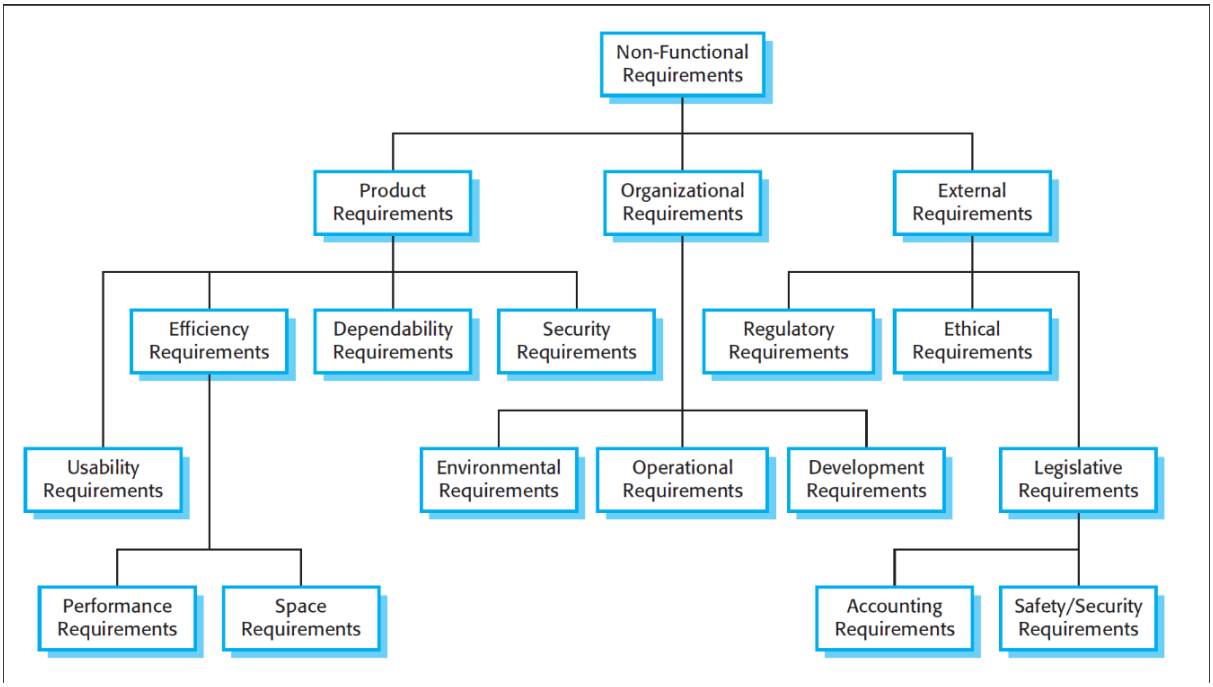
Non-Functional Requirements

- 定义：非功能性需求描述系统的操作质量和约束。它们详细说明系统应该如何表现，而不是应该做什么。非功能性需求与系统的特定服务或操作无关，而是关注系统的属性，如**可靠性、响应时间、扩展性、安全性或可用性**等。这些需求通常是对系统整体特性的约束。
    - 例子: 系统必须在 2 秒内响应用户请求，或系统的可用性应达到 99.99%。
- **系统整体性要求**：非功能性需求通常适用于整个系统，而不仅仅是单个功能或服务。

▼ **Important of Non-Functional Requirements**

- 非功能性需求有时比功能性需求更为关键，因为如果系统不能满足非功能性需求，那么整个系统可能会不可用。
- 非功能性需求的实现通常会分布在系统的各个部分，而不是单一的组件。例如：
    - 它们可能影响系统的整体架构，而不仅仅是某个具体模块。
    - 一个非功能性需求可能会生成多个相关的功能性需求。

- 特征:
    - **质量属性（Quality Attributes）**：关注性能、安全性、可用性、扩展性、可靠性等方面。
    - **以系统为中心（System-Centric）**：通常源于技术约束或业务需求。
    - **可测量（Measurable）**：非功能性需求应该是可量化和可测试的，以确保系统符合这些要求。



Classification of Non-Functional Requirements

- **产品需求（Product Requirements）**：这些需求定义或限制了软件的行为。
- **组织需求（Organizational Requirements）**：这些需求来自客户或开发者组织的政策和程序。
- **外部需求（External Requirements）**：这些需求来自于系统外部的因素，影响系统及其开发过程。

▼ Example

- **产品需求**：系统在正常工作时间内应该可用，停机时间不得超过一天中的五秒。
- **组织需求**：用户必须通过其健康卡身份验证。
- **外部需求**：系统必须实现有关患者隐私保护的法律要求。

Testability(可测试性)

非功能性需求的一个常见问题是，用户或客户往往以笼统的目标形式提出这些需求，这会导致需求难以测试。例如：

- **不明确的需求**：系统应易于医疗人员使用，且应按一定方式组织，以尽量减少用户错误。
  - 这种陈述是模糊的，无法有效测试。

- **明确的需求**：医疗人员应能在四小时培训后使用所有系统功能。培训结束后，经过培训的用户每小时的平均错误数不应超过两次。
    - 这种陈述明确且可测量，因此具有可测试性。

▼ 属性及其度量(Property and Measure)

- **速度**：
    - 度量标准：每秒处理的事务数、用户/事件响应时间、屏幕刷新时间。
- **大小**：
    - 度量标准：存储大小（MB）、ROM芯片数量。
- **易用性**：
    - 度量标准：培训时间、帮助页面数量。
- **可靠性**：
    - 度量标准：平均故障间隔时间、不可用概率、故障发生率、可用性。
- **健壮性**：
    - 度量标准：故障后重启时间、导致故障的事件百分比、故障时数据损坏的概率。
- **可移植性**：
    - 度量标准：目标依赖语句的百分比、目标系统的数量。

- 特征:
    - **具体（Specific）**：清楚地定义预期内容。
    - **可测量（Measurable）**：提供可以量化的度量标准。
    - **可实现（Achievable）**：在项目限制内，现实且可实现。
    - **相关性（Relevant）**：与系统的目标和目的直接相关。
    - **上下文（Context）**：指定需求应满足的背景条件。

▼ Example

- 非功能性需求的可测试示例
    - **性能需求（Performance Requirement)**：例如，系统应在峰值负载条件下每秒处理 1000 次事务，响应时间不应超过 2 秒。
        - 这种需求是具体的、可测量的、可实现的，并且有上下文背景，适合通过性能测试工具进行验证。

- 不可测试的非功能性需求示例
  - **可用性需求（Usability Requirement)**：系统应用户友好且直观。
    - **测试性分析**：这个需求含糊不清(Vagueness)，缺乏具体的标准和度量(Lack of Specific Metrics), 歧义性(Ambiguity)。不同用户对"用户友好"和"直观"的理解不同，因此难以评估。
- **改进后的陈述**：系统应在至少 50 位用户的反馈调查中，获得 80% 以上的系统可用性评分（SUS），调查样本量至少为 50 人。
  - 这种改进后的陈述明确了具体的衡量标准（SUS评分），并包含了可测量的目标和上下文（50人样本）。

Emergent System Properties(涌现系统属性)

涌现属性是指通过系统组件的相互作用和集体行为而产生的特性。这些属性并不归因于任何单个组件，而是从整个系统中涌现出来的。

- **涌现属性的例子：**
  - **性能（Performance）**：系统的总体速度和效率。
  - **可靠性（Reliability）**：系统在不同条件下随时间正确运行的能力。
  - **安全性（Security）**：系统保护数据并抵御攻击的能力。
  - **可扩展性（Scalability）**：系统处理增加的负载而不降低性能的能力。

Important of Highlighting Emergent System Properties(必要性)

- **系统范围的影响(System-Wide Impact):** 涌现属性影响整个系统，对于系统的整体成功至关重要。
- **可见性与意识（Visibility and Awareness）**：明确突出这些需求，确保所有利益相关者意识到它们的重要性和影响。
- **设计考虑（Design Considerations）**：涉及涌现属性的非功能性需求往往会影响系统的架构和设计决策。例如，性能和可靠性的需求可能决定系统的技术架构。
- **测试与验证（Testing and Validation）**：这些需求需要特定的测试策略，以确保系统达到期望的性能和可靠性标准。
- **风险管理（Risk Management）**：提前识别和处理这些需求有助于降低与系统性能和可靠性相关的风险。

Feasibility Study(可行性研究)

- **技术可行性（Technical Feasibility）**
  - **评估（Assessment）**：评估技术栈、技术资源和所需的技能。
  - **挑战（Challenges）**：识别潜在的技术挑战和风险。
  - **结果（Outcome）**：确定项目在技术上是否可行。
- **经济可行性（Economic Feasibility）**
  - **成本效益分析（Cost-Benefit Analysis）**：将预计成本与预期收益进行比较。
  - **预算（Budgeting）**：评估所需的财务资源和可用资源。
  - **结果（Outcome）**：确定项目在财务上是否可行。
- **运营可行性（Operational Feasibility）**
  - **组织适配（Organizational Fit）**：评估系统与组织运营的适配程度。
  - **支持与维护（Support and Maintenance）**：评估组织是否有能力支持和维护该系统。
  - **结果（Outcome）**：确定项目在运营上是否可行。

Requirement Elicitation and Analysis

需求提取与分析是一个迭代的过程，软件工程师通过与客户和系统的最终用户合作来发现和分析需求。这个过程可能涉及多个利益相关者，如最终用户、业务经理、工程师等。

▼ **需求提取过程的步骤：**

1. **需求发现（Requirements Discovery）**：通过与系统的利益相关者互动，发现他们的需求。
2. **需求分类与组织（Requirements Classification and Organization）**：将未结构化的需求收集起来，分组相关需求并将其组织为连贯的集群。
3. **需求优先排序与谈判（Requirements Prioritization and Negotiation）**：当涉及多个利益相关者时，需求之间可能会发生冲突。这一步通过谈判解决冲突并确定需求的优先级。
4. **需求规格说明（Requirements Specification）**：文档化需求并输入到下一轮迭代中。

▼ **Challenges**

- **利益相关者的需求不明确**：利益相关者可能不知道他们具体想要什么，尤其是在计算机系统方面，他们可能只能表达一些笼统的需求。
- **难以表达需求**：有时利益相关者无法准确地表达需求，或者他们提出不切实际的要求，因为他们不清楚什么是可行的。
- **利益相关者的隐含知识**：利益相关者通常使用他们的术语和对自身工作经验的隐含知识来表达需求，这使得需求工程师很难完全理解。
- **不同利益相关者的需求冲突**：不同的利益相关者有各自的优先级和需求，这些需求可能相互冲突，需求工程师需要在其中进行调解。
- **政治因素的影响**：组织中的政治因素可能会影响系统需求的制定，一些需求可能出于权力关系或部门间的利益考量。
- **经济与业务环境的变化**：在需求分析过程中，经济或业务环境可能会动态变化，影响到系统需求的准确性和优先级。

Requirements Discovery

需求发现是通过收集与现有系统相关的信息，整理用户和系统需求的过程。此过程涉及从多个来源获取信息，包括文档、利益相关者以及类似系统的规格说明。

▼ **需求发现的常用技术：**

访谈（Interviews）：

- 定义: 与系统利益相关者进行正式或非正式访谈，了解他们对系统的期望和需求。访谈适合于获取利益相关者的总体理解、他们如何使用系统以及当前系统所面临的困难等信息。
- **封闭式访谈**：预先设定一组问题，利益相关者回答这些问题。
- **开放式访谈**：没有预设议程，需求工程师与利益相关者讨论各种问题，收集需求。

▼ Challenges

- **领域知识的提取难度**：有些利益相关者使用特定领域的术语和行话，这些术语可能对需求工程师来说比较陌生，导致信息误解。
- **组织需求的获取难度**：由于组织内部的权力关系，利益相关者通常不愿意讨论影响需求的政治或组织问题。

▼ Characteristics of Effective interviews

- 思维开放，避免预设想法，愿意倾听。
- 使用引导性问题或提案促进讨论。

观察法（Observation）：

- 定义: 通过观察用户如何与系统交互，获取真实的需求信息。

场景法（Scenarios）：

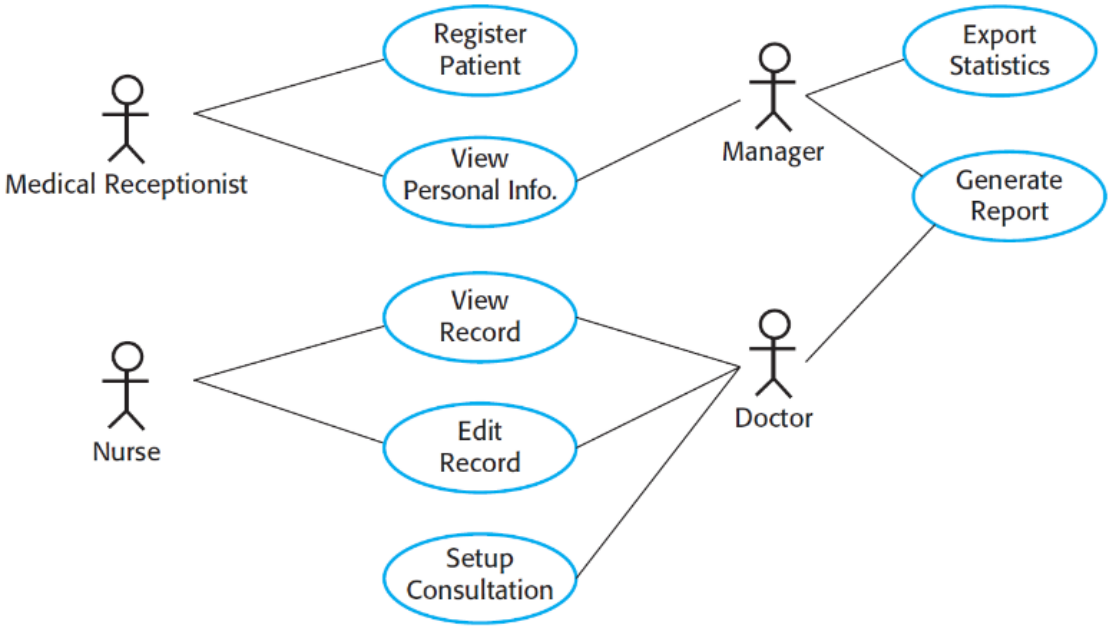
- **场景的定义和作用**：场景分析是通过与利益相关者（如用户、客户）合作，确定系统的各种场景，并捕捉这些场景的细节。这有助于更好地理解系统如何与外部环境互动。
- **文档化**：场景可以以文字形式写成，也可以通过图表、屏幕截图等补充信息。这样可以清楚地描述系统的操作流程，确保所有利益相关者都能理解系统的功能。
- **结构化方法**：有时也可以使用更结构化的方法，如事件场景或用例（use cases）来补充或代替场景分析。

用例法（Use Cases）：

- **用例的定义**：用例分析是一种需求获取技术，用例用来标识系统中的参与者及他们之间的交互。每个用例通常代表一个场景，描述系统的某一特定功能。
- **补充信息**：用例分析可以通过附加的文本描述或图形模型（如UML图）来补充，详细说明交互是如何进行的。
- **用例图的使用**：用例图是一种高层次的图表，用来展示所有可能的交互。图中每个用例表示一个场景，多个场景可以集成到一个用例图中。

▼ Limitation—用例主要关注系统的交互，但对以下内容的获取效果不佳：

- 约束条件(constraints)
- 高层次的业务需求(high-level business)
- 非功能性需求（如系统性能、可靠性）
- 域要求（特定领域内的技术和规定）(domain requirements)



原型法（Prototyping）：

- 定义: 通过构建系统的原型，向利益相关者展示系统的初步功能，收集他们的反馈以调整需求。

民族志(Ethnography)

- **民族志的定义和作用**：
  - 系统经常用于某种社会和组织环境中，而系统需求可能会因为这个环境而衍生或受到限制。
  - 满足这些社会和组织需求对系统成功至关重要。
  - 民族志是一种观察技术，分析员会融入到实际工作环境中，观察参与者如何完成任务，从而推导出隐含的系统需求。这些需求反映了实际工作的方式，而不是理论上应当如何进行。
  - 通过这种方式，可以发现那些参与者在工作中不明显表达的系统需求。
- **民族志发现的需求类型**：
  - **基于实际操作的需求**：分析人员通过观察实际操作，能够发现与系统互动的真实工作方式，而不是流程定义中所描述的理想方式。
  - **基于合作的需求**：观察过程中能够识别基于团队合作和参与者对他人行为的理解所衍生的需求。

- **民族志方法的局限性**：
  - **专注于最终用户**：民族志的优势在于它对最终用户行为的深入观察，但它并不总是适用于发现组织或领域需求。
  - **难以发现新特性**：民族志方法通常不擅长发现应该添加到系统中的新功能，因为它主要基于现有的工作方式，而不是探索创新的解决方案。

Requirement Specification

这是将用户需求和系统需求编写成文档的过程。文档中的需求需要满足以下几个条件：

- **清晰、无歧义、易于理解、完整且一致**，以确保不同的利益相关者能够准确理解需求，减少冲突或不一致的情况。
- 系统需求文档应包括**功能性需求和非功能性需求**，并以一种让没有详细技术知识的用户能够理解的方式进行描述。

此外，用户需求文档应该：

- 只描述系统的**外部行为**，即系统在不同情况下的反应，不应该包含设计或系统架构的详细信息。
- 采用简单的自然语言、表格、直观的图表等形式，确保读者能轻松理解。

▼ 系统需求和用户需求的区别

- **用户需求**：通常是功能和非功能需求的概述，描述系统应做什么。
- **系统需求**：是用户需求的扩展版，包含了更多的技术细节，并解释如何提供这些用户需求。

▼ 外部行为案例(External Behaviors)

- 系统必须**按时生成月度报告**。
- 系统必须**在每月的最后一天计算佣金**。
- 系统为了提高响应速度，配置数据应提前加载到内存缓存中。

这些外部行为是系统对外部触发的直接响应，而不涉及系统内部的具体实现细节。

▼ 尽管需求文档应避免包含详细的设计信息，但实际过程中完全排除设计信息是困难的。设计信息可能包括：

- 需要设计初始架构来帮助定义系统需求。
- 现有系统的互操作性可能对新系统提出额外的设计要求。

- 为满足非功能需求，可能需要指定特定的架构。

Natural Language Specification

自然语言的使用是普遍且直观(intuitive)的，但它也存在一些缺陷：

- 自然语言有时可能导致歧义或模棱两可的理解，读者的背景可能会影响对文本的不同解释。

▼ Natural Language Specification

- **避免误解**：写需求时应使用一致的格式，并确保所有需求定义都遵守这种格式。这样可以减少遗漏，并使需求更易于检查。
- **语言一致性**：在区分强制性和可选需求时，使用一致的语言。并使用文本高亮来强调需求的关键部分。
- **读者背景**：不应假设读者了解技术软件工程的语言，因此需要明确解释。
- **添加需求的理由**：每个用户需求应附有合理性说明，解释为什么包括这个需求。

Structure Specification

- 使用结构化的自然语言编写系统需求，在预定义的结构或模板中记录需求，确保需求清晰且易于理解。
- 需求通常可以记录在需求卡片上，每个需求一张卡片，并包含一些必要的字段，如需求的理由、对其他需求的依赖、需求的来源等。

▼ Sample

Insulin Pump/Control Software/SRS/3.3.2

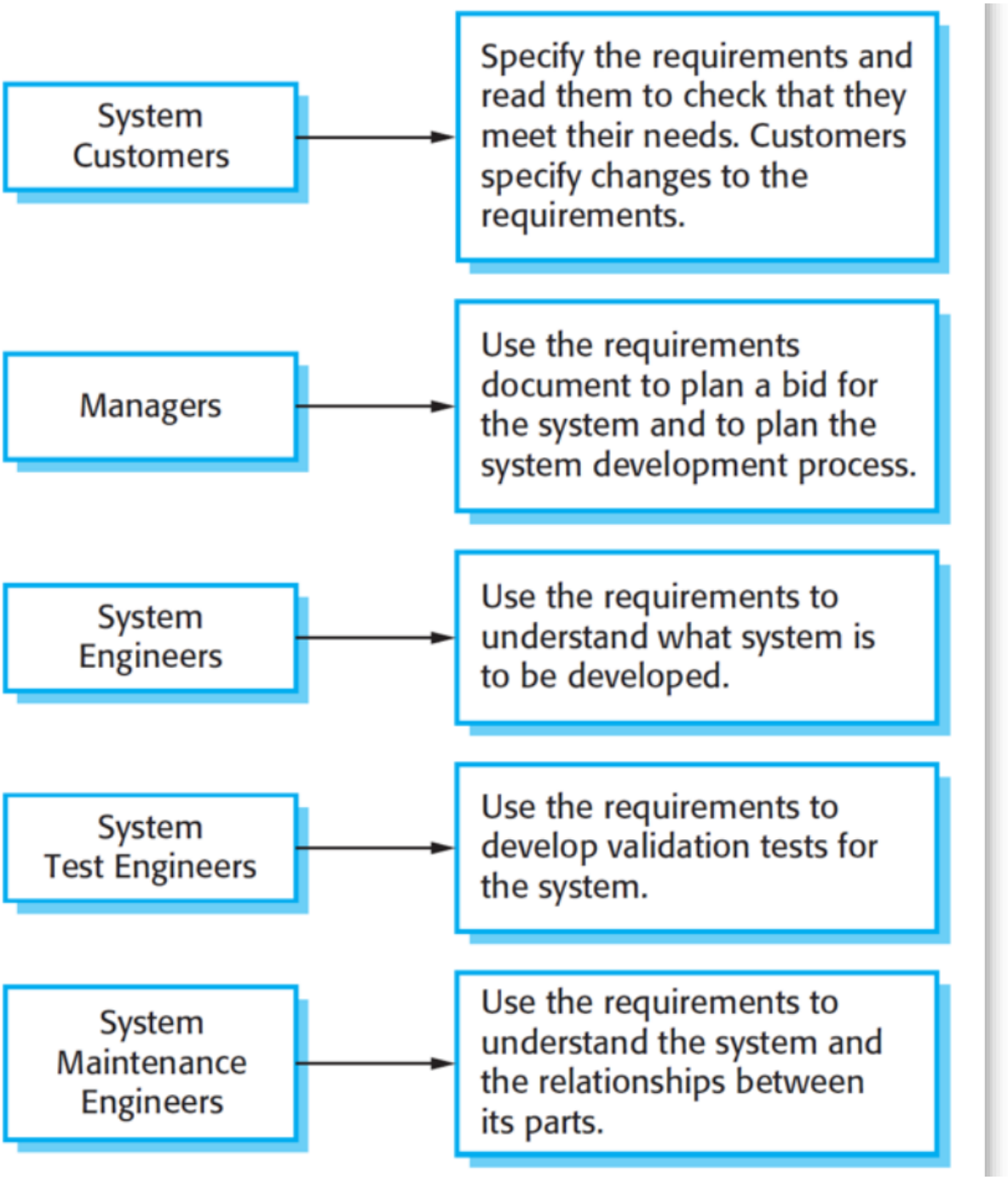
Function	Compute insulin dose: Safe sugar level.
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.
Inputs	Current sugar reading (r2), the previous two readings (r0 and r1).
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose—the dose in insulin to be delivered.
Destination	Main control loop.
Action	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requirements	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin.
Post-condition	r0 is replaced by r1 then r1 is replaced by r2.
Side effects	None.

Software Requirements Document

- **定义**：需求规格文档（SRD）是一份正式的文档，捕获和明确了软件系统的功能和非功能需求。它为开发团队和其他利益相关者提供蓝图。
- **目的**：需求文档的目的是促进利益相关者之间的**沟通**，提供开发、测试和设计的详细**指导**，并作为合同协议的一部分，确保利益相关者对系统功能的期望和约束达成**一致**。
- **敏捷开发中的要求**：尽管敏捷开发方法强调需求文档很快过时，但仍然有必要编写简短的辅助文档，定义业务和可靠性要求。

▼ Important of Software Requirements Document

- **清晰与理解(Clarity and Understanding)**：确保所有利益相关者对系统需求有清晰、共享的理解。
- **项目规划(Project Planning)**：有助于资源分配、时间估算、风险管理等项目规划。
- **范围管理(Scope Management)**：帮助管理项目的范围，避免需求膨胀。
- **测试基础(Basis for Testing)**：为创建测试用例和验证系统是否满足需求提供基础。
- **变更管理(Change Management)**：有助于在项目生命周期中管理需求的变化。



▼ Best Practices for Creating a Software Requirements Document

- **吸引利益相关者(Engage Stakeholders)**：确保所有相关人员参与需求收集过程，以确保覆盖全面。
- **清晰明确(Be Clear and Unambiguous)**：使用清晰的语言，避免误解。
- **需求优先化(Prioritize Requirements)**：识别并优先处理关键需求。
- **使用多种技术(Use Multiple Techniques)**：结合不同方法（例如，访谈、调查）全面收集需求。

- **验证与确认(Validate and Verify)**：不断验证和确认需求，确保准确性。
- **维护可追踪性(Maintain Traceability)**：确保每个需求可追溯到其来源，并贯穿开发过程。
- **迭代优化(Iterative Refinement)**：基于反馈和测试结果逐步优化需求。

▼ Level of Detail(详细程度)

定义正确细节级别：编写需求文档时，选择合适的细节级别至关重要，它可以确保清晰、有效的沟通，并有助于项目的成功。过多的细节会使文档繁冗复杂，而太少的细节则可能导致需求理解的差异和缺失。

- **避免过度详细(Avoid Over-Specification)**：提供太多细节会使文档臃肿且难以管理。
- **避免不充分详细(Avoid Under-Specification)**：过少的细节可能导致误解和需求的缺失。
- **使用可视化工具(Use Visual Aids)**：利用图表、模型等来清晰传达复杂信息。
- **迭代优化(Iterative Refinement)**：不断通过反馈和测试结果优化文档。

▼ **Factor Influencing the Level of Detail**

- **项目复杂性(Project Complexity)**：复杂项目通常需要更详细的需求文档。
- **干系人的需求(Stakeholder Needs)**：不同干系人可能需要不同的细节级别。
- **开发方法(Development Methodology)**：不同的软件开发方法（例如，敏捷开发、瀑布模型）可能会影响所需的细节程度。
- **法规要求(Regulatory and Compliance Requirements)**：受到监管和合规要求的项目可能需要更详细的文档。
- **风险管理(Risk Management)**：高风险项目通常需要更多的细节，以减少潜在问题。

Requirements Validation

- **定义**：需求验证是确保需求正确描述了客户期望的系统功能的过程。验证的目的是减少开发中后期发现问题所带来的成本。
- **不同类型的检查**：

- **有效性检查(Validation checks)**：确保所提出的功能与系统实际需求一致。
- **一致性检查(Consistency checks)**：确保文档中描述的功能不会互相冲突。
- **完整性检查(Completeness checks)**：确保文档覆盖所有功能和约束。
- **现实性检查(Realism checks)**：确保需求可以在技术上实现。
- **可验证性检查(Verifiability)**：确保需求可被检验和测试。

Validation Techniques

- **需求评审(Requirements reviews)**：系统性地分析需求，检查错误和不一致。
- **原型设计(Prototyping)**：使用可执行的模型向用户和客户展示系统。
- **测试用例生成(Test-case generation)**：确保需求是可测试的。如果难以生成测试用例，说明需求可能过于复杂或难以实现。

Best Practices for Software Specification

- **Engage Stakeholders（参与股东）**：
  - 在整个过程中应尽早并持续地让所有相关的股东参与进来，确保全面覆盖并获得支持和认同。
- **Use Multiple Techniques（使用多种技术）**：
  - 结合不同的需求获取和分析技术（如访谈、问卷调查、工作坊等）来收集和精炼需求。
- **Communicate Clearly（清晰沟通）**：
  - 使用明确和精确的语言，避免产生误解。
- **Prioritize Requirements（需求优先级排序）**：
  - 确定并优先处理最重要的需求，确保最关键的方面得到优先解决。
- **Validate Continuously（持续验证）**：
  - 持续与干系人验证需求，确保其准确性和完整性。
- **Document Thoroughly（全面记录）**：
  - 保持清晰和完整的需求文档，详细记录需求获取和分析的所有活动。
- **Iterate and Refine（迭代和精炼）**：
  - 基于反馈和测试结果迭代优化需求，逐步细化和完善。

Appendix—Structure of the Software Requirements Document(软件需求文档结构)

- **Introduction（介绍）**
  - **Purpose（目的）**：阐明SRD的目标和用途。
  - **Scope（范围）**：定义软件系统的范围。
  - **Definitions, Acronyms, and Abbreviations（定义、首字母缩写和缩略语）**：提供文档中使用的术语、首字母缩写和缩略语的定义。
  - **References（参考文献）**：列出引用的参考资料或标准。
  - **Overview（概述）**：简要概述文档的结构。
- **Overall Description（总体描述）**
  - **Product Perspective（产品视角）**：描述系统的背景和运行环境。
  - **Product Functions（产品功能）**：总结系统的主要功能。
  - **User Characteristics（用户特征）**：描述预期用户的特点。
  - **Constraints（约束条件）**：列出影响系统的任何约束。
  - **Assumptions and Dependencies（假设和依赖关系）**：描述与系统相关的假设和依赖条件。
- **Specific Requirements（具体需求）**
  - **Functional Requirements（功能性需求）**：系统需要提供的具体功能。
  - **Non-Functional Requirements（非功能性需求）**：系统操作中的质量属性和约束（例如性能、可靠性、安全性等）。
  - **External Interface Requirements（外部接口需求）**：描述系统与外部系统、硬件、软件的交互方式。
  - **System Features（系统特性）**：提供系统特性详细描述，包括使用案例和用户故事。
- **Appendices（附录）**
  - **Appendix A（附录A：术语表）**：定义文档中使用的术语和概念。
  - **Appendix B（附录B：分析模型）**：包含支持需求的模型和图示。

- **Appendix C（附录C：问题列表）**：包含讨论问题、未解决问题或其他需求相关内容。

TTL

What is software specification in the context of software engineering? How is software specification different from requirements engineering?

▼ Answer

Software specification is the process of finding out, analyzing, documenting, and checking these needs and constraints.

Software specification and requirements engineering can be used interchangeably.

In a normal circumstance, system requirements could be part of a software contract. Explain why system requirements are better candidate compared to user requirements to be expressed in the contract.

▼ Answer

A software contract is a legally bind document stating the expectation and responsibility among others. Compared to user requirements, system requirements included more specific information that spelled out the exact requirements. Therefore, system requirements are more suitable to be used in a software contract.

List the differences between functional requirements and non-functional requirements.

▼ Answer

Functional requirements are statements of services the system should provide. How the services should react and behave in certain condition.

Non-functional requirements are statements of services the system should provide. How the services should react and behave in certain condition. It is often more critical than individual functional requirements. Failing to meet a nonfunctional requirement can mean that the whole system is unusable.

The implementation of non-functional requirements may be intricately dispersed throughout the system.

Re-write the following non-functional requirement into a testable requirement. (Add some detail and restricted condition)

▼ The patient registration system in the hospital must be reliable.

Every patient registration terminal in the hospital must have an additional terminal to act as the backup. The down time for a terminal must not exceed 20 minutes in the peak hours (7AM to 11AM); not exceed 30 minutes in the non-peak hours (11AM to 5PM)

▼ The self-service ticketing system in the cinema must be fast and responsive.

The barcode/QR scanner must respond in 0.5 second. The system must retrieve the ticket information within 1 second after the code is scanned. The ticket must be completely printed within 3 seconds after the user confirm to print the tickets.

What would happen if the Software Requirement Document (SRS) for a mission critical system is missing out important details?

▼ Answer

If the missing information is in the early stage before the project officially started, the project schedule would be affected by the missing information. If the project is to be outsourced to a third-party software company, the contract of the software will not include the missing requirements.

If the missing information is discovered in the later stage of the development, the project may be delayed because engineers must revisit the requirements engineering stage to fix the missing pieces. If the project is outsourced to a third-party company, contract need to be re-negotiated.

If the missing information is not being discovered, after the system being used for many years, the maintenance would become difficult because of the missing information.

Express the iterative process of the requirements engineering in the form of spiral view(螺旋图).

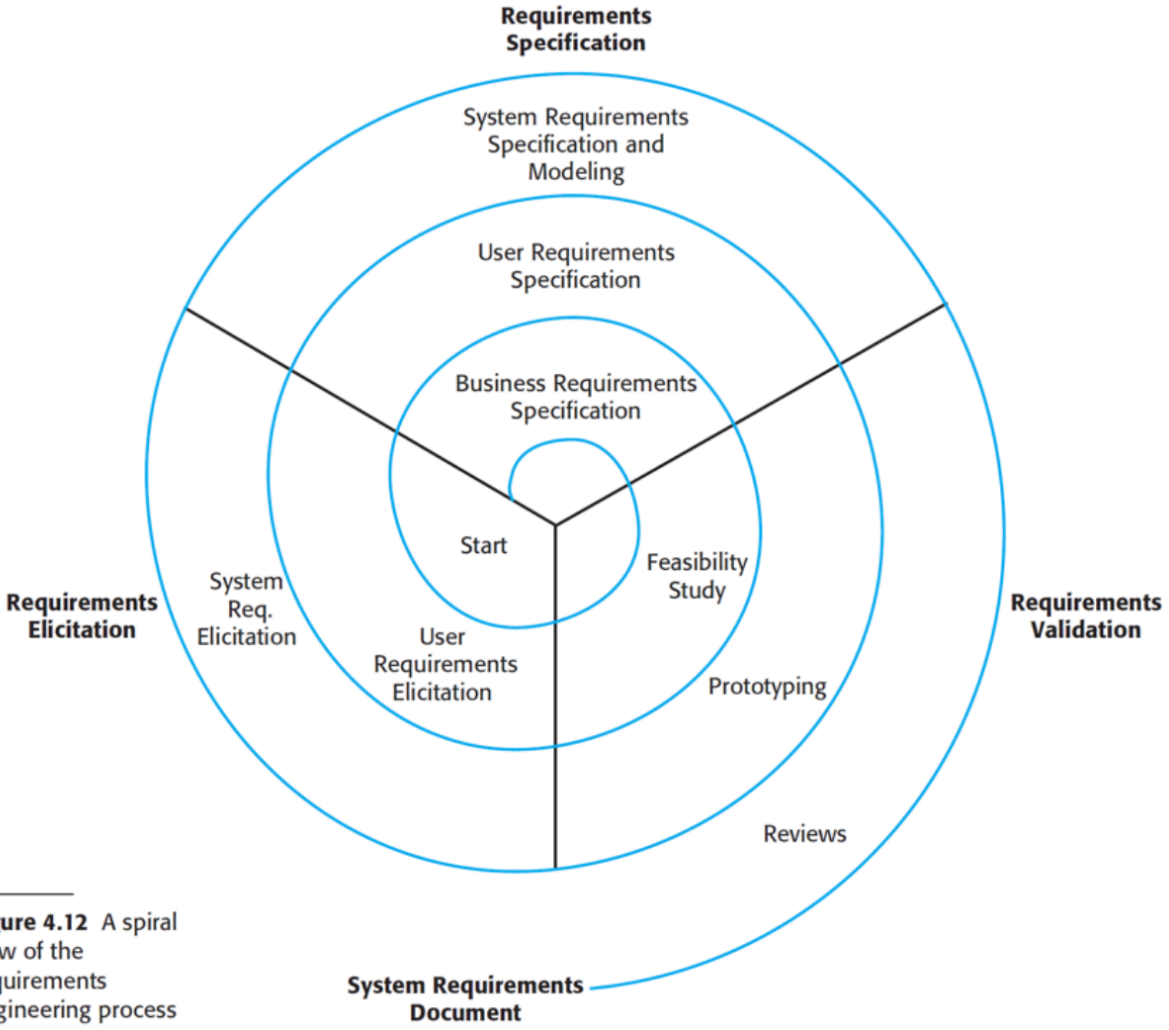


Figure 4.12 A spiral view of the requirements engineering process

What is requirements validation?

▼ Answer

It is the process to check if the requirements correctly define the system, which the customer really wants.

What are the aspects that the requirements validation checks?

▼ Answer

Validity checks, Consistency checks, Completeness checks, Realism checks,
Verifiability