软件架构认证专业人士

课程大纲(CPSA)®

基础级

2024.1-rev4-CN-202400412

目录

本课程大纲只能在满足以下前提条件下使用：

1. 您希望获得CPSA®软件架构师基础级认证证书或CPSA®软件架构师高级认证证书。为了获得证书，您应该在自己的计算机上创建工作副本来使用这些文本文件和/或课程大纲。如果打算以其他方式使用文件和/或课程大纲，例如将其传播给第三方、用于广告等，请写邮件至info@isaqb.org询问是否允许这样做。然后还须签订一份单独的许可协议。
2. 如果您是培训师或培训机构，一旦获得使用许可，您就可以使用这些文件和/或课程大纲。如有任何问题都可发送至info@isaqb.org咨询。在许可协议内对所有方面都有全面的规定。
3. 如果既不属于以上情况1，也不属于情况2，但仍想使用这些文件和/或课程大纲，请写邮件联系iSAQB® e.V. info@isaqb.org。您将获得通过现有许可协议获得相关许可的可能性，从而获得所需的使用授权。

缩写“e.V.”是iSAQB®官方名称的一部分，代表“注册协会”，根据德国法律，该协会描述其作为法律实体的地位。为简便起见，iSAQB® e.V.在下文中称为iSAQB®，不再使用上述缩写（iSAQB® e.V.）。

学习目标列表

- LG1-1：讨论软件架构的定义（R1）
- LG1-2：理解和解释软件架构的目标和收益（R1）
- LG1-3：将软件架构理解为软件生命周期的一部分（R2）
- LG1-4：了解软件架构师的任务和职责（R1）
- LG1-5：将软件架构师的角色与其他利益相关方联系起来（R1）
- LG1-6：能解释开发方法和软件架构之间的相关性（R2）
- LG1-7：区分短期和长期目标（R2）
- LG1-8：区分显性陈述和隐性假设（R1）
- LG1-9：在大的架构背景下软件架构师的职责（R3）
- LG 1-10：区别IT系统的不同类型（R3）
- LG1-11：分布式系统的挑战（R3）
- LG 2-1：选择和使用架构开发的方法和启发式方法（R1，R3）
- LG 2-2：设计软件架构（R1）
- LG 2-3：识别并考虑影响软件架构的因素（R1-R3）
- LG 2-4：设计和实施横切（Cross-Cutting）概念（R1）
- LG 2-5：描述、解释并适当应用重要的解决方案模式（R1，R3）
- LG 2-6：解释和使用设计原则（R1-R3)
- LG 2-7：管理构建块（Building Blocks）之间的依赖关系（R1）
- LG 2-8：用适当的方法和技术达到质量需求（R1）
- LG 2-9：设计和定义接口（R1-R3）
- LG 2-10：了解软件部署的基本原则（R3）
- LG 3-1：解释并考虑技术文档的需求（R1）
- LG 3-2：描述和沟通软件架构（R1-R3）
- LG 3-3：解释并应用符号/模型来描述软件架构（R2-R3）
- LG 3-4：解释和使用架构视图（R1）
- LG 3-5：解释和应用系统的上下文视图（R1）
- LG 3-6：记录和沟通横切概念（R2）
- LG 3-7：描述接口（R1）
- LG 3-8：解释和记录架构决策（R1-R2）
- LG 3-9：了解文档化的其他资源和工具（R3）
- LG 4-1：讨论质量模型和质量特性（R1）
- LG 4-2：明确软件架构的质量需求（R1）
- LG 4-3：软件架构的定性分析（R2-R3）
- LG 4-4：软件架构的定量评估（R2）
- LG 5-1：了解需求、约束和解决方案之间的关系（R3）
- LG 5-2：了解解决方案的技术实施原理（R3）

引言

基础级培训教授什么？

获得授权认证的软件架构专业人员 – 基础级（CPSA-F®: Certified Professional Software Architect – Foundation Level）培训将为参与者提供在设计、阐述并记录软件架构所需的知识和技能，以满足中小型系统的相应需求。参与者基于他们个人的实践经验和现有技能，将学习从现有系统愿景和详细需求中得出系统架构决策。CPSA-F®培训过程教授的是独立于特定开发过程的软件架构的设计、文档化以及评估等的方法和原则。

重点是以下技能的教育和培训：

- 与需求、管理、开发、运营和测试的利益相关方讨论并协调基本架构决策
- 理解软件架构的基本活动，并为中小型系统开展这些活动
- 基于架构视图、架构模式和技术概念来记录和沟通软件架构。

此外，培训还包括：

- 软件架构的术语及其含义
- 软件架构师的任务和职责
- 软件架构师在开发项目中的角色
- 最新的软件架构开发方法和技术。

培训中不包括的内容

本课程大纲反映了iSAQB®成员目前认为对实现 CPSA-F®的学习目标是必要的和有用的内容，但它不是对“软件架构”整个领域的全面描述。

以下主题或概念不属于CPSA-F®培训内容：

- 具体的实现技术、框架或库
- 编程或编程语言
- 特定的过程模型
- 建模符号（如UML）的基础知识或建模本身的基础知识
- 系统分析和需求工程（请参阅IREB® e.V.的培训和认证计划。https://ireb.org，国际需求工程委员会）
- 软件测试（请参阅ISTQB® e.V.的培训和认证计划。https://istqb.org，国际软件测试认证委员会）
- 项目或产品管理
- 介绍特定的软件工具。

> 培训的目的是为获得相应的应用所需的高级知识和技能而提供基础知识。

前提条件

iSAQB® e.V.可以通过相应的问题检查以下认证考试的先决条件。

参与者应具备以下知识和/或经验。特别是，具有在团队中丰富的软件开发实践经验是理解学习材料和成功认证的重要前提:

- 正式教育之外，通过团队开发多个系统获得的超过18个月的软件开发实践经验
- 至少具备一种高级编程语言的知识和实践经验，尤其是：
  - 针对如下概念
    - 模块化（包、命名空间等）
    - 参数传递（按值调用，按引用调用）
    - 范围，即类型和变量的声明和定义
  - 类型系统的基础知识（静态与动态类型、通用数据类型）
  - 软件中的错误和异常处理
  - 全局状态和全局变量的潜在问题

如下基本知识：

  - 建模和抽象
  - 算法和数据结构（即列表、树、哈希表、字典、映射）
  - UML（类、封装/包、组件和序列图）及其与源代码的关系
  - 软件测试方法（例如单元测试和验收测试）

此外，以下内容将有助于对多个概念的理解：

- 命令式、声明式、面向对象和函数式编程的基础和区别
- 如下的实践经验
  - 一种高级的程序设计语言
  - 设计和实现分布式应用程序，如客户端-服务器系统或web应用程序
  - 技术文档，尤其是记录源代码、系统设计或技术概念

（课程大纲）结构、时长和教学方法

以下课程大纲中给出的学习时间只是建议。培训课程的持续时间至少为三天，但也可以更长。培训机构可能在持续时间、教学方法、练习的类型和结构以及详细的课程大纲方面有所不同。示例和练习的类型（领域和技术）可以由培训机构自主确定。

内容	推荐的时长（分钟）
1. 软件架构基本概念	120
2. （软件架构的）设计与开发	420
3. （软件架构的）规格说明与沟通	240
4. 软件架构和质量	120
5. 示例	90
总计	990

学习目标与考试的关联性

课程大纲各章的结构中对学习目标进行了优先级划分。对于每个学习目标，明确说明了该学习目标或它的子目标的考试相关性（通过R1、R2、R3分类，见下表）。每个学习目标都描述了要教授的内容，包括其关键术语和概念。

关于（学习目标）与考试的相关性，本课程大纲使用了以下类别：

标识	学习目标类别	含义	与考试的关联性
R1	能够或掌握	这些内容是参与者在完成课程后能够独立实践的内容。在课程中，这些内容将通过练习和讨论来涵盖。	该内容将作为考试的一部分。
R2	理解	这些是参与者原则上应该理解的内容。它们通常不会是培训中练习的主要重点。	该内容可能是考试的一部分。
R3	知道	这些内容（术语、概念、方法、实践或类似内容）可以增强对主题的理解和启发。如果需要，这些内容可以包含在培训中。	该内容不属于考试内容。

如果需要，可以进一步阅读学习目标中的参考资料，包括标准或其他来源。每章的“术语和概念”部分列出了与本章内容相关的术语，其中一些术语用于描述学习目标。

当前版本和公共存储库

您可以在官方下载页面上找到此文档的最新版本 https://public.isaqb.org/。

该文档保存在公共存储库中，位于 https://github.com/isaqb-org/curriculum-foundation，所有变更和修改都是公开的。

请在我们的公共问题跟踪器中报告（您发现的）任何问题 https://github.com/isaqb-org/curriculum-foundation/issues。

1. 软件架构的基本概念

时长：120 分钟	练习：无

相关术语

Software architecture（软件架构）；architecture domains（架构域）；structure（结构）；building blocks（构建块）；components（组件）；interfaces（接口）；relationships（关系）；cross-cutting-concepts（横切概念）；software architects and their responsibilities（软件架构师及其职责）；tasks and required skills（任务和所需技能）；stakeholders and their concerns（利益相关方及其关注点）；requirements（需求）；constraints（约束）；influencing factors（影响因素）；types of IT systems (embedded systems; real-time systems; information systems etc.)（IT系统的类型 （嵌入式系统、实时系统、信息系统 等））

学习目标

LG 1-1：讨论软件架构的定义（R1）

软件架构师知道软件架构的若干定义（包括ISO 42010/IEEE 1471、SEI、Booch 等），并可以说出它们的相似之处：

- 具有接口和关系的组件/构建块
- 构建块是一个通用术语，组件是其特殊形式
- 结构、横切概念、原则
- 架构决策及其对整个系统及其生命周期的影响

LG 1-2：理解和解释软件架构的目标和收益（R1）

软件架构师可以说明软件架构的以下基本目标和收益：

- 支持系统的设计、实现、维护和运行
- 实现功能需求或确保能够满足这些需求
- 实现可靠性、维护性、可互换性、信息安全性、性能效率等要求
- 确保所有相关利益相关方理解系统的结构和方案
- 系统化地降低复杂性
- 为实现和运营指定与架构相关的指导方针

LG 1-3：将软件架构理解为软件生存周期的一部分（R2）

软件架构师理解他们自身的任务，并可以将其成果集成到IT系统的整个生命周期中。他们可以：

- 识别与软件架构相关的需求、技术或系统环境变更的后果
- 详细说明IT系统与支持的业务和运营过程之间的关系

LG 1-4：了解软件架构师的任务和职责（R1）

软件架构师负责满足需求并创建解决方案的架构设计，根据所使用的实际方法或过程模型，他们必须将这一职责与项目管理和/或其他角色的总体职责保持一致。

软件架构师的任务和职责：

- 澄清和仔细检查需求和约束条件，并在必要时对其进行细化，包括所需特征和所需约束条件
- 决定如何将系统分解为构建块，同时确定构建块之间的依赖关系和接口
- 确定并选定横切概念（实例持久化、通信、图形用户界面等）
- 基于视图、架构模式、横切和技术概念来沟通和记录软件架构
- 陪同架构的实现和实施；如有必要，将相关利益相关方的反馈整合到架构中；评审并确保源代码和软件架构的一致性
- 分析和评估软件架构，特别是有关涉及满足需求的风险，请参见 LG 4-3：软件架构的定性分析（R2-R3）和 LG 4-4：软件架构的定量评估（R2）
- 为其他利益相关方识别、强调、表明架构决策识别架构决策的后果

他们应该独立地认识到所有任务中迭代的必要性，并指出适当和相关反馈的可能性。

LG 1-5：将软件架构师的角色与其他利益相关方联系起来（R1）

软件架构师能够解释他们的角色，他们应该在特定的背景下，结合其他利益相关方和组织单元，调整他们对软件开发的贡献，特别是与：

- 产品管理和产品负责人
- 项目经理
- 需求工程师（需求或业务分析师、需求经理、系统分析师、业务负责人、领域专家等）
- 开发人员
- 质量保证人员和测试人员
- IT运营人员和管理员（主要适用于生产环境或信息系统的数据中心）
- 硬件开发人员
- 企业架构师和架构委员会成员

LG 1-6：能解释开发方法和软件架构之间的相关性（R2）

- 软件架构师能够解释迭代方法对架构决策的影响（关于风险和可预测性）。
- 由于固有的不确定性，软件架构师通常必须迭代式地工作和做出决策。为此，他们必须系统性地从其他利益相关方那里获得反馈。

LG 1-7：区分短期和长期目标（R2）

软件架构师可以解释短期和长期目标之间的潜在冲突，以便为所有利益相关方找到合适的解决方案

LG 1-8：区分显性陈述和隐性假设（R1）

软件架构师：

- 应明确提出假设或先决条件，从而防止隐性假设
- 知道隐性假设可能会导致利益相关方之间的潜在误解

LG 1-9：在大的架构背景下软件架构师的责任（R3）

iSAQB®CPSA®基础级的重点是单一软件系统的结构和方案。

此外，软件架构师还应熟悉其他架构领域，例如：

- 企业IT架构：应用程序环境的结构
- 业务和过程架构：业务流程等的结构
- 信息架构：跨系统的结构以及信息和数据的使用
- 基础设施或技术架构：技术基础设施、硬件、网络等的结构。
- 硬件或处理器架构（用于硬件相关系统），这些架构领域不属于 CPSA-F®的重点内容。

LG 1-10：区分IT系统类型（R3）

软件架构师了解不同类型的IT系统，例如：

- 信息系统
- 决策支持、数据仓库或商业智能系统
- 移动系统
- 云原生系统（请参阅[Cloud-Native]）
- 批处理过程或系统
- 基于机器学习或人工智能的系统
- 硬件相关系统；在这里，他们理解硬件/软件协同设计的必要性（硬件和软件设计的时间和内容相关性）

LG 1-11：分布式系统的挑战（R3）

软件架构师能够：

- 识别给定软件架构中的分布
- 分析给定业务问题的一致性准则
- 解释分布式系统中事件的因果关系

软件架构师知道：

- 分布式系统中的通信可能会失败
- 现实世界数据库一致性方面的限制
- “大脑分裂/脑裂”问题是什么以及为什么很难解决
- 确定分布式系统中事件的时间顺序是不可能的

参考资料

[Bass+ 2021], [Gharbi+2020], [iSAQB® References], [Lorz+2021], [Starke 2020], [vanSteen+Tanenbaum]

2. 软件架构的设计与开发

时长：330 分钟	练习：90分钟

相关术语

Design（设计）；design approach（设计方法）；design decision（设计决策）；views（视图）；interfaces（接口）；technical and cross-cutting concepts（技术和横切概念）；architectural patterns（架构模式）；pattern languages（模式语言）；design principles（设计原则）；dependencies（依赖）；coupling（耦合）；cohesion（内聚）；top-down and bottom-up approaches（自顶向下和自底向上方法）；model-based design（基于模型的设计）；iterative/incremental design（迭代/增量设计）；domain-driven design（领域驱动设计）

学习目标

LG 2-1：选择和使用架构开发的方法和启发方法(R1,R3)

软件架构师能够说出、解释和使用架构开发的基本方法，例如：

- 自顶而下和自底而上的设计方法（R1）
- 基于视图的架构开发（R1）
- 迭代和增量设计（R1）
  - 迭代的必要性，尤其是当决策受到不确定性的影响时（R1）
  - 设计决策反馈的必要性（R1）
- 领域驱动设计，见[Evans 2004]（R3）
- 演进式架构，参见[Ford 2017]（R3）
- 整体分析，见[Hofmeister+ 1999]（R3）
- 模型驱动架构（R3）

LG 2-2：设计软件架构(R1)

软件架构师能够：

- 基于为既不是安全关键型也不是业务关键型的软件系统，基于已知功能和质量需求，设计、适当沟通和记录软件架构
- 就系统分解和构建块结构做出与结构相关的决策，并有意识地设计构建块之间的依赖关系
- 识别并说明设计决策的相互依赖关系和权衡性的合理性
- 解释术语黑盒和白盒，并有目的地应用它们
- 逐步细化并定义构建块
- 设计架构视图，尤其是构建块视图、运行时视图和部署视图
- 解释这些决策对相应源代码的影响
- 将架构的技术和业务/领域相关元素分开，并说明这些决策的合理性

- 识别与架构决策相关的风险。

LG 2-3：识别并考虑影响软件架构的因素(R1-R3)

软件架构师能够澄清和考虑需求（包括限制他们决策的约束），他们明白，他们的决策可能会对正在设计的系统、其架构或开发过程产生额外的需求（包括约束）。

他们应该认识到并说明以下方面的影响：

- 产品相关需求，如（R1）
  - 功能需求
  - 质量需求
- 技术约束，例如
  - 现有或计划中的硬件和软件基础设施（R1）
  - 数据结构和接口的技术约束（R2）
  - 参考架构、库、组件和框架（R1）
  - 程序设计语言/编程语言
- 组织约束，例如
  - 开发团队和客户的组织结构（R1），特别是康威（Conway’s）定律（R2）。
  - 公司和团队文化（R3）
  - 伙伴关系和合作协议（R2）
  - 标准、指南和过程模型（例如批准和发布过程）（R2）
  - 可用资源，如预算、时间和人员（R1）
  - 员工的可用性、技能组合和承诺（R1）
- 监管约束，如（R2）
  - 地方和国际法律约束
  - 合同和责任问题
  - 数据保护和隐私法
  - 合规问题或提供举证责任的义务
- 趋势，如（R3）
  - 市场趋势
  - 技术趋势（如区块链、微服务）
  - 方法论趋势（例如敏捷）
  - 利益相关方进一步的关注和授权设计决策的（潜在）影响

软件架构师能够描述这些因素如何影响设计决策，并可以通过为其中一些因素提供示例来详细说明影响因素变化的后果（R2）。

LG 2-4：设计和实施横切概念 (R1)

软件架构师能够：

- 解释横切概念的重要性

- 决定并设计横切概念，例如持久化、通信、GUI、错误处理、并发性、性能效率
- 识别和评估这些决策之间潜在的相互依赖关系。

软件架构师知道这样的横切概念可以在整个系统中重复使用。

LG 2-5：描述、解释并适当应用重要的解决方案模式（R1，R3）

软件架构师知道：

- 各种架构模式，并可以在适当的时候应用它们
- 模式是在给定的上下文环境中为给定的问题和需求实现特定质量的一种方式
- 存在各种类型的模式（R3）
- 与模式相关的特定技术或应用领域的其他资源（R3）

软件架构师可以解释并提供以下模式的示例（R1）：

- 分层式：
  - 抽象层隐藏细节，例如：ISO/OSI网络层，或“硬件抽象层”。请参阅 https://en.wikipedia.org/wiki/Hardware_abstraction
  - 另一种解释是分层（逻辑上）分离功能或职责，请参阅 https://en.wikipedia.org/wiki/Multitier_architecture
- 管道和过滤器：代表数据流模式，将逐步的过程拆分处理为一系列处理活动（“过滤器”）和关联的数据传输/缓冲功能（“管道”）
- 微服务将应用程序拆分为多个通过网络进行通信的、独立可执行文件
- 依赖注入作为依赖反转原则的可能解决方案[Newman 2015]

软件架构师可以解释以下几种模式，解释它们与具体系统的关联性，并提供示例（R3）：

- 黑板：处理确定性算法无法解决但需要多样化知识的问题
- 代理（Broker）：负责协调提供者和消费者之间的通信，应用于分布式系统。负责转发请求和/或传输结果和异常
- 组合子（同义词：闭合操作），对于T类型的域对象，寻找输入和输出类型都为T的运算。参见[Yorgey 2012]
- CQRS（Command-Query-Responsibility-Segregation 命令查询责任分离）：分离信息系统中的读写问题。需要一些数据库/持久化技术的环境，以了解“读取”和“写入”操作的不同特性和需求
- 事件溯源(event sourcing)：通过一系列事件处理对数据的操作，每个事件都记录在一个仅能添加的存储（append-only store）中
- 解释器：将领域对象或DSL表示为语法，提供独立于领域对象本身，实现领域对象语义解释的功能
- 集成和消息传递模式（例如，来自[Hohpe+2004]）
- MVC（模型视图控制器）、MVVM（模型视图视图模型）、MVU（模型视图更新）、PAC（表示抽象控制）模式系列，将外部表示（视图）与数据、服务及其配合分离

- 接口模式，如适配器、外观、代理。这样的模式有助于子系统的集成和/或依赖关系的简化架构师应该知道，这些模式可以独立于（对象）技术使用
  - 适配器：解耦消费者和供方 —— 供方的接口与消费者的接口不完全匹配，适配器将一方与另一方的接口更改从而解耦
  - 外观（Facade）：通过提供简化的访问，简化消费者使用供方的过程
  - 代理(Proxy)：消费者和供方之间的中介，实现时间解耦、结果缓存、控制对提供者的访问等。
- 观察者：感兴趣的对象（观察者）与另一个对象（主题）登记注册，以便主题在有变更时通知观察者。
- 插件：扩展组件的行为
- 端口和适配器（类似于洋葱架构、六边形架构、干净架构）：将领域逻辑集中在系统的中心，在边缘与外部世界（数据库、UI）建立连接，依赖关系仅由外向内，从不由内向外[Lange 2021][Martin 2017]
- 远程过程调用：使函数或算法在不同的地址空间中执行
- SOA（面向服务的架构）：一种向系统用户提供抽象服务而非具体实现的方法，以促进跨部门和公司之间服务的复用
- 模板和策略：通过封装特定算法，使其变得灵活
- 访问者：将数据结构的遍历与具体处理分开

软件架构师知道架构模式的初始来源，如POSA（例如[Buschmann+1996]）和 PoEAA（[Fowler 2002]）（用于信息系统）（R3）。

LG 2-6：解释和使用设计原则 (R1-R3)

软件架构师能够解释什么是设计原则。他们可以概述它们在软件架构方面的总体目的和应用。（R2）软件架构师能够：

- 解释下面列出的设计原则，并可以举例说明
- 解释如何应用这些原则
- 解释需求如何确定哪些原则应该使用
- 解释设计原则对实现的影响
- 分析源代码和架构设计，以评价这些设计原则是否已经应用或应该应用

抽象（R1）

- 从某种意义上说，这是一种得出有用且泛化的方法
- 作为一种设计技术，构建块依赖于抽象而非实现
- 接口作为抽象

模块化（R1-R3）

- 信息隐藏和封装（R1）
- 关注点分离-SoC（R1）

- 松散但功能足够的构建块耦合（R1），见LG 2-7
- 高内聚（R1）
- SOLID原则（R1-R3），在一定程度上与架构级别相关
  - S：单一责任原则（R1）及其与SoC的关系
  - O：开闭原则（R1）
  - L：里氏替换原则（R3）作为 OO 设计中实现一致性和概念完整性的一种方法
  - I：接口隔离原则（R2），包括其与LG 2-9的关系
  - D：通过接口或类似抽象的依赖反转原则（R1）

概念完整性（R2）

- 表示类似问题解决方案的统一性（同质性、一致性）（R2）
- 作为实现最小意外原则的手段（R3）

简单性（R1-R2）

- 作为降低复杂性的手段（R1）
- 作为KISS（R3）和YAGNI背后的驱动因素（R3）

预期错误（R1-R2）

- 作为稳健和弹性系统的设计手段（R1）
- 作为鲁棒性原则（伯斯塔尔法则）的泛化

其他原则（R3）

软件架构师知道其他原则（如CUPID，请参见[Nygard 2022]），并可以应用它们。

LG 2-7：管理构建块（Building Blocks）之间的依赖关系（R1）

软件架构师理解构建块之间的依赖关系和耦合，并可以有针对性地使用它们。他们：

- 了解和理解构建块的不同类型的依赖关系（例如，通过调用/委派耦合、消息传递/事件、组合、创建、继承、时间耦合、通过数据耦合、数据类型或硬件）
- 理解依赖关系如何增加耦合
- 可以有针对性地使用这些类型的耦合，并可以评估这种依赖关系的后果
- 了解并能够应用减少或消除耦合的可能性，例如：
  - 模式（见LG 2-5）
  - 基本设计原则（见LG 2-6）
  - 依赖关系的外部化，即在安装或运行时定义的具体依赖关系，例如通过使用依赖注入。

LG 2-8：用适当的方法和技术达到质量需求（R1）

软件架构师理解并考虑质量需求对架构和设计决策的巨大影响，例如：

- 效率、运行时性能
- 可用性
- 维护性、易修改性、易扩展性和适应性

- 能效性

他们可以：

- 解释和应用解决方案选项、架构策略、适当的实践以及技术可能性，以实现软件系统的重要质量需求（对于嵌入式系统或信息系统不同）
- 识别并沟通此类解决方案及其相关风险之间可能的权衡

LG 2-9：设计和定义接口(R1-R3)

软件架构师知道接口的重要性。他们能够设计或指定架构构建块之间的接口，以及系统与系统外部元素之间的外部接口。

他们了解：

- 接口的所需特性，并可在设计中使用：
  - 易于学习、易于使用、易于扩展
  - 难以误用
  - 从用户或使用它们的构建块的角度来看，功能是齐全的。
- 区别对待内部和外部接口的必要性
- 实现接口的不同方法（R3）：
  - 面向资源的方法（REST，Representational State Transfer）
  - 面向服务的方法（请参阅基于WS-*/SOAP-基于web的服务）

LG 2-10：了解软件部署的基本原则 (R3)

软件架构师：

- 了解软件部署是向用户提供可用的新软件或更新软件的过程
- 能够命名和解释软件部署的基本概念，例如：
  - 自动化部署
  - 可重复构建
  - 一致的环境（例如使用不可变的一次性基础设施）
  - 将所有内容置于版本控制之下
  - 发布容易被撤消

参考资料

[Bass+ 2021], [Fowler 2002], [Gharbi+2020], [Gamma+94], [Hohpe+2004], [Martin 2003], [Buschmann+ 1996], [Buschmann+ 2007], [Starke 2020], [Lilienthal 2019], [Lorz+2021]

3. 软件架构的规格说明与沟通

时长：180 分钟	练习：60分钟

相关术语

(Architectural) Views（（架构）视图）；structures（结构）；(technical) concepts（(技术)概念）；documentation（文档）；communication（沟通）；description（描述）；stakeholder-oriented（面向利益相关方）；meta structures and templates for description and communication（用于描述和交流的元结构和模板）；system context（系统上下文）；building blocks（构建块）；building-block view（构建块视图）；runtime view（运行时视图）；deployment view（部署视图）；node（节点）；channel（通道）；deployment artifacts（部署工件）；mapping building blocks onto deployment artifacts（将构建块映射到部署工件上）；description of interfaces and design decisions（接口和设计决策的描述）；UML, tools for documentation（UML，文档工具）

学习目标

LG 3-1：解释并考虑技术文档的需求（R1）

软件架构师了解技术文档的需求，在记录系统时可以考虑并满足这些需求：

- 可理解性、正确性、效率、适当性和维护性
- 为利益相关方量身定制的形式、内容和详细程度

他们知道，只有目标受众才能评估技术文件的可理解性。

LG 3-2：描述和沟通软件架构（R1-R3）

软件架构师使用文档来支持系统的设计、实现和进一步开发（也称为维护或演进）（R2）

软件架构师能够（R1）：

- 为相应的利益相关方记录和沟通架构，从而针对不同的目标群体，例如管理层、开发团队、质量保证-QA、其他软件架构师，以及可能的其他利益相关方
- 整合和协调不同发起方群体的风格和内容
- 制定和实施措施，支持书面和口头沟通的融合，并适当平衡两者

软件架构师知道（R1）：

- 基于模板的文档的好处
- 文档的各种属性取决于系统的特定属性、其需求、风险、开发过程、组织或其他因素。

例如，软件架构师可以根据情况调整以下文档特征（R3）：

- 所需文件的数量和详细程度
- 文件格式
- 文件的可访问性
- 文件形式（例如，符合元模型的图表或简单图纸）

- 文件的正式评审和签字过程

LG 3-3：解释并应用符号/模型来描述软件架构 (R2-R3)

软件架构师至少了解以下UML（参见[UML]）图来描述架构视图：

- 类、封装/包、组件（所有 R2）和复合结构图（R3）
- 部署图（R2）
- 序列图和活动图（R2）
- 状态机图（R3）

软件架构师了解UML图的替代符号，例如：（R3）

- Archimate，参见【Archimate】
- 用于运行时视图，例如流程图、编号列表或业务过程建模符号（BPMN）。

LG 3-4：解释和使用架构视图 (R1)

软件架构师可以使用以下架构视图：

- 上下文视图
- 构建块或组件视图（软件构建块的组成）
- 运行时视图（动态视图、运行时软件构建块之间的交互、状态机）
- 部署视图（硬件和技术基础设施以及软件构建块到基础设施的映射）

LG 3-5：解释和应用系统上下文视图 (R1)

软件架构师能够：

- 描述系统上下文，例如以带有解释的上下文图的形式
- 在上下文视图中表示系统的外部接口
- 区分业务和技术上下文。

LG 3-6：记录和沟通横切概念 (R2)

软件架构师能够充分地记录和沟通典型的横切（跨领域）概念（同义词：原则、观点），例如，持久化、工作流管理、UI、部署/集成、日志记录。

LG 3-7：描述接口(R1)

软件架构师能够描述和指定内部和外部接口。

LG 3-8：解释和记录架构决策 (R1-R2)

软件架构师能够：

- 系统地采取、论证、沟通和记录架构决策
- 识别、沟通和记录设计决策之间的相互依赖关系

软件架构师了解架构决策记录（ADR，见[Nygard 2011]），并可以将其应用于记录决策（R2）。

LG 3-9：了解文档化的其他资源和工具 (R3)

软件架构师了解：

- 用于描述软件架构的几个已发布框架的基础知识，例如：
  - ISO/IEEE-42010（以前为1471），参见[ISO 42010]
  - arc42，参见[arc42]
  - C4，参见[Brown]
  - FMC，参见[FMC]
- 软件架构的创建、文档编制和测试的检查表的想法和示例
- 创建和维护架构文档的可能工具

参考资料

[arc42], [Archimate], [Bass+ 2021], [Brown], [Clements+ 2010], [FMC], [Gharbi+2020], [Lorz+2021], [Nygard 2011], [Starke 2020], [UML], [Zörner 2021]

4. 软件架构和质量

时长：60分钟	练习：60分钟

相关术语

Quality（质量）；quality characteristics (also called quality attributes)（质量特性（也称作质量属性））；DIN/ISO 25010（德国工业标准/国际标准组织 25010）；quality scenarios（质量场景）；quality tree（质量树）；trade-offs between quality characteristics（质量特性之间的权衡）；qualitative architecture analysis（定性架构分析）；metrics and quantitative analysis（度量和定量分析）

学习目标

LG 4-1：讨论质量模型和质量特性（R1）

软件架构师可以解释：

- 质量和质量特性的概念（基于例如[ISO 25010]或[Bass+2021]）
- 通用质量模型（如[ISO 25010]、[Bass+2021]或[Q42]）
- 质量特性的相关性和权衡，例如：
  - 可配置性与可靠性
  - 内存需求与性能效率
  - 信息安全与易用性
  - 运行时灵活性与维护性。

LG 4-2：明确软件架构的质量需求（R1）

软件架构师可以：

- 明确和制定要开发的软件及其架构的具体质量需求，例如以场景和质量树的形式
- 解释并应用场景和质量树。

LG 4-3：软件架构的定性分析(R2-R3)

软件架构师：

- 了解软件架构定性分析的方法（R2），例如ATAM 规定的方法（R3）；
- 能定性分析较小的系统（R2）
- 应知晓以下信息来源有助于架构的定性分析（R2）：
  - 质量需求，例如以质量树和场景的形式
  - 架构文档
  - 架构和设计模型
  - 源代码

- 度量
- 系统的其他文档，如需求、操作或测试文档。

LG 4-4：软件架构的定量评估 (R2)

软件架构师知道软件定量评估和评价的方法。

他们了解：

- 定量评估有助于识别系统内的关键部分
- 进一步的信息有助于评估架构，例如：
  - 需求和架构文档
  - 源代码和相关度量，如代码行、（圈）复杂性、向内（inbound）和向外（outbound）依赖关系
  - 源代码中的已知错误，尤其是错误集群
  - 测试用例和测试结果。
- 将度量作为目标会导致其无效，如古德哈特(Goodhart)定律所述（R3）

参考资料

[Bass+ 2021], [Clements+ 2002], [Gharbi+2020], [Lorz+2021], [Martin 2003], [Starke 2020]

5. 软件架构的示例

时长：90 分钟	练习：无

本节与考试无关。

学习目标

LG 5-1：了解需求、约束和解决方案之间的关系 (R3)

软件架构师应识别和理解需求和约束以及至少一个示例采用的解决方案之间的相关性。

LG 5-2：了解解决方案技术实施的基本原理(R3)

软件架构师应至少了解一个解决方案的技术实现（实施、技术概念、使用的产品、架构决策、解决方案策略）。

参考资料

- [arc42] arc42, the open-source template for software architecture communication, online: https://arc42.org. Maintained on https://github.com/arc42
- [Archimate] The ArchiMate® Enterprise Architecture Modeling Language, online: https://www.opengroup.org/archimate-forum/archimate-overview
- [Bass+ 2021] Len Bass, Paul Clements, Rick Kazman: Software Architecture in Practice. 4th Edition, Addison Wesley 2021.
- [Brown] Simon Brown: The C4 model for visualising software architecture. https://c4model.com https://www.infoq.com/articles/C4-architecture-model.
- [Buschmann+ 1996] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal: Pattern-Oriented Software Architecture (POSA): A System of Patterns. Wiley, 1996.
- [Buschmann+ 2007] Frank Buschmann, Kevlin Henney, Douglas C. Schmidt: Pattern-Oriented Software Architecture (POSA): A Pattern Language for Distributed Computing, Wiley, 2007.
- [Clements+ 2002] Paul Clements, Rick Kazman, Mark Klein: Evaluating Software Architectures. Methods and Case Studies. Addison Wesley, 2002.
- [Clements+ 2010] Paul Clements, Felix Bachmann, Len Bass, David Garlan, David, James Ivers, Reed Little, Paulo Merson and Robert Nord. *Documenting Software Architectures: Views and Beyond*, 2nd edition, Addison Wesley, 2010
- [Cloud-Native] The Cloud Native Computing Foundation, online: https://www.cncf.io/
- [Evans 2004] Eric Evans: *Domain-Driven Design: Tackling Complexity in the Heart of Software,* Addison- Wesley, 2004.
- [FMC] Siegfried Wendt: Fundamental Modeling Concepts, online: http://www.fmc-modeling.org/
- [Ford 2017] Neil Ford, Rebecca Parsons, Patrick Kua: Building Evolutionary Architectures: Support Constant Change. OReilly 2017
- [Fowler 2002] Martin Fowler: Patterns of Enterprise Application Architecture. (PoEAA) Addison- Wesley, 2002.
- [Gharbi+2020] Mahbouba Gharbi, Arne Koschel, Andreas Rausch, Gernot Starke: Basiswissen Softwarearchitektur. 4. Auflage, dpunkt Verlag, Heidelberg 2020.
- [Geirhos 2015] Matthias Geirhos. Entwurfsmuster: Das umfassende Handbuch (in German). Rheinwerk Computing Verlag. 2015
- [Gamma+94] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. 1994.
- [Goll 2014] Joachim Goll: Architektur- und Entwurfsmuster der Softwaretechnik: Mit

lauffähigen Beispielen in Java (in German). Springer-Vieweg Verlag, 2. Auflage 2014.

- [Hofmeister et. al 1999] Christine Hofmeister, Robert Nord, Dilip Soni: *Applied Software Architecture*, Addison-Wesley, 1999
- [Hohpe+2004] Hohpe, G. and WOOLF, B.A.: *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley Professional, 2004
- [ISO 42010] ISO/IEC/IEEE 42010:2011, Systems and software engineering — Architecture description, online: https://www.iso-architecture.org/ieee-1471/
- [ISO 25010] ISO/IEC DIS 25010(en) Systems and software engineering — Systems and software
- Quality Requirements and Evaluation (SQuaRE) — Product quality model. Terms and definitions online: https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:dis:ed-2:v1:en
- [iSAQB® References] Gernot Starke et. al. Annotated collection of Software Architecture References, for Foundation and Advanced Level Curricula. Freely available https://leanpub.com/isaqbreferences.
- [Keeling 2017] Michael Keeling. Design It!: From Programmer to Software Architect. Pragmatic Programmer.
- [Lange 2021] Kenneth Lange: The Functional Core, Imperative Shell Pattern, online: https://www.kennethlange.com/functional-core-imperative-shell/
- [Lilienthal 2019] Carola Lilienthal: Langlebige Softwarearchitekuren. 3. Auflage, dpunkt Verlag 2019.
- [Lilienthal 2019] Carola Lilienthal: Sustainable Software Architecture: Analyze and Reduce Technical Debt. dpunkt Verlag 2019.
- [Lorz+2021] Alexander Lorz, Gernot Starke: Software Architecture Foundation, CPSA® Foundation® Exam Preparation. Van Haaren Publishing, 2021. Alexander Lorz, Gernot Starke
- [Martin 2003] Robert Martin: Agile Software Development. Principles, Patterns, and Practices. Prentice Hall, 2003.
- [Martin 2017] Robert Martin. Clean Architecture: A craftsman's guide to software structure and design. Pearson, 2017.
- [Miller et. al] Heather Miller, Nat Dempkowski, James Larisch, Christopher Meiklejohn: Distributed Programming (to appear, but content-complete) https://github.com/heathermiller/dist-prog-book.
- [Newman 2015] Sam Newman. Building Microservices: Designing Fine-Grained Systems. O'Reilly. 2015.
- [Nygard 2011] Michael Nygard: Documenting Architecture Decision. https://cognitect.com/blog/ 2011/11/15/documenting-architecture-decisions. See

also https://adr.github.io/

- [Nygard 2022] Michael Nygard: CUPID - for joyful coding. See https://dannorth.net/2022/02/10/cupid- for-joyful-coding/.
- [Pethuru 2017] Raj Pethuru et. al: Architectural Patterns. Packt 2017.
- [Q42] arc42 Quality Model, online: https://quality.arc42.org.
- [Starke 2020] Gernot Starke: Effektive Softwarearchitekturen - Ein praktischer Leitfaden (in German).
- 9. Auflage, Carl Hanser Verlag 2020. Website: https://esabuch.de
- [Eilebrecht+2019] Karl Eilebrecht, Gernot Starke: Patterns kompakt: Entwurfsmuster für effektive Software-Entwicklung (in German). 5th Edition Springer Verlag 2019.
- [UML] The UML reading room, collection of UML resources https://www.omg.org/technology/ readingroom/UML.htm. See also https://www.uml-diagrams.org/.
- [vanSteen+Tanenbaum] Andrew Tanenbaum, Maarten van Steen: Distributed Systems, Principles and Paradigms. https://www.distributed-systems.net/.
- [Yorgey 2012] Brent A. Yorgey, Monoids: Theme and Variations. Proceedings of the 2012 Haskell Symposium, September 2012 https://doi.org/10.1145/2364506.2364520
- [Zörner 2021] Stefan Zörner: Softwarearchitekturen dokumentieren und kommunizieren. 3. Auflage, Carl Hanser Verlag, 2021.