



iSAQB®工作组

# 软件架构 术语

版次2022



# iSAQB® 软件架构术语表

2022. 1-EN-20221216

中文版本号：1.0



目录

介绍..... 5

    Personal Comments 个人评论..... 5

    Terms Can Be Referenced 术语可被引用..... 5

    License 许可证..... 6

    Acknowledgements 致谢..... 6

    Contributing 贡献..... 6

Terms 术语..... 7

    A..... 7

    B..... 13

    C..... 14

    D..... 20

    E..... 22

    F..... 23

    G..... 24

    H..... 24

    I..... 25

    J..... 27

    K..... 27

    L..... 27

    M..... 28

    N..... 30

    O..... 30

    P..... 31

    Q..... 33

    R..... 36

    S..... 39

    T..... 45

    U..... 47

    V..... 48

    W..... 48

    X..... 49

    Y..... 49

    Z..... 49

Translation Tables 翻译表 ..... 50

    English to German 英德中对照 ..... 50

    German to English 德英中对照 ..... 56

References and Resources 参考文献与资源 ..... 62

Appendix 附录 ..... 65

    The iSAQB® e. V. Association 国际软件架构认证委员会 (iSAQB®) ..... 65

About the Authors 关于作者 ..... 66

关于我们的事业 ..... 69

# 介绍

本书包含软件架构术语的词汇表。

它可以作为准备 iSAQB® e. V. 考试认证软件架构专业人士 - 基础级®的参考资料。

注意：本词汇表并非作为软件架构的入门书或教材，仅是定义以及更广泛信息链接的集合。

此外，您还可以找到关于 iSAQB® 术语的 [翻译建议](#)，目前涉及英语与德语、中文（及互译）。

最后，本书包含了大量对书籍和 [其他资源](#) 的引用，其中部分我们已在定义中引用。



本书正在制作中。

任何错误或遗漏可以在我们的 [GitHub](#) 问题跟踪器中报告，作者在那里维护本书的原始来源。

## Personal Comments 个人评论

本书中包含的许多术语被若干作者进行了评论：



评论 (Gernot Starke)

某些术语可能特别重要，有时可能涉及一些细微差别。此类评论仅提供个人观点，并不一定反映 iSAQB® 的立场。

## Terms Can Be Referenced 术语可被引用

词汇表中的所有术语都有唯一的 URL，指向书籍的（免费）在线版本，因此它们可以被普遍引用，无论是来自在线文档还是印刷品。

我们的 URL 方案非常简单：

- 基本 URL 是 <https://public.isaqb.org/glossary/glossary-en.html>
- 我们在要引用的术语前加上前缀 #term-，其后是术语本身，用连字符（“-”）代替空格。

例如，我们的软件架构术语描述可以通过以下链接引用（超链接）：

<https://public.isaqb.org/glossary/glossary-en.html#term-software-architecture>

几乎所有术语都是用它们的全名进行超链接的，只有极少数例子是通过它们的（常见）缩写来引用的，比如 UML 或 DDD。

## License 许可证



本书采用知识共享署名4.0国际许可协议授权以下仅为简要概述，并不替代正式许可协议。

知识共享署名4.0许可意味着您可以：

- 分享 — 以任何媒介或格式复制和重新分享材料。
- 改编 — 对材料进行混编、转换以及基于商业用途在内的任何目的的构建。
- 只要您遵守许可条款，许可方不能撤销这些权利。

您必须：

- 给予适当的信用。
- 提供指向许可证的链接 (<https://creativecommons.org/licenses/by/4.0/>)。
- 如果（以及哪些）对原始作品进行更改，应指明。

## Acknowledgements 致谢

本词汇表的许多部分由以下志愿者和赞助商贡献（除众多作者之外）。

- 大约120个术语的定义由Gernot Starke贡献，这些定义最初是为他的著作之一编译的。
- 系统改进和演化上下文中的许多定义由aim42开源项目贡献。

## Contributing 贡献



### 欢迎贡献

如果您发现错误、遗漏或错别字，或者想要增加新内容，可以采用以下几种方法：

1. 在我们的GitHub仓库中开启一个issue。
2. fork仓库并创建一个pull请求。
3. 给作者写一封电子邮件。

非常感谢您的参与。

# Terms 术语

## A

### Abstraction 抽象

去除细节以关注更重要方面的过程。本质上类似于泛化的过程。  
一种关注与特定目的相关的信息，而忽略其他额外信息的元素视图。  
一个设计构造，如“构建块应该依赖于抽象而不是实现”。

### Abstractness 抽象性

面向对象系统源代码的度量：抽象类型（接口和抽象类）的数量除以类型的总数。

### Accessibility Quality Attribute 易访问性质量属性

在指定的使用周境中，为达到指定的目标，产品或系统被具有最广泛的特征和能力的个体所使用的程度。它是[易用性](#)这一质量属性的一个子特性。有关更多信息，请参考[ISO 25010](#)网站。

### Accountability Quality Attribute 可核查性[H1] 质量属性

实体的活动可以被唯一地追溯到该实体的程度。它是[信息安全性](#)这一质量属性的一个子特性。有关详细信息，请参考[ISO 25010](#)网站。

### Accreditation 认证

由授权的认证机构（这里指iSAQB®）进行的确定程序和鉴定，确认申请人满足作为[培训提供商](#)的组织、技术和质量要求。

### Accreditation Body 认证机构

[申请资格认证](#)必须通过iSAQB®指定的认证机构提交。认证机构是培训提供商在认证过程中所有问题的联系点。它协调认证程序，对提交的文件进行正式评估，并在[审计工作小组](#)中组织技术评估。

### Accredited Training Provider 经认证的培训提供商

拥有由iSAQB®颁发的有效认证的[培训提供商](#)。

### ACL 访问控制列表

访问控制列表（Access Control Lists）用于控制[主体](#)对特定实体的授权访问。附属于某个[实体](#)的ACL列出了主体及其访问权限。许多文件系统，包括Windows和POSIX文件系统，都支持ACL来控制访问。

[\[H1\]](#)GB25000.10中叫可核查性。

由于ACL在大规模基础上扩展性不佳，因此通常基于角色（[RBAC](#)）来构建访问控制模型。

### Acyclic Dependencies Principle 无环依赖原则

用于设计软件系统结构的一种基本原则（参见[包原则](#)）。其要求一个系统的依赖图中无环，这也是[层次分解](#)的必要条件。

避免依赖环对[低耦合](#)和[维护性](#)是必不可少的，如果一个依赖环中的所有组件实际上（即使是间接地）依赖彼此，这使得单独理解、更改或替换循环中的任何部分都变得困难（参见[Lilienthal-2019]）。

尽管直到Robert C. Martin([Martin-2003])才进行“就面向对象软件的大量组件而言ADP是通用原则”的表述，但至少可追溯到一份早期关于软件架构的文章——1972年的经典论文“用于系统分解为模型的标准”([Parnas-1972])——就总结出了“干净的”分解的层次结构是任何系统理想的属性。可以说，甚至在考虑其各种的实际问题前，依赖环就已经在逻辑上含有[循环论证](#)或者[循环定义](#)的缺陷。基于结构化的矛盾，环对于系统的固有性质和目的而言，是不适当且无意义的模型。概念上的分歧势必引起[原则性](#)方法完全防范的（无法预期的）问题。

### Adaptability Quality Attribute 适应性质量属性

产品或系统能够有效地、有效率地适应不同的或演变的硬件、软件、或者其他运行（或使用）环境的程度。是[可移植性](#)的子特性。参考ISO 25010 网址。

### Adapter 适配器

适配器是一种设计模式，允许另一个接口使用现有组件的接口。常用于现有组件与其他组件的协作，无需修改双方源码。

### Aggregate 聚合

聚合是[领域驱动设计](#)的构建块。聚合是[实体](#)和[值对象](#)组成的复杂对象结构。每个聚合都有一个根实体，在更改时被视为一个单元。聚合确保了其所包含实体的一致性和完整性。

### Aggregation 聚合关系

面向对象编程中的一种对象[组合](#)的形式。与组合关系不同，聚合关系不意味着所有权。当元素被销毁时，所含有的元素会保持完整。

### Analysability Quality Attribute 易分析性质量属性

可以评估预期变更（变更产品或系统的一个或多个部分）对产品或系统的影响、诊断产品的缺陷或失效原因、识别待修改部分的有效性和效率的程度。是[维护性](#)的一个子特性。参见ISO 25010网址。

### Appropriateness 适应性

（同义词：充分性）对特定目的的适用性。

### Appropriateness Recognizability Quality Attribute 可辨识性质量属性



用户能够辨识产品或系统是否适合他们的要求的程度。是[易用性](#)的一个子特性。参见[ISO 25010](#)网址。

## arc42

用于软件架构沟通和文档化的免费、开源[模板](#)。arc42包含12个（可选）部分或章节。

## Architectural (Architecture) Pattern 架构模式

“架构模式描述了软件系统的一种基本的架构组织范式。它提供一组预定义子系统，详细说明其任务，且包含组织子系统之间关系的规则和指南。” ([[Buschmann+1996](#)], page 12)，与[架构样式](#)类似。

例如包含：

- 层
- 管道过滤器
- 微服务
- [命令查询职责分离](#) (Command Query Responsibility Segregation, CQRS)

## Architectural Decision 架构决策

对架构具有持续的或本质的影响的决策。

例如：关于数据库技术或用户界面技术基础的决策。

遵循ISO/IEC/IEEE 42010，架构决策涉及系统关注点。然而，两者之间经常没有简单映射。决策能够通过几种方式影响架构方式。这些可以反映在架构描述（同ISO/IEC/IEEE 42010的定义）中。

## Architectural Tactic 架构策略

帮助实现一个或多个质量需求的技术、策略、方法或决策。术语由[[Bass et al. 2022](#)]创造。

## Architecture 架构

见[软件架构](#)。

## Architecture Description 架构描述

用于表达架构的工作产品（同ISO/IEC/IEEE 42010的定义）。

## Architecture Description Element 架构描述元素

架构描述元素是架构描述的任意构建块。架构描述元素是在ISO/IEC/IEEE 42010中讨论的最原始的构建块。ISO/IEC/IEEE 42010中定义的所有术语是架构描述元素（同ISO/IEC/IEEE 42010的定义）的专业化概念。

## Architecture Description Language 架构描述语言

架构描述语言（ADL）架构描述语言是架构描述中使用的任何一种表达形式（同ISO/IEC/IEEE 42010定义）。

例如 Rapide, Wright, SysML, ArchiMate和RM-ODP的视点语言[ISO 10746]。

### Architecture Evaluation 架构评价

软件或系统架构的定量或定性评估。确定架构是否能够实现其目标质量或质量属性。

参见[评估](#)。



评论（Gernot Starke）：在我看来，术语“架构分析”或“架构评估”更适合，由于评价包含值，意味着数值评估或度量，而这些仅是进行架构分析时应做的一部分。

### Architecture Framework 架构框架

用于描述在特定应用领域和/或利益相关方团体建立的架构的约定、原则和实践（同ISO/IEC/IEEE 42010定义）。

例如：

- 通用企业参考架构和方法论（GERAM）[ISO 15704]是一种架构框架。
- 开放分布式处理参考模型（RM-ODP）[ISO/IEC 10746]是一种架构框架。

### Architecture Goal 架构目标

（同义词：架构质量目标，架构质量需求）：系统需要实现的质量属性和被理解为架构问题的质量属性。

因此，架构需要以一种满足架构目标的方法进行设计。与（短期）项目目标相比，这些目标通常具有长期特征。

### Architecture Model 架构模型

架构视图由一个或多个架构模型组成。架构模型使用建模协定适应于将被处理的业务。这些协定由管理该模型的模型类指定。在架构描述中，一个架构模型可以属于多个架构视图（同ISO/IEC/IEEE 42010定义）。

### Architecture Objective 架构对象

参见[架构目标](#)。

### Architecture Quality Requirement 架构质量需求

参见[架构目标](#)。

### Architecture Rationale 架构基本原理

架构基本原理记录了对已做的架构决策的解释、正当理由或推理。决策的理由可以包含决策的基础，考虑的替代和权衡，决策的潜在后果和附加信息来源的引用（同ISO/IEC/IEEE 42010定义）。

### Architecture Style 架构风格

元素和关系类型及其使用约束的描述。常称为[架构模式](#)。例如：管道过滤器、模型视图控制器、层。



评论 (Alexander Lorz):

取决于你询问的对象，有人可能认为架构风格是架构模式的泛化。也就是说，“分布式系统”是一种风格，而“客户端服务器，CQRS，代理和对等”是属于这种风格的更特定的模式。但是，从实用性视角来看，这些区别不是必需的。

## Architecture View 架构视图

从特定的角度表示一个系统。重要的和众所周知的视图如下：

- [上下文视图](#)
- 构建块视图
- 运行时视图
- 部署视图

[\[Bass et al. 2022\]](#)和[\[Rozanski & Woods 2011\]](#)广泛地讨论这个概念。

遵循ISO/IEC/IEEE 42010，架构视图是从特定系统关注点的视角表达系统架构的工作产品（同ISO/IEC/IEEE 42010定义）。

## Architecture Viewpoint 架构视角

创建、解读和使用架构视图以框定（一个或多个）特定系统关注点的一组约定（同ISO/IEC/IEEE 42010定义）。

## Artifact 工件、制品

在软件开发过程中创造或产生的有形的副产品。例如，用例、各种图表、UML模型、需求和设计文档、源代码、测试用例、类文件、档案。

## Assessment 评估

参见[评估](#)。

收集系统有关状态、风险或漏洞的信息。评估可能关注所有方面（开发、组织、架构、代码等）。

## Asset 资产

在信息安全、计算机安全和网络安全领域，资产是任何数据、设备或其他环境组件用以支持相关信息活动。资产通常包括硬件（如服务器和交换机），软件（如关键任务应用和支持系统）以及机密信息“。

（引用自[维基百科](#)）

## Association 关联

定义对象（通常组件之间）之间关系。每个关联能够由基数和（角色）名称 进行详细描述。

参见[耦合](#)，[依赖](#)和[关系](#)。

### Asymmetric Cryptography 非对称密码学

非对称密码学算法设计为用于加密和解密的密钥是不同的。用于加密的密钥称为“公钥”，用于解密的密钥称为“私钥”。任何人发布和使用公钥对信息进行加密，只有拥有私钥的组织才能对读取的信息进行解密。参见[第17页](#)。

非对称密码学是PKI(公钥基础设施)和数字签名的基础。

### ATAM

架构权衡分析法。基于（分层）质量树和具体的质量场景的定性的架构评价方法。基本理念：将细颗粒度的质量场景（“[质量需求](#)”）与相应的架构方法进行比较，以识别风险和权衡。

### Attack Tree 攻击树

描述攻击者达到确定目标的不同路径的形式化方式。树的结构通常以攻击目标作为根节点，不同路径作为子节点。每个路径可能有再次被列为子节点的依赖项。可以通过给每个节点分配附加属性来分析确定方式攻击IT系统的可能性。例如通过参考对策可以估算攻击成本或者攻击路径是否可行。参见[Bruce Schneier “安全威胁建模”](#)。

### Audit Working Group 审计工作组：

审计工作组负责培训材料的技术评价以及培训课程的监测和评价。审计工作组成员由iSAQB®授权并独立于[培训提供方](#)。评估结论（审计工作组的各种认证提议）由[认证机构](#)传达给[培训提供方](#)。

### Authentication 身份认证

身份认证是对给定实体声明的身份进行确认的过程。通常是通过验证至少一个系统已知的身份认证因素来完成的：

- 知识（例如，密码）
- 所有权（例如，安全令牌）
- 固有属性（例如，生物特征识别）

通过请求多个因素或者至少两个类型的因素来增强身份认证。

### Authenticity Quality Attribute 真实性质量属性

对象或资源的身份标识能够被证实符合其声明的程度。是[信息安全性](#)的一个子特性。参考[ISO 25010](#)网址。

### Authorization 授权

“授权或许可的作用是详细说明涉及信息安全和计算机安全资源的通用访问权限及特定访问控制，更正式的说法，”授权“定义访问策略”。

（引用自[维基百科](#)）

### Availability 可用性

基本[信息安全目标](#)之一，描述系统在需要时能够提供所需信息。从信息安全角度举例，拒绝服务攻击可能会妨碍可用性。

### Availability Quality Attribute 可用性质量属性

系统、产品或组件在需要使用时可操作和可访问的程度。是[可靠性](#)的一个子特性。参考[ISO 25010](#)网址。

## B

### Black Box 黑盒

查看隐藏内部结构的[构建块](#)（或[组件](#)）。黑盒遵守[信息隐藏](#)原理。明确定义输入和输出接口以及精确制定任务或目标。（可选）黑盒定义一些质量属性，例如计时行为、吞吐量或安全方面。

### Bottom-Up Approach 自下而上方法

建模和设计工作（或过程策略）方向。从详细的或具体的事物开始朝向更通用的或抽象的。

“在自下而上方法中，首先对系统单个基础元素进行详尽说明。然后将这些元素连接在一起形成更大的子系统。”（引用自[维基百科](#)）

### Bounded Context 限界上下文

限界上下文是[领域驱动设计](#)的策略设计原则。“限界上下文明确地定义了软件系统适用的[领域模型](#)中的上下文。理论上，相同领域的所有软件系统有一个单一、统一的模型更可取。这是一个崇高的目标，实际上通常会分割成多个模型。认识生活事实有助于与之共事。”（引用自[维基百科](#)）

“多个领域模型在任何大型项目中发挥作用。然而，基于不同模型的代码组合在一起时，软件变得缺陷多、不可靠且难以理解。组员之间的沟通变得混乱。常常不清楚什么情况下模型不适用。因此，在团队组织结构、应用特定部分的用途和物理表现（如代码库和数据库模式）方面显性地设置限界。严格维持这些限界内模型的一致性，但不要被外部问题分散或混淆。”（引用自[维基百科](#)）

### Bridge 桥接

设计模式中抽象从实现中解耦，以便于两者可以独立变化。假如你发现这些难于理解（像大多数人一样）——[看这里](#)。

### Broker 代理

一种架构模式用于采用解耦组件来构造分布式软件系统，由（通常是远程）服务调用进行组件间交互。

代理负责协调通信，例如转发请求，以及传输结果和异常。

### Building Block 构建块

用于构建软件的各种制品的通用或抽象术语。软件架构的静态结构（[构建块视图](#)）的一部分。构建块可以是分层结构的，可能包含其他（更小的）构建块。构建块的替代（具体的）名称：组件、模块、包、命名空间、类、文件、程序、子系统、函数、配置、数据定义。

### Building Block View 构建块视图

展示系统的静态结构，以及源码如何进行组织。通常一个分层方式，从[上下文视图](#)开始。一个或多个[运行时视图](#)补充。

### Business Architecture 业务架构

提供组织的共同理解，并用于调整战略目标和战术需求的企业蓝本。

### Business Context 业务上下文

从业务角度将完整系统包括其环境展示为一个[黑盒](#)，其包括所有通讯方（用户、IT系统等）并阐明领域特定输入和输出或接口的详细说明。注意，与外部参与者交互的特定技术解决方案由于其受制于[技术上下文](#)通常应从业务上下文中忽略。

参见[上下文视图](#)。

## C

### C4 Model C4模型

[软件架构文档的C4模型](#)由Simon Brown开发，包括一组分层的软件架构框图，用于上下文、容器、组件和代码。C4框图的层次结构提供了不同的抽象级别，每个等级与不同的受众有关。

## CA

证书颁发机构给指定主体颁发[PKI](#)数字证书。通常与此颁发机构建立信任使得颁发的证书具有相同的信任等级。

一个例子是广泛使用的TLS-PKI，其中每个浏览器包括已定义的CAs列表的根证书。这些CAs接着检查指定的网络域名所有者的身份，并使用[TLS](#)对其证书进行数字签名。

### Capacity Quality Attribute 容量质量属性

产品或系统参数最大限度满足需求的程度。是[性能效率](#)的一个子特性。参考[ISO 25010](#)网址。

### Cardinality 基数

描述关联或关系的定量评级。其指定关联的参与者（对象、实体、模块等）数量。

### Certification Program 认证程序

iSAQB®CPSA®认证程序包括组织结构、文档（培训文档、合同）和过程。

CPSA®版权的缩写和术语是指软件架构认证专家。

### CIA Triad CIA三要素

参见[安全目标](#)。

### Cloud 云

“云计算是一种模型，用于实现对可配置的计算资源（例如网络、服务器、存储器、应用和服务）共享池无处不在的、方便的、按需的网络访问，且资源可以在最小程度的管理影响或服务提供交互影响下快速提供和释放。”

定义引用自[NIST](#)（美国国家标准与技术研究院）。

NIST 定义包含以下五个特征（引用但是缩写也来自上述NIST源）：

- 按需自助服务：消费者可以单方面地提供计算能力，例如服务器时间和网络存储，而无需与每个服务提供商进行人机交互。
- 宽泛的网络访问：计算能力可通过网络获得，通过标准机制进行访问，这些机制促进异构客户端平台的使用。
- 资源池化：提供方的计算资源池化，采用多租户模型，根据消费者需求动态分配和重新分配不同物理的和虚拟的资源，来服务多种多样的消费者。
- 快速弹性：计算能力有时可以自动弹性提供和释放，以便随需求快速缩放。对消费者而言，所提供的计算能力通常似乎是无限的，且可以随时以任何数量进行占用。
- 精确服务：云系统通过杠杆化适合于服务类型（例如，存储，处理，贷款和激活用户账户）的一些抽象级别的计量能力，来自动控制和优化资源使用。可以监视、控制和报告资源利用率，对已使用服务的提供方和消费者提供透明度。

### Co-Existence Quality Attribute 共存性质量属性

在与其他产品共享通用的环境和资源的条件下，产品能够有效执行其所需的功能并且不会对其他产品造成负面影响的程度。是[兼容性](#)的一个子特性。参考[ISO 25010](#)网址。

### Cohesion 内聚

构建块、组件或模型的元素在一起的程度称为[内聚](#)。其衡量给定组件的各功能之间关系的强度。在内聚系统中，功能密切相关。通常以“高内聚”或“低内聚”作为特征描述。力求高内聚，因高内聚常意味着可重用性，低耦合和易理解性。

### Command 命令模式

这种设计模式的对象用于封装一个动作。这个动作可能在后续调用或执行。

### Common Closure Principle 共同封闭原则

设计软件系统结构的基本原则（参见[包原则](#)）。其对较大组件直接明确重申[单一职责原则](#)。



组件的子组件理论上应有确切且一致的理由才进行变更。影响其中一个子组件的变更请求将影响所有子组件，但不能影响封闭组件以外的任何其他内容。

因此，每个预期的变更请求将影响最小数量的组件。或者换种说法：每个组件在最大数量预期变更请求时将[关闭](#)。这里的术语“预期”指少数重要影响：

1. 系统的内在概念/任务比其行为的表层描述更深入。
2. 系统更深的概念/任务不完全客观，但是能用不同的方式进行建模。
3. 确定系统的概念/任务不只是被动描述，也会主动制定策略。

此原则导致高聚合组件。同样意味着低耦合组件由于相关概念变更使得绑定相同组件一同变更。当每个单独概念由一个单独组件表示时，组件间没有不必要的耦合。

### Common Reuse Principle 通用复用原则

设计软件系统结构的基本原则（参见[包原则](#)）。组件的子组件（类）应正是那些正在同时使用/复用的。或者另一种说法：正在同时使用/复用的组件应封装成一个大组件。这同样意味着不经常与其他子组件联合使用的子组件不应位于相应的组件中。

这个观点有助于决策哪些属于组件，哪些不属于组件。其目的在于系统分解成低耦合和高内聚组件。

显著回应单一职责原则。其同样回应接口隔离原则，因其确保客户端不会被迫依赖其不关心的概念。

### Compatibility Quality Attribute 兼容性质量属性

在共享相同的硬件或软件环境的条件下，产品、系统或组件能够与其他产品、系统或组件交换信息，和/或执行其所需的功能的程度。由以下子特性组成：[共存性](#)、[互操作性](#)。参考[ISO 25010](#)网址。

### Complexity 复杂度

“复杂度通常用来表征具有多个部分且部分之间通过多种方式相互作用的事物。”（引用维基百科）

- 基本复杂度是我们不得不解决的问题核心，由软件合理的难题部分组成。大多数软件难题包括一些复杂度。
- 偶然复杂度是所有没必要与解决方案直接关联，但无论如何不得不处理的事物。

（引用自 [Mark Needham](#)）

架构师应努力减少偶然复杂。

### Component 组件

参见[构建块](#)。架构的结构化元素。

### Composition 组合

组合简单的元素（例如，函数、数据类型、构建块）来构建更复杂、更强大或更重要的部分。

在UML中，当拥有的元素被销毁时，包含的元素会一并销毁。



## Concept 概念

如何解决特定问题的计划、原则或规则。

概念在某种意义上经常横跨，多个架构元素可能被单个概念影响。意味着实现单元（构建块）的实现者应遵守相应的概念。

概念形成[概念完整性](#)的基础。

## Conceptual Integrity 概念完整性

概念、规则、模式和类似解决方案方法在整个系统中以一致的（同质的、相似的）方式应用。类似的问题以相似的或相同的方法进行解决。

## Concern 关注点

对架构的关注点是利益相关方对架构的需求、对象、约束条件、意图或愿望。（引用自[Rozanski & Woods 2011], 第8章）

遵循ISO/IEC/IEEE 42010，关注点定义为，与一个或多个利益相关方有关的（或重要的）（系统）事项。

注意，关注点涉及系统在其环境中的任何影响，包括开发、技术、业务、运营、组织、政治、经济、法律、监管、生态和社会影响。

## Concurrency 并发

并发是程序、算法或问题的不同部分或单元乱序或部分顺序执行而不影响最终结果的能力。并发并不意味着并行。（引自[维基百科](#)）

## Confidentiality 保密性

基本[安全目标](#)之一，描述系统仅对授权方披露和提供信息。

## Confidentiality Quality Attribute 保密性质量属性

产品或系统确保数据只有在被授权时才能被访问的程度。是信息[安全性](#)的一个子特性。参考ISO 25010网址。

## Consistency 一致性

一个一致的系统不存在不一致。

- 相同的（至少相似的）方法解决相同的问题。
- 数据及其关系符合验证规则的程度。
- （数据库的）客户进行相同查询得到相同结果（例如，读一致性、写一致性、读写一

致性)。

- 重视行为：系统行为合乎逻辑、可复现和合理的程度。

### Constraint 约束

对您在创建、设计、实现或以其他方式提供解决方案时对自由度限制的程度。约束经常时全局需求，例如有限的开发资源或高级管理层的决策限制了规划、设计、开发或运作系统的方式。

基于[Scott Ambler](#)的定义。

### Context (of a System) (系统) 上下文

“定义系统与其环境之间的关系、依赖和交互。环境特指人、系统和外部交互实体。”（引用自[Rozanski-Woods](#)）

来自[arc42](#)的另一个定义：“顾名思义，系统范围和上下文将你的系统（也就是你的范围）与所有通讯方（相邻系统和用户，也就是你系统的上下文）进行界定。因此其指定了外部接口。”（引用自[docs.arc42.org](#)）

区分业务和技术上下文：

- **业务**上下文（过去称为逻辑上下文）从业务或非技术视角显示外部关系。其从技术、硬件或实现细节进行抽象。输入/输出关系根据其业务含义而不是其技术属性来命名。
- **技术**上下文展示技术细节，比如传输通道、技术协议、IP地址、总线或类似硬件细节。例如，嵌入式系统在开发早期经常关注硬件相关信息。

### Context View 上下文视图

从业务角度（[业务上下文](#)）和/或从技术或开发角度（[技术上下文](#)）将完整系统在其环境中展示成**黑盒**。上下文视图（或上下文图）展示系统与其环境之间的边界，展示其环境（其邻居）中与之交互的实体。

邻居可以是其他软件、硬件（如传感器）、人、用户角色、甚至是使用系统的组织。

参见[上下文](#)。

### Correspondence 对应关系

对应关系定义了架构描述元素之间的关系。对应关系用于表达架构描述内（或架构描述之间）的（已识别或命名）的关系（如ISO/IEC/IEEE 42010中定义的）。

### Correspondence Rule 对应规则

对应关系可以由对应规则来管理。对应规则用于强制执行架构描述内（或架构描述之间）的关系（如ISO/IEC/IEEE 42010中定义的）。

同义词：完整性、一致性、概念完整性。

## Coupling 耦合

**耦合**是指软件构建块之间的相互依赖关系的种类和程度；其衡量两个组件连接的紧密程度。

您应该始终追求低耦合。耦合通常与内聚相对比。低耦合通常与**高内聚**相关联，反之亦然。低耦合通常是结构良好系统的标志。

当与高内聚结合时，它支持可理解性和可维护性。

## CPSA® 认证软件架构专家

认证软件架构专家-**iSAQB®**颁发的不同级别认证的通用名称。最常见的认证级别包括基础级（CPSA-F）和高级（CPSA-A）。

## CQRS （命令查询职责分离）

命令查询职责分离（command query responsibility segregation）：将操作（命令）数据的元素与仅读取（查询）数据的元素分开。这种分离使得可以针对读写数据采用不同的优化策略。（例如，缓存只读数据比缓存也会被修改的数据要容易得多。）

关于这个主题，[Mark Nijhof](#)有一本有趣的书。

## Cross-Cutting Concept 横切概念

参见**概念**。

同义词：原则，规则。

## Cross-Cutting Concern 横切关注点

架构或系统的功能性，它影响多个元素。这类关注点的例子有日志、事务、安全、异常处理、缓存等。

通常，这些关注点将通过**概念**在系统中得到处理。

## Curriculum 课程

学校（此处：**iSAQB®**作为管理软件架构教育的机构）提供的学习过程。它包括课程内容（教学大纲）、采用的方法和其他方面。

以及与教育组织方式相关的内容，如规范、价值观，包括认证和考试。

## Cyclomatic Complexity 循环复杂度

循环复杂度是一种定量度量，表示程序源代码中独立路径的数量。它大致与代码中条件语句（**if**、**while**）的数量加1相关。没有**if**或**while**的线性语句序列的循环复杂度为1。许多软件工程师认为，更高的复杂度与缺陷的数量相关。

## D

### Decomposition 分解

（同义词：分解）将一个复杂的系统或问题分解成几个更小、更容易理解、实现或维护的部分。

### Dependency 依赖

参见[耦合](#)。

### Dependency Injection (DI) 依赖注入 (DI)

传给或通过属性设置器传递所需的依赖，而非让对象或工厂创建依赖。因此，你使得特定依赖的创建成为其他人的问题。

### Dependency Inversion Principle 依赖倒置原则

高级（抽象）元素不应依赖于低级（具体）元素。“细节应依赖于抽象”（[\[Martin-2003\]](#)）。这是SOLID原则之一，[Brett Schuchert](#)对此有很好的解释，并且与SDP和SAP密切相关。

### Deployment 部署

将软件引入其执行环境（硬件、处理器等）。使软件投入运行。

### Deployment View 部署视图

架构视图，展示了系统或工件将被部署和执行的技术基础设施。

“该视图定义了系统预期运行的物理环境，包括系统需要的硬件环境（例如，处理节点、网络连接和磁盘存储设施）、系统中每个节点（或节点类型）的技术环境要求，以及将软件元素映射到将执行它们的运行时环境。”（根据[Rozanski+2011](#)的定义）

### Design Pattern 设计模式

设计模式是针对在特定设计背景下普遍存在的问题的通用或一般的可重用解决方案。这个概念最初是由著名建筑师[Christopher Alexander](#)提出的，后来被软件工程师采纳。

在我们看来，每个认真的软件开发人员至少应该了解一些来自Erich Gamma（《[设计模式](#)》一书中的先驱[四人帮](#)）及其三位同行的设计模式。

### Design Principle 设计原则

一组指导原则，帮助软件开发人员设计和实施更好的解决方案，其中“更好”可能意味着以下一个或多个方面：

- 低[耦合](#)。
- 高[内聚](#)。

- 关注点分离或遵守单一职责原则。
- 遵守信息隐藏原则。
- 避免僵化：系统或元素难以改变，因为每一次改变都可能影响许多其他元素。
- 避免脆弱性：当元素被改变时，其他元素会出现意外结果、缺陷或其他负面后果。
- 避免不灵活性：一个元素难以复用，因为它无法与系统的其余部分分离。

## Design Rationale 设计原理

设计任何架构元素时所做决策背后的理由的明确文档记录。

## Document 文档

传达信息的（通常是书面的）工件。

## Documentation 文件编制

系统有序地收集文档和其他任何类型的材料，以便于使用或评价。其他材料的例子包括：演示文稿、视频、音频、网页、图像等。

## Documentation Build 文档构建

自动构建过程，将工件收集到一致的文档中。

## Domain Model 领域模型

领域模型是领域驱动设计（Domain-Driven Design, DDD）的概念。它提供了一套抽象，描述了领域的选定方面，并可用于解决与该领域相关的问题。

## Domain-Driven Design (DDD) 领域驱动设计 (DDD)

“领域驱动设计（DDD）是一种开发软件的方法，用于解决复杂需求，通过将实现与不断演化的核心业务概念模型深度连接。”

（引自DDDCommunity）。参见[Evans-2004]。

“另请参见： ”

- 实体 (Entity)
- 值对象 (Value Object)
- 聚合 (Aggregate)
- 服务 (Service)
- 工厂 (Factory)
- 仓库 (Repository)

- 通用语言 (Ubiquitous Language)

### Drawing Tool 绘图工具

架构文档中用于创建绘图的工具。例如：MS Visio, OmniGraffle, PowerPoint等。绘图工具将每个绘图视为一个单独的事物，在更新架构中已在多个图表中显示的元素时，会导致维护开销（与建模工具相对）。

## E

### Economicalness 经济性

以相对较低的努力实现的经济、简单、精简或可达性。

### Embedded System 嵌入式系统

嵌入在更大的机械或电气系统中的系统。嵌入式系统通常具有实时计算约束。嵌入式系统的典型特性包括低功耗、有限的内存和处理资源以及小巧的尺寸。

### Encapsulation 封装

封装有两种略有不同的概念，有时是这两种概念的组合：

- 限制对对象某些组件的访问。
- 将数据与操作该数据的方法或函数捆绑在一起。

封装是一种信息隐藏的机制。

### Enterprise IT Architecture 企业IT架构

同义词：企业架构。

整个企业的IT支持的结构和概念。企业架构的基本主题是单个软件系统，也称为“应用程序”。

### Entity 实体

实体是领域驱动设计的一个构建块。实体是业务领域的核心对象，具有不可变身份和明确定义的生命周期。实体将它们的状态映射到值对象，并且几乎总是持久的。

### Entropy 熵

在信息理论中，熵被定义为消息的“信息量”或“信息内容的不确定性”。一个密码系统的熵是通过密钥空间的大小来测量的。更大的密钥空间具有更高的熵，如果算法本身没有缺陷，那么比小密钥空间更难破解。为了进行安全的加密操作，不仅强制使用随机值作为输入，它们还应该具有高熵。在计算机系统中创建高熵是有意义的，并可能影响系统的性能。

参见[Bruce Schneier](#)和Whitewood Inc. 关于“[Understanding and Managing Entropy](#)”的文章，或者SANS关于“[Randomness and Entropy - An Introduction](#)”。

## Environment 环境

（系统）上下文确定了对系统的所有影响的设置和情况（周境）（如ISO/IEC/IEEE 42010中所定义）。

注意，一个系统的环境包括开发、技术、商业、运营、组织、政治、经济、法律、监管、生态和社会影响。

## F

## Facade 外观模式

结构设计模式。外观模式为复合的或复杂的构建块（提供者）提供简化的接口，而无需对提供者进行任何修改。

## Factory 工厂模式

（设计模式）在基于类或面向对象的编程中，工厂方法模式是一个创建性设计模式，它使用工厂方法或工厂组件来创建对象，而无需为将创建的对象指定确切的类。

在[领域驱动设计](#)中：工厂封装了[聚合](#)、[实体](#)和[值对象](#)的创建。工厂只在领域内工作，并且无法访问技术构建块（例如，一个数据库）。

## Fault Tolerance Quality Attribute 容错性质量属性

系统、产品或组件在存在硬件或软件故障的情况下，仍然能够按照预期运行的程度。它是[可靠性](#)的一个子特性。参考[ISO 25010](#)网站。

## Filter 过滤器

管道和过滤器架构风格的一部分，用于创建或转换数据。过滤器通常与其他过滤器独立执行。

## Fitness Function 适应度函数

“架构适应度函数为某些架构特性提供了一个客观的完整性评估。”（[\[Ford+2017\]](#)）。

适应度函数来源于手动评价和自动化测试，显示了架构或质量要求得到满足的程度。

## Function Signature 函数签名

（同义词：类型或方法签名）定义了函数或方法输入和输出。一个签名可以包括：

- 参数及其类型

- 返回值及其类型
- 抛出的异常或错误

#### Functional Appropriateness Quality Attribute 功能适合性质量属性

功能在促进完成指定任务和目标方面的程度。它是[功能适用性](#)的一个子特性。参考[ISO 25010](#)网站。

#### Functional Completeness Quality Attribute 功能完备性质量属性

一组功能覆盖所有指定任务和用户目标的程度。它是[功能适用性](#)的一个子特性。参考[ISO 25010](#)网站。

#### Functional Correctness Quality Attribute 功能正确性质量属性

功能集对指定的任务和用户目标的覆盖程度。它是[功能适用性](#)的一个子特性。参考[ISO 25010](#)网站。

#### Functional Suitability Quality Attribute 功能适用性质量属性

产品或系统在特定条件下提供满足明确和隐含需求的功能的程度。它由以下子特性组成：[功能完备性](#)、[功能正确性](#)、[功能适合性](#)。参考[ISO 25010](#)网站。

#### Fundamental Modeling Concepts (FMC) 基本建模概念 (FMC)

[基本建模概念 \(FMC\)](#) 是用于建模和文档化软件系统的图形表示法。从他们的网站来看：

“FMC为软件密集型系统的全面描述提供了一个框架。它基于精确的术语，并得到图形表示法的支持，这种表示法易于理解”。

## G

#### Gateway 网关

一种（设计或架构）模式：封装对（通常是外部）系统或资源的访问的元素。也参见[包装器 \(wrapper\)](#)、[适配器 \(adapter\)](#)。

#### Global Analysis 全局分析

一种系统化的方法，用于实现期望的质量属性。由Christine Hofmeister（Siemens Corporate Research）开发和记录。全局分析在[\[Hofmeister+2000\]](#)中有描述。

## H

#### Heterogeneous Architectural Style 异构架构风格

参见[混合架构风格](#)。



## Heuristic 启发式方法

非正式规则，经验法则。任何解决问题的方式，虽然没有保证是最优的，但某种程度上足够好。例如，来自[面向对象设计](#)或[用户界面设计](#)的启发式方法。

## Hybrid Architecture Style 混合架构风格

两种或多种现有架构风格或模式的组合。例如，嵌入层结构的 MVC 结构。

## I

### IEEE-1471

标准“软件密集型系统架构描述的推荐实践”，即ISO/IEC 42010:2007。它定义了一个用于软件架构描述的（抽象）框架。

## Incremental Development 增量开发

参见[迭代和增量开发](#)。

## Information Hiding 信息隐藏

软件设计的一个基本原则：持续隐藏那些可能发生改变的设计或实现决策，从而保护系统的其他部分在这些决策或实现发生变化时不被修改。这是[黑盒](#)的一个重要属性。它将接口与实现分离。

术语[封装](#)通常与信息隐藏互换使用。

## Installability Quality Attribute 易安装性质量属性

在指定环境中，产品或系统能够成功地安装和/或卸载的有效性和效率的程度。它是[可移植性](#)的一个子特性。参考ISO 25010网站。

## Integrity 完整性

有多种含义：

基本[安全目标](#)之一，意味着维护和确保数据的准确性和完整性。通常这是通过使用加密算法来创建数字签名来实现的。

数据或行为完整性：

- 客户端（数据库）对相同查询获得相同结果的程度（例如，单调读一致性、单调写一致性、读写一致性等）。
- 系统行为一致性、可复现性和合理性的程度。

另请参见[一致性](#)。

### Integrity Quality Attribute 完整性质量属性

系统、产品或组件防止未经授权访问或修改计算机程序或数据的程度。它是信息[安全性](#)的一个子特性。参考[ISO 25010](#)网站。

### Interface 接口

有多重含义，具体取决于上下文：

1. 两个构件之间交互或通信的边界。
2. 设计构造，它提供了一个对具体组件行为的抽象，声明了与这些组件可能的交互以及这些交互的约束。

第二个含义的一个例子是面向对象语言Java (™) 中的编程语言构造接口：

```
/* File name : Animal.java */
interface Animal {
    public void eat();
    public void move();
}

/* File name : Horse.java */
public class Horse implements Animal {

    public void eat() {
        System.out.println("Horse eats");
    }

    public void move() {
        System.out.println("Horse moves");
    }
}
```

### Interface Segregation Principle (ISP) 接口隔离原则 (ISP)

构件（类、组件）不应被迫依赖于它们不使用的方法。ISP将较大的接口拆分为更小且更具体针对客户端的接口，这样客户端就只需要了解它们实际使用的那些方法。

### Interoperability Quality Attribute 互操作性质量属性

两个或多个系统、产品或组件能够交换信息并使用已交换的信息的程度。它是[兼容性](#)的一个子特性。参考[ISO 25010](#)网站。

### iSAQB®

国际软件架构认证委员会（International Software Architecture Qualification Board）——这是一个国际活跃组织，致力于推动软件架构教育的发展。有关详细讨论，请参阅[附录](#)。

### ISO 25010

用于描述（和评价）软件产品质量的标准。“质量模型确定在评价软件产品特征时将考虑哪些质量特性。”（来自[ISO网站](#)的引用）

有关ISO 25010标准定义的质量属性的列表，请参阅[\[ISO-25010\]](#)。

## ISO 9126

（已废止）用于描述（和评价）软件产品质量的标准。已被[ISO 25010](#)所替代。

## Iterative and Incremental Development 迭代和增量开发

软件开发的迭代和增量方法的结合。这些是各种敏捷开发方法（例如Scrum和XP）的基本组成部分。

## Iterative Development 迭代开发

“从收集需求到交付可工作的发布版本，贯穿开发阶段的开发方法。”（引自[c2-wiki](#)）

这样的周期会重复进行，以单独或同时改善功能、质量。

与[瀑布开发](#)形成对照。

## J

## K

## Kerckhoffs' Principle 柯克霍夫原则

荷兰密码学家和语言学家Auguste Kerckhoffs在1883年发表的一篇文章“La cryptographie militaire”中描述的六个密码学公理之一。这个公理至今仍然相关，因此被称为“Kerckhoffs' Principle”。

它描述了加密方法不需要保密才能实现加密消息的安全性。

“敌人知道这个系统”是数学家Claude Shannon提出的另一个表达，也被称为香农箴言。

参见[Bruce Schneiers Crypto-Gram, May 15, 2002](#)

## L

## Latency 延迟

延迟是指在系统中的某个变化的原因和效果之间的时间延迟。

在计算机网络中，延迟描述了适量数据（数据包）从特定位置传输到另一个位置所需的时间。

在交互式系统中，延迟是指系统接收到某些输入到音频视觉响应之间的时间间隔。通常存在延迟，通常由网络延迟引起。

## Layer 层次架构

构件或组件的集组（共同）为其他层提供一组内聚服务。层次之间通过允许使用的有序关系相互关联。

#### Learnability Quality Attribute 易学性质量属性

在指定的使用周境中，产品或系统在有效性、效率、抗风险和满意度特性方面为了学习使用该产品或系统这一指定的目标可为指定用户使用的程度。它是[易用性](#)的一个子特性。参考[ISO 25010](#)网站。

#### Liskov Substitution Principle 里氏替换原则

指面向对象编程：如果你使用继承，就要正确地使用：派生类型（子类）的实例必须能够完全替代其基类型。如果代码使用基类，这些引用可以被任何派生类的实例替换，而不会影响该代码的功能。

## M

#### Maintainability Quality Attribute 易维护性质量属性

产品或系统在改进、更正或适应环境变化和需求变化时，能够进行修改的有效性和效率程度。它由以下子特性组成：[模块化](#)、[可重用性](#)、[易分析性](#)、[易修改性](#)、[易测试性](#)。参考[ISO 25010](#)网站。

#### Maturity Quality Attribute 成熟性质量属性

系统、产品或组件在正常运行时满足可靠性要求的程度。它是[可靠性](#)的一个子特性。参考[ISO 25010](#)网站。

#### MFA 多因素认证

多因素认证，请参见[认证](#)。

#### Microservice 微服务

一种架构风格，提出将大型系统分割成小的单元。“微服务必须作为虚拟机实现，也可以作为更轻量级的替代方案，如Docker容器或单个进程。这样，它们可以很容易地单独部署到生产环境中。”（引自[Eberhard Wolff](#)关于[微服务的免费LeanPub小册子](#)）

#### Model Driven Architecture (MDA) 模型驱动架构 (MDA)

[模型驱动架构 \(MDA\)](#) 是OMG（对象管理组）为基于模型的软件开发制定的标准。定义为：“一种IT系统规范的方法，它将功能规范与特定技术平台上该功能的实现规范相分离。”

#### Model Kind 模型种类

（通过关键特性和建模）约定区分的模型类型（如ISO/IEC/IEEE 42010中所定义）。

注意，模型种类的例子包括数据流图、类图、Petri网、资产负债表、组织图表和状态转换模型。

#### Model-Driven Software Development (MDSD) 模型驱动的软件开发 (MDSD)

基本思想是从更抽象的需求或领域模型生成代码。

### Model-View-Controller 模型-视图-控制器 (MVC)

架构模式，通常用于实现用户界面。它将系统划分为三个相互连接的部分（模型、视图和控制器），以分离以下职责：

- 模型管理系统的数据和逻辑。“真理”将由一个或多个视图显示或展示。模型不知道（依赖于）其视图。
- 视图可以是（模型）信息的任何数量（任意）的输出表示。同一个模型可以有多个视图。
- 控制器接受（用户）输入并将这些输入转换为模型或视图的命令。

### Modeling Tool 建模工具

创建模型的工具（例如UML或BPMN模型）。可以用于创建一致的图表进行文档记录，因为它具有优点：每个模型元素只存在一次，但可以在许多图表中一致地显示（与仅[绘图工具](#)相对）。

### Modifiability Quality Attribute 易修改性质量属性

产品或系统可以被有效地、有效率地修改，且不会引入缺陷或降低现有产品质量的程度。它是[维护性](#)的一个子特性。参考[ISO 25010](#)网站。

### Modular Programming 模块化编程

“一种软件设计技术，它将程序的功能性分离成独立、可互换的模块，这样每个模块只包含实现所需功能的一个方面的所有必要元素。

模块具有表达模块提供的和所需的元素的接口。接口中定义的元素可以被其他模块检测到。”（引自[维基百科](#)）

### Modularity Quality Attribute 模块化质量属性

由多个独立组件组成的系统或计算机程序，其中一个组件的变更对其他组件的影响最小的程度。它是[维护性](#)的一个子特性

参考[ISO 25010](#)网站。

### Module 模块

（也参见[模块化编程](#)）

1. 结构元素或构件，通常被视为具有明确定义责任的黑盒。它封装了数据和代码，并提供公共接口，以便客户端可以访问其功能。这种含义首次在David L. Parnas的一篇开创性和基础性的论文“[On the Criteria to be Used in Decomposing Software into Modules](#)”中描述。
2. 在一些编程语言中，模块是一个用于聚合较小编程单元的结构，例如在Python中。在其他语言（如Java）中，模块称为包。
3. CPSA®-高级目前分为几个模块，可以单独学习或教学，且学习顺序没有规定。每个模块

之间的具体关联及其内容都在各自的课程大纲中进行定义。

## N

### Node (in UML) 节点(在UML中)

一个处理资源（执行环境、处理器、机器、虚拟机、应用服务器），可以在其上部署和执行工件。

### Node (Node.js) 节点(Node.js)

在现代Web开发中：[Node.js](#)®是开源JavaScript运行时的简称，此环境由Chrome的V8 JavaScript引擎构建。Node.js以其事件驱动、非阻塞的I/O模型和庞大的支持库生态系统而闻名。

### Non Functional Requirement (NFR) 非功能性需求(NFR)

指限制解决方案的需求。非功能性需求也被称为质量属性需求或[质量需求](#)。术语NFR实际上是具有误导性的，因为许多涉及的属性直接与特定系统功能相关（所以现代需求工程更喜欢将这些称为必要约束）。

### Non-repudiation Quality Attribute 抗抵赖性质量属性

能够证明行动或事件已经发生，以便后续无法否认这些行动或事件的程度。它是[安全性](#)的一个子特性。参考[ISO 25010](#)网站。

### Notation 标记

一套用于表示信息的标识、符号、数字或字符。例如：散文、表格、项目符号列表、编号列表、UML、BPMN。

## O

### Observer 观察者模式

（设计模式）“…在其中，一个对象，称为主体，维护一个其依赖对象的列表，称为观察者，并在任何状态变化时自动通知它们，通常通过调用它们的一个方法。”

（引自[维基百科](#)）

观察者模式是补充[模型-视图-控制器 \(MVC\)](#) 架构模式的关键模式。

### Open-Close-Principle (OCP) 开闭原则 (OCP)

“软件实体（类、模块、函数等）应该对扩展开放，但对修改封闭”（Bertrand Meyer, 1998）。简单来说：为了向系统添加功能（扩展），你可以不修改现有代码。这是Robert Martin针对面向对象系统的“SOLID”原则之一。在面向对象语言中，可以通过接口继承实现，更通用的做法是作为插件。

## Operability Quality Attribute 可操作性质量属性

产品或系统具有易操作和控制的属性的程度。它是[可用性](#)质量属性的一个子特性。参考[ISO 25010](#)网站。

## OWASP 开放式web应用安全项目

开放式web应用安全项目（Open Web Application Security Project）是一个于2001年成立的全球非营利在线组织，旨在改善软件的安全性。它是web安全领域信息和最佳实践的丰富资源。参见<https://owasp.org/>。

OWASP-Top-10是一个经常被引用的基于项目数据调查的攻击类别列表。

## P

## Package Principles 包原则

设计软件系统结构的基本原则([[Martin-2003](#)])：

- [复用/发布等价原则](#)（REP）
- [共同复用原则](#)（CRP）
- [共同封闭原则](#)（CCP）
- [无循环依赖原则](#)（ADP）
- [稳定依赖原则](#)（SDP）
- [稳定抽象原则](#)（SAP）

Robert C. Martin，他创造了“[SOLID](#)”这个缩写，也提出了[包原则](#)，并且经常同时重申这两个原则。尽管SOLID原则针对类级别，包原则针对包含多个类且可能独立部署的大组件级别。

包原则和SOLID原则共享一个明确的目标：保持软件的[可维护性](#)，并避免设计退化的症状：僵化、脆弱、不灵活和粘性。

虽然Martin用大规模组件的形式表达了包原则，但它们也适用于其他规模。它们的核心是通用原则，如低耦合、高内聚、单一职责、层次（无循环）分解，以及一个深刻的见解：有意义的依赖关系是从具体/不稳定概念到更抽象/稳定概念的（这与[DIP原则](#)相呼应）。

## Pattern 模式

在软件设计或架构中，一种可重用或可重复的解决常见问题的方案。

查看[架构模式](#)或[设计模式](#)。

## Perfect Forward Secrecy 完全前向保密



加密协议的一个特性，攻击者即使获取到长期密钥也无法获得有关短期会话密钥的任何信息。

具有完全前向保密性的协议示例包括TLS和OTR。如果为TLS启用了此功能，即使攻击者获得了服务器的私钥，之前记录的通信会话仍然无法被解密。

### Performance Efficiency Quality Attribute 性能效率质量属性

性能与在指定条件下所使用的资源量有关。

资源可以包括其他软件产品、系统的软硬件配置以及材料（例如，打印纸、存储介质）。

由以下子特征组成：[时间特性](#)、[资源利用性](#)、[容量](#)。

请参阅[ISO 25010](#)网站。

### Perspective 视角

视角用于考虑系统的一组相关质量特性和关注点。

架构师将视角迭代应用于系统的架构视图，以评估架构设计决策在多个角度和架构视图中的效果。

[[Rozanski & Woods 2011](#)]还将术语视角与活动、策略和指南联系起来，如果系统需要提供一组相关的质量特性，则必须考虑这些点，并建议以下视角：

- 易访问性
- 可用性和易恢复性
- 开发资源
- 演进
- 国际化
- 本地化
- 性能和可扩展性
- 法规
- 信息安全性
- 易用性

### Pikachu 皮卡丘

来自（相当著名的）[宝可梦世界](#)的类似黄色老鼠的角色。实际上，你不需要知道这个。但了解它也无妨——你可能会用这个知识给你的孩子留下深刻印象……

### Pipe 管道

在管道与过滤器架构风格中的连接器，它将数据流或数据块从一个过滤器的输出传输到另一个过滤器的输入，而不修改数据的值或顺序。



## PKI 公钥基础设施

公钥基础设施的简称。通常涉及非对称加密的数字证书管理概念。术语“公共”大多数时候指的是使用的[加密密钥类型](#)，而不必然指对公众开放的基础设施。为了防止语义混淆，可以使用“开放PKI”或“封闭PKI”等术语，参见[\[Anderson-2008\]](#)第21.4.5.7章PKI，第672页。

PKI通常基于[CA或信任网](#)。

## Port 端口

是在组件图中使用的UML构件。它是一个接口，定义组件与其环境的交互点。

## Portability Quality Attribute 可移植性质量属性

系统、产品或组件从一个硬件、软件或其他操作或使用环境转移到另一个环境的有效性和效率的程度。由以下子特征组成：[适应性](#)、[易安装性](#)、[易替换性](#)。请参阅[ISO 25010](#)网站。

## POSA 面向模式的软件架构

面向模式的软件架构。关于软件架构模式丛书。

## Principal 主体

在安全上下文中，主体是已经被认证并可以被分配权限的实体。主体可以是用户，但也可以是其他服务或系统上运行的进程。该术语用于[Java环境](#)以及各种认证协议中使用（参见[GSSAPI RFC2744](#)或[Kerberos RFC4121](#)）。

## Proxy 代理

（设计模式）“一个由客户端调用以访问后台真实服务对象的包装器或代理对象。使用代理可以简单地转发给真实对象，或可以提供额外的逻辑。在代理中，可以提供额外的功能，例如，当对真实对象的操作资源密集时进行缓存，或在真实对象的操作被调用之前检查前置条件。对客户端而言，使用代理对象与使用真实对象类似，因为两者都实现了相同的接口。”（引自[维基百科](#)）

## Pseudo-Randomness 伪随机性

通常与伪随机数生成器一起使用。以高[熵](#)收集随机性是资源密集型的，通常许多应用程序不需要这样做，但密码学除外。为了解决这个问题，伪随机生成器采用数据种子初始化，并基于这种子创建随机值。数据将随机生成，但如果生成器用相同的种子初始化，则数据总是相同的。这被称为伪随机性，其性能密集度较低。

## Q

## Qualitative Evaluation 定性评价

发现关于系统所需质量属性的风险。分析或评估系统或其架构是否能满足所需或要求的质量目标。

定性评价不是计算或测量系统或架构的某些特性，而是关注风险、权衡和敏感点。

另见[评估](#)。

## Quality 质量

参见[软件质量](#)和[质量属性](#)。

## Quality Attribute 质量属性

软件质量是系统拥有所需属性组合的程度（参见：[软件质量](#)）。

国际标准ISO-25010定义了以下质量属性：

- 功能性
  - 功能完备性
  - 功能正确性
  - 功能适合性
- 性能效率
  - 时间特性
  - 资源利用性
  - 容量
- 兼容性
  - 共存性
  - 互操作性
- 易用性
  - 可辨识性
  - 易学性
  - 易操作性
  - 用户差错防御性
  - 用户界面舒适性
  - 易访问性
- 可靠性
  - 成熟性
  - 可用性
  - 容错性
  - 易恢复性
- 信息安全性

- 保密性
- 完整性
- 抗抵赖性
- 可核查性
- 真实性
- 维护性
  - 模块化
  - 可重用性
  - 易分析性
  - 易修改性
  - 易测试性
- 可移植性
  - 适应性
  - 易安装性
  - 易替换性

区分以下类型的质量属性可能有帮助：

- 运行时质量属性（在系统执行时可以观察到的）。
- 非运行时质量属性（在系统执行时无法观察到的）。
- 商业质量属性（成本、进度、市场性、组织适宜性）。

运行时质量属性的示例包括功能性、性能效率、信息安全性、可靠性、易用性和互操作性。

非运行时质量属性的示例包括易修改性、可移植性、易理解性和易测试性。


## Quality Characteristic 质量特性

同义词：质量属性。

## Quality Model 质量模型

（来自 ISO 25010）定义与软件的静态性质以及计算机系统和软件产品的动态性质相关的质量特性的模型。质量模型提供一致的术语，用于指定、测量和评价系统和软件产品的质量。

质量模型的应用范围包括支持从不同角度对软件和软件密集型计算机系统进行规范和评价，包括相关的采购、需求、开发、使用、评价、支持、维护、质量保证和控制以及审计。



评论（Gernot Starke）

像ISO-25010这样的质量模型只提供了术语的分类，但并不提供指定或评价质量的任何手段。我同意上述“一致的术语”，但强烈反对“测量和评价”。对于测量和评价，你绝对需要额外的工具和/或方法，纯粹模型并无帮助。

Quality Requirement 质量需求

系统组件的特性或属性。示例包括运行时性能、（人身或生产）安全、信息安全性、可靠性或维护性。另见[软件质量](#)。

Quality Tree 质量树

（同：质量属性效用树）。一个用来描述产品质量的层次模型：根节点为“质量”，其下按领域或主题层层细化，最终细化到质量场景，质量场景是整棵树的叶子节点。

- 产品质量标准，如ISO 25010，提出了通用的质量树。
- 特定系统的质量可以通过一个特定的质量树来描述（见下面的示例）。

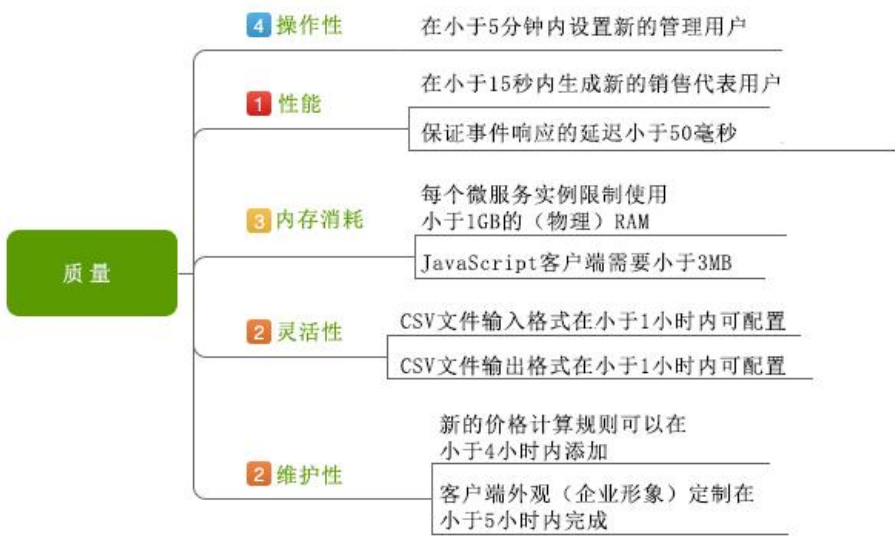


图1：质量树示例

Quantitative Evaluation 量化评价

（同：定量分析）：测量或计算软件制品的值，例如耦合度、圈复杂度、大小、测试覆盖率。这样的度量可以帮助识别系统的关键部分或元素。

R

Randomness 随机性

参见[熵](#)或[伪随机性](#)。

### Rationale 理由

解释构成架构决策背后的推理或论点。

### RBAC（基于角色的访问控制）

角色是通常分配给一组[主体](#)的固定权限集。这允许基于角色的访问控制通常比基于[ACL](#)的系统更有效地实现，并且例如可以使代理安排成为可能。

### Recoverability Quality Attribute 易恢复性质量属性

在发生中断或失效时，产品或系统能够恢复直接受影响的数据并重建期望的系统状态的程度。是[可靠性](#)的一个子特性。参见[ISO 25010](#)网站。

### Redesign 重新设计

以这样一种方式修改软件单元，使其满足与之前相似的目的，但以不同的方式和可能的不同手段。经常被错误地称为重构。

### Refactoring 重构

一个表示通过改变内部结构而不改变行为来改进软件单元的术语。（参见“重构是以这样一种方式改变软件系统的过程，它不改变代码的外部行为，但改进了内部结构。” – 重构，Martin Fowler，1999年）不要与重新设计混淆。

### Registry 注册表

“一个众所周知的对象，其他对象可以使用它来找到常用的对象和服务。”（引自[PoEAA](#)）。通常实现为[单例](#)（也是一个众所周知的设计模式）。

### Relationship 关系

表示架构元素之间某种依赖的通用术语。在架构中使用不同类型的关系，例如调用、通知、所有权、包含、创建或继承。

### Reliability Quality Attribute 可靠性质量属性

系统、产品或组件在指定条件下、指定时间内执行指定功能的程度。由以下子特征组成：[成熟性](#)、[可用性](#)、[容错性](#)、[易恢复性](#)。参见[ISO 25010](#)网站。

### Replaceability Quality Attribute 易替换性质量属性

在相同的环境中，产品能够替换另一个相同用途的指定软件产品的程度。是[可移植性](#)的一个子特性。参见[ISO 25010](#)网站。

## Repository 仓库

在架构文档中：在自动构建过程将其收集成一个一致的文档之前存储工件的地方。

在领域驱动设计中：仓库是领域驱动设计的构建块之一。仓库隐藏了基础设施层的技术细节，对域层来说是透明的。仓库返回数据库中持久化的实体。

## Resource Utilization Quality Attribute 资源利用性质量属性

产品或系统执行其功能时，所使用资源数量和类型满足需求的程度。是性能效率的一个子特性。参见ISO 25010网站。

## Reusability Quality Attribute 可重用性质量属性

资产能够被用于多个系统，或其他资产建设的程度。是维护性的一个子特性。参见ISO 25010网站。

## Reuse/Release Equivalence Principle 重用/发布等价原则

设计软件系统结构的基本原则（可参见包原则）。它要求大型组件“发布”并受版本控制，特别是如果系统从多个点使用它们。即使我们不公开发布它们，我们也应该从系统中提取这些组件，并通过具有适当版本控制的外部依赖管理器提供它们。

REP包含两个不同的洞察：

1. 在大规模上，模块化和低耦合需要的不仅仅是类型分离。
2. 组件的可重用性（即使所有的“重用”都是内部的）转化为整体的维护性。

## Risk 风险

简单来说，风险是问题发生的可能性。风险涉及对活动或决策的效果、后果或含义的不确定性，通常与某个特定价值（如健康、金钱或系统的可用性或安全性等质量）的负面含义有关。

为了量化风险，发生的可能性乘以潜在价值，通常是损失——否则风险就是一种机会，鉴于不确定性，这可能是某些风险的潜在结果。

## RM/ODP

开放分布式处理的参考模型是信息系统文档的一个（抽象的）元模型。在ISO/IEC 10746中定义。

## Round-trip Engineering 双向工程

“能够对模型以及从该模型生成的代码进行任何类型更改的概念。更改始终双向传播，并且两种工件始终保持一致。”（引自维基百科）



评论 (Gernot Starke)

在我个人看来，这在实际情况中是行不通的，只有在类似hello-world这样的场景中才可能，因为从低级源代码到高级架构元素的逆向抽象通常涉及设计决策，而且现实中无法实现自动化。



评论 (Matthias Bohlen)

最近我看到了一些源自领域驱动设计 (DDD) 的代码，其中的逆向工程确实有效。

## Ruby

一种精彩的编程语言。

## Runtime View 运行时视图

展示了在具体场景中运行时构建块（或其实例）的合作或协作。应该引用[构建块视图](#)的元素，例如可以（但不必须）用UML序列或活动图来表达。

## S

### S. O. L. I. D. 原则

SOLID（单一职责、开闭原则、里氏替换、接口隔离和依赖倒置）是一些原则的首字母缩略词（由[Robert C. Martin](#)命名），用以改进面向对象的编程和设计。这些原则使开发者更有可能编写易于维护和随时间扩展的代码。

有关额外的资料来源源，参见[\[SOLID-principles\]](#)。

## Scenario 场景

质量场景记录所需的质量属性。它们“…是从开发和最终用户角度，对系统的使用预测或期望进行概述”（[\[Kazman+1996\]](#)）因此，它们有助于以实用和非正式的方式描述系统所需或期望的质量，使“质量”这一抽象概念具体化和实用化。

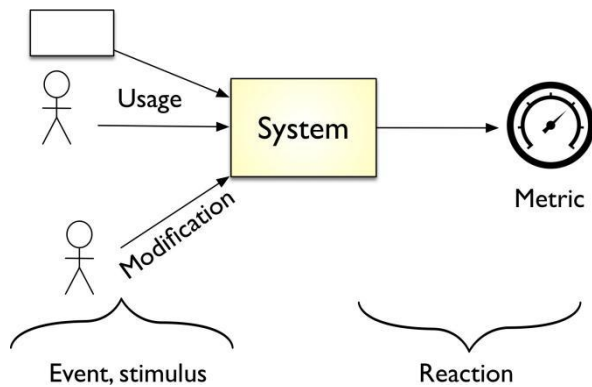


图2. (质量) 场景的通用形式

- 事件/刺激：任何到达系统的条件或事件
- 系统（或系统的一部分）由事件刺激。
- 响应：当受到刺激后进行的活动。
- 度量（响应度量）：响应应以某种方式可度量。

通常场景分为：

- 使用场景（应用场景）
- 变更场景（修改或扩展场景）
- 失败场景（边界、压力或探索性场景）

## SDL

安全开发生命周期是公司在通常的软件开发过程中增加的安全软件工程实践。这包括例如代码审查、架构风险分析、黑盒/白盒和渗透测试等多种补充措施。SDL应覆盖应用程序的整个生命周期，从最初的需求工程任务开始，到通过修复安全问题来运行已发布软件的反馈结束。

参见 [McGraw-2006]，第239页。

## Security Goals 信息安全目标

这些目标是信息安全的关键点。它们是一组基本的信息属性，这些属性能否被满足取决于系统的架构和流程。

最常见的一组安全目标是所谓的“CIA三元组”：

- 保密性
- 完整性
- 可用性

“信息保证与安全参考模型”（RMIAS）通过可核查性、可审计性、真实性/可信度、抗抵赖性和私密性扩展了这个列表。

这些是与信息安全相关的非功能性需求的典型例子。



参见 [Anderson-2008]，第11页或 [RMIAS-2013]。

### Security Quality Attribute 信息安全质量属性

产品或系统保护信息和数据的程度，使个人或产品或系统具有与其类型和授权级别相适应的数据访问权限。由以下子特性组成：保密性、完整性、抗抵赖性、可核查性、真实性。参见ISO 25010网站。

### Self-Contained System (SCS) 自包含系统 (SCS)

一种与微服务类似的架构风格。引用scs-architecture.org网站的话说：

“自包含系统 (SCS) 方法是一种架构，它专注于将功能分离到许多独立系统中，使整个系统成为许多较小软件系统的协作。这避免了不断增长并最终变得无法维护的大型单体应用程序的问题。”

### Sensitivity Point 敏感点

（如ATAM中的定性评价）：影响多个质量属性的架构或系统的元素。例如，如果一个组件既负责运行时性能又负责健壮性，那么该组件就是一个敏感点。

形象地说，如果你搞砸了一个敏感点，你通常会有不止一个问题。

### Separation of Concerns (SoC) 关注点分离 (SoC)

架构的任何元素都应具有排他性和单一性的责任和目的：没有元素应该共享另一个元素的责任或包含不相关的责任。

另一种定义是“将系统分解为尽可能少重叠的元素。”

著名的Edgar Dijkstra在1974年说：“关注点分离……即使不是完全可能，也是有效整理一个人思想的唯一可用技术”。

类似于单一职责原则。

### Sequence Diagram 序列图

用来说明架构元素如何交互以实现某个特定场景的图表类型。它展示了元素之间消息的顺序（流程）。作为平行的垂直线，它展示了对象或组件的生命周期，水平线描述了这些组件之间的交互。参见下面的例子。

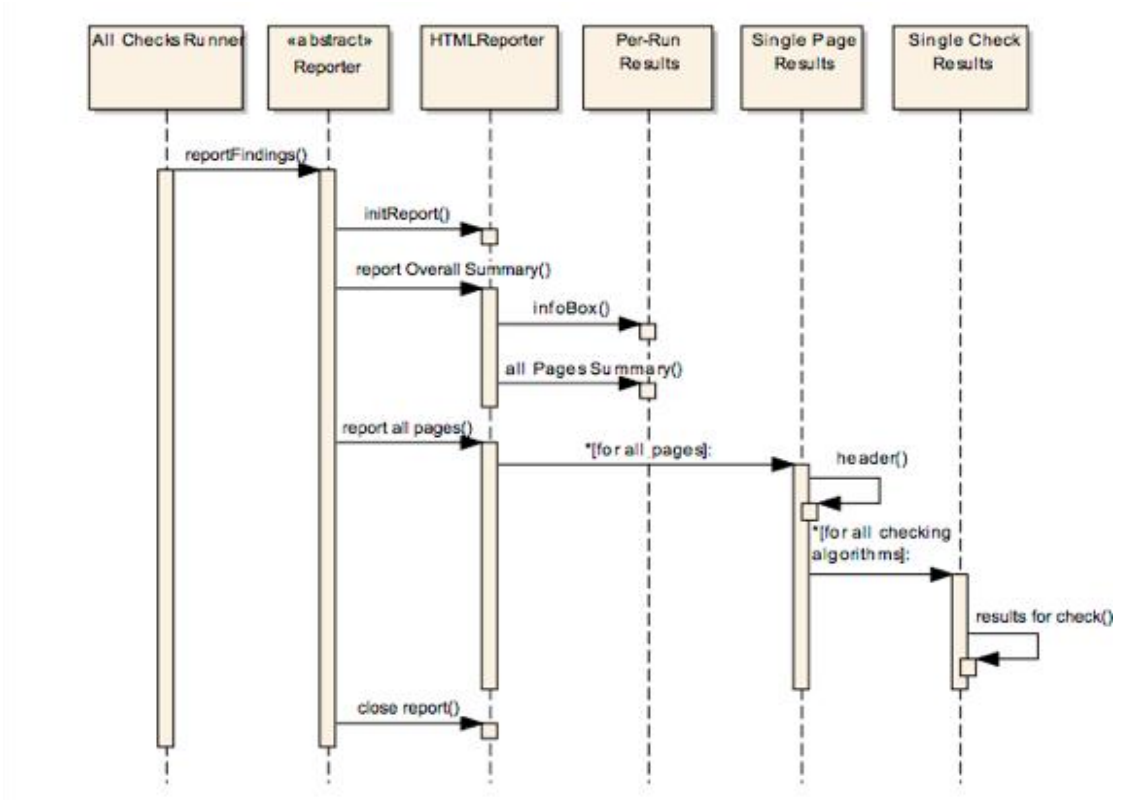


图3. 序列图示例

Service 服务

服务是一个技术上的、自治的单元，它捆绑了相关的功能，通常是围绕一个主题，并通过一个明确的接口使它们可供其他构建块使用。

服务理想地抽象了内部的技术功能，以至于使用服务时不需要知道、甚至理解内部实现细节。

典型的外部可访问服务示例包括浏览器服务、网络服务、系统服务或电信服务。

（基于[维基百科](#)）

Service (DDD) 服务 (DDD)

服务是[领域驱动设计](#)的一个构建块。服务实现了业务领域的逻辑或过程，这些逻辑或过程不仅仅由实体执行。服务是无状态的，其操作的参数和返回值是[实体](#)、[聚合](#)和[值对象](#)。

Signature 签名

函数或方法的签名：参见[函数签名](#)。

数字签名：用于验证数据或文档真实性的方法。

Single Responsibility Principle (SRP) 单一责任原则 (SRP)

系统或架构中的每个元素应有单一责任，且其所有功能或服务应与责任保持一致。

有时[内聚性](#)被认为是与SRP相关联。

## Singleton 单例模式

“一种设计模式，它限制一个类仅实例化一个对象。当需要一个对象来协调整个系统的操作时，这是非常有用的。”（引自[维基百科](#)。）

## Software Architecture 软件架构

存在若干个（！）有效且合理的软件架构术语定义。

以下定义由[IEEE 1471](#)标准提出：

架构：系统的基本组织，它体现在其组件中，组件彼此之间以及与环境的关系，以及指导其设计和演变的原软件则。

新的ISO/IEC/IEEE 42010:2011标准采纳并修订了这一定义：

架构：（系统）系统在其环境中的基本概念或特性，体现在其元素、关系以及设计和演变的原则中。

这一定义中的关键术语需要一些解释：

- 组件：子系统、模块、类、函数或更一般的术语[构建块](#)：软件的结构元素。组件通常用编程语言实现，但也可以是其他构成系统的工件。
- 关系：接口、依赖、关联 - 同一特征的不同名称：组件需要与其他组件互动以实现[关注点分离](#)。
- 环境：每个系统都与其环境有一些关系：数据、控制流或实践可能会与不同类型的邻居传输。
- 原则：适用于一个系统或其几个部分的规则或约定。通常对系统的多个元素有效的决策或定义。常称为[概念](#)或甚至解决方案模式。原则（概念）是[概念完整性](#)的基础。

软件工程研究所[进一步维护了定义的集合](#)。

虽然这个术语经常指的是IT系统的软件架构，但它也用来指软件架构作为一门工程科学。

## Software Quality 软件质量

（来自IEEE标准1061）：软件质量是软件拥有期望属性组合的程度。这个期望的属性组合需要被清晰定义；否则，质量的评估取决于直觉。

（来自ISO/IEC标准25010）：系统的质量是指系统满足其各种利益相关方的明确和隐性需求的程度，因而提供价值。指这些陈述和暗示的需求在ISO 25000质量模型中被分类为特性，这些特性在某些情况下进一步细分为子特性。

## Stable Abstractions Principle 稳定抽象原则

设计软件系统结构的一个基本原则（可参见[包原则](#)）。它要求组件的抽象性与它们的稳定性成正比。与之密切相关的SDP也在此背景下解释了稳定性的概念。

我们希望代表抽象概念和责任的组件仅需较少修改或无需修改，因为许多概念上更具体（具体的）组件依赖于它们。我们希望那些应该或不容易改变的组件至少足够抽象，以便我们可以扩展它们。这与OCP有关。

SAP听起来可能像是一个循环论证，直到其背后的思想显现：具体的事物和概念自然比抽象的更易变、更具体、更随意且更多。系统的组件结构简单就反映出这一点。一般逻辑、系统的物理工件以及其功能和技术概念都应该一致。

SAP与SDP密切相关。它们的结合等同于DIP的一个更通用且可以说更深刻的版本：具体概念自然依赖于更抽象的概念，因为它们是由更通用的构建块组成或派生的。依赖概念自然更具体，因为它们由比其依赖更多的信息定义（假设没有依赖循环）。

### Stable Dependencies Principle 稳定依赖原则

设计软件系统结构的一个基本原则（可参见[包原则](#)）。它要求经常变化的组件依赖于更稳定的组件。

组件的部分波动性是[预期的](#)，并自然隐含在其特定责任中。

但这种情况下的稳定性也是传入和传出依赖关系的函数。一个被其他组件严重依赖的组件更难以改变，被认为更稳定。而一个严重依赖于其他组件的组件有更多改变的理由，被认为是不稳定的。

因此，就依赖性而言，拥有许多客户端的组件不应依赖于一个有许多依赖的组件。同时拥有这两种属性的单一组件也是一个警示标志。这样的组件有许多改变的理由，但同时难以改变。

SDP的原始定义（如[\[Martin-2003\]](#)）涉及一个[不稳定性度量](#)。不幸的是，该度量没有捕捉到预期/固有的波动性、传递依赖或上述提到的红色预警。但不管它如何被测量，我们都应重视SDP的想法。

SDP与SAP密切相关。它们的结合等同于DIP的一个版本（更多关于这个在SAP下）。

### Stakeholder 利益相关方

可能受系统、系统的开发或执行影响或与系统利益相关的个人或组织。

以下是一些示例，包括用户、员工、所有者、管理员、开发人员、设计师、项目或产品经理、产品所有者、项目经理、需求工程师、业务分析师、政府机构、企业架构师等。

根据ISO/IEC/IEEE 42010的定义，利益相关方是指对（按照ISO/IEC/IEEE 42010所定义的）系统中拥有利益的（角色、职位、）个人、团队、组织或其阶层。

这种利益可以是积极的（例如，利益相关方希望从系统中获益）、中立的（利益相关方必须测试或验证系统）、或者是消极的（利益相关方与系统竞争或希望其失败）。

### Structural Element 结构元素

参见[构建模块或组件](#)

## Structure 结构

结构是系统中相互关联的元素的一种排列、顺序或组织。结构包括构建块（结构元素）及其关系（依赖关系）。

软件架构中的结构通常在[架构视图](#)中使用，例如[构建块视图](#)。文档模板（例如[arc42](#)）也是一种结构。

## Symmetric Cryptography 对称加密

对称加密是基于相同的密钥对数据进行加密和解密的。发送方和接收方必须就通信的密钥达成一致。详见[\[Schneier, Bruce\]](#)，第17页。

## System 系统

系统是为了实现共同目的而组织在一起的元素（构建块、组件等）的集合。

在ISO/IEC/IEEE标准中提供了若干个系统的定义：

- 根据[ISO/IEC 15288]中的描述，系统是人为构建的，可能配置有以下一项或多项组成部分：硬件、软件、数据、人员、流程（例如，为用户提供服务的流程）、程序（例如，操作员指令）、设施、材料和自然存在的实体。
- 软件产品和服务在[ISO/IEC 12207]中进行了描述。
- 根据[IEEE Std 1471:2000]中的描述，软件密集型系统被定义为在设计、构建、部署和演化整个系统过程中，软件对系统产生至关重要的影响的任何系统，包括“单个应用程序、传统意义上的系统、子系统、系统集、产品线、产品系列、整个企业以及其他感兴趣的聚合体”。

## System-of-Interest 利益系统

利益相关系统（或简称为系统）是指在准备架构描述时正在考虑其架构的（按照ISO/IEC/IEEE 42010中定义的）系统。

## T

## Technical Context 技术上下文

以技术或部署的角度展示完整的系统，将其作为一个[黑盒](#)在其环境中呈现。这其中特别包括技术接口和通信渠道，以及相邻系统的相关技术细节。技术上下文通过将领域特定的与相邻系统的交互映射到具体的通信渠道和技术协议，从而对[业务上下文](#)进行补充。

参见[上下文视图](#)。

## Template (for Documentation) 模板（为了文档化）

在软件开发中使用的标准化工件顺序。它可以帮助为其他文件提供基础，特别是使文档建立在一个预定义结构中，而不规定这些单个文件的内容。

这种模板的一个著名示例是[arc42](#)。

## Temporal Coupling 时间耦合

各种来源有不同的解释。时间耦合

- 意味着正在通信的进程都必须启动并处于运行状态。详见[Tanenbaum+2016]。
- 当你经常同时提交(修改)不同的组件时。参见[tornhill - 2015]。
- 当一个类中的两个或多个成员之间存在隐式关系，要求客户端在调用另一个成员之前先调用另一个成员时。Mark Seemann，参见[设计气味时间耦合](#)。
- 意味着一个系统需要等待另一个系统的响应才能继续处理。参见[Rest反模式](#)。

## Testability Quality Attribute 可测性质量属性

为系统、产品或组件建立测试标准并进行测试以确定是否满足这些标准的有效性和效率程度。是[可维护性](#)的一个子特性。请参阅[ISO 25010网站](#)。

## Time Behaviour Quality Attribute 时间行为质量属性

产品或系统在执行其功能时，响应和处理时间以及吞吐率满足需求的程度。是[性能效率](#)的一个子特性。参见[ISO 25010网站](#)。

## TLS 传输层安全

传输层安全是一套通过[CIA三位一体](#)的方式对双方通信进行加密保护的协议，被广泛用于互联网上的安全通信，是HTTPS的基础。

TLS起初是SSL（安全套接字层）的V3.0更新版本，并且现在应该代替SSL使用。请参阅[RFC7568《弃用安全套接字层版本3.0》](#)。

## TOGAF 开放群组架构框架(TOGAF)

[开放群组架构框架](#)（The Open Group Architecture Framework，简称TOGAF）是用于规划和维护企业IT架构的概念框架。

## Tools-and-Material-Approach 工具和材料方法

“工具与材料方法”是一个术语，指的是一种利用工具和材料作为主要资源来实现目标或完成任务的方法或策略。在不同的背景下，这种方法可能涉及利用特定的工具、设备或资源以及适当的材料，以高效和有效地实现目标。例如，在制造、建筑或软件开发中，工具与材料方法可能涉及使用专门的工具和适当的材料来优化流程并实现预期的结果。

## Top-Down 自上向下

“工作方向”或“沟通顺序”：从一个抽象或一般的构想开始，朝着更具体、更特殊或更详细的表示方向进行工作。



## Traceability 可追溯性

（更准确地说，需求的可追溯性）文档中：

1. 所有的需求都由系统的元素进行了解决（正向可追溯）；
2. 所有系统的元素至少被一个需求所证明（反向可追溯）。

我的个人观点是，如果可能的话，应该尽量避免可追溯性，因为它会造成大量的文档工作量。

## Trade-Off 权衡

（同义词：妥协）。在两个期望或要求但通常是不兼容或矛盾的特征之间达到或协商达成的平衡。例如，软件开发通常必须权衡内存消耗和运行时速度。

更口语化地说，如果一件事增加，那么另一件事就必须减少。

更加口语化地说：没有免费的午餐。每个质量属性都有与其他质量属性相对应的代价。

## Trainer 讲师、培训师

培训师是指在获得认可的[培训提供商](#)的授权框架内自行进行培训课程的人员。因此，只有获得认证的培训供应商才能组织和开展由认证培训师进行的CPSA®培训课程。只有获得认证的培训供应商才能申请培训师的[认证资质](#)。

## Training Level 培训级别

iSAQB® CPSA®教育计划目前分为两个培训级别：基础级和高级。培训级别应该包含相互建立的知识。每个级别之间以及培训级别的内容之间的确切关系在各自的课程大纲中定义。

## Training Provider 培训供应商

拥有[授权](#)培训材料使用权或已经购买培训材料授权的组织或个人，提供培训师和基础设施，并开展培训课程。

## U

## Ubiquitous Language 通用语言

[领域驱动设计](#)的一个概念：通用语言是围绕[领域模型](#)构建的语言。所有团队成员都使用它以便将团队的所有活动与软件连接起来。通用语言是一个活跃的实体，在项目期间会不断演变，并在软件整个生命周期中进行变更。

## Unified Modeling Language (UML) 统一建模语言

[统一建模语言（UML）](#)是一种用于可视化、指定和记录软件系统的工件和结构的图形化语言。

- 对于构建块视图或上下文视图，可以使用组件图，其中组件、包或类表示构建块。
- 对于运行时视图，可以使用序列图或活动图（带有泳道）。理论上可以使用对象图，但在实践中不建议使用，因为即使对于小型场景，它们也会变得混乱不堪。
- 对于部署视图，可以使用带有节点符号的部署图。

## Unit Test 单元测试

对系统中最小可测试部分的测试，以确定它们是否适合使用。

根据实现技术的不同，一个单元可能是一个方法、函数、接口或类似的元素。

## Usability Quality Attribute 易用性质量属性

指产品或系统在指定使用环境中，能够根据特定用户的需求，以有效、高效和满意的方式实现指定目标的程度。其由以下子特性组成：[适合性](#)、[可辨识性](#)、[易学性](#)、[易操作性](#)、[用户差错防御性](#)、[用户界面舒适性和易访问性](#)。请参阅[ISO 25010](#)网站。

## User Error Protection Quality Attribute 用户差错防御性质量属性

指系统保护用户不出错的程度。它是“[易用性](#)”的一个子特性。请参阅[ISO 25010](#)网站。

## User Interface Aesthetics Quality Attribute 用户界面舒适性质量属性

指用户界面使用户感到愉悦和满意的交互的程度。它是“[易用性](#)”的一个子特性。请参阅[ISO 25010](#)网站。

## Uses Relationship 使用关系

存在于两个构建块之间的依赖关系。如果A使用B，则A的执行依赖于B的正确实现的存在。

## V

## Value Object 值对象

值对象是[领域驱动设计](#)的构建块。值对象没有自己的概念标识，应被视为是不可变的。它们用于描述[实体](#)的状态，可以由其他值对象组成，但永远不能由[实体](#)组成。

## View 视图

参见[架构视图](#)

## W

## Waterfall Development 瀑布开发



在这种开发方法中，你预先收集所有需求，进行所有必要的设计，直到详细设计级别，然后将规范交给编码人员编写代码；然后进行测试（可能会经历一段集成地狱），最终一次性并在一个大的最终版本中交付整个项目。一切都是大规模的，包括失败的风险。”（引自[C2维基](#)）

也可参见[迭代开发](#)

### Web of Trust 信任网络

由于单个CA（证书授权）很容易成为攻击者的目标，因此信任网络将信任的任务委托给用户。每个用户通常通过验证给定密钥的指纹来确定信任哪些其他用户的身份证明。该信任通过对另一个用户的密钥进行签名来表示，然后后者可以发布该密钥，附加签名。然后，第三个用户可以验证该签名，并决定是否信任该身份。

电子邮件加密PGP（完美隐私）是基于信任网络的PKI（公钥基础设施）的示例。

### White Box 白盒

展示系统或构建块的内部结构，由黑盒以及内部/外部关系和接口组成。

也可参考[黑盒](#)

### Workflow Management System (WFMS) 工作流管理系统 (WFMS)

“为以工作流形式定义的任务序列提供用于设置、执行和监控的基础设施。”（引用自维基百科）。

### Wrapper 包装器

（同义词:装饰器、适配器、网关）模式，用于抽象化组件的具体接口或实现，动态地为对象附加其他职责。

根据来源不同，术语包装器的语义可能不同。



注释（Gernot Starke）

在文献中发现的关于这个词的微小差异在实际生活中通常并不重要。在单个软件系统中，对组件或构建块进行包装应具有明确清晰的语义。

X

Y

Z

Translation Tables 翻译表


在这里，你可以找到英语和德语及中文（见下文）以及德语到英语及中文（下一章节）之间的术语翻译。

其中几个术语基于iSAQB®协会的法律和组织基础（因此与软件架构无关）。

以下翻译是在本书的开源[GitHub存储库](#)中维护的，存储在一个简单的JSON输入文件中。

[^生成翻译]：在链接<https://github.com/isaqb-org/glossary>中文档包含生成翻译表所需的所有信息。目前仅支持英语和德语。翻译表以JSON格式维护，欢迎提出改进建议！

English to German 英德中对照



请注意：该翻译表不一定完整，一些英文术语将不会被翻译，而是最好使用其原始语言（例如，一些模式的名字）。

English 英语	German 德语	中文
Accessibility	Barrierefreiheit, Zugänglichkeit	易访问性
Accountability	Rechenschaft, Verantwortlichkeit	可核查性
Accreditation contract	Akkreditierungsvertrag	认证合同
Accreditation fee	Akkreditierungsgebühr	认证费用
Action	Maßnahme	软件演进，动作
Adaptability	Adaptierbarkeit	适应性
Adaption	Anpassung	适应
Adequacy	Angemessenheit	充分性
Analysability	Analysierbarkeit	可分析性
Approach	Ansatz	方法
Appropriateness	Angemessenheit	适合性
Appropriateness Recognizability	Erkennbarkeit der Brauchbarkeit, Verständlichkeit	可辨识性
Architectural objective	Architekturziel	架构目标
Architectural pattern	Architekturmuster	架构模式
Architectural view	Architektursicht, Sicht	架构视图
Architecture assessment	Architekturanalyse, Architekturbewertung	架构评估
Architecture evaluation	Architekturbewertung, Architekturanalyse	架构评价
Architecture objective	Architekturziel	架构目标

Articles of association	Satzung des Vereins	协会章程
Artifact	Artefakt	工件、制品
Aspect	Aspekt, Belang	方面

Assessment	Bewertung, Begutachtung, Einschätzung, Untersuchung	评估
Association	Verein, Beziehung	协会
AttackTree	Angriffsbäume	攻击树
Authenticity	Authentifizierbarkeit	真实性
Availability	Verfügbarkeit	可用性
BoundedContext	Kontextgrenze	限界上下文
Buildingblock	Baustein	构建块
Buildingblock view	Bausteinsicht	构建块视图
Business	Fachlichkeit, Domäne	业务
Businessarchitecture	fachliche Architektur, Geschäftsarchitektur	业务架构
Businesscontext	Fachlicher Kontext	业务上下文
Cabinet (as methaphor for template)	Schrank (als Metapher für Template)	柜子(隐喻模板)
Capacity	Kapazität	能力
Cash audit	Rechnungsprüfung	现金审计
Cash auditor	Rechnungsprüfer	现金审计员
Certification authority	Zertifizierungsstelle	证书准发机构
Certification body	Zertifizierungsstelle	认证机构
Chairman	Vorsitzender	主席
Channel	Kanal	渠道
Co-Existence	Koexistenz	共存性
Cohesion	Kohäsion, innerer Zusammenhalt	内聚
Commensurability	Angemessenheit, Messbarkeit, Vergleichbarkeit	可公度性, 可通约性
Compatibility	Kompatibilität	兼容性
Compliance	Erfüllung, Einhaltung	履约, 合规性
Component	Baustein, Komponente	组件
Concern	Belang	关注点
Confidentiality	Vertraulichkeit	保密性
Constraint	Randbedingung, Einschränkung	限制, 约束
Context (of a term)	Einordnung (eines Begriffes) in einen Zusammenhang	(关于一个定义)的上下文, 周境
Context view	Kontextabgrenzung	上下文视图

Coupling	Kopplung, Abhängigkeit	耦合
Cross-cutting	Querschnittlich	横切
Curriculum	Lehrplan	课程, 教学计划
Decomposition	Zerlegung	分解
Dependency	Abhängigkeit, Beziehung	依赖
Deployment	Verteilung	部署
Deployment unit	Verteilungsartefakt	部署单元

Deployment view	Verteilungssicht	部署视图
Deputy chairman	Stellvertretender Vorsitzender	副主席
Design	Entwurf	设计
Design approach	Entwurfsansatz, Entwurfsmethodik	设计方案, 设计方法
Design decision	Entwurfsentscheidung	设计决策
Design principle	Entwurfsprinzip	设计原则
Domain	Fachdomäne, Fachlicher Bereich, Geschäftsbereich	域, 领域
Domain event	Fachliches Event	域事件, 领域事件
Domain-related architecture	fachliche Architektur	与领域相关的架构
Drawing Tool	Mal-/Zeichenprogramm	绘画/绘图工具
Economicalness	Sparsamkeit, Wirtschaftlichkeit	经济性
Embedded	Eingebettet	嵌入式
Encapsulation	Kapselung	封装
Enterprise IT architecture	Unternehmens-IT- Architektur	企业IT架构
Estimation	Schätzung	估计, 估算
Evaluation	Bewertung	评价
Examination question	Prüfungsfrage	考试题目
Examination rules and regulations	Prüfungsordnung	考试条例, 考试规则
Examination sheet	Prüfungsbogen	考试试卷
Examination task	Prüfungsaufgabe	考试任务
Examinee	Prüfling	考生
Examiner	Prüfer	监考人员, 考官
Executive board	Vorstand	执行董事会
Fault Tolerance	Fehlertoleranz	容错性
Fees rules and regulations	Gebührenordnung	收费条例, 收费标准
Fitness Function	Fitnessfunktion	适应度函数

Functional Appropriateness	Funktionale Angemessenheit	功能适合性
Functional Completeness	Funktionale Vollständigkeit	功能完备性
Functional Correctness	Funktionale Korrektheit	功能正确性
Functional Suitability	Funktionale Eignung	功能适用性
General meeting	Mitgliederversammlung	大会
Improvement	Verbesserung	改善
Improvement action	Verbesserungsmaßnahme	改善措施
Influencing Factor	Einflussfaktor	影响因素
Information hiding principle	Geheimnisprinzip	信息隐藏原则
Installability	Installierbarkeit	易安装性
Integrity	Integrität	完整性
Interdependency (between design decisions)	Abhängigkeit (zwischen Entwurfsentscheidungen)	(设计决策之间的)相互依赖性
Interface	Schnittstelle	接口
Interfacedescription	Schnittstellenbeschreibung, Schnittstellendokumentation	接口描述
Interoperability	Interoperabilität	互操作性
Learnability	Erlernbarkeit	易学性
Learning goal	Lernziel	学习目标
License fee	Akkreditierungsgebühr	证书费用
Licensee	Lizenznehmer	证书被授予人
Licensingagreement	Lizenzvertrag, Lizenzvereinbarung, Akkreditierungsvertrag	许可协议, 许可约定
Local court	Amtsgericht	地方法院
Maintainability	Wartbarkeit	易维护性
Maturity	Reifegrad	成熟度
Means for describing	Beschreibungsmittel	描述手段
Means for documenting	Beschreibungsmittel	记录手段
Measurability	Messbarkeit	可测性
Members' meeting	Mitgliederversammlung	会员会议
message-driven	Nachrichten-zentrisch	消息驱动
ModelingTool	Modellierungswerkzeug	建模工具
Modifiability	Modifizierbarkeit	易修改性
Modularity	Modularität	模块化度
Module	Komponente, Modul, Baustein	模块
Node	Knoten	节点
Non-exclusive license	Einfache Lizenz	非独占许可, 非专有许可

Non-profit	Gemeinnützig	公益，非盈利
Non-repudiation	Nichtabstreitbarkeit	不可否认性
Normal case	Normalfall	正常情况
Notification	Benachrichtigung	通知
Objective	Ziel	目标
Operability	Bedienbarkeit	易操作性
Operational processes	Betriebsprozesse (von Software)	(软件的)操作过程
Pattern	Muster	模式
Pattern language	Mustersprache, Musterfamilie	模式语言
Performance Efficiency	Leistungseffizienz, Performance	性能效率
Perspective	Perspektive	视角
Portability	Portierbarkeit	可移植性
Principle	Prinzip, Konzept	原理
Quality attribute	Qualitätsmerkmal, Qualitätseigenschaft	质量属性
Qualitycharacteristic	Qualitätsmerkmal, Qualitätseigenschaft	质量特性

Quality feature	Qualitätsmerkmal, Qualitätseigenschaft	质量特征
Rationale	Begründung, Erklärung	基本原理
Real-time system	Echtzeitsystem	实时系统
Recoverability	Widerherstellbarkeit	易恢复性
Registered trademark	Marke (gesetzlich geschützt)	注册商标
Relationship	Beziehung	关系
Relationship (kind of)	Beziehungsart	关系(类型)
Reliability	Zuverlässigkeit	可靠性
Replaceability	Austauschbarkeit	易替换性
Repository	Ablage	存储库
Requirement	Anforderung	需求
resilient	unverwüstlich, selbstwiederherstellend	弹性
Resolution	Beschluss	决议
ResourceUtilization	Ressourcenverbrauch	资源利用性
Responsibility	Verantwortlichkeit	责任
responsive	reaktionsfähig	响应的
Reusability	Wiederverwendbarkeit	易复用性
Rightsof use	Nutzungsrecht	使用权
Runtime	Laufzeit	运行时

Runtime view	Laufzeitsicht	运行时视图
Security	Sicherheit	信息安全性
Security Goals	Schutzziele, Sachziele	信息安全目标
Skill	Fähigkeit, Fertigkeit	能力, 技巧
Specification (of software architecture)	Beschreibung (von Softwarearchitektur)	(软件架构的) 规范说明
sponsoring (board) member	materiell förderndes Mitglied	赞助成员
statutory	satzungsgemäß	符合章程的
Structure	Struktur	结构
Task	Aufgabe	任务
Team regulations	Arbeitsgruppenordnung	工作组条例, 团队条例
Technical context	Technischer Kontext	技术上下文
Term	Begriff	术语
Testability	Testbarkeit	易测试性
Thriftness	Sparsamkeit, Wirtschaftlichkeit	节约, 经济性
Time Behaviour	Zeitverhalten	时间行为, 时间特性
Tools	Arbeitsmittel, Werkzeug	工具
Tools-and-material-approach	Werkzeug-Material-Ansatz	工具-材料-方法
Tradeoff	Kompromiss, Abwägung, Wechselwirkung	权衡, 平衡
Training provider	Schulungsanbieter	培训单位, 培训机构、培训供应商
Treasurer	Schatzmeister	出纳员, 会计

Ubiquitous language	Allgegenwärtige Sprache	通用语言
Usability	Benutzbarkeit, Benutzerfreundlichkeit	易用性
User Error Protection	Schutz vor Fehlbedienung	用户差错防御性
User Interface Aesthetics	Ästhetikder Benutzeroberfläche	用户界面舒适性
Uses relationship	Benutzt-Beziehung, Nutzungsbeziehung	使用关系
View	Sicht, Architektursicht	视图
Workflow management	Ablaufsteuerung	工作流程管理
Working environment	Arbeitsumgebung	工作环境
Working group	Arbeitsgruppe	工作组
Working group head	Arbeitsgruppenleiter	工作组组长

## German to English 德英中对照

在本节中，我们收集了iSAQB®从德语到英语和中文的术语翻译。



请注意：该翻译表不一定完整，一些英文术语将不会被翻译，而是最好尽量其原始语言（例如，许多设计模式的名字）。

实际的翻译表是从翻译/translations目录中的JSON文件生成的。

German 德文	English 英文	中文
Abhängigkeit	Coupling, Dependency	依赖
Abhängigkeit (zwischen Entwurfsentscheidungen)	Interdependency (between design decisions)	(设计决策之间的)相互依赖性
Ablage	Repository	存储库
Ablaufsteuerung	Workflow management	工作流程管理
Abwägung	Tradeoff	权衡，平衡
Adaptierbarkeit	Adaptability	适应性
Akkreditierungsgebühr	Accreditation fee, License fee	认证费、许可费用
Akkreditierungsvertrag	Accreditation contract, Licensing agreement	认证合同；许可协议，许可约定
Allgegenwärtige Sprache	Ubiquitous language	通用语言
Amtsgericht	Local court	地方法院
Analysierbarkeit	Analysability	可分析性
Anforderung	Requirement	需求
Angemessenheit	Adequacy, Appropriateness, Commensurability	充分性；适合性；可公度性、可通约性
Angriffsbäume	Attack Tree	攻击树
Anpassung	Adaption	适应
Ansatz	Approach	方法
Arbeitsgruppe	Working group	工作组
Arbeitsgruppenleiter	Working grouphead	工作组组长
Arbeitsgruppenordnung	Team regulations	工作组条例，团队条例
Arbeitsmittel	Tools	工具
Arbeitsumgebung	Working environment	工作环境
Architekturanalyse	Architecture assessment, Architecture evaluation	架构评估；架构评价
Architekturbewertung	Architecture assessment, Architecture evaluation	架构评估、架构评价
Architekturmuster	Architectural pattern	架构模式
Architektursicht	Architectural view, View	架构视图；视图
Architekturziel	Architectural objective,	架构目标



	Architecture objective	
Artefakt	Artifact	工件、制品
Aspekt	Aspect	方面
Aufgabe	Task	任务
Austauschbarkeit	Replaceability	易替换性

Authentifizierbarkeit	Authenticity	真实性
Barrierefreiheit	Accessibility	易访问性
Baustein	Building block, Component, Module	构建块；组件；模块
Bausteinsicht	Building blockview	构建块视图
Bedienbarkeit	Operability	易操作性
Begriff	Term	概念
Begründung	Rationale	基本原理
Begutachtung	Assessment	评估
Belang	Aspect, Concern	方面；关注点
Benachrichtigung	Notification	通知
Benutzbarkeit	Usability	易用性
Benutzerfreundlichkeit	Usability	易用性
Benutzt-Beziehung	Uses relationship	使用关系
Beschluss	Resolution	决议
Beschreibung (von Softwarearchitektur)	Specification (of software architecture)	(软件架构的) 规格说明
Beschreibungsmittel	Means for describing, Means for documenting	描述手段；记录手段
Betriebsprozesse (von Software)	Operational processes	(软件的) 操作过程
Bewertung	Assessment, Evaluation	评估，评价
Beziehung	Association, Dependency, Relationship	关联；依赖性；关系
Beziehungsart	Relationship (kindof)	关系(类型)
Domäne	Business	业务
Echtzeitsystem	Real-time system	实时系统
EinfacheLizenz	Non-exclusive license	非独占许可，非专有许可
Einflussfaktor	Influencing Factor	影响因素
Eingebettet	Embedded	嵌入式
Einhaltung	Compliance	履约，合规性
Einordnung (eines Begriffes) in einen Zusammenhang	Context (of a term)	(关于一个定义)的上下文，周境
Einschränkung	Constraint	限制，约束
Einschätzung	Assessment	评估

Entwurf	Design	设计
Entwurfsansatz	Design approach	设计方法
Entwurfsentscheidung	Design decision	设计决策
Entwurfsmethodik	Design approach	设计方法论
Entwurfsprinzip	Design principle	设计原则
Erfüllung	Compliance	履约，合规性
Erkennbarkeit der Brauchbarkeit	Appropriateness Recognizability	可辨识性
Erklärung	Rationale	基本原理
Erlernbarkeit	Learnability	易学性

Fachdomäne	Domain	域，领域
fachlicheArchitektur	Business architecture, Domain-related architecture	业务架构；与领域相关的架构
Fachlicher Bereich	Domain	域，领域
Fachlicher Kontext	Business context	业务上下文
Fachliches Event	Domain event	域事件，领域事件
Fachlichkeit	Business	业务
Fehlertoleranz	Fault Tolerance	容错性
Fertigkeit	Skill	能力，技巧
Fitnessfunktion	Fitness Function	适应度函数
Funktionale Angemessenheit	Functional Appropriateness	功能适合性
Funktionale Eignung	Functional Suitability	功能性
Funktionale Korrektheit	Functional Correctness	功能正确性
Funktionale Vollständigkeit	Functional Completeness	功能完备性
Fähigkeit	Skill	能力，技巧
Gebührenordnung	Fees rulesand regulations	收费条例，收费标准
Geheimnisprinzip	Information hidingprinciple	信息隐藏原则
Gemeinnützig	Non-profit	公益，非盈利
Geschäftsarchitektur	Business architecture	业务架构
Geschäftsbereich	Domain	域，领域
innererZusammenhalt	Cohesion	内聚性
Installierbarkeit	Installability	易安装性
Integrität	Integrity	完整性
Interoperabilität	Interoperability	互操作性
Kanal	Channel	渠道
Kapazität	Capacity	能力
Kapselung	Encapsulation	封装
Knoten	Node	节点
Koexistenz	Co-Existence	共存性

Kohäsion	Cohesion	内聚性
Kompatibilität	Compatibility	兼容性
Komponente	Component, Module	组件；模块
Kompromiss	Tradeoff	权衡，平衡
Kontextabgrenzung	Context view	上下文/场景视图，周境视图
Kontextgrenze	Bounded Context	限界上下文
Konzept	Principle	原理
Kopplung	Coupling	耦合性
Laufzeit	Runtime	运行时
Laufzeitsicht	Runtime view	运行时视图
Lehrplan	Curriculum	课程大纲、教学计划

Leistungseffizienz	Performance Efficiency	性能效率
Lernziel	Learning goal	学习目标
Lizenznehmer	Licensee	证书被授予人
Lizenzvereinbarung	Licensing agreement	许可协议
Lizenzvertrag	Licensing agreement	许可合同
Mal-/Zeichenprogramm	Drawing Tool	绘画/绘图工具
Marke (gesetzlich geschützt)	Registered trademark	注册商标
materiell förderndes Mitglied	sponsoring (board) member	赞助成员
Maßnahme	Action	软件演进，动作
Messbarkeit	Commensurability, Measurability	可公度性，可通约性；可测性
Mitgliederversammlung	General meeting, Members' meeting	会员大会
Modellierungswerkzeug	Modeling Tool	建模工具
Modifizierbarkeit	Modifiability	易修改性
Modul	Module	模块
Modularität	Modularity	模块化度
Muster	Pattern	模式
Musterfamilie	Pattern Family	模式系列
Mustersprache	Pattern language	模式语言
Nachrichten-zentrisch	message-driven	消息驱动
Nichtabstreitbarkeit	Non-repudiation	不可否认性
Normalfall	Normal case	正常情况
Nutzungsbeziehung	Uses relationship	使用关系
Nutzungsrecht	Rights of use	使用权
Performance	Performance	性能
Perspektive	Perspective	视角
Portierbarkeit	Portability	可移植性

Prinzip	Principle	原则
Prüfer	Examiner	监考人员，考官
Prüfling	Examinee	考生
Prüfungsaufgabe	Examination task	考试任务
Prüfungsbogen	Examination sheet	考试试卷
Prüfungsfrage	Examination question	考试题目
Prüfungsordnung	Examination rules and regulations	考试条例和规则
Qualitätseigenschaft	Quality attribute, Quality characteristic, Quality feature	质量属性；质量特性；质量特征
Qualitätsmerkmal	Quality attribute, Quality characteristic, Quality feature	质量属性；质量特性；质量特征
Querschnittlich	Cross-cutting	横切
Randbedingung	Constraint	边界条件、约束

reaktionsfähig	responsive	响应的
Rechenschaft	Accountability	可核查性
Rechnungsprüfer	Cash auditor	现金审计员
Rechnungsprüfung	Cash audit	现金审计
Reifegrad	Maturity	成熟性
Ressourcenverbrauch	Resource Utilization	资源利用性
Sachziele	Security Goals	信息安全目标
Satzungdes Vereins	Articles of association	协会章程
satzungsgemäß	statutory	符合章程的
Schatzmeister	Treasurer	出纳，会计
Schnittstelle	Interface	接口
Schnittstellenbeschreibung	Interface description	接口描述
Schnittstellendokumentation	Interface description	接口描述
Schrank (alsMetapher für Template)	Cabinet (as methaphor for template)	柜子(隐喻模板)
Schulungsanbieter	Training provider	培训机构
Schutz vor Fehlbedienung	User ErrorProtection	用户差错防御性
Schutzziele	Security Goals	安全目标
Schätzung	Estimation	估算
selbstwiederherstellend	resilient	弹性
Sicherheit	Security	信息安全性
Sicht	Architectural view, View	架构视图；视图
Sparsamkeit	Economicalness, Thriftyness	经济性

Stellvertretender Vorsitzender	Deputy chairman	副主席
Struktur	Structure	结构
Technischer Kontext	Technical context	技术上下文
Testbarkeit	Testability	易测试性
Unternehmens-IT-Architektur	Enterprise IT architecture	企业IT架构
Untersuchung	Assessment	评估
unverwüstlich	resilient	弹性
Verantwortlichkeit	Accountability, Responsibility	可核查性、责任、职责
Verbesserung	Improvement	改善
Verbesserungsmaßnahme	Improvement action	改善措施
Verein	Association	协会
Verfügbarkeit	Availability	可用性
Vergleichbarkeit	Commensurability	可公度性，可通约性
Verständlichkeit	Appropriateness Recognizability	可辨识性
Verteilung	Deployment	部署
Verteilungsartefakt	Deployment unit	部署单元
Verteilungssicht	Deployment view	部署视图

Vertraulichkeit	Confidentiality	保密性
Vorsitzender	Chairman	主席
Vorstand	Executive board	执行委员会
Wartbarkeit	Maintainability	易维护性
Wechselwirkung	Tradeoff	权衡
Werkzeug	Tools	工具
Werkzeug-Material-Ansatz	Tools-and-material-approach	工具和材料方法
Widerherstellbarkeit	Recoverability	可恢复性
Wiederverwendbarkeit	Reusability	可复用性
Wirtschaftlichkeit	Economicalness, Thriftiness	经济性，节约性
Zeitverhalten	Time Behaviour	时间行为
Zerlegung	Decomposition	分解
Zertifizierungsstelle	Certification authority, Certification body	认证机构
Ziel	Objective	目标
Zugänglichkeit	Accessibility	可访问性
Zuverlässigkeit	Reliability	可靠性
Ästhetik der Benutzeroberfläche	User Interface Aesthetics	用户界面舒适性

## References and Resources 参考文献与资源

本节包含词汇表或某一课程中引用的参考文献。

### A

- [Anderson-2008] Ross Anderson, *Security Engineering – A Guide to Building Dependable Distributed Systems*, 2nd edition 2008, John Wiley & Sons. One of the most comprehensive books about information security available.

信息安全领域最全面的书籍之一。

### B

- [Bachmann et al. 2000] Bachmann, F., L. Bass, et al.: *Software Architecture Documentation in Practice*. Software Engineering Institute, CMU/SEI-2000-SR-004.
  - [Bass et al. 2022] Bass, L., Clements, P. und Kazman, R. (2003): *Software Architecture in Practice*. 4th edition 2022, Addison-Wesley.
- 尽管标题另有说明，但这是一本相当基础（有时也是抽象的）的书。作者在超大型（通常是军事）系统方面有着深厚的背景，因此他们的建议有时可能与小型或精简型项目相冲突。
- [Buschmann+1996] Buschmann, Frank/Meunier, Regine/Rohnert, Hans/Sommerlad, Peter: *A System of Patterns: Pattern-Oriented Software Architecture 1*, 1st edition, 1996, John Wiley & Sons.

也被称为POSA-1。它很可能是关于架构模式的最著名和最具开创性的图书。

### C

- [Clements et al. 2003] Clements, P., F. Bachmann, L. Bass, D. Garlan, J. Ivers et al.: *Documenting Software Architectures – Views and Beyond*. Addison Wesley, 2003.

### E

- [Evans-2004] Evans, Eric: *Domain-Driven Design: Tackling Complexity in the Heart of Software*, 1st edition, Addison-Wesley, 2004.

### F

- [Ford+2017] Neil Ford, Rebecca Parsons, Patrick Kua: *Building Evolutionary Architectures: Support Constant Change*. O'Reilly 2017

### G

- [GoF:设计模式] Gamma, Erich/Helm, Richard/Johnson, Ralph/Vlissides, John M.

Design Patterns: Elements of Reusable Object-Oriented Software, 1st edition, 1994, Addison-Wesley, 1994.

设计模式领域的经典之作。

- [Gang-of-Four, short: GoF]出自[GoF:设计模式]

## H

- [Hargis 2004] Hargis, Gretchen et al.: Quality Technical Information: A Handbook for Writers and Editors. Prentice Hall, IBM Press, 2004.
- [Hofmeister+2000] Hofmeister, Christine/Nord, Robert/Soni, Dilip]]]: Applied Software Architecture, 1st edition, Addison-Wesley, 1999

## I

- [ISO-25010] ISO/IEC DIS 25010, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model. Online: link:<https://www.iso.org/obp/ui/#iso:std:iso-iec:25010>

## K

- [Kazman+1996] Kazman, R., Abowd, G., Bass, L., & Clements, P.: *Scenario-based analysis of software architecture*, IEEE software, 13(6), 47–55, 1996.
- [Kruchten 1995] Kruchten, P.: Architectural Blueprints - The 4-1 View Model of Architecture. IEEE Software November 1995; 12(6), p. 42–50.

## L

- [Lilienthal-2019] Lilienthal, Carola: *Langlebige Software-Architekturen: Technische Schulden analysieren, begrenzen und abbauen* 3rd edition, dpunkt.verlag, 2019

## M

- [Martin-2003] Martin, Robert C.: Agile Software Development: Principles, Patterns and Practices, Prentice Hall, 2003
- [SOLID-原则] Martin, Robert: SOLID-principles. S.O.L.I.D是Robert C.Martin的前五个面向对象设计原则的缩写。一些原始论文已被转移到不同的位置-参见[维基百科](#)
- [McGraw-2006] Garry McGraw, "Software Security - Building Security In", Addison-Wesley 2006, 通过风险管理、代码审查、风险分析、渗透测试、安全测试滥用案例开发, 从安全角度涵盖软件设计的全过程。

## P

- [Parnas-1972] Parnas, David: On the criteria to be used in decomposing systems into modules", Communications of the ACM, volume 15, issue 12,

Dec 1972. 软件工程领域有史以来最具影响力的文章之一，介绍了封装和模块化。  
谢谢你，大卫！

## R

- [RMIAS-2013] Yulia Cherdantseva, Jeremy Hilton, A Reference Model of Information Assurance & Security, 2013 Eight International Conference on Availability, Reliability and Security (ARES), DOI: 10.1109/ARES.2013.72, <http://users.cs.cf.ac.uk/Y.V.Cherdantseva/RMIAS.pdf>: Yulia Cherdantseva和Jeremy Hilton为RMIAS准备的会议文件。
- [Rozanski&Woods 2011] Eoin Woods and Nick Rozanski: Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives. 2nd edition 2011, Addison-Wesley. 呈现一组架构视点和架构视角。

## S

- [Schmidt, Douglas C/Stal, Michael/Rohnert, Hans/Buschmann, Frank.] Pattern-Oriented Software Architecture, volume 2: Patterns for Concurrent and Networked Objects, Wiley & Sons, 2000
- [Schneier, Bruce] Applied Cryptography, 2nd Edition 1996, John Wiley & Sons. 现代密码学综述。
- [Starke 2019] Starke, G. Effektive Software-Architekturen – Ein praktischer Leitfaden. 9. Auflage 2019, Carl Hanser Verlag.

## T

- [Tanenbaum+2016] Andrew Tanenbaum, Maarten van Steen: Distributed Systems, Principles and Paradigms, 2016. <https://www.distributed-systems.net/>
- [Tornhill-2015] Adam Tornhill: Your Code as a Crime Scene. Use Forensic Techniques to Arrest Defects, Bottlenecks, and Bad Design in Your Programs. Pragmatic Programmers, 2015. <https://www.adamtornhill.com/>



## Appendix 附录

### The iSAQB® e. V. Association 国际软件架构认证委员会 (iSAQB®)

国际软件架构认证委员会 (iSAQB®, <http://isaqb.org>) 是一个非营利组织，成员来自工业届、发展开发和咨询公司、教育、学术界和其他组织机构。

它是根据德国法律成立的一个协会，目标如下：

- 创建和维护面向软件架构师的一致性课程体系。
- 根据各种CPSA®课程定义认证考试。
- 确保软件架构师的高质量教学。
- 确保软件架构的高质量认证。

iSAQB®定义并规定了培训和考试规则，但本身不进行任何培训或考试。iSAQB®培训由（获得许可的）培训和考试机构进行。

iSAQB®监管和审计这些培训和所有相关流程（如认证程序）的质量。

# About the Authors 关于作者

## Gernot Starke 格诺特·斯塔克

Gernot Starke博士（[INNOQ](#)研究员）是（开源）[arc42](#)文档模板的联合创始人和忠实用户。在20多年的时间里，他担任软件架构师、辅导教师和顾问，攻克了不同行业客户在创建有效软件架构中面临的挑战。

2008年，Gernot参与共同创立了国际软件架构认证委员会（[iSAQB®](#)），并从那时起作为活跃成员为其提供支持。

Gernot撰写了若干本关于软件架构和相关主题的（德语）书籍，并发起了本词汇表。

他和妻子(Cheffe Uli)住在科隆。

## Ulrich Becker 乌尔里希·贝克尔

Ulrich Becker在[Method Park](#)担任首席顾问，专注于软件架构和应用程序生命周期管理。

Ulrich曾在汉堡大学和埃尔兰根-纽伦堡大学学习计算机科学。2003年，他因在基于模型的分布式配置方面的工作获得了埃尔兰根-纽伦堡大学的博士学位。随后，他成为弗劳恩霍夫集成电路研究所（[Fraunhofer IIS](#)）自适应系统软件小组的组长。

自2005年以来，Ulrich在Method Park担任培训师、顾问和辅导教师，支持客户改进开发流程和方法。他的大多数客户来自汽车行业或其他监管严格的行业。

Ulrich是[iSAQB® e.V.](#)的创始成员，为基础级和高级工作组做出了贡献。他和家人住在爱尔兰根。

## Matthias Bohlen 马蒂亚斯·鲍伦

Matthias Bohlen是高效产品开发方面的[独立专家](#)，1980年开始从事软件开发工作。他为摩托罗拉的MC68020处理器编写了编译器，该处理器在当时还没有IBM PC的时代是一款革命性的设备。这款编译器的销量在当时非常好。

从那以后，Matthias与无数软件团队合作，帮助他们推出能够正常运行的软件，同时避免陷入困境。截至今天，他仍在做同样的事情。

Matthias是[国际软件架构认证委员会](#)的活跃成员，不仅撰写[博客](#)，还在精益/敏捷领域广为人知，并曾经在各类软件开发大会上发表演讲。

## Phillip Ghadir 菲利普·加迪尔

Phillip是INNOQ Deutschland GmbH的董事会成员。多年以来他为来自各个行业的客户提供软件架构、技术和开发方面的咨询服务。他是[iSAQB®](#)的联合创始人，并定期举办软件架构方面的培训。

## Carola Lilienthal 卡罗拉·利林塔尔

Carola Lilienthal博士是[WPS Workplace Solutions](#)的软件架构师和联合创始人。20年来，她一直担任开发人员、项目经理、辅导教师、顾问和架构师。Carola是领域驱动设计和敏捷开发活动的早

期倡导者，并成功地为来自各个领域的众多客户提供服务，主要涵盖金融、保险和物流领域。

自2003年以来，她一直致力于分析基于Java、C++、C#、PHP、ABAP编写的软件系统，并为开发团队提供如何提高代码可持续性方面的建议。Carola定期在各类会议上发表演讲，并撰写了各种文章以及一本关于可持续软件架构的书。

自2008年以来，Carola一直是国际软件架构认证委员会(iSAQB®)的活跃成员，并为其提供支持。

### **Mahbouba Gharbi 马赫布巴·加尔比**

Mahbouba Gharbi是iTech Progress的首席信息官、图书作者和各类会议发言人。

几年前，Mahbouba成为iSAQB的主席。目前她和家人住在曼海姆。

### **Simon Kölsch 西蒙·柯尔施**

Simon Kölsch是INNOQ的高级顾问，专注于网络架构和信息安全。

Simon热衷于传统的单体企业应用程序之外的解决方案，涵盖分布式系统及其基础设施的架构、日志记录和监控。

他不局限于某一特定的技术或编程语言，但有很强的JVM背景。

### **Alexander Lorz 亚历山大·洛兹**

Alexander Lorz博士是一名自由职业软件架构培训师、IT顾问和开发人员。他第一次接触IT系统可以追溯到20世纪80年代中期，从那以后，他一直拒绝放弃对于开发复杂系统的科学和工艺的迷恋。

作为国际软件架构认证委员会(iSAQB®)及其基础级工作组的活跃成员，他为基础级课程的发展做出了贡献。

### **Michael Mahlberg 迈克尔·马赫伯格**

Michael Mahlberg在德国经营着自己的公司method consultancy，大部分时间都在帮助客户寻求更有效的工作方法。他主要通过应用精益和敏捷的理念来提供方法。

他从18岁开始经营自己的计算机和软件相关公司，很快就意识到软件架构和（开发）流程在某种程度上是一门永恒的技艺。

现在，他的许多工作都集中在流程和人际交往方面，他既从事专业工作，也热心公益活动（例如，他是Limited WIP Society Cologne协会的发起和负责人之一）。

因此，Michael的架构工作倾向于处理架构和流程决策对彼此的影响和作用，以及相关的优化策略。

### **Andreas Rausch 安德列亚斯·劳施**

Andreas Rausch博士是克劳斯塔尔工业大学软件系统工程的终身教习教授。

他师从Manfred Broy教授并于2001在慕尼黑工业大学获得博士学位。他在软件系统工程领域的主要研究兴趣包括软件架构、基于模型的软件开发和过程模型。并在这些领域发表了300多篇国际论文。

## Roger Rhoades 罗杰·罗阿德

Roger Rhoades是德国[埃尔比翁](#)培训和咨询公司的创始人。

Roger在企业、商业和软件架构以及国际团队和项目管理领域拥有超过25年的实践经验。这种实践经验被整合到他的培训课程中，以确保学员不仅理解理论内容，还了解其实施中的真实挑战。

自2012年以来，Roger定期在各类国际会议上发表演讲（例如EAMKon、Lean42 EAM、IT战略与治理）。

自2014年以来，Roger一直是国际软件架构资格认证委员会（iSAQB®e.V.）的活跃成员。除了iSAQB®词汇表，他还积极支持基础和高级课程、考试题目和案例研究的演进。

## Sebastian Fichtner 塞巴斯蒂安·菲希特纳

[flowtoolz.com](#)的创始人。应用程序工程师和顾问。1995年开始编码。自那时以来一直热衷于架构。为各种客户创建原创应用程序、开源项目和项目。热爱苹果平台和Swift语言。

## 关于我们的事业



**ELECTRONIC FRONTIER FOUNDATION**

**eff.org**

这本书的所有版税都将捐赠给 EFF。通过购买这本书，您支持了他们的事业：

电子前沿基金会（Electronic Frontier Foundation，简称 EFF）是领先的非营利组织，捍卫数字世界的公民自由。EFF 成立于 1990 年，通过影响诉讼、政策分析、基层活动和技术开发，捍卫用户隐私、自由表达和创新。我们致力于确保随着我们对技术的使用不断增长，权利和自由得到增强和保护。

即使在互联网发展的初期阶段，EFF 也意识到保护对新兴技术的访问对于促进所有人的自由至关重要。在接下来的几年里，EFF 通过我们独立坚定的声音为开源软件、加密、安全研究、文件共享工具和涌现技术扫清了道路。

如今，EFF 运用领先技术人员、活动家和律师的独特专业知识，努力捍卫网络言论自由，抵制非法监视，为用户和创新者发声，支持增进自由的技术。

在全球范围内，我们建立了一个庞大的关注会员和合作伙伴组织网络。EFF 向决策者提供建议，通过全面分析、教育指南、活动工作坊等方式教育新闻界和公众。EFF 通过我们的行动中心赋予成千上万的个人力量，成为在线权利辩论中的领军声音。

EFF是一个由捐助者资助的美国501（c）（3）非营利组织，我们依赖您的支持来继续为用户而战。”

（摘自[eff.org/about](https://eff.org/about)）