Syllabus

Certified Professional for Software Architecture (CPSA®)

Foundation Level

2021.1-PT-20210209

Índice remissivo

Aviso legal

A abreviação "e. V." faz parte do nome oficial da iSAQB e significa "associação registrada", que descreve seu status como uma pessoa jurídica legal de acordo com a lei alemã. Por uma questão de simplicidade, o iSAQB e.V. será referida como iSAQB daqui em diante sem o uso esta abreviação.

Índice dos objetivos de aprendizagem

- OA 4-3: Analisar e avaliar qualitativamente as arquiteturas de software (R2-R3)
- OA 4-4: Avaliar quantitativamente as arquiteturas de software (R2)
- OA 5-1: Compreender a relação entre requisitos, restrições e soluções (R3)
- OA 5-2: Compreender a lógica da implementação técnica de uma solução (R3)

Introdução

Qual é o escopo do treinamento Foundation Level?

Os treinamentos licenciados pelo *Certified Professional for Software Architecture – Foundation Level* (CPSA-F) transmitem conhecimentos e habilidades necessárias para o projeto, especificação e documentação de uma arquitetura de software adequada para sistemas de TI de pequeno e médio portes. Os participantes serão capazes de expandir e aprofundar suas experiências práticas e habilidades, aprendendo a tomar decisões sobre arquiteturas a partir da visão de sistemas existentes e requisitos adequadamente detalhados. O treinamento CPSA-F ensina métodos e princípios de projetos, documentação e avaliação de arquiteturas de software, independente do processo de desenvolvimento utilizado.

O foco do treinamento está na aquisição das seguintes habilidades:

- Discutir e negociar decisões básicas de arquitetura com outros stakeholders das áreas de gerenciamento de requisitos, gerenciamento de projetos, desenvolvimento e testes de software.
- Compreender as atividades básicas da arquitetura de software, bem como executá-las de forma independente para sistemas de pequeno e médio portes.
- Documentar e comunicar arquiteturas de software baseadas em visões e padrões arquiteturais e conceitos técnicos.

Além disso, o treinamento abrange:

- O conceito e o significado da arquitetura de software.
- As tarefas e responsabilidade dos arquitetos de software.
- O papel dos arquitetos de software em projetos de desenvolvimento.
- O estado da arte de métodos e técnicas para o desenvolvimento de arquiteturas de software.

Escopo Negativo

Este programa reflete o conteúdo considerado necessário e útil pela perspectiva atual do iSAQB e.V. para alcançar os objetivos de aprendizagem do CPSA-F. Não representa uma descrição completa da área de conhecimento "arquitetura de software".

Los siguientes temas y conceptos **no forman parte del CPSA-F**:

- Implementações específicas de tecnologias, frameworks ou bibliotecas.
- Programação ou linguagens de programação.
- Modelos de processo específicos.
- Fundamentos de notações de modelagem (como UML) ou fundamentos de modelagem em si.
- Análise de sistemas e engenharia de requisitos (por favor, consulte o programa de treinamento e certificação do IREB e. V., http://ireb.org, Conselho Internacional de Engenharia de Requisitos).
- Testes de Software (consulte o programa de treinamento e certificação do ISTQB e. V., http://istqb.org, Conselho Internacional de Qualificações para Testes de Software).
- Gerenciamento de projetos ou produtos.
- Introdução às ferramentas de software específicas.

O objetivo do treinamento é fornecer a base para aquisição de conhecimentos e habilidades exigidas para a respectiva aplicação.

Pré-requisitos

O iSAQB e. V. pode verificar os requisitos aqui mencionados nos exames de certificação, fazendo as perguntas apropriadas.

Os participantes devem ter os conhecimentos e/ou experiência mencionados abaixo. Em particular, a experiência prática substancial de desenvolvimento de software em equipe é um requisito importante para a compreensão da matéria transmitida e para o sucesso da certificação.

- Mais de 18 meses de experiência prática no desenvolvimento de software baseado na colaboração em equipes, adquirida através do desenvolvimento de diferentes sistemas, indo além de sua formação educacional.
- Conhecimento e experiência prática em, pelo menos, uma linguagem de programação, e em particular:
  - Conceitos de:
    - Modularização (pacotes, namespaces).
    - Passagem de parâmetros (chamada por valor, chamada por referência).
    - Escopo, por exemplo, declaração e definição de tipo ou de variável.
  - Fundamentos de tipagem (tipagem estática e dinâmica, tipos de dados genéricos).
  - Tratamento de erros e exceções em software.
  - Possíveis problemas de status global e variáveis globais.
- Conhecimentos básicos de:
  - Modelagem e abstração.
  - Algoritmos e estruturas de dados (por exemplo, listas, árvores, tabelas hash, dicionário/mapa).
  - UML (diagramas de classe, pacote, componentes e sequência) e suas relação com o código-fonte.

Além disso, também são úteis para a compreensão de alguns conceitos:

- Conceitos básicos e diferenças entre programação imperativa, declarativa, orientada a objetos e funcional.
- Experiência prática em:
  - Um alto nível de linguagem de programação.
  - Projeto e implementação de aplicações distribuídas, tais como sistemas cliente/servidor ou aplicativos da web.
  - Documentação técnica, em particular, documentação do código-fonte, projetos de sistemas ou conceitos técnicos.

Estrutura, duração, didática

Os tempos de duração dos estudos mencionados nos capítulos seguintes do programa são apenas recomendações. A duração de um treinamento deve ser de, pelo menos, 3 dias, mas pode ser maior. Os provedores podem diferir em estratégias de duração, didática, tipo, estrutura de exercícios e estrutura detalhada do curso. Os tipos (domínios e tecnologias) dos exemplos e exercícios podem ser determinados individualmente pelos respectivos provedores do treinamento.

Conteúdo	**Duração recomendada (min)**
1. Conceitos básicos de arquitetura de software	120
2. Projeto e desenvolvimento	420
3. Especificação e comunicação	240
4. Arquitetura de software e qualidade	120
5. Exemplos	90
Total	990

Objetivos de aprendizagem e relevância para o exame

Os capítulos do programa são estruturados de acordo com as prioridades dos objetivos de aprendizagem. A relevância de cada objetivo de aprendizagem ou de seus subitens, em relação à certificação, está explicitamente identificada (pelos indicadores R1, R2 ou R3, conforme exposto na tabela abaixo).

Em termos de relevância para o exame, o programa utiliza as seguintes categorias:

ID	Categoria do Objetivo de aprendizagem	Significado	Relevância para o exame
R1	Ser capaz de	Os participantes devem ser capazes de aplicar estes conteúdos de forma independente após o treinamento. Durante o treinamento, esses conteúdos são cobertos por exercícios e discussões.	Os conteúdos **serão** examinados.
R2	Compreender	Em princípio, os participantes devem compreender este conteúdo. Em geral, não são foco principal dos exercícios do treinamento.	Os conteúdos **podem** ser examinados.
R3	Saber	Estes conteúdos (termos, conceitos, métodos, práticas ou similares) podem aprimorar a compreensão do tópico ou motivar a aprendizagem. Eles podem ser abordados durante o treinamento, se for necessário.	Os conteúdos **não serão** examinados.

Se necessário, os objetivos de aprendizagem podem conter referências para leituras complementares, normas ou outras fontes. As seções "Conceitos essenciais" de cada capítulo listam palavras que estão relacionadas com o conteúdo do capítulo e, às vezes, são usadas nos objetivos de aprendizagem.

Versão atual e repositório público

Você encontrará a versão mais atual deste documento na página oficial de download em https://isaqborg.github.io/.

O documento é mantido num repositório público em: https://github.com/isaqb-org/curriculum-foundation. Todas as alterações e modificações são públicas.

Por favor, informe quaisquer questões no nosso rastreador de edições públicas em https://github.com/isaqborg/curriculumfoundation/issues.

1. Conceitos básicos de arquitetura de software

Duração: 120 min.	Tempo de exercícios: Nenhum.

Conceitos essenciais

Arquitetura de software; domínios de arquitetura; estrutura; blocos de construção; componentes; interfaces; relações; conceitos transversais (cross-cutting); arquitetos de software e suas responsabilidades; tarefas e habilidades necessárias; partes interessadas (stakeholders) e suas preocupações; requisitos funcionais; requisitos de qualidade; restrições; fatores influenciadores; tipos de sistemas de TI (sistemas embarcados; sistemas em tempo real; sistemas de informação, etc.)

Objetivos de aprendizagem

OA 1-1: Discutir definições de arquitetura de software (R1)

Os arquitetos de software conhecem várias definições de arquitetura de software (incluindo ISO 42010 / IEEE 1471, SEI, Booch etc.) e podem identificar suas similaridades:

- Componentes/blocos de construção com interfaces e relações.
- Blocos de construção como conceito geral, componentes com caracterização especial.
- Estruturas, conceitos transversais, princípios.
- Decisões de arquitetura e suas consequências em todo o sistema ou ciclo de vida.

OA 1-2: Entender e identificar os objetivos e benefícios da arquitetura de software (R1)

Os arquitetos de software podem garantir os seguintes objetivos e benefícios essenciais da arquitetura de software:

- Dar suporte à criação e manutenção de software, especialmente apoiando a implementação; apoiar o projeto, implementação, manutenção e operação dos sistemas.
- Atingir requisitos de qualidade como confiabilidade, mantenabilidade, modificabilidade, segurança, etc. e requisitos funcionais.
- Atingir os requisitos funcionais ou garantir que eles possam ser atendidos.
- Garantir que as estruturas e conceitos do sistema sejam compreendidos por todas as partes interessadas relevantes.
- Reduzir sistematicamente a complexidade.
- Especificar diretrizes arquiteturalmente relevantes para implementação e operação.

OA 1-3: Entender a arquitetura de software como parte do ciclo de vida do software (R2)

Os arquitetos de software entendem suas tarefas e podem integrar seus resultados em todo o ciclo de vida dos sistemas de TI. Eles são capazes de:

- Reconhecer as consequências de alterações nos requisitos funcionais, nos requisitos de qualidade, nas tecnologias ou no ambiente do sistema em relação à arquitetura do software.
- Criar conexões entre os sistemas de TI e os negócios e processos operacionais suportados.

OA 1-4: Entender as tarefas e responsabilidade dos arquitetos de software (R1)

Os arquitetos de software são responsáveis por atingir a qualidade exigida ou necessária e criar o projeto de arquitetura da solução. Dependendo da abordagem ou do modelo de processo utilizado, eles devem alinhar estas responsabilidades com a responsabilidade geral do gerenciamento do projeto e/ou outros papéis.

Tarefas e responsabilidades dos arquitetos de software:

- Esclarecer, questionar e, se necessário, refinar requisitos e restrições. Além dos requisitos funcionais (características necessárias), eles incluem, em particular, as características de qualidade exigidas (restrições necessárias).
- Tomar decisões estruturais relativas à decomposição do sistema em estruturas dos blocos de construção, definindo dependências e interfaces entre estes.
- Determinar e decidir sobre conceitos transversais (por exemplo, persistência, comunicação, GUI).
- Comunicar e documentar a arquitetura de software baseada em visões, padrões de arquitetura, bem como conceitos transversais e conceitos técnicos.
- Acompanhar a realização e implementação da arquitetura, incorporando o feedback dos stakeholders conforme necessário, verificando e assegurando a consistência do código-fonte e da arquitetura de software.
- Analisar e avaliar a arquitetura de software, especialmente em relação aos riscos relativos aos atributos de qualidade exigidos.
- Reconhecer, destacar e justificar as consequências das decisões de arquitetura para os stakeholders.

Eles devem, independentemente, identificar a necessidade de iterações em todas as tarefas e apresentar possibilidades de feedback adequado e relevante.

OA 1-5: Relacionar o papel dos arquitetos de software com outras partes interessadas (stakeholders) (R1)

Os arquitetos de software devem ser capazes de explicar os seus papéis. Devem adaptar suas contribuições para o desenvolvimento de sistemas em um contexto específico e em relação a outros stakeholders, em particular para:

- Gerenciamento de produtos e Product Owner
- Gerentes de projeto
- Engenheiros de requisitos, analistas de requisitos ou negócios, gerentes de requisitos, analistas de sistemas, empresários, especialistas no assunto, etc.)
- Desenvolvedores
- Liderança e gerenciamento de projetos.
- Garantia de qualidade e testadores.
- Operadores de TI e administradores, (aplicáveis principalmente para ambiente de produção ou data centers para sistemas de informação).
- Desenvolvimento de hardware.
- Arquitetos corporativos e membros do conselho de arquitetura.

OA 1-6: Ser capaz de explicar a correlação entre abordagens de desenvolvimento e arquitetura de software (R1)

- Os arquitetos de software devem poder explicar o impacto de procedimentos iterativos nas decisões de arquitetura (no que diz respeito a riscos e previsibilidade).
- Devido à incerteza inerente, eles precisam trabalhar e tomar decisões de forma iterativa. Para isso, eles devem obter, sistematicamente, feedbacks de outras partes interessadas.

OA 1-7: Diferenciar entre objetivos de curto e longo prazos (R1)

Os arquitetos do software podem:

- Explicar os requisitos de qualidade a longo prazo e sua diferenciação dos objetivos do projeto (a curto prazo).
- Explicar potenciais conflitos entre os objetivos de curto e longo prazo, a fim de encontrar uma solução viável para todos os envolvidos.
- Identificar e especificar os requisitos de qualidade.

OA 1-8: Distinguir declarações explícitas de suposições implícitas (R1)

Arquitetos de software:

- Devem apresentar explicitamente suposições ou pré-requisitos e evitar suposições implícitas.
- Sabem que suposições implícitas causam potenciais mal-entendidos entre os stakeholders.
- Podem formular implicitamente, se for apropriado no contexto em questão.

OA 1-9: Entender as responsabilidades dos arquitetos de software dentro um contexto arquitetural maior (R3)

O foco do iSAQB CPSA Foundation Level está nas estruturas e conceitos de sistemas de software individuais.

Além disso, os arquitetos de software estão familiarizados com outros domínios de arquitetura, por exemplo:

- Arquitetura de TI corporativa: estrutura de aplicativos landscape.
- Arquitetura de negócios ou processos (Business and Process Architecture): Estrutura de, entre outros, processos de negócios.
- Arquitetura de informação: estrutura entre sistemas e uso de informações e dados.
- Arquitetura de infraestrutura ou de tecnologia: Estrutura da infraestrutura técnica, hardware, redes, etc.
- Arquitetura de hardware ou de processador (para sistemas relacionados a hardware).

Estes domínios de arquitetura não são o foco do CPSA-F.

OA 1-10: Saber diferenciar os tipos de sistemas de Tecnologia da Informação TI (R3)

Os arquitetos de software devem conhecer diferentes tipos de sistemas de TI, por exemplo:

- Sistemas de informação.
- Suporte a decisões, Data Warehouse ou sistemas de inteligência de negócios.
- Sistemas móveis.

- Processos ou sistemas em lote.
- Sistemas relacionados a hardware; aqui eles entendem a necessidade do projeto de código de hardware/software (dependências de tempo e conteúdo de projeto de hardware e software).

OA 1-11: Desafios dos sistemas distribuídos (R3)

Os arquitetos de software possuem a habilidades de:

- Identificar a distribuição em uma arquitetura de software.
- Analisar critérios de consistência para um determinado problema empresarial.
- Explicar a causalidade dos eventos num sistema distribuído.

Os arquitetos de software sabem …

- … a comunicação pode falhar num sistema distribuído.
- … limitações relativas à coerência em bases de dados do mundo real.
- … qual é o problema do "cérebro dividido" e porque é difícil.
- … que é impossível determinar a ordem temporal dos acontecimentos num sistema distribuído.

Referências

[Bass+ 2012], [Gharbi+2020], [iSAQB References], [Starke 2020], [vanSteen+Tanenbaum]

2. Projeto e desenvolvimento de arquiteturas de software

Duração: 330 min.	Tempo de exercícios: 90 min.

Conceitos essenciais

Projeto; abordagens do projeto; decisão de projeto; visões; interfaces; conceitos técnicos e transversais; padrões de arquitetura; padrões de linguagem; princípios de projeto; dependências; acoplamento; coesão; arquiteturas funcionais e técnicas; abordagens *top-down* e *bottom-up*; projeto baseado em modelos; projeto iterativo/incremental; projeto orientado por domínio

Objetivos de aprendizagem

OA 2-1: Ser capaz de selecionar e aplicar procedimentos e heurísticas para o desenvolvimento de arquiteturas (R1, R3)

Os arquitetos de software podem nomear, explicar e aplicar abordagens básicas de desenvolvimento de arquiteturas, por exemplo:

- Abordagens de projeto top-down e bottom-up. (R1)
- Desenvolvimento de arquiteturas baseado em visões. (R1)
- Projeto iterativo e incremental. (R1)
  - Necessidade de iterações, em especial em caso de incerteza quanto a tomada de decisão. (R1)
  - Necessidade de feedback sobre as decisões de projeto. (R1)
- Projeto orientado a domínio, ver [Evans 2004] (R3)
- Arquitetura orientada a modelos, ver [Ford 2017] (R3)
- Análise global, ver [Hofmeister et. al 1999] (R3)
- Arquitetura-orientada a modelo (R3).

OA 2-2: Projetar arquiteturas de software (R1)

Os arquitetos do software são capazes de:

- Projetar, comunicar e documentar apropriadamente arquiteturas de software com base em requisitos funcionais e de qualidade para sistemas de software que não são críticos do ponto de vista de segurança ou negócios;
- Tomar decisões estruturais relativas à decomposição do sistema e de estrutura dos blocos de construção e projetar dependências entre esses blocos.
- Reconhecer e justificar interdependências e trade-offs sobre decisões de projeto.
- Explicar os conceitos caixa-preta (Blackbox) e caixa-branca (Whitebox) e aplicá-los apropriadamente.
- Realizar refinamento gradual e especificar dos blocos de construção.
- Projetar visões de arquitetura, especialmente visão de blocos de construção, tempo de execução e distribuição.
- Explicar as consequências resultantes dessas decisões no código-fonte.

- Separar elementos técnicos e relacionados a domínios de arquiteturas e justificar essa separação.
- Identificar os riscos relacionados às decisões sobre a arquitetura.

OA 2-3: Identificar e considerar fatores que influenciam a arquitetura de software (R1-R3)

Os arquitetos de software são capazes de reunir e considerar restrições e fatores de influência que limitam as suas decisões. Compreendem que as suas decisões podem gerar requisitos e restrições adicionais para o sistema a ser concebido, a sua arquitetura, ou o processo de desenvolvimento. (R1-R2)

Devem reconhecer e levar em consideração do impacto de:

- Fatores relacionados com o produto, tais como (R1):
  - Requisitos funcionais.
  - Requisitos de qualidade e objetivos de qualidade.
  - Fatores adicionais como o custo do produto, modelo de licenciamento pretendido, ou modelo de negócio do sistema.
- Fatores tecnológicos tais como (R1-R3):
  - Decisões e conceitos técnicos mandatados externamente. (R1)
  - Infraestruturas de hardware e software existentes ou planejadas. (R1)
  - Restrições tecnológicas sobre estruturas de dados e interfaces. (R2)
  - Arquiteturas de referência, bibliotecas, componentes e estruturas. (R1)
  - Linguagens de programação. (R3)
- Fatores organizacionais tais como:
  - Estrutura organizacional da equipe de desenvolvimento e do cliente. (R1)
  - Cultura da empresa e da equipe. (R3)
  - Acordos de parceria e cooperação. (R2)
  - Normas, diretrizes e modelos de processo (por exemplo, processos de aprovação e libertação). (R2)
  - Recursos disponíveis como orçamento, tempo e pessoal. (R1)
  - Disponibilidade, conjunto de competências e empenho do pessoal. (R1)
- Fatores regulatórios tais como (R2):
  - Restrições legais locais e internacionais.
  - Questões contratuais e de responsabilidade.
  - Leis de proteção de dados e privacidade.
  - Questões de conformidade ou obrigações de fornecer o ônus da prova.
- Tendências tais como (R3):
  - Tendências de mercado.
  - Tendências tecnológicas (por exemplo, Blockchain, microsserviços).
  - Tendências metodológicas (e.g. ágil).
  - (Potencial) impacto de outras preocupações das partes interessadas e decisões obrigatórias do projeto. (R3)

Os arquitetos de software são capazes de descrever como esses factores podem influenciar as

decisões de projeto e podem detalhar as consequências da mudança dos fatores de influência, fornecendo exemplos para alguns deles. (R2)

OA 2-4: Projetar e implementar conceitos transversais (cross-cutting) (R1)

Os arquitetos de software são capazes de:

- Explicar a importância dos conceitos transversais;
- Decidir e projetar conceitos transversais, por exemplo, persistência, comunicação, GUI, tratamento de erros, simultaneidade;
- Identificar e avaliar as possíveis interdependências entre essas decisões.

Os arquitetos de software sabem que tais conceitos transversais podem ser reutilizados em todo o sistema.

OA 2-5: Descrever, explicar e aplicar adequadamente padrões de arquitetura importantes (R1, R3)

Os arquitetos de software sabem:

- Vários padrões de arquitetura e podem aplicá-los quando apropriado.
- Que os padrões são uma forma de alcançar determinadas qualidades para determinados problemas e requisitos dentro de determinados contextos.
- Que existem várias categorias de padrões. (R3)
- Fontes adicionais para padrões relacionados com o seu domínio técnico ou de aplicação específico. (R3)

Os arquitetos de software podem explicar e dar exemplos dos seguintes padrões (R1):

- Camadas:
  - Camadas de abstração escondem detalhes, exemplo: Camadas de rede ISO/OSI, ou "camada de abstração de hardware". Ver: https://en.wikipedia.org/wiki/Hardware_abstraction
  - Outra interpretação são Camadas para separar (fisicamente) funcionalidade ou responsabilidade. Ver: https://en.wikipedia.org/wiki/Multitier_architecture
- Tubos-e-Filtro: Representativo para padrões de fluxo de dados, decompondo o processamento por etapas numa série de actividades de processamento ("Filtro") e capacidades associadas de transporte/buffering de dados ("Pipes").
- Microsserviços dividem a aplicação em executáveis separados que comunicam via rede.
- Injeção de Dependência como solução possível para o Princípio da Dependência-Inversão.

Os arquitetos de software podem explicar vários dos seguintes padrões, explicar a sua relevância para sistemas concretos, e fornecer exemplos. (R3)

- Quadro negro: lidar com problemas que não podem ser resolvidos por algoritmos determinísticos mas que requerem diversos conhecimentos.
- Corretor: responsável pela coordenação da comunicação entre fornecedor(es) e consumidor(es), aplicado em sistemas distribuídos. Responsável pelo encaminhamento de pedidos e/ou transmissão de resultados e excepções.
- Combinador (sinônimo: encerramento de operações), para objeto de domínio de tipo T, procure operações tanto entrada, como saída do tipo T. Ver: [Yorgey 2012]

- CQRS (Command-Query-Responsibility-Segregation): Separa as preocupações de leitura da de escrita em sistemas de informação. Requer algum contexto sobre a tecnologia de base de dados/persistência para compreender diferentes propriedades e requisitos das operações de "ler" versus "escrever.
- Fonte de eventos: tratar operações sobre dados através de uma sequência de eventos, cada um dos quais é registrado numa loja apenas de anexos.
- Intérprete: representar o objeto de domínio ou DSL como sintaxe, fornecer função implementando uma interpretação semântica do objeto de domínio separadamente do objeto de domínio em si.
- Integração e padrões de mensagens (por exemplo, de Hohpe+2004]).
- A família de padrões MVC, MVVM, MV-Update, PAC, separa a representação externa (view) de dados, serviços e sua coordenação.
- Padrões de interface como Adaptador, Fachada, Proxy. Tais padrões ajudam na integração de subsistemas e/ou simplificação das dependências. Os arquitetos devem saber que estes padrões podem ser utilizados independente da tecnologia (objeto).
    - Adaptador: dissociar consumidor e fornecedor - onde a interface do fornecedor não é exatamente corresponder ao do consumidor. O Adaptador desacopla uma parte das alterações de interface no outro.
    - Facade: simplifica a utilização de um fornecedor para o(s) consumidor(es), fornecendo um acesso simplificado.
    - Proxy: Um intermediário entre consumidor e fornecedor, permitindo o desacoplamento temporal, armazenamento em cache, controlando o acesso ao fornecedor, etc.
- Observador: um produtor de valores ao longo do tempo notifica um quadro de distribuição central onde os consumidores podem se registrar para ser notificado sobre eles
- Plug-In: ampliar o comportamento de um componente.
- Portas&Adaptadores (syn. Onion-Architecture, Hexagonal-Architecture): concentrar a lógica do domínio no centro do sistema, ter conexões com o mundo exterior (base de dados, UI) nos limites, dependências só de fora para dentro, nunca de dentro para fora.
- Chamada de procedimento remoto: faz uma função ou algoritmo executar num espaço de endereçamento diferente.
- SOA: Arquitetura orientada a serviços. Uma abordagem para fornecer serviços abstratos em vez de Implementações concretas aos usuários do sistema para promover a reutiização dos serviços entre departamentos e empresas.
- Modelo e Estratégia: tornar os algoritmos específicos flexíveis encapsulando-os
- Visitante: separar dados estruturados transversais do processamento específico.

Os arquitetos de software conhecem fontes essenciais para padrões arquitetônicos, como POSA (por exemplo [Buschmann+1996]) e PoEAA ([Fowler 2002]) (para sistemas de informação). (R3)

OA 2-6: Explicar e aplicar princípios de projeto concepção (R1-R3)

Os arquitetos de software são capazes de explicar o que são princípios de concepção do projeto. Eles podem delinear os seus objetivos gerais e aplicações no que diz respeito à arquitetura de software. (R2)

Com relevância para o exame, dependendo do princípio específico listado abaixo, os arquitetos de software são hábeis em:

- Explicar los principios de diseño enumerados a continuación y de ilustrarlos mediante ejemplos.

- Explicar os princípios de concepção listados abaixo e podem ilustrá-los com exemplos.
- Explicar como esses princípios devem ser aplicados.
- Explicar como os requisitos de qualidade determinam os princípios que devem ser aplicados.
- Explicar o impacto dos princípios de concepção na implementação.
- Analisar projetos de código fonte e arquitetura para avaliar se estes princípios de concepção foram aplicados ou devem ser aplicados.

Abstração (R1)

- No sentido de um meio para derivar generalizações úteis.
- Como uma construção de desenho, onde os blocos de construção dependem das abstracções em vez de depender de implementações.
- Interfaces como abstrações.

Modularização (R1-R3)

- Ocultação e encapsulamento de informação. (R1)
- Separação de preocupações – SoC. (R1)
- Acoplamento (R1) de blocos de construção soltos, mas funcionalmente suficientes.

  Ver: OA 2-7.
- Alta coesão. (R1)
- Princípios SOLID (R1-R3), que têm, em certa medida, relevância no nível de arquitetura:

Single	• S:	Princípio da responsabilidade única (R1) e a sua relação com SoC.
Open/closed	• O:	Princípio aberto/fechado. (R1)
Liskov	• L:	Princípio de substituição Liskov (R3) como forma de alcançar consistência e integridade conceitual em desenho.
Interface segregation	• I:	Princípio de segregação da interface (R2), incluindo a sua relação com a OA 2-9
Dependency inversion	• D:	Princípio de inversão de dependência (R1) por meio de interfaces ou abstrações similares.

Integridade conceitual (R2)

- Significa uniformidade (homogeneidade, consistência) de soluções para problemas semelhantes. (R2)
- Como meio para alcançar o princípio da menor surpresa. (R3)

Simplicidade (R1-R2)

- Como meio para reduzir a complexidade. (R1)

- Como fator de condução por trás e KISS (R1) e YAGNI. (R2)

Erros de expectativa (R1-R2)

- Como meio de conceber sistemas robustos e resilientes. (R1)
- Como uma generalização do princípio da robustez, também conhecido por lei Postel. (R2)

OA 2-7: Gerenciar dependências entre blocos de construção (building blocks) (R1)

Os arquitetos de software entendem dependências e acoplamento entre os blocos de construção e podem aplicá-los de forma direcionada:

- Conhecem e compreendem diferentes tipos de dependências entre blocos de construção (por exemplo, acoplamento estrutural através de uso/delegação, mensagens/eventos, composição, criação, geração, herança, acoplamento temporal, através de dados, tipos de dados ou hardware).
- Entendem como as dependências aumentam o acoplamento.
- Podem utilizar esses tipos de acoplamento de forma direcionada e avaliar as consequências de tais dependências.
- Conhecem possibilidades de eliminar ou reduzir o acoplamento e podem aplicá-las, por exemplo:
  - Padrões (ver: OA 2-5).
  - Princípios básicos de design (ver: OA 2-6).
  - Externalização de dependências, ou seja, definição de dependências concretas em tempo de instalação ou execução, por exemplo, através de Injeção de Dependência.

OA 2-8: Atingir requisitos de qualidade com abordagens e técnicas adequadas (R1)

Os arquitetos de software compreendem e consideram a forte influência dos requisitos de qualidade nas decisões de arquitetura e design, por exemplo para:

- Eficiência / desempenho.
- Disponibilidade.
- Mantenabilidade, modificabilidade, expansibilidade, adaptabilidade.

Eles são capazes de:

- Explicar e aplicar possíveis soluções, táticas de projeto arquitetural, práticas adequadas e possibilidades de técnicas para atingir critérios de qualidade importantes para os sistemas de software (diferentes para sistemas embarcados ou sistemas de informação).
- Identificar e comunicar as possíveis compensações entre tais soluções e os riscos relacionados.

OA 2-9: Projetar e definir interfaces (R1-R3)

Os arquitetos de software conhecem a grande importância das interfaces. Eles são capazes de projetar ou definir interfaces entre blocos de construção de arquiteturas bem como interfaces externas entre o sistema e elementos externos a ele.

Eles conhecem:

- As características desejáveis das interfaces e podem aplicá-las nos projetos:
  - Fáceis de aprender, fáceis de usar, fáceis de estender.
  - Difíceis de usar indevidamente.
  - Funcionalmente completas do ponto de vista do usuário ou dos blocos de construção.
- A necessidade de tratar as interfaces internas e externas de forma diferente.
- Diferentes abordagens de implementação de interfaces (R3):
  - Abordagem orientada a recursos (REST, Representational State Transfer).
  - Abordagem orientada a serviços (como em serviços da Web baseados em WS*/SOAP).

Referências

[Bass+2012], [Fowler 2002], [Gharbi+2020], [Gamma+94], [Martin 2003], [Buschmann+1996] e [Buschmann+2007], [Starke 2020], [Lilienthal 2018]

3. Especificação e comunicação de arquiteturas de software

Duração: 180 min.	Tempo de exercícios: 60 min.

Conceitos essenciais

Visões (arquiteturais); estruturas; conceitos (técnicos); documentação; comunicação; descrição; orientação a partes interessadas (stakeholders); meta-estruturas e modelos para descrição e comunicação; delimitação do contexto do sistema; blocos de construção; visão de blocos de construção; visão de tempo de execução; visão de implantação; nós; canais; artefatos para implantação; mapeamento de blocos de construção em artefatos de implantação; descrição de interfaces e decisões do projeto; UML, ferramentas para documentação

Objetivos de aprendizagem

OA 3-1: Explicar e considerar a qualidade da documentação técnica (R1)

Os arquitetos de software conhecem os requisitos de qualidade da documentação técnica e podem levá-las em consideração ou cumpri-las ao documentar sistemas:

- Compreensibilidade, corretude, eficiência, adequação, mantenabilidade.
- Forma, conteúdo e nível de detalhe para os stakeholders.

Eles sabem que a compreensibilidade da documentação técnica só pode ser avaliada pelo seu público-alvo.

OA 3-2: Descrever e comunicar arquiteturas de software (R1, R3)

Arquitetos do software devem ter capacidade de:

- Documentar e comunicar arquiteturas de forma orientada a stakeholders e assim abordar diferentes públicos-alvo, por exemplo, gerenciamento, equipes de desenvolvimento, QA, outros arquitetos de software e possivelmente, stakeholders adicionais.
- Consolidar e harmonizar as contribuições de diferentes grupos de autores em termos de estilo e conteúdo.
- Conhecer os benefícios da documentação baseada em modelos.

Os arquitetos de software sabem que as várias propriedades da documentação dependem das especificidades do sistema, seus requisitos, riscos, processo de desenvolvimento, organização ou outros fatores.

Estes fatores têm impacto:

- Se deve ser dada prioridade à comunicação escrita ou verbal.
- A quantidade e o nível de detalhe da documentação necessária em cada fase de desenvolvimento.
- O formato de documentação.
- A acessibilidade à documentação.
- Formalidade da documentação. (por exemplo, diagramas em conformidade com um meta modelo ou desenhos simples).
- Revisões formais e processos de cancelamento de assinatura para documentação.

Os arquitetos de software estão conscientes destes fatores e podem ajustar as características da documentação de acordo com a situação.

OA 3-3: Explicar e aplicar notações/modelos para a descrição de arquitetura de software (R2-R3)

Os arquitetos de software estão familiarizados com, pelo menos, os seguintes diagramas UML para a notação de visões de arquiteturas (ver: [UML]):

- Classe, pacote, componente (todos R2) e diagramas de estrutura composta. (R3)
- Diagramas de implantação. (R2)
- Diagramas de sequência e atividade. (R2)
- Diagramas de máquina de estado. (R3)

Os arquitetos de software conhecem notações alternativas aos diagramas UML, por exemplo (R3):

- Arquimate, ver: [Archimate].
- Para visualizações de tempo de execução, por exemplo, fluxogramas, listas numeradas ou notação de modelagem de processos de negócios (BPMN).

OA 3-4: Explicar e aplicar visões de arquiteturas (R1)

Os arquitetos de software podem usar as seguintes visões arquiteturais:

- Visão de contexto.
- Visão de blocos de construção ou de componentes (estrutura do sistema a partir de blocos de construção de software).
- Visão de tempo de execução (visão dinâmica, interação dos blocos de construção de software em tempo de execução, máquinas de estado).
- Visão de distribuição/implantação (infraestrutura técnica e de hardware, bem como mapeamento de blocos de construção de software para essa infraestrutura).

OA 3-5: Explicar e aplicar a visão de contexto dos sistemas (R1)

Os arquitetos de software podem:

- Descrever o contexto dos sistemas, por exemplo, sob a forma de diagramas de contexto com explicações .
- Representar interfaces externas de sistemas na visão de contexto.
- Diferenciar entre o contexto comercial e técnico.

OA 3-6: Documentar e comunicar conceitos transversais (cross-cutting) (R2)

Os arquitetos do software podem documentar e comunicar adequadamente os conceitos (sinônimos: princípios, aspectos) transversais típicos, por exemplo, persistência, controle de fluxo, UI, distribuição/integração, logging.

OA 3-7: Descrever interfaces (R1)

Os arquitetos de software podem descrever e especificar interfaces internas e externas.

OA 3-8: Explicar e documentar decisões de arquitetura (R1-R2)

Os arquitetos do software devem ter capacidade de:

- Provocar, justificar, comunicar e documentar sistematicamente as decisões de arquitetura.
- Identificar, comunicar e documentar as interdependências de decisões de projeto.

Os arquitetos de software sabem sobre Architecture-Decision-Records (ADR, consulte [Nygard 2011]) e podem aplicar estes para documentar as decisões (R2).

OA 3-9: Usar documentação como comunicação escrita (R2)

Os arquitetos de software usam documentação para dar suporte ao design, implementação e desenvolvimento (também chamado de manutenção ou evolução) de sistemas.

OA 3-10: Conhecer recursos adicionais e ferramentas para documentação (R3)

Os arquitetos do software conhecem:

- O básico de vários frameworks publicados para descrever arquiteturas de software, por exemplo:
  - ISO/IEEE-42010 (anteriormente 1471), ver [ISO 42010]
  - arc42, ver [arc42]
  - C4, ver [Brown]
  - FMC, ver [FMC]
- Ideias e exemplos de listas de verificação (checklists) para a criação, documentação e teste de arquiteturas de software.
- Ferramentas possíveis para a criação e manutenção de documentação de arquitetura.

Referências

[arc42], [Archimate], [Bass+2012], [Brown], [Clements+2010], [FMC], [Gharbi+2020], [Starke 2020], [UML], [Zörner 2015]

4. Arquitetura de software e qualidade

Duração: 60 min.	Tempo de exercícios: 60 min.

Conceitos essenciais

Qualidade; características de qualidade (também chamadas de atributos de qualidade); DIN/ISO 25010; cenários de qualidade; árvore de qualidade; trade-offs entre características de qualidade; avaliação qualitativa da arquitetura; métricas e avaliação quantitativa

Objetivos de aprendizagem

OA 4-1: Discutir modelos e características de qualidade (R1)

Os arquitetos do software podem:

- Explicar o conceito de qualidade (baseado na DIN/ISO 25010, anteriormente 9126) e das características da qualidade
- Explicar modelos genéricos de qualidade (tais como DIN/ISO 25010)
- Explicar correlações e trade-offs das características de qualidade, por exemplo:
  - Configurabilidade versus confiabilidade.
  - Requisitos de memória versus eficiência de desempenho.
  - Segurança versus usabilidade.
  - Flexibilidade de tempo de execução versus mantenabilidade.

OA 4-2: Esclarecer os requisitos de qualidade das arquiteturas de software (R1)

Os arquitetos do software podem:

- Esclarecer e formular requisitos de qualidade específicos para o software a ser desenvolvido e suas arquiteturas, por exemplo, na forma de cenários e árvores de qualidade.
- Explicar e aplicar cenários e árvores de qualidade.

OA 4-3: Analisar e avaliar qualitativamente as arquiteturas de software (R2-R3)

Arquitetos do software:

- Conhecem procedimentos metodológicos para a análise qualitativa e avaliação de arquiteturas de software (R2), por exemplo, de acordo com o ATAM (R3).
- Podem analisar e avaliar qualitativamente sistemas menores (R2).
- Sabem que as seguintes fontes de informação podem ajudar a analisar e avaliar qualitativamente as arquiteturas:
  - Requisitos de qualidade, por exemplo, sob a forma de árvores e cenários de qualidade.
  - Documentação de arquitetura.
  - Modelos de arquitetura e design.
  - Código fonte.
  - Métricas.
  - Outra documentação do sistema, tal como documentação de requisitos, operacional ou

de teste.

OA 4-4: Avaliar quantitativamente as arquiteturas de software (R2)

Os arquitetos de software conhecem abordagens para a análise quantitativa e avaliação (medição) de software.

Eles sabem que:

- A avaliação quantitativa pode ajudar a identificar partes críticas dos sistemas.
- Para a avaliação das arquiteturas, informações adicionais podem ser úteis, como por exemplo:
    - Documentação de requisitos e arquitetura.
    - Código fonte e métricas relacionadas, tais como linhas de código, complexidade (ciclomática), dependências de entrada e de saída.
    - Erros conhecidos no código fonte, especialmente agrupamentos de erros.
    - Casos de teste e resultados de testes.

Referências

[Bass+2012], [Clements+2002], [Gharbi+2020], [Martin 2003], [Starke 2020]

5. Exemplos de arquiteturas de software

Duração: 90 min.	Tempo de exercícios: Nenhum.

Esta seção não é relevante para o exame.

Objetivos de aprendizagem

OA 5-1: Compreender a relação dos requisitos, restrições e soluções (R3)

Espera-se que os arquitetos de software reconheçam e compreendam a correlação entre requisitos e restrições, e as soluções escolhidas usando pelo menos um exemplo.

OA 5-2: Compreender a lógica da implementação técnica de uma solução (R3)

Os arquitetos de software são capazes de entender a realização técnica (implementação, conceitos técnicos, produtos utilizados, decisões arquiteturais, estratégias de solução) de uma solução, com base em, pelo menos, um exemplo.

Referências

- [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+2012] Len Bass, Paul Clements, Rick Kazman: Software Architecture in Practice. 3rd edition, Addison Wesley, 2012.
- [Brown] Simon Brown: Brown, Simon: 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, second edition, Addison Wesley, 2010.
- [Eilebrecht+2019] Karl Eilebrecht, Gernot Starke: Patterns kompakt: Entwurfsmuster für effektive Software-Entwicklung (in German). 5th Edition Springer Verlag, 2019.
- [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. O'Reilly, 2017.
- [Fowler 2002] Martin Fowler: Patterns of Enterprise Application Architecture. (PoEAA) Addison- Wesley, 2002.
- [Gamma+94] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.
- [Geirhosk 2015] Matthias Geirhos. Entwurfsmuster: Das umfassende Handbuch (in German). Rheinwerk Computing Verlag, 2015 ISBN: 9783836227629.
- [Gharbi+2020] Mahbouba Gharbi, Arne Koschel, Andreas Rausch, Gernot Starke: Basiswissen Softwarearchitektur. 4th edition, published by dpunkt, Heidelberg 2020.
- [Goll 2014] Joachim Goll: Architektur- und Entwurfsmuster der Softwaretechnik: Mit lauffähigen Beispielen in Java. Published by Springer-Vieweg, second edition 2014.
- [Hofmeister et. al 1999] Christine Hofmeister, Robert Nord, Dilip Soni: Applied Software Architecture, Addison-Wesley, 1999.
- [ISO 42010] ISO/IEC/IEEE 42010:2011, Systems and software engineering — Architecture description, online: http://www.iso-architecture.org/ieee-1471/.
- [iSAQB Working Groups] Gernot Starke et. al. Annotated collection of Software Architecture References, for Foundation and Advanced Level Curricula. Available for free on https://leanpub.com/isaqbreferences.

- [Keeling 2017] Michael Keeling. Design It!: From Programmer to Software Architect. Pragmatic Programmer. ISBN 978-1680502091.
- [Lilienthal 2018] Carola Lilienthal: Langlebige Softwarearchitekuren. Second edition, published by dpunkt Verlag, 2018.
- [Lilienthal 2019] Carola Lilienthal: Sustainable Software Architecture: Analyze and Reduce Technical Debt. dpunkt Verlag, 2019.
- [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. MITP.
- [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. ISBN 9781491950357.
- [Nygard 2011] Michael Nygard: Documenting Architecture Decision. https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions. See also https://adr.github.io/.
- [Pethuru 2017] Raj Pethuru et. al: Architectural Patterns. Packt, 2017.
- [Starke 2020] Gernot Starke: Effektive Softwarearchitekturen - Ein praktischer Leitfaden (in German). 9th edition, published by Carl Hanser, 2020. Sito web: https://esabuch.de.
- [UML] The UML reading room, collection of UML resources https://www.omg.org/technology/readingroom/UML.htm. See also https://www.uml-diagrams.org/.
- [van Steen+Tanenbaum] Andrew Tanenbaum, Maarten van Steen: Distributed Systems, Principles and Paradigms. https://www.distributed-systems.net/.
- [Yorgey 2012] Brent A. Yorgey, Proceedings of the 2012 Haskell Symposium, September 2012 https://doi.org/10.1145/2364506.2364520.
- [Zörner 2015] Stefan Zörner: Softwarearchitekturen dokumentieren und kommunizieren. Second edition, published by Carl Hanser, 2015.