Programa de Estudio de

Certified Professional for Software Architecture (CPSA®)

Foundation Level

2021.1-ES-20210209

Contenido

Información legal

La abreviatura "e. V." forma parte del nombre oficial de iSAQB y significa "eingetragener Verein" (asociación registrada), que describe su condición de entidad legal según la ley alemana. En aras de la simplicidad, en adelante se denominará iSAQB e. V. sin el uso de dicha abreviatura.

Relación de objetivos de aprendizaje

- OA 1-1: Debatir acerca de las definiciones de la arquitectura de software (R1)
- OA 1-2: Comprender y explicar los objetivos y beneficios de la arquitectura de software (R1)
- OA 1-3: Entender la arquitectura de software como parte del ciclo de vida del software (R2)
- OA 1-4: Comprender las tareas y responsabilidades de los arquitectos de software (R1)
- OA 1-5: Relacionar el papel de los arquitectos de software con otros implicados (R1)
- OA 1-6: Ser capaz de explicar la correlación entre los enfoques de desarrollo y la arquitectura de software (R1)
- OA 1-7: Diferenciar entre los objetivos a corto y largo plazo (R1)
- OA 1-8: Distinguir afirmaciones explícitas y supuestos implícitos (R1)
- OA 1-9: Responsabilidades de los arquitectos de software dentro de un contexto arquitectónico más amplio (R3)
- OA 1-10: Diferenciar los tipos de sistemas TI (R3)
- OA 1-11: Desafíos de los sistemas distribuidos (R3)
- OA 2-1: Seleccionar y utilizar enfoques y heurísticas para el desarrollo de la arquitectura (R1, R3)
- OA 2-2: Diseñar arquitecturas de software (R1)
- OA 2-3: Identificar y considerar los factores que influyen en la arquitectura del software (R1-R3)
- OA 2-4: Diseñar e implementar conceptos transversales (R1)
- OA 2-5: Describir, explicar y aplicar de forma adecuada patrones de solución relevantes (R1, R3)
- OA 2-6: Explicar y aplicar los principios de diseño (R1-R3)
- OA 2-7: Gestionar las dependencias entre bloques constructivos (R1)
- OA 2-8: Lograr los requisitos de calidad con enfoques y técnicas adecuadas (R1)
- OA 2-9: Diseñar y definir interfaces (R1-R3)
- OA 3-1: Explicar y considerar la calidad de la documentación técnica (R1)
- OA 3-2: Describir y comunicar arquitecturas de software (R1, R3)
- OA 3-3: Explicar y aplicar notaciones/modelos para describir una arquitectura de software (R2-R3)
- OA 3-4: Explicar y utilizar vistas arquitectónicas (R1)
- OA 3-5: Explicar y aplicar la vista de contexto de los sistemas (R1)
- OA 3-6: Documentar y comunicar conceptos transversales (R2)
- OA 3-7: Describir interfaces (R1)
- OA 3-8: Explicar y documentar las decisiones arquitectónicas (R1-R2)
- OA 3-9: Utilizar la documentación como comunicación escrita (R2)
- OA 3-10: Conocer recursos y herramientas adicionales para la documentación (R3)
- OA 4-1: Debatir sobre modelos de calidad y características de calidad (R1)
- OA 4-2: Precisar los requisitos de calidad de las arquitecturas de software (R1)
- OA 4-3: Llevar a cabo un análisis cualitativo y una evaluación de las arquitecturas de software (R2-R3)

- OA 4-4: Llevar a cabo una evaluación cuantitativa de las arquitecturas de software (R2)
- OA 5-1: Conocer la relación entre las necesidades, restricciones y soluciones (R3)
- OA 5-2: Conocer los fundamentos de la implementación técnica de una solución (R3)

Introducción

¿Qué aporta una Formación a Foundation Level?

Los cursos de formación con licencia *Certified Professional for Software Architecture - Foundation Level* (CPSA-F) proporcionarán a los participantes los conocimientos y aptitudes necesarios para diseñar, especificar y documentar una arquitectura de software adecuada para cumplir los requisitos correspondientes a sistemas de tamaño pequeño y mediano. Los participantes aprenderán a tomar decisiones arquitectónicas en función de su experiencia práctica individual y de sus competencias actuales, a partir de una visión de sistema existente y de los requisitos adecuadamente detallados. En los cursos de formación del CPSA-F se enseñan métodos y principios para el diseño, la documentación y la evaluación de arquitecturas de software, independientemente de los procesos de desarrollo específicos.

La atención se centra en la educación y la formación de las siguientes competencias:

- Debatir y conciliar las decisiones arquitectónicas fundamentales con los implicados provenientes de requisitos, gestión, desarrollo, operaciones y prueba.
- Comprender las actividades esenciales de la arquitectura de software y llevar a cabo aquellas correspondientes a los sistemas de tamaño pequeño y mediano.
- Documentar y comunicar arquitecturas de software basadas en vistas arquitectónicas, patrones de arquitectura y conceptos técnicos.

Adicionalmente, esta formación abarca:

- El concepto y el significado de la arquitectura de software.
- Las tareas y responsabilidades de los arquitectos de software.
- El papel de los arquitectos de software en los proyectos de desarrollo.
- Métodos y técnicas correspondientes al estado del arte para desarrollar arquitecturas de software.

Fuera de alcance

Este programa de estudio refleja los contenidos que los miembros del iSAQB consideran, en la actualidad, necesarios y útiles para alcanzar los objetivos de aprendizaje del CPSA-F. No es una descripción exhaustiva de todo el dominio de la "arquitectura de software".

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

- Tecnologías, marcos de trabajo o librerías de implementación específicos.
- Programación o lenguajes de programación.
- Modelos de proceso específicos.
- Fundamentos de las notaciones de modelado (como el UML) o fundamentos del propio modelado.
- Análisis de sistemas e ingeniería de requisitos (por favor, consulte el programa de estudio y certificación de IREB e. V., https://ireb.org, International Requirements Engineering Board).
- Prueba de software (por favor, consulte el programa de estudio y certificación de ISTQB e.V., https://istqb.org, International Software Testing Qualification Board).
- Gestión de proyectos o producto.
- Introducción a herramientas de software específicas.

El objetivo de la formación es proporcionar los fundamentos para adquirir los conocimientos y competencias avanzadas que se requieren para sus respectivas aplicaciones.

Prerrequisitos

El iSAQB e. V. puede comprobar los siguientes prerrequisitos en los exámenes de certificación mediante las preguntas correspondientes.

Los participantes deberán contar con los siguientes conocimientos y/o experiencia. En particular, la experiencia práctica sustancial en el desarrollo de software en equipo es un prerrequisito importante para la comprensión del material de aprendizaje y la obtención de la certificación.

- Más de 18 meses de experiencia práctica en el desarrollo de software, obtenida mediante el desarrollo en equipo de varios sistemas fuera de la formación académica.
- Conocimiento y experiencia práctica con al menos un lenguaje de programación de alto nivel, especialmente:
  - Conceptos de
    - Modularización (paquetes, espacio de nombres, etc).
    - Paso de parámetros (llamada por valor, llamada por referencia).
    - Alcance (es decir tipo), declaración y definición de variables.
  - Fundamentos de sistemas de tipos (tipos estáticos frente a tipos dinámicos, tipos genéricos de datos).
  - Tratamiento de errores y excepciones en el software.
  - Problemas potenciales de estado global y variables globales.
- Conocimientos básicos de:
  - Modelado y abstracción.
  - Algoritmos y estructuras de datos (por ejemplo, Listas, Árboles, Tablas Hash, Diccionario, Mapa).
  - UML (diagramas de clase, de paquetes, de componentes y de secuencia) y su relación con el código fuente.

Además, los siguientes temas serán útiles para comprender diferentes conceptos:

- Fundamentos y diferencias de la programación imperativa, declarativa, orientada a objetos y funcional.
- Experiencia práctica en
  - Un lenguaje de programación de alto nivel.
  - Diseñar e implementar aplicaciones distribuidas, como sistemas cliente-servidor o aplicaciones web.
  - Documentación técnica, especialmente la documentación del código fuente, el diseño del sistema o conceptos técnicos.

Estructura, duración y métodos de enseñanza

Los tiempos lectivos que se indican en las siguientes secciones del programa de estudio son sólo recomendaciones. La duración de un curso de formación debe ser de, al menos, tres días, pero también puede ser más prolongada. Los proveedores pueden variar la duración, los métodos de enseñanza, el tipo y la estructura de los ejercicios, así como el esquema detallado del curso. Los tipos (dominios y tecnologías) de los ejemplos y ejercicios pueden ser determinados individualmente por los proveedores de formación.

Índice	**Duración recomendada (min)**
1. Conceptos básicos de la arquitectura de software	120
2. Diseño y desarrollo	420
3. Especificación y comunicación	240
4. Arquitectura de software y calidad	120
5. Ejemplos	90
Total	990

Objetivos de aprendizaje y su relevancia con respecto al examen

La estructura de los capítulos del programa de estudio sigue un conjunto de objetivos de aprendizaje priorizados. Para cada objetivo de aprendizaje, se indica claramente la relevancia para el examen de este objetivo de aprendizaje o sus subelementos (según la clasificación R1, R2, R3, véase el cuadro que figura a continuación). Cada objetivo de aprendizaje describe los contenidos que deben enseñarse, incluidos sus términos y conceptos clave.

En lo que respecta a la relevancia para el examen, en este plan de estudios se utilizan las siguientes categorías:

ID	Categoría del objetivo de aprendizaje	Significado	Relevancia para el examen
R1	Ser capaz de	Estos son los contenidos que se espera que los participantes puedan poner en práctica de forma independiente al finalizar el curso. Dentro del curso, estos contenidos serán cubiertos a través de ejercicios y debates.	El contenido **será** parte del examen.
R2	Comprensión	Estos son los contenidos que se espera que los participantes comprendan en principio. Normalmente no serán el foco principal de los ejercicios de la formación.	El contenido **puede ser** parte del examen.
R3	Conocimiento	Estos contenidos (términos, conceptos, métodos, prácticas o similares) pueden mejorar la comprensión y motivar el tema. Pueden ser cubiertos en la capacitación si fuera necesario.	El contenido **no será** parte del examen.

Cuando se considere necesario, los objetivos de aprendizaje incluirán referencias a lecturas adicionales, normas u otras fuentes. En las secciones "Términos y conceptos" de cada capítulo se enumeran las palabras que están asociadas al contenido del capítulo. Algunas de ellas se utilizan en las descripciones de los objetivos de aprendizaje.

Versión actual y repositorio público

Encontrará la versión más reciente de este documento en la página oficial de descargas en https://isaqborg.github.io/.

Este documento se mantiene en un repositorio público en https://github.com/isaqb-org/curriculum-foundation, todos los cambios y modificaciones son públicos.

Por favor, notifique cualquier incidencia en nuestro gestor de incidencias público en https://github.com/isaqborg/curriculumfoundation/issues.

1. Conceptos básicos de la arquitectura de software

Duración: 120 min.	Tiempo propuesto para ejercicios: sin tiempo

Términos fundamentales

Arquitectura de software; dominios de arquitectura; estructura; bloque estructural; componente; interfaz; relación; conceptos transversales; arquitectos de software y sus responsabilidades; tareas y competencias necesarias; implicados y sus intereses; requisitos funcionales y de calidad de los sistemas; restricción; factor de influencia; tipos de sistemas TI (sistemas embebidos, sistemas en tiempo real, sistemas de información, etc.)

Objetivos

OA 1-1: Debatir acerca de las definiciones de la arquitectura de software (R1)

Los arquitectos de software tienen conocimiento de diversas definiciones en el ámbito de la arquitectura de software (incluyendo ISO 42010/IEEE 1471, SEI, Booch, etc.) y pueden señalar sus similitudes:

- Componentes/bloques estructurales con interfaces y relaciones.
- Bloques estructurales como un término general, componentes como una forma especial de los mismos.
- Estructuras, conceptos transversales, principios.
- Decisiones respecto a la arquitectura y sus consecuencias sobre la totalidad de los sistemas y sus ciclos de vida.

OA 1-2: Comprender y explicar los objetivos y beneficios de la arquitectura de software (R1)

Los arquitectos de software pueden justificar los siguientes objetivos y beneficios básicos de la arquitectura de software:

- Apoyar el diseño, la implementación, el mantenimiento y la operación de los sistemas.
- Lograr el cumplimiento de los requisitos de calidad, como la fiabilidad, la mantenibilidad, la capacidad para ser modificado, la seguridad, etc.
- Lograr el cumplimiento de los requisitos funcionales o garantizar que se puedan cumplir.
- Asegurar que las estructuras y los conceptos del sistema son comprendidos por todos los implicados relevantes.
- Reducir la complejidad de forma sistemática.
- Especificar las directrices arquitectónicas pertinentes para la implementación y operación.

OA 1-3: Entender la arquitectura de software como parte del ciclo de vida del software (R2)

Los arquitectos de software entienden sus tareas y pueden integrar sus resultados a lo largo de todo el ciclo de vida de los sistemas TI. Ellos pueden:

- Identificar las consecuencias de los cambios en requisitos funcionales, requisitos de calidad, tecnologías o el entorno del sistema con respecto a la arquitectura de software.
- Explicar detalladamente las relaciones entre los sistemas TI y los procesos de negocio y los

procesos operativos soportados.

OA 1-4: Comprender las tareas y responsabilidades de los arquitectos de software (R1)

Los arquitectos de software son responsables de lograr la calidad solicitada o necesaria y de crear el diseño de la arquitectura de una solución. Dependiendo del enfoque real o del modelo de proceso utilizado, deben alinear esta responsabilidad con las responsabilidades globales de la gestión del proyecto y/u otros roles.

Las tareas y la responsabilidad de los arquitectos de software:

- Aclarar y examinar los requisitos y restricciones, y mejorarlos si fuera necesario. Junto con los requisitos funcionales (prestaciones necesarias), esto incluye las características de calidad requeridas (restricciones necesarias).
- Decidir cómo descomponer el sistema en bloques estructurales, al tiempo que se determinan las dependencias e interfaces entre los bloques estructurales.
- Determinar y decidir respecto de conceptos transversales (por ejemplo, la persistencia, la comunicación, la interfaz gráfica de usuario, etc.).
- Comunicar y documentar la arquitectura de software basada en vistas, patrones arquitectónicos, conceptos transversales y técnicos.
- Acompañar la realización e implementación de la arquitectura; integrar la retroalimentación de los implicados relevantes en la arquitectura si fuera necesario; revisar y asegurar la consistencia del código fuente y la arquitectura del software.
- Analizar y evaluar la arquitectura de software, especialmente con respecto a los riesgos que implican los requisitos de calidad.
- Identificar, resaltar y justificar las consecuencias de las decisiones arquitectónicas a otros implicados.

Deberían reconocer, de forma independiente, la necesidad de las iteraciones en todas las tareas y señalar las posibilidades de una retroalimentación adecuada y relevante.

OA 1-5: Relacionar el papel de los arquitectos de software con otros implicados (R1)

Los arquitectos de software son capaces de explicar su rol. Deben adaptar su contribución al desarrollo de software en un contexto específico y en relación con otros implicados y unidades organizativas, en particular a:

- Gestión de producto, propietario de producto.
- Jefes de proyecto.
- Ingenieros de requisitos (analistas de requisitos o de negocio, gestores de requisitos, analistas de sistemas, propietarios de negocios, expertos en la materia, etc.).
- Desarrolladores.
- Personal de aseguramiento de la calidad y probadores.
- Operadores y administradores TIC (aplica principalmente al entorno de producción o a los centros de datos de sistemas de información).
- Desarrolladores de hardware.
- Arquitectos corporativos y miembros del comité de arquitectura.

OA 1-6: Ser capaz de explicar la correlación entre los enfoques de desarrollo y la arquitectura de software (R1)

- Los arquitectos de software pueden explicar la influencia de los enfoques iterativos en las decisiones arquitectónicas (con respecto a los riesgos y la previsibilidad).
- Los arquitectos de software, a menudo, tienen que trabajar y tomar decisiones de forma iterativa debido a la incertidumbre inherente. Para ello, tienen que obtener retroalimentación en forma sistemática por parte de otros implicados.

OA 1-7: Diferenciar entre los objetivos a corto y largo plazo (R1)

Los arquitectos de software pueden:

- Explicar los requisitos de calidad a largo plazo y sus diferencias en relación con los objetivos del proyecto (a corto plazo).
- Explicar los posibles conflictos entre los objetivos a corto y a largo plazo, a fin de encontrar una solución adecuada para todos los implicados.
- Identificar y especificar los requisitos de calidad.

OA 1-8: Distinguir afirmaciones explícitas y supuestos implícitos (R1)

Los arquitectos de software:

- Deben exponer explícitamente los supuestos o prerrequisitos y evitar supuestos implícitos.
- Deben saber que las suposiciones implícitas pueden dar lugar a posibles malentendidos entre los implicados.
- Pueden plantear implícitamente, si es adecuada en el contexto dado.

OA 1-9: Responsabilidades de los arquitectos de software dentro de un contexto arquitectónico más amplio (R3)

La atención del programa de estudio de Profesional Certificado del iSAQB® en Arquitectura de Software se centra en las estructuras y conceptos de sistemas de software individuales.

Adicionalmente, los arquitectos de software están familiarizados con otros dominios arquitectónicos, por ejemplo:

- Arquitectura TIC corporativa: estructura de panoramas de aplicación.
- Arquitectura de negocio y de procesos: estructura de, entre otras cosas, los procesos de negocio.
- Arquitectura de la información: estructura transversal de los sistemas y uso de la información y los datos.
- Infraestructura o arquitectura tecnológica: estructura de la infraestructura técnica, hardware, redes, etc.
- Arquitectura del hardware o del procesador (para sistemas relacionados con el hardware).

Estos dominios arquitectónicos no son el foco de contenido del CPSA-F.

OA 1-10: Diferenciar los tipos de sistemas TI (R3)

Los arquitectos de software conocen diferentes tipos de sistemas informáticos, por ejemplo:

- Sistemas de información.
- Sistemas de soporte a decisiones, almacenes de datos o inteligencia de negocio.

- Sistemas móviles.
- Procesos o sistemas por lotes.
- Sistemas relacionados con el hardware; aquí entienden la necesidad de diseño del código de hardware/software (dependencias temporales y relacionadas con el contenido del diseño del hardware y el software).

OA 1-11: Desafíos de los sistemas distribuidos (R3)

Los arquitectos de software son capaces de:

- Identificar la distribución en una determinada arquitectura de software.
- Analizar los criterios de consistencia para un determinado problema de negocio
- Explicar la causalidad de eventos en un sistema distribuido.

Los arquitectos de software …

- … saben que la comunicación puede fallar en un sistema distribuido.
- … conocen las limitaciones relativas a la consistencia en bases de datos reales.
- … saben en qué consiste el problema del "cerebro dividido" y por qué es una dificultad.
- … saben que es imposible determinar el orden temporal de los acontecimientos en un sistema distribuido.

Referencias

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

2. Diseño y desarrollo de arquitecturas de software

Duración: 330 min.	Tiempo propuesto para ejercicios: 90 min.

Términos fundamentales

acoplamiento; arquitectura funcional y técnica; cohesión; conceptos técnicos y transversales; decisión de diseño; dependencia; diseño basado en modelos; diseño guiado por el dominio; diseño iterativo/incremental; diseño; enfoque de diseño; enfoques descendente y ascendente; interfaces; lenguaje de patrón; patrón arquitectónico; principio de diseño; vista;

Objetivos

OA 2-1: Seleccionar y utilizar enfoques y heurísticas para el desarrollo de la arquitectura (R1, R3)

Los arquitectos de software son capaces de mencionar, explicar y utilizar los enfoques básicos del desarrollo de la arquitectura, por ejemplo:

- Enfoques de diseño descendente y ascendente. (R1)
- Desarrollo de arquitectura basada en vistas. (R1)
- Diseño iterativo e incremental. (R1)
  - Necesidad de las iteraciones, especialmente cuando la toma de decisiones se ve afectada por incertidumbres. (R1)
  - Necesidad de retroalimentación sobre las decisiones de diseño. (R1)
- Diseño guiado por el dominio, véase [Evans 2004] (R3)
- Arquitectura evolutiva, véase [Ford 2017] (R3)
- Análisis global, véase [Hofmeister et. al 1999] (R3)
- Arquitectura guiada por modelos (R3).

OA 2-2: Diseñar arquitecturas de software (R1)

Los arquitectos de software tienen la capacidad para:

- Diseñar y comunicar y documentar, de forma adecuada, arquitecturas de software basadas en requisitos funcionales y de calidad conocidos para sistemas de software que no son críticos para la seguridad ni para el negocio.
- Tomar decisiones relevantes para la estructura con respecto a la descomposición del sistema, la estructura de los bloques estructurales y diseñar, de forma deliberada, las dependencias entre los bloques estructurales.
- Reconocer y justificar las interdependencias y compromisos de las decisiones de diseño.
- Explicar los términos *caja negra* y *caja blanca* y aplicarlos de forma deliberada.
- Aplicar un refinamiento de forma gradual y especificar los bloques estructurales.
- Diseñar vistas de arquitectura, especialmente la vista de bloques estructurales, la vista de tiempo de ejecución y la vista de despliegue.
- Aclarar las consecuencias que tienen estas decisiones de cara al código fuente.
- Separar los elementos técnicos y los relacionados con el dominio de las arquitecturas y justificar estas decisiones.

- Identificar los riesgos de las decisiones de diseño.

OA 2-3: Identificar y considerar los factores que influyen en la arquitectura del software (R1-R3)

Los arquitectos de software son capaces de recopilar y tener en cuenta las restricciones y factores de influencia que limiten la libertad en las decisiones de diseño. Entienden que sus decisiones pueden implicar otros requisitos y restricciones sobre el sistema que se está diseñando, su arquitectura o el proceso de desarrollo. (R1-R2)

Deben reconocer y tener en cuenta el impacto de:

- factores relacionados con el producto, como: (R1)
  - Requisitos funcionales.
  - Requisitos de calidad y objetivos de calidad.
  - Factores adicionales como el coste del producto, el modelo de licencia previsto o el modelo de negocio del sistema.
- factores tecnológicos como: (R1-R3)
  - Las decisiones y conceptos técnicos impuestos externamente. (R1)
  - La infraestructura de hardware y software existente o prevista. (R1)
  - Restricciones tecnológicas sobre las estructuras de datos y las interfaces. (R2)
  - Arquitecturas, librerías, componentes y marcos de trabajo de referencia. (R1)
  - Los lenguajes de programación. (R3)
- factores organizativos como:
  - La estructura organizativa del equipo de desarrollo y del cliente. (R1)
  - La cultura de la empresa y del equipo. (R3)
  - Alianzas y acuerdos de cooperación. (R2)
  - Normas, directrices y modelos de procesos (por ejemplo, procesos de aprobación y entrega). (R2)
  - Los recursos disponibles, como el presupuesto, el tiempo y el personal. (R1)
  - Disponibilidad, competencia y compromiso del personal. (R1)
- factores normativos como: (R2)
  - Restricciones legales locales e internacionales.
  - Cuestiones de carácter contractual y relativos a la responsabilidad.
  - Leyes de protección de datos y privacidad.
  - Cuestiones relativas al cumplimiento o a la obligación de aportar la carga evidencias.
- tendencias como: (R3)
  - Tendencias del mercado.
  - Tendencias tecnológicas (por ejemplo, blockchain, microservicios).
  - Tendencias metodológicas (por ejemplo, ágil).
  - Impacto (potencial) de otros asuntos de interés y decisiones de diseño obligatorias. (R3)

Los arquitectos de software son capaces de describir cómo esos factores pueden influir en las decisiones de diseño y pueden explicar con detalle las consecuencias de los cambios en los factores de influencia proporcionando ejemplos de algunos de ellos (R2).

OA 2-4: Diseñar e implementar conceptos transversales (R1)

Los arquitectos de software son capaces de:

- Explicar el significado de tales conceptos transversales.
- Decidir y diseñar con respecto a los conceptos transversales, por ejemplo, la persistencia, la comunicación, la interfaz gráfica de usuario, el tratamiento de los errores, la concurrencia.
- Identificar y valorar las posibles interdependencias entre estas decisiones.

Los arquitectos de software saben que estos conceptos transversales pueden ser reutilizados en todo el sistema.

OA 2-5: Describir, explicar y aplicar de forma adecuada patrones de solución relevantes (R1, R3)

Los arquitectos de software conocen:

- Varios patrones arquitectónicos y pueden aplicarlos cuando sea conveniente.
- Que los patrones son una forma de lograr ciertas cualidades para determinados problemas y requisitos dentro de contextos determinados.
- Que existen varias categorías de patrones. (R3)
- Fuentes adicionales de patrones relacionados con su dominio técnico o de aplicación específica. (R3)

Los arquitectos de software pueden explicar y proporcionar ejemplos para los siguientes patrones (R1):

- Capas:
  - Las capas de abstracción ocultan detalles, ejemplo: Capas de red ISO/OSI, o "capa de abstracción de hardware". Véase https://en.wikipedia.org/wiki/Hardware_abstraction
  - Otra interpretación son las capas para separar (físicamente) la funcionalidad o la responsabilidad, véase https://en.wikipedia.org/wiki/Multitier_architecture
- Tuberías y filtros: Representante de los patrones de flujo de datos, dividiendo el procesamiento por etapas en una serie de actividades de procesamiento ("Filtro") y capacidades de transporte/almacenamiento de datos asociadas ("Tuberías").
- Los microservicios dividen la aplicación en ejecutables separados que se comunican a través de una red.
- La Inyección de Dependencia como posible solución al Principio de Inversión de la Dependencia.

Los arquitectos de software pueden explicar varios de los siguientes patrones, explicar su relevancia para sistemas concretos y proporcionar ejemplos. (R3)

- Pizarra: trata problemas que no pueden ser resueltos por algoritmos deterministas pero que requieren conocimientos diversos.
- Intermediario: responsable de coordinar la comunicación entre proveedor(es) y consumidor(es), se aplica en sistemas distribuidos. Responsable de reenviar las peticiones y/o transmitir los resultados y las excepciones.
- Combinador (sinónimo: cierre de operaciones), para objetos de dominio de tipo T, busca operaciones con entrada y salida de tipo T. Véase [Yorgey 2012]
- (SRCC) Segregación de responsabilidades de Consultas y Comandos: Separa los asuntos de

interés relativos a la lectura de los de la escritura en los sistemas de información. Requiere algo de contexto en la tecnología de bases de datos/persistencia para entender las diferentes propiedades y requisitos de las operaciones de "lectura" frente a las de "escritura".

- Aprovisionamiento de Eventos: trata las operaciones sobre los datos mediante una secuencia de eventos, cada uno de los cuales se registra en un almacén exclusivo de apéndices.

- Intérprete: representa el objeto de dominio o un Lenguaje Específico de Dominio como sintaxis, proporciona una función que implementa una interpretación semántica del objeto de dominio de forma independiente al propio objeto de dominio.
- Patrones de integración y mensajería. (por ejemplo, de Hohpe+2004])
- La familia de patrones Modelo-Vista-Controlador (por sus siglas en inglés, MVVM), Modelo–Vista–Actualización, Presentación–Abstracción–Control (por sus siglas en inglés, PAC), que separa la representación externa (vista) de los datos, los servicios y su coordinación.
- Patrones de interfaz como Adaptador, Fachada, Representante. Estos patrones ayudan a la integración de los subsistemas y/o a la simplificación de las dependencias. Los arquitectos deben saber que estos patrones pueden utilizarse independientemente de la tecnología (de objetos).
    - Adaptador: desacopla al consumidor y al proveedor, cuando la interfaz del proveedor no coincide exactamente con la del consumidor. El Adaptador desacopla una parte de los cambios de interfaz de la otra.
    - Fachada: simplifica el uso de un proveedor para el consumidor o los consumidores proporcionando un acceso simplificado.
    - Representante: Un intermediario entre el consumidor y el proveedor, que permite el desacoplamiento temporal, el almacenamiento en caché de los resultados, el control del acceso al proveedor, etc.
- Observador: un productor de valores a lo largo del tiempo notifica a centros de conmutación donde los consumidores pueden registrarse para ser notificados.
- Complemento: amplía el comportamiento de un componente.
- Puertos y Adaptadores (sin. Arquitectura de Cebolla, Arquitectura Hexagonal): concentran la lógica del dominio en el centro del sistema, tienen conexiones con el mundo exterior (base de datos, interfaz de usuario) en los bordes, dependencias sólo de fuera hacia adentro, nunca de dentro hacia afuera.
- Llamada a Procedimiento Remoto: hacer que una función o algoritmo se ejecute en un espacio de direcciones diferente.
- Arquitectura Orientada a Servicios (por sus siglas en inglés, SOA): Un enfoque para proporcionar servicios abstractos en lugar de implementaciones concretas a los usuarios del sistema para promover la reutilización de servicios entre departamentos y entre empresas.
- Plantilla y Estrategia: confieren flexibilidad a algoritmos específicos encapsulándolos.
- Visitante: separa el recorrido de la estructura de datos de un procesamiento específico.

Los arquitectos de software conocen fuentes consideradas fundamentales para los patrones arquitectónicos, como POSA (por ejemplo, [Buschmann+ 1996]) y PoEAA ([Fowler 2002]) (para los sistemas de información). (R3)

OA 2-6: Explicar y aplicar los principios de diseño (R1-R3)

Los arquitectos de software son capaces de explicar qué son los principios de diseño. Pueden exponer sus objetivos generales y sus posibles aplicaciones con respecto a la arquitectura del software. (R2)

Cada uno de los siguientes principios de diseño cuenta con un nivel de relevancia específico en lo que

respecta al examen. Los arquitectos de software son capaces de:

- Explicar los principios de diseño enumerados a continuación y de ilustrarlos mediante ejemplos.
- Explicar cómo se deben aplicar esos principios.
- Explicar cómo los requisitos de calidad determinan qué principios deben aplicarse.
- Explicar el impacto de los principios de diseño en la implementación.
- Analizar el código fuente y los diseños de la arquitectura para evaluar si estos principios de diseño se han aplicado o deberían aplicarse.

Abstracción (R1)

- En el sentido de un medio para obtener generalizaciones útiles.
- Como una construcción del diseño, donde los bloques estructurales dependen de las abstracciones en lugar de depender de las implementaciones.
- Interfaces como abstracciones.

Modularización (R1-R3)

- Ocultación de información y encapsulado. (R1)
- Separación de asuntos de interés - sdai. (R1)
- Acoplamiento débil, pero funcionalmente suficiente (R1) de los bloques estructurales, véase OA 2-7.
- Alta cohesión. (R1)
- Principios SOLID (R1-R3), que tienen, hasta cierto punto, relevancia a nivel de arquitectura:

Single	S:	Principio de responsabilidad única (R1) y su relación con SdAI.
Open/closed	O:	Principio de apertura/cierre (R1)
Liskov	L:	Principio de sustitución de Liskov (R3) como forma de lograr consistencia e integridad conceptual en un diseño OO
Interface segregation	I:	Principio de segregación de interfaces (R2), incluida su relación con el OA 2-9
Dependency inversion	D:	Principio de inversión de la dependencia (R1) mediante interfaces o abstracciones similares

Integridad conceptual (R2)

- Que significa uniformidad (homogeneidad, consistencia) de las soluciones para problemas similares. (R2)
- Como medio para lograr el principio de menor sorpresa. (R3)

Simplicidad (R1-R2)

- Como medio para reducir la complejidad. (R1)
- Como factor impulsor de kiss (R1) y yagni. (R2)

Esperar Errores (R1-R2)

- Como medio para diseñar sistemas robustos y resilientes. (R1)
- Como generalización del principio de robustez, también conocido como ley de postel. (R2)

OA 2-7: Gestionar las dependencias entre bloques constructivos (R1)

Los arquitectos de software comprenden las dependencias y el acoplamiento entre los bloques estructurales y pueden utilizarlos de manera específica. Ellos:

- Conocen y comprenden los diferentes tipos de dependencias de los bloques estructurales (por ejemplo, el acoplamiento mediante el uso/delegación, la mensajería/los eventos, la composición, la creación, la herencia, el acoplamiento temporal, el acoplamiento mediante los datos, los tipos de datos o el hardware).
- Comprenden cómo las dependencias aumentan el acoplamiento.
- Saben utilizar estos tipos de acoplamiento de forma selectiva y valorar las consecuencias de tales dependencias.
- Conocen y pueden aplicar las posibilidades de reducir o eliminar el acoplamiento, por ejemplo:
  - Patrones (véase OA 2-5).
  - Principios fundamentales de diseño (véase OA 2-6).
  - Externalización de dependencias, es decir, definición de dependencias concretas en tiempo de ejecución o instalación, por ejemplo, mediante el uso de la Inyección de Dependencia.

OA 2-8: Lograr los requisitos de calidad con enfoques y técnicas adecuadas (R1)

Los arquitectos de software comprenden y consideran la significativa influencia de los requisitos de calidad en las decisiones de diseño y arquitectura, por ejemplo, para

- Eficiencia, rendimiento en tiempo de ejecución.
- Disponibilidad.
- Mantenibilidad, capacidad para ser modificado, extensibilidad, adaptabilidad.

Ellos pueden:

- Explicar y aplicar las opciones de solución, las Tácticas de Arquitectura, las prácticas adecuadas, así como las posibilidades técnicas para lograr requisitos de calidad importantes de los sistemas de software (diferentes para los sistemas embebidos o los sistemas de información).
- Identificar y comunicar posibles compromisos entre dichas soluciones y sus riesgos asociados.

OA 2-9: Diseñar y definir interfaces (R1-R3)

Los arquitectos de software conocen la importancia de las interfaces. Son capaces de diseñar o especificar interfaces entre bloques estructurales arquitectónicos, así como interfaces externas entre el sistema y elementos ajenos al sistema.

Ellos conocen:

- Las características deseadas de las interfaces y pueden utilizarlas en el diseño:
  - Fácil de aprender, fácil de utilizar, fácil de ampliar.
  - Difícil de utilizar de forma incorrecta.
  - Funcionalmente completo desde la perspectiva de los usuarios o de los bloques estructurales que las utilizan.
- La necesidad de tratar las interfaces internas y externas de manera diferente.
- Diferentes enfoques para implementar las interfaces (R3):
  - Enfoque orientado a los recursos (REST, REpresentational State Transfer).
  - Enfoque orientado a los servicios (véase servicios web basados en WS-*/SOAP).

Referencias

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

3. Especificación y comunicación de arquitecturas de software

Duración: 180 min.	Tiempo propuesto para ejercicios: 60 min.

Términos fundamentales

Vista (arquitectónica); estructura; concepto (técnico); documentación; comunicación; descripción; orientado a los implicados; meta estructuras y plantillas para la descripción y la comunicación; contexto del sistema; bloque estructural; vista de bloque estructural; vista en tiempo de ejecución; vista de despliegue; nodo; canal; artefacto de despliegue; mapear bloques estructurales en los artefactos de despliegue; descripción de las interfaces y decisiones de diseño; UML; herramienta para la documentación

Objetivos

OA 3-1: Explicar y considerar la calidad de la documentación técnica (R1)

Los arquitectos de software conocen los requisitos de calidad de la documentación técnica y pueden tenerlos en cuenta y cumplirlos al documentar los sistemas:

- Capacidad de ser comprendido, corrección, eficiencia, adecuación, mantenibilidad.
- Forma, contenido y nivel de detalle se adaptan a los implicados.

Saben que sólo la audiencia objetivo puede evaluar la capacidad de ser comprendida de la documentación técnica.

OA 3-2: Describir y comunicar arquitecturas de software (R1, R3)

Los arquitectos de software están capacitados para:

- Documentar y comunicar las arquitecturas a los implicados correspondientes, dirigiéndose así a los diferentes grupos objetivo, por ejemplo, la dirección, los equipos de desarrollo, el control de calidad, otros arquitectos de software y posiblemente implicados adicionales.
- Consolidar y armonizar el estilo y el contenido de las contribuciones de los diferentes grupos de autores.
- Conocer los beneficios de la documentación basada en plantillas.

Los arquitectos de software saben que diversas propiedades de la documentación dependen de las características específicas del sistema, sus requisitos, los riesgos, el proceso de desarrollo, la organización u otros factores.

Estos factores condicionan:

- Si se debe dar prioridad a la comunicación escrita o verbal.
- La cantidad y el nivel de detalle de la documentación necesaria en cada fase de desarrollo.
- El formato de la documentación.
- La accesibilidad de la documentación.
- La formalidad de la documentación (por ejemplo, diagramas conformes a un metamodelo o dibujos sencillos).
- Revisiones formales y procesos de aprobación de la documentación.

Los arquitectos de software son conscientes de estos factores y pueden ajustar las características de la documentación en función de la situación.

OA 3-3: Explicar y aplicar notaciones/modelos para describir una arquitectura de software (R2-R3)

Los arquitectos de software conocen al menos los siguientes diagramas UML (véase [UML]) para describir vistas arquitectónicas:

- Diagramas de clase, paquete, componente (todos ellos R2) y composición-estructura. (R3)
- Diagramas de despliegue. (R2)
- Diagramas de secuencia y actividad. (R2)
- Diagramas de estado. (R3)

Los arquitectos de software conocen notaciones alternativas a los diagramas UML, por ejemplo (R3)

- Archimate, véase [Archimate].

  Ppara vistas en tiempo de ejecución, por ejemplo, diagramas de flujo, listas numeradas o Business Process Model & Notation (BPMN).

OA 3-4: Explicar y utilizar vistas arquitectónicas (R1)

Los arquitectos de software son capaces de utilizar las siguientes vistas arquitectónicas:

- Vista de contexto.
- Vista de bloque estructural o componente (composición de bloques estructurales de software).
- Vista en tiempo de ejecución (vista dinámica, interacción entre los bloques estructurales en tiempo de ejecución, máquinas de estado).
- Vista de despliegue (infraestructura de hardware y técnica, así como el mapeo de los bloques estructurales en la infraestructura).

OA 3-5: Explicar y aplicar la vista de contexto de los sistemas (R1)

Los arquitectos de software son capaces de:

- Describir el contexto de los sistemas, por ejemplo, en forma de diagramas de contexto con explicaciones.
- Representar las interfaces externas de los sistemas en la vista de contexto.
- Diferenciar el contexto de negocio y el contexto técnico.

OA 3-6: Documentar y comunicar conceptos transversales (R2)

Los arquitectos de software son capaces de documentar y comunicar adecuadamente los conceptos transversales típicos (sinónimo: principios, aspectos), por ejemplo, la persistencia, la gestión del flujo de trabajo, la interfaz de usuario, el despliegue/integración, el registro.

OA 3-7: Describir interfaces (R1)

Los arquitectos de software son capaces de describir y especificar tanto las interfaces internas como las externas.

OA 3-8: Explicar y documentar las decisiones arquitectónicas (R1-R2)

Los arquitectos de software son capaces de:

- Tomar, justificar, comunicar y documentar de forma sistemática decisiones arquitectónicas.
- Identificar, comunicar y documentar las interdependencias entre las decisiones de diseño.

Los arquitectos de software conocen los Registros de Decisiones de Arquitectura (por sus siglas en inglés ADR, ver [Nygard 2011]) y pueden aplicarlos para documentar las decisiones (R2).

OA 3-9: Utilizar la documentación como comunicación escrita (R2)

Los arquitectos de software utilizan la documentación para dar soporte al diseño, la implementación y el desarrollo ulterior (también llamado mantenimiento o evolución) de los sistemas.

OA 3-10: Conocer recursos y herramientas adicionales para la documentación (R3)

Los arquitectos de software conocen:

- Fundamentos de diferentes marcos de trabajo públicos para la descripción de las arquitecturas de software, por ejemplo:
  - ISO/IEEE-42010 (antes 1471), ver [ISO 42010]
  - arc42, véase [arc42]
  - C4, ver [Brown]
  - FMC, ver [FMC]
- Ideas y ejemplos de listas de control para la creación, documentación y prueba de arquitecturas de software.
- Posibles herramientas para crear y mantener la documentación arquitectónica.

Referencias

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

4. Arquitectura de software y calidad

Duración: 60 min.	Tiempo propuesto para ejercicios: 60 min.

Términos fundamentales

Calidad; características de calidad (también llamados atributos de calidad); DIN/ISO 25010; escenarios de calidad; árbol de calidad; compromisos entre las características de calidad; valoración cualitativa de la arquitectura; métricas y valoración cuantitativa

Objetivos

OA 4-1: Debatir sobre modelos de calidad y características de calidad (R1)

Los arquitectos de software pueden:

- Explicar el concepto de calidad (basado en la norma DIN/ISO 25010, anteriormente 9126) y las características de calidad.
- Explicar los modelos de calidad genéricos (como DIN/ISO 25010).
- Explicar las correlaciones y compromisos de las características de calidad, por ejemplo:
  - Configurabilidad frente a fiabilidad.
  - Requisitos de memoria frente a la eficiencia de desempeño.
  - Seguridad frente a usabilidad.
  - Flexibilidad en tiempo de ejecución frente a mantenibilidad.

OA 4-2: Precisar los requisitos de calidad de las arquitecturas de software (R1)

Los arquitectos de software pueden:

- Aclarar y formular requisitos de calidad específicos para el software que se va a desarrollar y sus arquitecturas, por ejemplo, en forma de escenarios y árboles de calidad.
- Explicar y aplicar escenarios y árboles de calidad.

OA 4-3: Llevar a cabo un análisis cualitativo y una evaluación de las arquitecturas de software (R2-R3)

Los arquitectos de software:

- Conocen los enfoques metódicos para el análisis cualitativo y la evaluación de las arquitecturas de software (R2), por ejemplo, como se especifica en ATAM (R3).
- Pueden analizar y evaluar cualitativamente sistemas más pequeños (R2).
- Saben que las siguientes fuentes de información pueden ayudar en el análisis cualitativo y la evaluación de las arquitecturas (R2):
  - Requisitos de calidad, por ejemplo, en forma de árboles y escenarios de calidad.
  - Documentación de la arquitectura.
  - Arquitectura y modelos de diseño.

- Código fuente.
- Métricas.
- Otra documentación del sistema, como los requisitos, la documentación operativa o de prueba.

OA 4-4: Llevar a cabo una evaluación cuantitativa de las arquitecturas de software (R2)

Los arquitectos de software conocen los enfoques para el análisis cuantitativo y la evaluación (medición) del software.

Ellos saben que:

- La evaluación cuantitativa puede contribuir a identificar las partes críticas dentro de los sistemas.
- Información adicional puede ser útil para la evaluación de las arquitecturas, por ejemplo:
  - Documentación de requisitos y arquitectura.
  - Código fuente y métricas relacionadas como líneas de código, complejidad (ciclomática), dependencias de entrada y salida.
  - Defectos conocidos en el código fuente, especialmente en las agrupaciones de defectos.
  - Casos de prueba y resultados de prueba.

Referencias

[Bass2003], [Clements+2002], [Gharbi+2020], [Martin 2003], [Starke 2020]

5. Ejemplos de arquitecturas de software

Duración: 90 min.	Tiempo propuesto para ejercicios: sin tiempo

Esta sección no es relevante para el examen.

Objetivos

OA 5-1: Conocer la relación entre las necesidades, restricciones y soluciones (R3)

Los arquitectos de software han utilizado por lo menos un ejemplo para identificar y comprender la relación entre los requisitos y las restricciones, y las decisiones de solución.

OA 5-2: Conocer los fundamentos de la implementación técnica de una solución (R3)

Los arquitectos de software entienden la realización técnica (implementación, conceptos técnicos, productos utilizados, decisiones arquitectónicas, estrategias de solución) de al menos una solución.

Referencias

- [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, 2nd 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. [Conocimientos básicos de la arquitectura de software.] 4.ª edición, dpunkt Verlag, Heidelberg 2020.
- [Goll 2014] Joachim Goll: Architektur- und Entwurfsmuster der Softwaretechnik: Mit lauffähigen Beispielen in Java. [Patrones arquitectónicos y de diseño en la ingeniería de software. Con ejemplos ejecutables en Java.] Springer-Vieweg Verlag, 2.ª edición 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 References] Gernot Starke et. al. Annotated collection of Software Architecture References, for Foundation and Advanced Level Curricula. Freely available https://leanpub.com/isaqbreferences.

- [Keeling 2017] Michael Keeling. Design It!: From Programmer to Software Architect. Pragmatic Programmer. ISBN 978-1680502091.
- [Lilienthal 2018] Carola Lilienthal: Langlebige Softwarearchitekuren. [Arquitectura de software duradera.] 2.ª edición, 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. [Eficiencia en la arquitectura de software. Una guía práctica.] 9.ª edición, Carl Hanser Verlag 2020. Página 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. [Documentar y comunicar la arquitectura de software.] 2.ª edición, Carl Hanser Verlag 2015.