Piano di Studio

Certified Professional for Software Architecture (CPSA®)

Foundation Level

2025.1-rev2-IT-20250427

Indice dei contenuti

Note Legali

L'abbreviazione "e. V." è parte del nome ufficiale di iSAQB e significa "eingetragener Verein" (associazione registrata), che descrive il proprio status come persona giuridica in base alle normative tedesche. A scopo di semplicità, di seguito iSAQB e.V. verrà denominata solamente "iSAQB" senza l'utilizzo di tale abbreviazione.

Indice degli obiettivi di apprendimento

OA 05-03: Valutare la conformità alle decisioni architetturali (R2)

OA 06-01 [ex OA 5-1]: Conoscere la relazione tra requisiti, vincoli e soluzioni (R3)

OA 06-02 [ex OA 5-2]: Comprendere l’implementazione tecnica di una soluzione (R3)

Introduzione

Cosa contiene questo piano di studio

Questo piano di studio per Certified Professional for Software Architecture – Foundation Level (CPSA-F) include gli obiettivi di apprendimento che si dovrebbero padroneggiare per assumere il ruolo di architetto/architetta software.

La sua struttura si orienta alle attività e responsabilità fondamentali dell'architettura software come ruolo atto a:

- Chiarire requisiti e vincoli degli stakeholder
- Progettare e sviluppare architetture software, prendendo decisioni strutturali e concettuali
- Comunicare e documentare l'architettura per stakeholder diversi
- Analizzare e valutare architetture software

Cosa trasmette una formazione di livello Foundation?

Le formazioni con licenza per *Certified Professional for Software Architecture – Foundation Level (CPSA-F)* trasmettono conoscenze e competenze di base per progettare e documentare un'architettura software adeguata a soddisfare i rispettivi requisiti per sistemi software di piccole e medie dimensioni. I/le partecipanti ampliano e approfondiscono le loro esperienze e capacità già esistenti nello sviluppo software, acquisendo approcci, metodi e principi rilevanti per lo sviluppo di architetture software. Grazie a quanto appreso, sono in grado di progettare, comunicare, analizzare, valutare ed evolvere ulteriormente un'adeguata architettura software sulla base di requisiti e vincoli sufficientemente dettagliati. Le formazioni CPSA-F trasmettono conoscenze di base indipendentemente da specifici metodi di progettazione, modelli di processo, linguaggi di programmazione o strumenti. In questo modo, i partecipanti possono applicare le competenze acquisite a un ampio spettro di casi d'uso.

L'attenzione è posta sull'acquisizione delle seguenti capacità:

- Allineare decisioni architetturali essenziali con stakeholder da ambiti quali gestione dei requisiti, management, sviluppo e test
- Comprendere i passaggi essenziali della progettazione dell'architettura software e svolgerli autonomamente per sistemi di piccole e medie dimensioni
- Documentare e comunicare le architetture software basandosi su viste architetturali, pattern architetturali e concetti tecnici

Inoltre, le formazioni CPSA-F trattano:

- Il termine architettura software e il suo significato
- Le attività e le responsabilità degli/delle architetti/e software
- Il ruolo degli/delle architetti/e software nei progetti di sviluppo
- Metodi e tecniche all'avanguardia per lo sviluppo di architetture software

Argomenti non trattati

Questo piano di studio riflette il contenuto che, secondo l'attuale visione di iSAQB, è necessario e sensato per raggiungere gli obiettivi di apprendimento del CPSA-F. Non costituisce una descrizione completa dell'intero dominio di "architettura software".

I seguenti temi o concetti non fanno parte del CPSA-F:

- Tecnologie di implementazione specifiche, framework o librerie
- Programmazione o linguaggi di programmazione
- Modelli di processo specifici
- Fondamenti della modellazione o delle relative notazioni (ad es. UML)
- Analisi di sistema e ingegneria dei requisiti (requirements engineering) (si veda il programma di formazione e certificazione di IREB e. V., https://ireb.org, International Requirements Engineering Board)
- Software testing (si veda il programma di formazione e certificazione di ISTQB e. V., https://istqb.org, International Software Testing Qualification Board)
- Project o product management
- Introduzione a strumenti software specifici

L'obiettivo della formazione è fornire le basi per acquisire le conoscenze e le competenze avanzate necessarie per i singoli casi applicativi.

Prerequisiti

iSAQB e. V. può verificare il rispetto dei requisiti qui indicati, durante gli esami di certificazione, mediante domande specifiche.

I/le partecipanti dovrebbero possedere le conoscenze e/o esperienze elencate di seguito. In particolare, una sostanziale esperienza pratica nello sviluppo software in un team costituisce un importante prerequisito per comprendere i contenuti e per una certificazione di successo.

- Più di 18 mesi di esperienza pratica nello sviluppo software, ottenuta attraverso lo sviluppo svolto in team, di diversi sistemi software al di fuori della formazione formale.
- Conoscenza ed esperienza pratica con almeno un linguaggio di programmazione di alto livello, in particolare:
  - Concetti di
    - Modularizzazione (package, namespace, ecc).
    - Passaggio di parametri (*Call-by-Value, Call-by-Reference*).
    - Ambito di validità (scope), e.g. dichiarazione variabili e definizione di tipo.
  - Principi base sui sistemi di tipi (tipizzazione statica o dinamica, tipi di dati generici).
  - Gestione degli errori e delle eccezioni nel software.
  - Problemi potenziali correlati a stato globale e variabili globali.
- Conoscenza di base di:
  - Modellazione e astrazione.
  - Algoritmi e strutture di dati (come Liste, Alberi, HashTable, Dizionari e Mappe).
  - UML, Unified Modeling Language (class, package, component e sequence diagram) e relativa relazione con il codice sorgente.
  - Approcci al testing del software (ad es. unit e acceptance testing)

Inoltre, i seguenti argomenti saranno utili per la comprensione di alcuni concetti:

- Nozioni di base e differenze tra programmazione imperativa, dichiarativa, object-oriented e funzionale.
- Esperienza pratica
  - In un linguaggio di programmazione ad alto livello.
  - Nella progettazione e implementazione di applicazioni distribuite, come sistemi Client/Server o applicazioni web.
  - Nella documentazione tecnica, in particolare nella documentazione di codice sorgente, progettazione di sistemi o concetti tecnici.

Struttura, durata e metodi didattici

I tempi di studio riportati nei seguenti capitoli del piano di studio si intendono esclusivamente come raccomandazioni. La durata di un corso per la certificazione dovrebbe essere almeno di tre giorni, ma può essere più lunga. I Training Provider possono variare il loro approccio nella durata, nei metodi didattici, nel tipo e nella struttura degli esercizi oltre che per la suddivisione dettagliata del corso. In particolare, la tipologia (domini e tecnologie) degli esempi e degli esercizi può essere definita individualmente dai Training Provider.

Contenuto	Durata suggerita (min)
1. Concetti Fondamentali dell'Architettura Software	120
2. Requisiti e Vincoli	180
3. Progettazione e Sviluppo	360
4. Specifica e Comunicazione	240
5. Analisi e Valutazione	90
6. Esempi	90
Totale	1080

Obiettivi di Apprendimento e Importanza per l'Esame

La struttura dei capitoli del piano di studio si basa su un insieme di obiettivi di apprendimento ordinati per priorità. L'importanza per l'esame di ogni Obiettivo di Apprendimento, o dei suoi sotto elementi, è riportata espressamente (secondo la classificazione R1, R2 o R3, come da tabella seguente). Ogni Obiettivo di Apprendimento descrive i contenuti che devono essere trattati, inclusi i termini e i concetti chiave.

Relativamente all'importanza per l'esame, il piano di studio utilizza le categorie seguenti:

ID	Categoria dell'Obiettivo di Apprendimento	Significato	Importanza per l'esame
R1	Essere in grado di	I/le partecipanti, dopo il corso, devono saper applicare autonomamente questi contenuti, i quali dovrebbero essere coperti tramite esercizi e discussioni durante il corso.	I contenuti **saranno** parte dell'esame.
R2	Comprendere	I/le partecipanti devono comprendere questi contenuti in linea di principio. In genere non vengono approfonditi tramite esercizi durante il corso.	I contenuti **possono** essere parte dell'esame.
R3	Conoscere	Questi contenuti (termini, concetti, metodi, pratiche o similari) possono aiutare la comprensione o motivare l'argomento. Se necessario, possono essere coperti durante il corso.	I contenuti **non sono** parte dell'esame.

Se necessario, gli obiettivi di apprendimento includono riferimenti a letteratura, standard o altre fonti. Le sezioni "Termini Rilevanti" all'inizio di ciascun capitolo mostrano parole collegate al contenuto del capitolo e in parte usate nelle descrizioni degli obiettivi di apprendimento.

Versione attuale e archivio pubblico

La versione più recente di questo documento è disponibile per il download dalla pagina ufficiale https://isaqb-org.github.io/.

Il documento è conservato in un archivio pubblico su https://github.com/isaqb-org/curriculum-foundation, tutti i cambiamenti e le modifiche sono visibili.

Si prega di segnalare eventuali problemi utilizzando il nostro issue tracker pubblico su https://github.com/isaqb-org/curriculum-foundation/issues.

1. Concetti Fondamentali dell’Architettura Software

Durata: 120 min.	Durata esercitazione: nessuna

Obiettivo

L'obiettivo di questa sezione è fornire ai/alle partecipanti una comprensione di base dei termini e concetti più importanti dell'architettura software. Fare conoscere diverse definizioni e i loro elementi comuni, far comprendere obiettivi e vantaggi essenziali dell'architettura software e mettere in grado i partecipanti di trasmetterli ad altri stakeholder. Inoltre, i partecipanti potranno indicare e spiegare le attività e le responsabilità principali degli/delle architetti/e software. Viene anche trattato il ruolo degli/delle architetti/e software nel contesto organizzativo e la loro interazione con altri stakeholder, per permettere ai/alle partecipanti di contribuire efficacemente a progetti di sviluppo software di vario tipo.

Termini Rilevanti

Architettura software; domini architetturali; struttura; elementi costitutivi (building block); componenti; interfacce; relazioni; cross-cutting concern; benefici dell'architettura software; architetti/e software e loro responsabilità; attività e competenze richieste; stakeholder e loro esigenze; requisiti; vincoli; fattori d'influenza;

Obiettivi di Apprendimento

OA 01-01: Comprendere le definizioni di architettura software (R1)

Gli/le architetti/e software conoscono e comprendono gli elementi comuni di molte definizioni di architettura software:

- Componenti/elementi costitutivi con interfacce e relazioni
- Elementi costitutivi come termine generale, componenti come forma speciale degli elementi costitutivi
- Strutture, cross-cutting concerns, principi
- Decisioni architetturali e loro impatto sull'intero sistema e sul suo ciclo di vita

Riferimenti
[ISO 42010], [Bass+2021], [Kruchten 2004], [Starke+2023a]

OA 01-02: Comprendere e spiegare obiettivi e benefici dell'architettura software (R1)

Gli/le architetti/e software possono motivare i seguenti obiettivi e benefici essenziali dell'architettura software:

- Supportare la progettazione, l’implementazione, la manutenzione e l’operatività dei sistemi
- Raggiungere requisiti funzionali o garantirne la loro raggiungibilità
- Raggiungere requisiti come affidabilità, manutenibilità, modificabilità, sicurezza,

efficienza energetica, ecc.

- Garantire la comprensione di strutture e concetti del sistema da parte di tutti gli stakeholder rilevanti
- Ridurre sistematicamente la complessità
- Specificare le linee guida architetturalmente rilevanti per l'implementazione e l'operatività

OA 01-03: Conoscere gli impatti a lungo termine dell'architettura software (R3)

Gli/le architetti/e software conoscono:

- La relazione tra decisioni architetturali e futura adattabilità e manutenibilità del sistema
- Gli impatti dei cambiamenti di requisiti, tecnologie o ambiente di sistema sulle decisioni architetturali esistenti
- Le conseguenze di lungo periodo delle decisioni architetturali sulle diverse caratteristiche di qualità del sistema
- Le interdipendenze tra sistemi IT e i processi di business e operativi che essi supportano

Sanno come analizzare gli impatti delle decisioni architetturali sull'evoluzione di lungo periodo di un sistema.

Riferimenti
[Bass+2021], [Lehman 1980], [Wiki-LehmansLaws], [Lilienthal 2024], [Ford+2023], [Rajlich+2000], [Richards+2020]

OA 01-04: Comprendere attività e responsabilità degli/delle architetti/e software (R1)

Gli/le architetti/e software sono responsabili dello sviluppo di un'architettura software che soddisfi i requisiti dati. Devono allineare tale responsabilità, a seconda dell'approccio o del modello di processo effettivamente utilizzato, con la responsabilità complessiva del progetto del project management o di altri ruoli.

Attività e responsabilità degli/delle architetti/e software:

- Chiarire e analizzare i requisiti e i vincoli. Coordinare e concordare affinamenti necessari con i rispettivi stakeholder
- Prendere decisioni sulla decomposizione del sistema in elementi costitutivi, determinando al contempo dipendenze e interfacce tra questi ultimi
- Decidere i cross-cutting concerns (ad es. persistenza, comunicazione, GUI)
- Comunicare e documentare l'architettura software tramite viste, pattern architetturali, cross-cutting concerns e temi tecnici
- Accompagnare la realizzazione e l'implementazione dell'architettura, integrare feedback degli stakeholder rilevanti nell'architettura se necessario, verificare e garantire la coerenza tra il codice sorgente e l'architettura software
- Analizzare e valutare l'architettura software, in particolare rispetto ai rischi che coinvolgono il

raggiungimento dei requisiti

- Identificare, evidenziare e giustificare le conseguenze delle scelte architetturali agli altri stakeholder.

Gli architetti software dovrebbero riconoscere autonomamente la necessità di iterazioni in tutte le attività e indicare le possibilità di fornire un feedback adeguato.

OA 01-05 [ex OA 1-9]: Distinzione tra architettura software e altri domini di architettura (R3)

Il focus del CPSA-Foundation Level di iSAQB è su strutture e concetti di sistemi software singoli. Inoltre gli/le architetti/e software conoscono altri domini di architettura, ad esempio:

- Architettura IT aziendale (Enterprise IT Architecture): struttura degli scenari applicativi
- Architettura di business e di processo (Business and Process Architecture): struttura, tra l'altro, dei processi di business
- Architettura dell'informazione: struttura e uso cross-sistema di informazioni e dati
- Architettura infrastrutturale o tecnologica: struttura dell'infrastruttura tecnica, hardware, reti, ecc.
- Architettura hardware o del processore (per sistemi relativi all'hardware)
- Architettura di sistema (può avere significati diversi a seconda della definizione del termine "sistema")

Questi domini di architettura non sono focus del contenuto del CPSA-F.

OA 01-06: Mettere in relazione il ruolo degli/delle architetti/e software con altri stakeholder (R1)

Gli/le architetti/e software sono in grado di spiegare il proprio ruolo. Dovrebbero adattare il proprio contributo allo sviluppo del sistema a seconda del contesto specifico e in relazione ad altri stakeholder e unità organizzative, in particolare rispetto a:

- Product management e product owner
- Project manager
- Requirement engineer (requirement/business analyst, requirement manager, system analyst, business owner, subject-matter expert, ecc.)
- Sviluppatori
- Quality assurance e tester
- IT operator e administrator (si applica principalmente ad ambienti di produzione o data center per sistemi informativi)
- Sviluppatori hardware e architetti di sistema, (si applica principalmente per sistemi embedded e relativi all'hardware)
- Enterprise architect e architecture board member

OA 01-07: Importanza dei dati e dei modelli di dati (R2)

Gli/le architetti/e software comprendono l'importanza dei dati e dei modelli di dati (indipendentemente dalla loro rappresentazione fisica) per l'architettura. Essi/esse:

- Possono identificare modelli di dati che influenzano in modo determinante l'architettura
- Possono progettare tali modelli di dati in modo sistematico
- Comprendono la differenza tra prodotti e somme nella modellazione dei dati

Gli/le architetti/e software:

- Comprendono l'importanza di disaccoppiare i modelli di dati dalla loro rappresentazione in database, file e protocolli di trasmissione
- Possono spiegare gli impatti dei dati sulle decisioni architetturali, ad es. rispetto a storage, sicurezza, scalabilità, affidabilità, performance, ecc.

Riferimenti:
[Felleisen+2014], [Sperber+2023], [Sperber+2024], [Kleppmann 2017], [Ford+2021]

2. Requisiti e Vincoli

Durata: 90 min.	Durata esercitazione: 90 min.

Obiettivo

Questa sezione approfondisce la comprensione delle esigenze degli stakeholder, dei requisiti e della qualità dell'architettura software. I/le partecipanti imparano a riconoscere l'influenza degli stakeholder sulle decisioni architetturali e a valutare conflitti e sinergie nel contesto di progetti di sviluppo software. Confrontandosi con diversi requisiti e vincoli, ottengono una panoramica su come considerare efficacemente le esigenze degli stakeholder e i vincoli di progetto. Riconoscono inoltre l'importanza delle qualità di un sistema software come driver per la progettazione architetturale e sanno formulare tali requisiti con l'aiuto di scenari.

Termini Rilevanti

Qualità; caratteristiche di qualità (chiamate anche attributi di qualità); DIN/ISO 25010; Q42; scenari di qualità; compromessi e interazioni tra caratteristiche di qualità; requisiti; vincoli; esigenze degli stakeholder.

Obiettivi di Apprendimento

OA 02-01: Comprendere le esigenze degli stakeholder (R1, R3)

Gli/le architetti/e possono identificare gli stakeholder e le loro esigenze, nonché gli impatti di tali esigenze sull'architettura software o sul processo di progettazione e sviluppo. (R1)

Esempi di stakeholder e relative esigenze (R3):

Stakeholder	**Esigenze degli stakeholder**
Product management e product owner	ad es. tempo necessario per implementare i requisiti
Sviluppatori	ad es. componenti e interfacce da implementare, protocolli, requisiti tecnici e vincoli
Requirements Engineer, Product Owner, Business Analyst	ad es. soddisfacimento dei requisiti
Project manager	ad es. tempo e budget necessari per l'implementazione, rischi connessi all'approccio architetturale scelto
Quality assurance e tester	ad es. testing isolato dei componenti
Operation	ad es. requisiti infrastrutturali legati all'operatività del sistema

Gli/le architetti/e software sono in grado di identificare potenziali conflitti tra obiettivi di breve e lungo termine (ad es. obiettivi di business e di progetto vs obiettivi architetturali e di manutenibilità) e comprendono che devono coinvolgere gli stakeholder rilevanti per risolverli. (R1)

Comprendono che non tutte le esigenze degli stakeholder possono o verranno trasformate in requisiti, ma devono comunque essere considerate. (R3)

Possono usare le esigenze degli stakeholder per scoprire requisiti mancanti o contraddittori e/o validare requisiti e vincoli sull'architettura, ad es. in interviste con gli stakeholder. (R3)

OA 02-02 [ex OA 2-3]: Chiarire e considerare requisiti e vincoli (R1-R3)

Gli/le architetti/e software comprendono che sia i requisiti sia i vincoli possono avere impatti sull'architettura e sul lavoro architetturale (R2). Sono in grado di chiarire requisiti e vincoli e di considerarli nella progettazione architetturale e nel processo di sviluppo. Comprendono che le loro decisioni possono introdurre nuovi requisiti o rendere necessari cambiamenti ai requisiti esistenti.

Riconoscono e considerano l'influenza di:

- Requisiti relativi al prodotto come (R1)
  - Requisiti funzionali
  - Requisiti di qualità
- Vincoli tecnologici come
  - Infrastruttura hardware e software esistente o pianificata (R1)
  - Vincoli tecnologici per strutture di dati e interfacce (R2)
  - Architetture di riferimento, librerie, componenti e framework (R1)
  - Linguaggi di programmazione (R2)
- Vincoli organizzativi come
  - Struttura organizzativa dei team di sviluppo e del cliente (R1), in particolare la legge di Conway (R2)
  - Cultura aziendale e di team (R3)
  - Partnership e cooperazioni (R2)
  - Standard, linee guida e modelli di processo (ad es. processi di approvazione e release) (R2)
  - Disponibilità di risorse come budget, tempo e personale (R1)
  - Disponibilità, insieme delle competenze e coinvolgimento del personale (R1)
- Vincoli normativi come (R2)
  - Vincoli legali locali e internazionali

- Questioni contrattuali e di responsabilità
- Leggi sulla protezione dei dati e della privacy
- Requisiti di conformità o obblighi di prova

• Trend come (R3)

- Trend di mercato
- Trend tecnologici (ad es. cloud, microservizi, container, IA generativa o LLM)
- Trend metodologici (ad es. Agile)

Gli/le architetti/e software sono in grado di descrivere come tali influenze impattano sulle decisioni architetturali e spiegare con esempi le conseguenze di un cambiamento dei fattori di influenza (R2).

Riferimenti
[IREB Foundation], [Bass+2021], [Gharbi+2024], [Starke 2024], [Richards+2020], [Pohl 2025]

OA 02-03 [ex OA 4-1]: Comprendere e spiegare le qualità di un sistema software (R1)

Gli/le architetti/e software sanno che il termine "qualità" viene usato diversamente in contesti differenti:

- Si riferisce, nel contesto del quality management, al senso di “bontà/qualità intrinseca”, e
- Si riferisce ad una "proprietà specifica (di un sistema software)" negli altri contesti

In questo obiettivo di apprendimento viene trattato il secondo significato.

Gli/le architetti/e software sono in grado di spiegare che:

- Esistono diverse tassonomie per categorizzare le qualità dei sistemi software
- Alcune categorizzazioni distinguono tra funzionalità e qualità, ad es. IREB [IREB Foundation]
- Gli/le architetti/e software sono in grado di influenzare le qualità di un sistema software
- Modificare una qualità può influenzarne altre, rendendo necessarie valutazioni (trade-off), come ad esempio:
  - configurabilità vs affidabilità
  - consumo di memoria vs efficienza prestazionale
  - sicurezza vs usabilità
  - flessibilità a runtime vs manutenibilità

Comprendono che un singolo requisito può riferirsi a più qualità.

Riferimenti
[IREB Foundation], [ISO 25010], [Bass+2021], [Q42]

OA 02-04 [ex OA 4-2]: Formulare i requisiti relativi alle qualità (R1-R3)

Gli/le architetti/e software:

- Possono formulare scenari per determinate qualità, includendo contesto, stimolo, reazione e misurazione, per diversi scopi, ad esempio chiarire requisiti, fornire input a valutazioni architetturali, ecc. (R1)
- Comprendono che un requisito per una determinata qualità dovrebbe specificare un metodo di analisi (vedi OA 05-02 (ex OA 4-3 e 4-4): Analizzare le qualità di un sistema software (R1, R2)) (R1)
- Sanno che usare una metrica come obiettivo può portare alla sua invalidazione (R2), ad esempio come descritto nella legge di Goodhart (R3)

Riferimenti
[Bass+2021], [Q42], [ISO 25010]

OA 02-05 [ex OA 1-8]: Preferire affermazioni esplicite ad assunzioni implicite (R1)

Gli/le architetti/e software:

- Sono in grado di rendere le assunzioni esplicite, evitando quindi assunzioni implicite
- Sanno che le assunzioni implicite possono causare fraintendimenti tra gli stakeholder coinvolti

3. Progettazione e Sviluppo delle Architetture Software

Durata: 270 min.	Durata esercitazione: 90 min.

Obiettivo

Questa sezione mira a mettere i/le partecipanti in condizione di prendere decisioni architetturali tali da soddisfare i requisiti degli stakeholder ottemperando i vincoli del contesto di sistema. Imparano a sviluppare progetti architetturali, prendere decisioni motivate sulla decomposizione del sistema e a modellare le dipendenze tra gli elementi costitutivi. A tal fine, imparano ad applicare approcci ed euristiche di base nello sviluppo dell'architettura. Riconoscono l'importanza di principi di progettazione e pattern di soluzione e sanno applicarli. Inoltre, in questa sezione vengono trattati la gestione dei cross-cutting concerns, i principi del deployment del software e le sfide dei sistemi distribuiti.

Termini Rilevanti

Progettazione; approcci alla progettazione; decisione architetturale; view (viste); interfacce; concetti tecnici e cross-cutting concerns; pattern architetturali; pattern di progettazione; linguaggi di pattern; principi di progettazione; dipendenze; accoppiamento; coesione; approcci top-down e bottom-up; progettazione model-based; progettazione iterativa; Domain-Driven Design.

Obiettivi di Apprendimento

OA 03-01 [ex OA 2-8, nuovi contenuti]: Soddisfare i requisiti tramite l'architettura (R1)

Gli/le architetti/e software:

- Comprendono che le attività architetturali dovrebbero essere guidate dalla necessità di raggiungere o migliorare qualità specifiche
- Sono in grado di proporre un'architettura adatta a soddisfare i requisiti
- Sono in grado di stimare quali qualità migliorano tramite specifiche attività o decisioni architetturali
- Sono in grado di identificare e comunicare trade-off tra alternative progettuali e i rischi associati

OA 03-02 [ex OA 2-2]: Progettare architetture software (R1)

Gli/le architetti/e software sono in grado di:

- Progettare, comunicare e documentare in modo adeguato architetture software per sistemi software non safety-critical o business-critical, sulla base di requisiti funzionali e di qualità noti
- Prendere decisioni strutturali sulla decomposizione del sistema e sulla struttura degli elementi costitutivi, definendo dipendenze tra gli elementi costitutivi (vedi OA 03-06 (ex OA 2-7) Gestire le dipendenze tra building blocks (elementi costitutivi) (R1))
- Riconoscere e motivare interdipendenze e trade-off legati alle decisioni architetturali
- Spiegare e applicare in modo mirato i concetti di black-box e white-box

- Effettuare affinamento e specifica graduale degli elementi costitutivi
- Progettare viste architetturali, in particolare vista degli elementi costitutivi, vista di runtime e vista di deployment (vedi OA 04-05 (ex OA 3-4) Spiegare e applicare le view (viste) architetturali)
- Spiegare le conseguenze delle decisioni sul codice sorgente corrispondente
- Distinguere nelle architetture gli elementi relativi al dominio da quelli tecnici e motivare tale distinzione
- Identificare i rischi relativi alle decisioni architetturali

Riferimenti:
[Kruchten 1995], [Rozanski+2011], [Starke+2023a], [arc42], [Brown], [Starke 2024]

OA 03-03 [ex OA 2-1]: Selezionare e usare approcci ed euristiche per lo sviluppo dell'architettura (R1, R3)

Gli/le architetti/e software sono in grado di nominare, spiegare e applicare approcci fondamentali allo sviluppo dell'architettura, ad esempio:

- Approccio top-down e bottom-up, vedi [Gharbi+2024], [Starke 2024] (R1)
- Sviluppo architetturale view-based, vedi [Rozanski+2021], [Kruchten 1995] (R1)
- Domain-Driven Design, vedi [Evans 2004] (R3)
- Architettura evolutiva, vedi [Ford+2023] (R3)
- Analisi globale, vedi [Hofmeister+1999] (R3)
- Progettazione model-based (R3)

OA 03-04 [ex OA 2-6]: Spiegare e applicare principi di progettazione (R1-R3)

Gli/le architetti/e software sanno spiegare cosa sono i principi di progettazione. Sono in grado di illustrarne gli obiettivi generali e la loro applicazione nell'ambito dell'architettura software. (R2)

Sono in grado di:

- Spiegare i principi di progettazione elencati di seguito e illustrarli con esempi
- Spiegare come questi principi debbano essere applicati
- Illustrare in che modo i requisiti influenzano l'applicazione di questi principi
- Illustrare gli impatti dei principi di progettazione sull'implementazione
- Analizzare il codice sorgente e la progettazione dell'architettura per valutare se tali principi di progettazione sono stati applicati o dovrebbero essere applicati

Astrazione (R2):

- Come mezzo volto alla derivazione di generalizzazioni utili
- Come tecnica di progettazione in cui gli elementi costitutivi dipendono dalle astrazioni e non

dalle implementazioni

- Interfacce come astrazioni

Modularizzazione (R1):

- Information hiding e incapsulamento
- Separazione delle responsabilità (Separation of Concerns - SoC)
- Accoppiamento debole ma funzionalmente sufficiente (R1) degli elementi costitutivi (vedi OA 03-06 [ex OA 2-7] Gestire le dipendenze tra building blocks(elementi costitutivi) (R1))
- Alta coesione
- Open/closed principle
- Dependency inversion principle tramite interfacce o astrazioni simili

Integrità concettuale (R2-R3):

- Significa ottenere l'uniformità (omogeneità, coerenza) delle soluzioni per problemi simili (R2)
- Come mezzo per raggiungere il principio della minima sorpresa (principle of least surprise, principle of least astonishment POLA) (R3)
- Liskov's substitution principle come modo per raggiungere coerenza, integrità concettuale e robustezza (nel senso di type safety) (R3)

Riduzione della complessità (R3):

- Come motivazione per KISS, YAGNI e CUPID vedi [Terhorst-North 2022]
- DRY (Don't Repeat Yourself) come modo per evitare ripetizioni

Aspettarsi errori (R2-R3):

- Come mezzo per progettare sistemi robusti e resilienti (R3)
- Come generalizzazione del principio di robustezza (legge di Postel) (R2)

Principi SOLID (R3):

Gli/le architetti/e software conoscono i vantaggi e i limiti dei principi SOLID: Single Responsibility Principle, Open/Closed Principle, Liskov's Substitution Principle, Interface Segregation Principle, Dependency Inversion Principle.

Riferimenti
[Liskov 1994], [SOLID]

OA 03-05 [ex OA 1-6]: Relazione tra cicli di feedback e rischi (R1, R2)

Gli/le architetti/e software comprendono la necessità di iterazioni, soprattutto per decisioni prese in condizioni di incertezza. Essi/esse:

- Sono in grado di spiegare l'impatto dell'approccio iterativo sulle decisioni architetturali (in termini di rischi e prevedibilità) (R1)
- Sono in grado di lavorare e decidere in modo iterativo (R1)
- Comprendono la necessità di feedback sulle decisioni architetturali (R1)
- Sono in grado di raccogliere sistematicamente feedback da altri stakeholder (R2)

OA 03-06 [ex OA 2-7]: Gestire le dipendenze tra building blocks (elementi costitutivi) (R1)

Gli/le architetti/e software comprendono le dipendenze e gli accoppiamenti tra gli elementi costitutivi e sono in grado di usarli in modo mirato. Essi/esse:

- Conoscono e comprendono diversi tipi di dipendenze degli elementi costitutivi (ad es. accoppiamento via uso/delega, messaggi/eventi, composizione, creazione, ereditarietà, accoppiamento temporale, accoppiamento via dati, tipi di dati o hardware)
- Comprendono come le dipendenze aumentino l'accoppiamento
- Sono in grado di distinguere almeno i seguenti tipi di accoppiamento:
    - accoppiamento statico e dinamico
    - accoppiamento efferente e afferente
- Sanno che sostituire l'accoppiamento statico con quello dinamico non riduce necessariamente l'accoppiamento sottostante
- Sanno riconoscere l'accoppiamento e valutarne gli effetti
- Sanno prendere decisioni motivate sul fatto che una dipendenza sia appropriata o vada rimossa alla luce dei requisiti e dei vincoli
- Conoscono le modalità per eliminare o ridurre l'accoppiamento e sono in grado di applicarle, ad esempio:
    - Pattern (modelli)
    - Principi fondamentali di progettazione
    - Esternalizzazione delle dipendenze, ovvero definire le dipendenze concrete solo al momento dell'installazione o del runtime, ad esempio tramite l'uso della Dependency Injection (R3) (vedi anche OA 03-08 [ex OA 2-5]: Descrivere, spiegare e applicare in modo adeguato i principali pattern architetturali (R1, R3))

Riferimenti
[Ford+2021]

OA 03-07 [ex OA 2-9]: Progettare e definire le interfacce (R1-R3)

Gli/le architetti/e software conoscono l'importanza critica delle interfacce per l'interazione tra elementi costitutivi architetturali o tra sistema ed elementi esterni e sono in grado di progettarle e specificarle.

Conoscono:

- Le proprietà desiderabili delle interfacce e come ottenerle in fase di progettazione (R1):
  - facili da imparare, usare ed estendere
  - difficili da usare impropriamente
  - funzionalmente complete dal punto di vista degli utenti o degli elementi costitutivi che le utilizzano
- La necessità di trattare diversamente interfacce interne ed esterne (R2)
- La separazione tra interfaccia e implementazione (R1):
  - le implementazioni possono essere sostituite se necessario
- Diverse caratteristiche delle interfacce, ad esempio (R3):
  - canali di trasporto (ad esempio: TCP/IP quale parte del modello OSI 7 livelli, memoria condivisa)
  - interne/esterne
  - locali/remote
  - sincrone/asincrone
  - binarie (leggibili solo dalle macchine) o testuali (leggibili anche dall'uomo)
  - stateless/stateful
  - punto-punto/multipunto (broadcast o multicast)
  - chiamata a funzione (remote procedure call) o scambio messaggi
  - batch, request-response o streaming
- Approcci di implementazione per interfacce di servizi remoti, ad esempio (R3):
  - orientati alla procedura (ad esempio: GraphQL, o servizi Web basati su WS/SOAP)
  - orientate alle risorse (REST, REpresentational State Transfer)

Vedi anche OA 04-06 [ex OA 3-7]: Documentare le interfacce (R1)

Riferimenti:
[Zimmermann+2022], [Geewax 2021]

OA 03-08 [ex OA 2-5]: Descrivere, spiegare e applicare in modo adeguato i principali pattern architetturali (R1, R3)

Gli/le architetti/e software sono in grado di spiegare i seguenti pattern architetturali e fornire esempi (R1):

- Layer
- Pipes and Filters
- Microservices

Gli/le architetti/e software sono in grado di spiegare alcuni dei seguenti pattern architetturali, illustrarne la rilevanza per sistemi concreti e fornire esempi (R3):

- Blackboard
- Broker
- CQRS (Command-Query-Responsibility-Segregation)
- Event Sourcing
- Dependency Injection (vedi anche OA 03-06 [ex OA 2-7]: Gestire le dipendenze tra building blocks (elementi costitutivi) (R1))
- Integration e Messaging-Patterns (ad esempio [Hohpe+2004])
- Remote Procedure Call
- MVC (Model View Controller), MVVM (Model View ViewModel), MVU (Model View Update), PAC (Presentation Abstraction Control)
- Plugin
- Ports and Adapters (sinonimi: Onion Architecture, Hexagonal Architecture, Clean Architecture)
- SOA (Service-Oriented Architecture)

Gli/le architetti/e software conoscono le principali fonti relative ai pattern architetturali, come ad esempio POSA (ad es. [Buschmann+1996]) e PoEAA ([Fowler 2002]) (per i sistemi informativi). (R3)

Sanno che:

- I pattern sono un modo per ottenere determinate qualità in relazione a problemi e requisiti specifici all'interno di contesti prestabiliti
- Esistono diverse categorie di pattern
- Esistono fonti aggiuntive di pattern specifici per settori tecnici o domini di applicazione

Riferimenti:
[Buschmann+1996], [Buschmann+2007], [Eilebrecht+2024], [Fowler 2002], [Gamma+ 1994], [Hohpe+2004], [Pethuru 2017]

OA 03-09 [ex OA 2-5]: Descrivere, spiegare e applicare correttamente i principali pattern di progettazione (R3)

Gli/le architetti/e software sono in grado di descrivere diversi dei seguenti pattern, spiegarne la rilevanza per l'architettura e per sistemi specifici nonché fare esempi:

- Combinator
- Pattern di interfaccia come Adapter, Facade e Proxy. Gli/le architetti/e dovrebbero sapere che questi pattern possono essere utilizzati indipendentemente da specifici linguaggi di programmazione o framework.
- Interpreter
- Observer
- Template Method e Strategy
- Visitor

Gli/le architetti/e software conoscono le principali fonti di pattern di progettazione, come ad esempio GOF e POSA.

Riferimenti
[Gamma+ 1994], [Buschmann+1996]

OA 03-10 [ex OA 2-4]: Identificare, progettare e implementare i cross-cutting concerns (R1)

Gli/le architetti/e software sono in grado di:

- Spiegare l'importanza dei cross-cutting concerns
- Identificarli
- Progettare concetti cross-cutting, tra i quali: persistenza, comunicazione, GUI, error handling, concorrenza, efficienza energetica
- Riconoscere e valutare interdipendenze potenziali

Gli/le architetti/e software sanno che tali concetti cross-cutting possono essere riutilizzabili trasversalmente nei sistemi.

Vedi anche OA 04-07 [ex OA 3-06]: Documentare e comunicare i cross-cutting concerns (R2).

Riferimenti
[Starke 2024], [Starke+2023a]

OA 03-11 [ex OA 2-10]: Conoscere i principi fondamentali del deployment del software (R3)

Gli/le architetti/e software:

- Sanno che il deployment del software è il processo con cui il software nuovo o aggiornato viene reso disponibile all'uso da parte degli utenti
- Possono elencare e spiegare concetti fondamentali del deployment del software:
  - Deployment automatizzati
  - Build ripetibili
  - Ambienti coerenti (ad esempio: attraverso l'uso di infrastruttura immutabile (immutable) e monouso (disposable))
  - Mettere tutto sotto version control
  - Le release possono essere facilmente riportate alla versione precedente (easy-to-undo)

Riferimenti:
[Humble+2010]

OA 03-12 [ex OA 1-11]: Conoscere le sfide dei sistemi distribuiti (R3)

Gli/le architetti/e software sono in grado di:

- Identificare la distribuzione in una data architettura software
- Analizzare i criteri di coerenza relativi ad un determinato problema funzionale
- Spiegare la causalità degli eventi in un sistema distribuito

Gli/le architetti/e software sanno:

- Che nei sistemi distribuiti la comunicazione può fallire
- Che nei sistemi distribuiti esistono limitazioni relative alla coerenza dei database

- Cos'è il problema dello "split-brain" e perché è difficile da risolvere
- Che in un sistema distribuito è impossibile determinare l'ordine temporale esatto degli eventi

Riferimenti
[Tanenbaum+], [Buschmann+2007], [Ford+2021], [Miller+]

4. Specifica e Comunicazione delle Architetture Software

Durata: 180 min.	Durata esercitazione: 60 min.

Obiettivo

Questa sezione mira a mettere i/le partecipanti in grado di documentare e comunicare architetture software in modo da soddisfare i bisogni dei principali stakeholder e supportare il processo di sviluppo. L'attenzione è rivolta alla comprensione dei requisiti fondamentali della documentazione tecnica, all'uso di modelli e notazioni adeguati alla descrizione delle architetture e all'applicazione delle principali viste architetturali. Inoltre, i/le partecipanti impareranno a documentare le decisioni architetturali rilevanti, le interfacce e i cross-cutting concerns, al fine di garantire una documentazione chiara, corretta e pertinente per gli stakeholder.

Termini Rilevanti

Viste (architetturali); strutture; concetti (tecnici); documentazione; comunicazione; descrizione; allineamento a gruppi target e alle esigenze degli stakeholders; strutture e template per la descrizione e comunicazione; contesto del sistema; elementi costitutivi; vista degli elementi costitutivi; vista di runtime; vista di deployment; nodo; canale; artefatti di deployment; mappatura degli elementi costitutivi sugli artefatti di deployment; mappatura degli artefatti di deployment sui nodi; descrizione delle interfacce e delle decisioni di progettazione; UML; strumenti per la documentazione.

Obiettivi di Apprendimento

OA 04-01 [ex OA 3-1]: Spiegare e considerare i requisiti della documentazione tecnica (R1)

Gli/le architetti/e software conoscono i requisiti fondamentali della documentazione tecnica e sono in grado di tenerne conto e soddisfarli quando si documentano i sistemi:

- Comprensibilità, correttezza, efficienza, appropriatezza, manutenibilità
- Forma, contenuto e livello di dettaglio adattati al pubblico destinatario della documentazione

Sanno che la comprensibilità della documentazione tecnica può essere valutata solo dai suoi destinatari.

Riferimenti
[Starke+2023a], [Zörner 2021], [Clements+2010]

OA 04-02 [ex OA 3-2]: Descrivere e comunicare le architetture software (R1-R3)

Gli/le architetti/e software usano la documentazione per supportare la progettazione, l'implementazione e successivi sviluppi (anche detti manutenzione o evoluzione) dei sistemi. (R2)

Gli/le architetti/e software (R1):

- Sono in grado di documentare e comunicare le architetture per le corrispondenti esigenze degli stakeholder, rivolgendosi così a diversi destinatari, ad esempio il management, i team di sviluppo, il QA, altri/e architetti/e software ed eventuali altri stakeholder
- Sono in grado di consolidare e armonizzare i contributi di diversi gruppi di autori dal punto di vista stilistico e di contenuto
- Sono in grado di sviluppare e implementare misure volte a supportare la coerenza tra la comunicazione orale e quella scritta e a trovare un adeguato equilibrio tra le due
- Conoscono i vantaggi della documentazione basata su template
- Sanno che diverse caratteristiche della documentazione dipendono dalle specifiche proprietà del sistema, dai suoi requisiti, dai rischi, dal processo di sviluppo, dall'organizzazione o da altri fattori

Sono in grado, ad esempio, di adattare le seguenti caratteristiche della documentazione in base alla situazione (R3):

- L'ampiezza e il livello di dettaglio della documentazione richiesta
- Il formato della documentazione
- L'accessibilità della documentazione
- Gli aspetti formali della documentazione (ad es. diagrammi conformi a un metamodello o semplici disegni)
- Le revisioni formali e i processi di approvazione della documentazione

OA 04-03 [ex OA 3-3]: Spiegare e applicare notazioni/modelli per descrivere l'architettura software (R2-R3)

Gli/le architetti/e software conoscono almeno i seguenti diagrammi UML per descrivere le viste architetturali:

- Class, package, component (ognuno R2) e composite-structure diagram (R3)
- Deployment diagram (R2)
- Sequence e activity diagram (R2)
- State machine diagram (R3)

Gli/le architetti/e software conoscono notazioni alternative ai diagrammi UML, ad esempio (R3):

- ArchiMate
- SysML
- C4 Vedi [Brown]
- Entity-Relationship diagram, vedi [Chen 1976]
- Per le viste di runtime ad esempio flow chart, elenchi numerati o la Business Process Modelling Notation (BPMN).

Riferimenti
[UML], [ArchiMate], [SysML], [Brown], [Chen 1976]

OA 04-04 [nuovo]: Obiettivo di apprendimento non trovato (R3)

Gli/le architetti/e software possono gestire con abilità situazioni inattese.

Nota: il titolo è stato scelto intenzionalmente e non rappresenta un errore tecnico.

Riferimenti
[IETF HTTP]

OA 04-05 [ex OA 3-4]: Spiegare e applicare le viste (views) architetturali (R1)

Gli/le architetti/e software sono in grado di utilizzare le seguenti viste architetturali:

- Vista di contesto (detta anche delimitazione del contesto):
  - include interfacce esterne dei sistemi
  - se necessario si distingue il contesto di business da quello tecnico
- Vista degli elementi costitutivi o dei componenti (composizione degli elementi costitutivi del software)
- Vista di runtime (vista dinamica, interazione tra gli elementi costitutivi del software in runtime, state machine)
- Vista di deployment (hardware e infrastruttura tecnica, mappatura degli elementi costitutivi del software sull'infrastruttura)

Se necessario, è possibile utilizzare ulteriori viste per tener conto di ulteriori esigenze o requisiti degli stakeholder, ad esempio la safety funzionale, viste relative alle informazioni, all'operatività e all'interfaccia utente (R3).

Riferimenti
[Kruchten 1995], [Rozanski+2011], [Starke+2023a], [arc42], [Brown]

OA 04-06 [ex OA 3-7]: Documentare le interfacce (R1)

Gli/le architetti/e software sono in grado di documentare interfacce interne ed esterne.

Vedi anche OA 03-07 [ex OA 2-9]: Progettare e specificare le interfacce (R1-R3).

OA 04-07 [ex OA 3-06]: Documentare e comunicare i cross-cutting concerns (R2)

Gli/le architetti/e software sono in grado di documentare e comunicare adeguatamente cross-cutting concerns tipici e relativi concetti di soluzione (concetti cross-cutting) ad esempio: persistenza, controllo del flusso, UI, deployment/integrazione, logging.

Vedi anche OA 03-10 [ex OA 2-4]: Identificare, progettare e implementare i cross-cutting concerns (R1).

OA 04-08 [ex OA 3-8]: Spiegare e documentare le decisioni architetturali (R1-R2)

Gli/le architetti/e software sono in grado di:

- Prendere decisioni architetturali in modo sistematico, motivarle, comunicarle e documentarle
- Riconoscere, comunicare e documentare interdipendenze tra decisioni architetturali

Gli/le architetti/e software conoscono gli Architecture Decision Records (ADR vedi [Nygard 2011]) e sono in grado di utilizzarli per documentare le decisioni (R2).

Riferimenti
[Nygard 2011]

OA 04-09 [ex OA 3-9]: Conoscere ulteriori risorse e strumenti per la documentazione (R3)

Gli/le architetti/e software conoscono:

- Principi fondamentali di diversi framework pubblicati per la descrizione delle architetture software, ad esempio:
  - ISO/IEC/IEEE 42010
  - arc42
  - C4 vedi [Brown]
- Idee ed esempi di checklist per la creazione, la documentazione e la verifica di architetture software
- Strumenti possibili per creare e mantenere la documentazione architetturale

Riferimenti
[ISO 42010], [arc42], [Brown]

5. Analisi e Valutazione delle Architetture Software

Durata: 60 min.	Durata esercitazione: 30 min.

Obiettivo

In questa sezione gli/le architetti/e software acquisiscono capacità e conoscenze necessarie per eseguire un'analisi architetturale efficace. Imparano a identificare rischi, valutare la conformità alle decisioni architetturali e giudicare la qualità complessiva di un sistema sulla base del suo design e della sua implementazione. Grazie alla comprensione dei diversi metodi di analisi come testing di accettazione, metriche architetturali, analisi basata su scenari e analisi costi-benefici, gli/le architetti/e possono garantire che un'architettura software soddisfi i requisiti degli stakeholder e sia coerente col design previsto.

Termini Rilevanti

Analisi architetturale; identificazione dei rischi; metodi di analisi della qualità; scenari; analisi basata su scenari; metriche; analisi supportata da strumenti

Obiettivi di Apprendimento

OA 05-01: Conoscere le ragioni dell'analisi architetturale (R1)

Gli/le architetti/e software comprendono che esistono varie ragioni possibili per svolgere un'analisi architetturale, ad esempio:

- Identificare i rischi e i possibili miglioramenti nella progettazione dell'architettura (prima, durante, o dopo l'implementazione)
- Verificare se il design architetturale soddisfa o soddisferà i requisiti
- Rilevare in che misura l'implementazione corrisponde a decisioni e design architetturali
- Verificare in che misura le esigenze degli stakeholder relative all'architettura siano state prese in considerazione

OA 05-02 [ex OA 4-3 e 4-4]: Analizzare le qualità di un sistema software (R1, R3)

Gli/le architetti/e software:

- Comprendono che, per una determinata qualità, potrebbero essere disponibili diversi metodi di analisi per un particolare sistema software, ad esempio:
  - Analisi dei risultati del testing di accettazione (R1)
  - Misurazioni quantitative del comportamento in runtime (R1)
  - Valutazioni qualitative tramite interviste, sondaggi, penetration test, ecc. (R1)
  - Analisi basata su scenari (R1)
  - Metriche architetturali relative all'accoppiamento come il grado delle dipendenze entranti ed uscenti (R1)
  - Analisi costi-benefici (R3)
  - ATAM - Architecture Trade-Off Analysis Method [Bass+2021] (R3)

- Conoscono le fonti informative per l'analisi della qualità:
    - Documentazione dei requisiti (R1)
    - Documentazione dell'architettura (R1)
    - Modelli architetturali e di progettazione (R1)
    - Codice sorgente (R1)
    - Metriche relative al codice sorgente come ad esempio linee di codice, complessità (ciclomatica) (R1)
    - Test case e risultati dei test (R1)
    - Errori e loro localizzazione nel codice sorgente, in particolare cluster di errori (R1)
    - Altra documentazione del sistema come ad esempio documentazione operativa e di test (R1)
    - Log degli eventi e metriche in runtime (R1)
    - Cronologia delle revisioni, ad es. tasso di modifica per ogni componente (R3)

Vedi anche OA 02-03 [ex OA 4-1]: Comprendere e spiegare le qualità di un sistema software (R1), OA 02-04 [ex OA 4-2]: Formulare i requisiti relativi alle qualità (R1-R3).

Riferimenti
[Bass+2021], [Starke+2023a], [Clements+2002], [ISO 25019], [Lilienthal 2019]

OA 05-03: Valutare la conformità alle decisioni architetturali (R2)

Gli/le architetti/e software sono in grado di valutare se l'implementazione del sistema è coerente con il design e le decisioni architetturali prese usando metodi come la revisione del codice e dell'architettura o applicando un'analisi supportata da strumenti.

6. Esempi di Architetture Software

Durata: 90 min.	Durata esercitazione: nessuna

Questa sezione non è rilevante per l'esame.

Obiettivi di Apprendimento

OA 06-01 [ex OA 5-1]: Conoscere la relazione tra requisiti, vincoli e soluzioni (R3)

Gli/le architetti/e software hanno individuato e compreso, usando almeno un esempio, il nesso tra requisiti e vincoli, e le soluzioni scelte.

Riferimenti
[arc42], [Starke+2023b], [Hruschka+2021], [Zörner 2021], [AOSA]

OA 06-02 [ex OA 5-2]: Comprendere l'implementazione tecnica di una soluzione (R3)

Gli/le architetti/e software sono in grado di comprendere, sulla base di almeno un esempio, la realizzazione tecnica (implementazione, concetti tecnici, prodotti utilizzati, strategie di soluzione) di una soluzione.

Riferimenti
[arc42], [Starke+2023b], [Hruschka+2021], [Zörner 2021], [AOSA]

Riferimenti

[AOSA] A. Brown and G. Wilson, Eds., *The Architecture of Open Source Applications*. [Online]. Available: https://aosabook.org/en/

[arc42] "arc42, the open-source template for software architecture communication." [Online]. Available: https://arc42.org; Maintained on: https://github.com/arc42

[ArchiMate] "The ArchiMate® Enterprise Architecture Modeling Language." [Online]. Available: https://www.opengroup.org/archimate-forum/archimate-overview

[Bass+2021] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 4th ed. Boston, MA, USA: Addison Wesley, 2021.

[Brown] Simon Brown: The C4 model for visualising software architecture. https://c4model.com; https://www.infoq.com/articles/C4-architecture-model.

[IREB Foundation] Stan Bühne, Martin Glinz, Hans van Loen, Stefan Staal: Certified Professional for Requirements Engineering - Foundation Level - Syllabus - Version 3.2.0, IREB, 2024.

[Burns 2018] Brendan Burns: Designing Distributed Systems, Patterns and Paradigms for Scalable, Reliable Services, OReilly 2018.

[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

[CloudNative] The Cloud Native Computing Foundation, online: https://www.cncf.io/

[Eilebrecht+2024] Karl Eilebrecht, Gernot Starke: Patterns kompakt: Entwurfsmuster für effektive Software-Entwicklung (in German). 6th Edition Springer Verlag 2024.

[Chen 1976] Chen, Peter (March 1976): *The Entity-Relationship Model - Toward a Unified View of Data*. ACM Transactions on Database Systems. 1 (1): 9–36.

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

[Felleisen+2014] Matthias Felleisen, Robert Bruce Findler, Matthew Flatt, Shriram Krishnamurthi: How to Design Programs. Second Edition. MIT Press, 2014. https://htdp.org/

[Ford+2023] Neil Ford, Rebecca Parsons, Patrick Kua, Pramod Sadalage: Building Evolutionary Architectures: Automated Software Governance, 2nd Edition, O'Reilly Media, 2023.

[Ford+2021] N. Ford, M. Richards, P. Sadalage, and Z. Dehghani, *Software Architecture: The Hard Parts*. Sebastopol, CA, USA: O'Reilly Media, 2021.

[Fowler 2002] Martin Fowler: Patterns of Enterprise Application Architecture. (PoEAA) Addison-Wesley, 2002.

[Ghandi+2024] Raju Gandhi, Mark Richards and Neal Ford. Head-First Software Architecture. OReilly 2024.

[Gamma+ 1994] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley. 1994.

[Geewax 2021] J. Geewax. API Design Patterns. Manning, 2021. This book lays out a set of design principles for building internal and public-facing APIs.

[Geirhos 2015] Matthias Geirhos. Entwurfsmuster: Das umfassende Handbuch (in German). Rheinwerk Computing Verlag. 2015

[Gharbi+2024] Mahbouba Gharbi, Arne Koschel, Andreas Rausch, Gernot Starke: Basiswissen Softwarearchitektur. 5. Auflage, dpunkt Verlag, Heidelberg 2024.

[Goll 2014] Joachim Goll: Architektur- und Entwurfsmuster der Softwaretechnik: Mit lauffähigen Beispielen in Java (in German). Springer-Vieweg Verlag, 2. Auflage 2014.

[Hofmeister+1999] Christine Hofmeister, Robert Nord, Dilip Soni: *Applied Software Architecture*, Addison-Wesley, 1999

[Hohpe+2004] Hohpe, G. and WOOLF, B.A.: Enter*prise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley Professional, 2004

[Hombergs 2024] Hombergs, Tom: Get Your Hands Dirty on Clean Architecture, Packt, 2nd edition 2024.

[Humble+2010] Jez Humble, David Farley. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Pearson International, 2010

[Hruschka+2021] P. Hruschka, I. Kostov, and W. Reimesch, *Arc42 by Example Volume 2: Architecture Documentation for Embedded Systems and IoT*. Victoria, BC, Canada: Leanpub, 2021. [Online]. Available: https://leanpub.com/arc42byexample-volume2

[IETF HTTP] Internet Engineering Task Force: RFC 9110, HTTP Semantics. Online: https://www.rfc-editor.org/rfc/rfc9110.html

[iSAQB Downloads] iSAQB public download site. https://public.isaqb.org. Contains curricula and mock-examination.

[iSAQB Glossary] Gernot Starke et. al. iSAQB Glossary of Software Architecture Terminology. Freely available from https://leanpub.com/isaqbglossary or its source repository https://github.com/isaqb-org/glossary/releases

[iSAQB References] Gernot Starke et. al. Annotated collection of Software Architecture References, for Foundation and Advanced Level Curricula. Freely available https://leanpub.com/isaqbreferences.

[ISO 25010] ISO/IEC 25010:2023(en) Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model. Terms and definitions online: https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-2:v1:en

[ISO 25019] ISO/IEC 25019:2023(en) Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Quality-in-use model. Terms and definitions online: https://www.iso.org/obp/ui/#iso:std:iso-iec:25019:ed-1:v1:en

[ISO 42010] ISO/IEC/IEEE 42010:2022, Software, systems and enterprise Architecture description, online: https://www.iso.org/standard/74393.html

[Keeling 2017] Michael Keeling. Design It!: From Programmer to Software Architect. Pragmatic Programmer.

[Kleppmann 2017] Martin Kleppmann: Designing Data-Intensive Applications. O'Reilly 2017.

[Kruchten 1995] Philippe Kruchten: Architectural Blueprints—The "4+1" View Model of Software

Architecture, IEEE Software 12 (6), November 1995, pp. 42-50

[Kruchten 2004] Philippe Kruchten: The Rational Unified Process: An Introduction. 3rd edition. Addison-Wesley Professional 2004.

[Lange 2021] Kenneth Lange: The Functional Core, Imperative Shell Pattern, online: https://www.kennethlange.com/functional-core-imperative-shell/

[Lehman 1980] Meir M. Lehman: Programs, Life Cycles, and Laws of Software Evolution. Proceedings of the IEEE, 68(9), 1060-1076, 1980.

[Wiki-LehmansLaws] Laws of Software Evolution. https://en.wikipedia.org/wiki/Lehman%27s_laws_of_software_evolution

[Lilienthal 2024] Carola Lilienthal: Langlebige Softwarearchitekuren. 4. Auflage, dpunkt Verlag 2024.

[Lilienthal 2019] Carola Lilienthal: Sustainable Software Architecture: Analyze and Reduce Technical Debt. dpunkt Verlag 2019.

[Liskov 1994] Barbara H. Liskov, Jeannette M. Wing: A behavioral notion of subtyping. ACM Transactions on Programming Languages and Systems, Volume 16, Issue 6, 1994. <doi:10.1145/197320.197383>

[Maguire 2019] Sandy Maguire: Algebra-Driven Design - Elegant Solutions from Simple Building Blocks. Leanpub, 2019.

[Miller+] Heather Miller, Nat Dempkowski, James Larisch, Christopher Meiklejohn: Distributed Programming (to appear, but content-complete) https://github.com/heathermiller/dist-prog-book.

[Newman 2021] Sam Newman. Building Microservices - Designing Fine-Grained Systems. O'Reilly 2nd edition 2021.

[Terhorst-North 2022] Daniel Terhorst-North: CUPID - for joyful coding. See https://dannorth.net/2022/02/10/cupid-for-joyful-coding/.

[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.

[Pohl 2025] Klaus Pohl: Requirements Engineering - Fundamentals, Principles and Techniques. Springer 2025

[Q42] arc42 Quality Model, online: https://quality.arc42.org.

[Rajlich+2000] Václav T. Rajlich, Keith H. Bennett: A Staged Model for the Software Life Cycle. IEEE Computer 33(7): 66-71, 2000.

[Read 2023] Jacqui Read: Communication Patterns - An Engineering Approach. A Guide for Developers and Architects. OReilly 2023.

[Richards+2020] Mark Richards, Neal Ford: Fundamentals of Software Architecture - An Engineering Approach. OReilly 2020.

[Rozanski+2011] Nick Rozanski, Eoin Woods: Software Systems Architecture - Working With Stakeholders Using Viewpoints and Perspectives. Addison-Wesley, 2nd edition 2011.

[SOLID] Samuel Oloruntoba and Anish Singh Walia: SOLID: The First 5 Principles of Object Oriented Design, https://www.digitalocean.com/community/conceptual-articles/s-o-l-i-d-the-first-five-principles-of-object-oriented-design.

[Sperber+2023] Michael Sperber, Herber Klaeren: Schreibe Dein Programm! Tübingen University Press, 2023. https://www.deinprogramm.de/sdp/.

[Sperber+2024] Michael Sperber, Stefan Wehr: Datenmodellierung mit Summen und Produkten, 2024. https://funktionale-programmierung.de/2024/11/25/sums-products.html. (English translation: Data Modeling with Sums and Products, 2024. <https://funktionaleprogrammierung.de/2024/11/25/sumsproducts-english.html>)

[Starke 2024] G. Starke, *Effektive Software-Architekturen - Ein praktischer Leitfaden*, 10th ed. Munich, Germany: Carl Hanser Verlag, 2024. Website: https://esabuch.de

[Starke+2023a] Gernot Starke, Alexander Lorz: Software Architecture Foundation, CPSA Foundation® Exam Preparation. Van Haren Publishing, 2nd edition, 2023.

[Starke+2023b] Gernot Starke, Michael Simons, Stefan Zörner, Ralf D. Müller, and Hendrik Lösch: arc42-by-Example - Software Architecture Documentation in Practice. Leanpub, 3rd edition 2023. https://leanpub.com/arc42byexample

[SysML] What is SysML https://sysml.org/. For diagrams, see also https://sysml.org/tutorials/sysml-diagram-tutorial/.

[Tanenbaum+] Andrew Tanenbaum, Maarten van Steen: Distributed Systems, Principles and Paradigms. https://www.distributed-systems.net/.

[UML] The UML reading room, collection of UML resources https://www.omg.org/technology/readingroom/UML.htm. See also https://www.uml-diagrams.org/.

[Yorgey 2012] Brent A. Yorgey, Monoids: Theme and Variations. Proceedings of the 2012 Haskell Symposium, September 2012 https://doi.org/10.1145/2364506.2364520

[Zimmermann+2022] Olaf Zimmermann, Mirko Stocker, Daniel Lübke, Uwe Zdun, Cesare Pautasso: Patterns for API Design: Simplifying Integration with Loosely Coupled Message Exchanges. Addison-Wesley, 2022.

[Zörner 2021] Stefan Zörner: Softwarearchitekturen dokumentieren und kommunizieren. 3. Auflage, Carl Hanser Verlag, 2021.