Piano di Studio

Certified Professional for Software Architecture (CPSA®)

Foundation Level

2021.1-IT-20210209

Indice dei contenuti

Note Legali

© (Copyright), International Software Architecture Qualification Board e. V. (iSAQB® e. V.) 2021

Il piano di studio può essere utilizzato esclusivamente in base alle seguenti condizioni:

1. Il partecipante desidera conseguire la certificazione CPSA Certified Professional for Software Architecture Foundation Level®. Ai fini del conseguimento del certificato dovrà essere consentito di usare questi documenti di testo e/o i piani di studio, generando una copia di lavoro per il proprio computer. Qualora si intendesse fare un utilizzo diverso dei documenti e/o dei piani di studio, ad esempio per la distribuzione a terze parti, pubblicità, ecc., si prega di scrivere all'indirizzo info@isaqb.org per chiedere se questo è permesso. In tal caso, si dovrebbe stipulare un proprio contratto di licenza separato.
2. Se si è trainer o Training Provider, dovrà essere possibile l'utilizzo dei documenti e/o dei piani di studio una volta ottenuta una licenza di utilizzo. Si prega di indirizzare qualsiasi richiesta all'indirizzo info@isaqb.org. Sono disponibili contratti di licenza con forniture appropriate per tutti gli aspetti inerenti.
3. Se non si rientra in alcuna delle precedenti categorie 1 e 2, ma comunque si desidera utilizzare questi documenti e/o piani di studio, si prega di contattare iSAQB e. V. scrivendo all'indirizzo info@isaqb.org. Verrete informati sulla possibilità di acquisto delle corrispondenti licenze attraverso contratti di licenza esistenti e potrete ottenere l'autorizzazione di utilizzo desiderato.

Avviso importante

Sottolineiamo che, in linea di principio, questo piano di studio è protetto da diritti d'autore. Tutti i diritti su questo Copyright sono di uso esclusivo dell'International Software Architecture Qualification Board e. V. (iSAQB® e. V.).

L'abbreviazione “e. V.” è parte del nome ufficiale di iSAQB e significa “eingetragener Verein" (“associazione registrata”), che descrive il proprio status come “entità legale” 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 4-3: Analisi qualitativa e assessment delle architetture software (R2-R3)
- OA 4-4: Valutazione quantitativa delle architetture software (R2)
- OA 5-1: Conoscere la relazione tra requisiti, vincoli e soluzioni (R3)
- OA 5-2: Conoscere il razionale dell'implementazione tecnica di una soluzione (R3)

Introduzione

Obiettivi di un corso Foundation Level

I corsi per la certificazione *Certified Professional for Software Architecture – Foundation Level* (CPSA-F) forniranno ai partecipanti le conoscenze e competenze richieste per progettare, specificare e documentare un'architettura software adeguata a soddisfare i rispettivi requisiti per sistemi di piccola e media dimensione. Sulla base dell'esperienza pratica individuale e delle competenze esistenti, i partecipanti impareranno a derivare decisioni architetturali dalla vision di un sistema esistente e da requisiti adeguatamente dettagliati. I corsi per la certificazione CPSA-F insegnano metodi e principi per la progettazione, documentazione e valutazione di architetture software, indipendenti da processi di sviluppo specifici.

Il focus è l'educazione e la formazione delle seguenti competenze:

- Discutere e concordare le decisioni architetturali principali con gli stakeholder relativi alla Gestione dei requisiti, allo Sviluppo, al Test, e alle attività di Operations.
- Comprendere le attività principali dell'architettura software, ed eseguirle per sistemi di piccola e media dimensione.
- Documentare e comunicare le architetture software sulla base di viste architetturali (architectural view), pattern (modelli) architetturali e concetti tecnici.

Inoltre, tali corsi coprono:

- Il termine e il significato di architettura software.
- I compiti e la responsabilità degli architetti software.
- I ruoli degli architetti software nei progetti di sviluppo.
- Metodi e tecniche sullo stato dell'arte per lo sviluppo di architetture software.

Out of scope

Questo piano di studio riflette i contenuti attualmente considerati dai membri iSAQB come necessari e utili per il raggiungimento degli Obiettivi di Apprendimento della certificazione CPSA-F. Non è una descrizione completa dell'intero dominio dell'"Architettura software".

I seguenti argomenti o concetti **non sono parte della certificazione CPSA-F**:

- Specifiche Tecnologie, framework o librerie di codifica.
- Programmazione o linguaggi di programmazione.
- Specifici modelli di processo.
- Fondamenti delle notazioni di modellazione (come UML, Unified Modeling Language) o fondamenti della modellazione stessa.
- Analisi di sistema e Requirements Engineering (far riferimento allo schema di formazione e certificazione IREB e. V., http://ireb.org, International Requirements Engineering Board).
- Testing del software (far riferimento allo schema di formazione e certificazione ISTQB e. V., http://istqb.org, International Software Testing Qualification Board).
- Project management o product management.
- Introduzione a specifici strumenti software.

L'obiettivo del corso di formazione è quello di trasmettere i principi basilari per l'acquisizione delle conoscenze e competenze approfondite necessarie per i singoli casi applicativi.

Prerequisiti

iSAQB e. V. può verificare i seguenti prerequisiti per gli esami di certificazione attraverso le corrispondenti domande.

I partecipanti dovrebbero possedere la seguente conoscenza e/o esperienza. In particolare, l'esperienza pratica sostanziale nello sviluppo software in un team costituisce un importante prerequisito per la comprensione del materiale didattico e per una certificazione di successo.

- Più di 18 mesi di esperienza pratica nello sviluppo software, ottenuto attraverso lo sviluppo svolto in team di diversi sistemi aldifuori della formazione formale.
- Conoscenza ed esperienza pratica in almeno un linguaggio di programmazione di livello più alto, in particolare:
  - Concetti di
    - Modularizzazione (package, namespace, ecc).
    - Trasferimento di parametri (*Call-by-Value, Call-by-Reference*).
    - Ambito, cioè dichiarazione e definizione di tipo/variabile.
  - Principi base di sistemi tipo (tipizzazione statica o dinamica, tipi di dati generici).
  - Gestione degli errori e delle eccezioni nel software.
  - Problemi potenziali di stato e variabili globali.
- Conoscenza di base di:
  - Modellazione e astrazione.
  - Algoritmi e strutture dati (come Liste, Alberi, HashTable, Dizionari/Mappe).
  - UML, Unified Modeling Language (class diagram, package diagram, component diagram e sequence diagram) e relativa relazione con il codice sorgente.

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 sistema o concetti tecnici.

Struttura, durata e metodi didattici

I tempi di studio riportati nei seguenti capitoli del piano di studio si intendono esclusivamente come suggerimenti. 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. I Training Provider possono determinare individualmente i tipi (domini e tecnologie) degli esempi e degli esercizi.

Contenuto	**Durata suggerita (min)**
1. Concetti Base dell'Architettura Software	120
2. Progettazione e Sviluppo	420
3. Specifica e Comunicazione	240
4. Architettura e Qualità	120
5. Esempi	90
Totale	990

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 prioritizzati. L'importanza per l'esame di ogni Obiettivo di Apprendimento, o dei suoi sottoelementi, è chiaramente riportata (secondo la classificazione R1, R2 o R3, come da tabella seguente). Ogni Obiettivo di Apprendimento descrive i contenuti che devono essere insegnati, 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	Sono i contenuti che ci si aspetta che i partecipanti siano in grado di mettere in pratica indipendentemente dal completamento del corso. Durante il corso, questi contenuti saranno coperti attraverso esercizi e discussioni.	I contenuti **saranno** parte dell'esame.
R2	Comprendere	Sono i contenuti che ci si aspetta che i partecipanti comprendano in linea di principio. Non saranno in generale il focus principale degli esercizi svolti durante il corso.	I contenuti **possono** essere parte dell'esame.
R3	Conoscere	Questi contenuti (termini, concetti, metodi, pratiche o similari) possono aumentare la comprensione e motivare l'argomento. Se richiesto, possono essere coperti durante il corso.	I contenuti **non sono** parte dell'esame.

Se richiesto, gli Obiettivi di Apprendimento includono riferimenti a ulteriori lettere, standard o altri fonti. I paragrafi "Termini Importanti" di ogni capitolo elencano parole che sono associate al contenuto del capitolo. Alcune di queste parole sono utilizzate nelle descrizioni degli Obiettivi di Apprendimento.

Versione attuale e archivio pubblico

La versione più recente di questo documento è disponibile per il download sulla pagina ufficiale: https://isaqborg.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 pubbliche.

Si prega di segnalare eventuali problemi utilizzando il nostro segnalatore di problemi pubblico: https://github.com/isaqborg/curriculum-foundation/issues.

1. Concetti Base dell'Architettura Software

Durata: 120 min.	Durata esercitazione: nessuna

Termini Importanti

Architettura software; domini architetturali; struttura; elementi costitutivi; componenti; interfacce; relazioni; cross-cutting concepts; architetti di software e loro responsabilità; compiti e competenze richieste; stakeholder e loro preoccupazioni; requisiti funzionali e di qualità dei sistemi; vincoli; fattori d'influenza; tipi di sistemi IT (sistemi embedded; sistemi real-time; sistemi informativi ecc.)

Obiettivi di Apprendimento

OA 1-1: Discutere le definizioni dell'architettura software (R1)

Gli architetti software conoscono diverse definizioni di architettura software (inclusi ISO 42010/IEEE 1471, SEI, Booch ecc.) e possono menzionare i loro aspetti comuni:

- Componenti/elementi costitutivi con interfacce e relazioni.
- Elementi costitutivi come termine generico, componenti come una sua caratteristica speciale
- Strutture, cross-cutting concepts, principi.
- Decisioni architetturali e loro conseguenze sul sistema completo e il relativo ciclo di vita.

OA 1-2: Comprendere e spiegare gli obiettivi e i benefici dell'architettura software (R1)

Gli architetti software possono motivare i seguenti obiettivi e vantaggi essenziali dell'architettura software:

- Supportare la progettazione, l'implementazione, la manutenzione e il funzionamento dei sistemi.
- Raggiungere requisiti di qualità come affidabilità, manutenibilità, modificabilità, sicurezza, ecc.
- Soddisfare requisiti funzionali o garantire che possano essere soddisfatti.
- Garantire che le strutture e i concetti del sistema siano compresi da tutte le parti interessate.
- Ridurre sistematicamente la complessità.
- Specificare le linee guida architettonicamente rilevanti per l'implementazione e il funzionamento.

OA 1-3: Comprendere l'architettura software come parte del ciclo di vita del software (R2)

Gli architetti software comprendono i propri compiti e possono integrare i loro risultati nell'intero ciclo di vita dei sistemi IT. Sono in grado di:

- Identificare le conseguenze delle modifiche ai requisiti funzionali, ai requisiti di qualità, alle tecnologie o all'ambiente di sistema, in relazione all'architettura software.

- Elaborare le relazioni tra i sistemi IT e i processi di business e operativi supportati

OA 1-4: Comprendere i compiti e le responsabilità degli architetti software (R1)

Gli architetti software sono responsabili del raggiungimento della qualità richiesta o necessaria, e della creazione della progettazione architetturale di una soluzione. In base all'approccio attuale o al modello di processo utilizzato, devono allineare questa responsabilità con le responsabilità globali del project management e/o di altri ruoli.

Compiti e responsabilità degli architetti software:

- Chiarire, analizzare in dettaglio ed eventualmente affinare requisiti e vincoli. Questi includono, oltre ai requisiti funzionali (Funzionalità Richieste), le caratteristiche di qualità richieste (Vincoli Richiesti).
- Decidere come decomporre il sistema in elementi costitutivi, determinando dipendenze e interfacce tra gli elementi costitutivi.
- Determinare e decidere sui cross-cutting concepts (ad esempio persistenza, comunicazione, GUI – Graphical User Interface, ecc).
- Comunicare e documentare l'architettura software sulla base di viste, pattern architetturali, concetti tecnici e cross-cutting concepts.
- Guidare la realizzazione e l'implementazione dell'architettura, se necessario integrare i feedback degli stakeholder rilevanti nell'architettura; svolgere la review e garantire la consistenza del codice sorgente e dell'architettura software.
- Analizzare e valutare l'architettura software, in particolare rispetto ai rischi correlati ai requisiti di qualità.
- Identificare, evidenziare e argomentare le conseguenze delle decisioni architetturali agli altri stakeholder.

Gli architetti software dovrebbero riconoscere autonomamente la necessità di iterazioni in tutti i compiti ed evidenziare possibilità per feedback importanti e appropriati.

OA 1-5: Correlare il ruolo degli architetti software agli altri stakeholder (R1)

Gli architetti software sono in grado di spiegare il loro ruolo. Dovrebbero adattare il proprio contributo ad uno sviluppo software in uno specifico contesto, e in relazione ad altri stakeholder e unità organizzative, in particolare:

- Product management e product owner.
- Project manager.
- Requirement engineer (requirement o business analyst, requirement manager, system analyst, business owner, subject-matter expert, etc.).
- Developer.
- Quality assurance e tester.
- Operatore e amministratore IT (si applica principalmente all'ambiente di produzione o ai data center per i sistemi informativi).
- Hardware developer.
- Enterprise architect e architecture board member.

OA 1-6: Saper spiegare la correlazione tra approcci di sviluppo e architettura software (R1)

- Gli architetti software sono in grado di spiegare come la procedura iterativa influisce sulle decisioni architetturali (in base ai rischi e alla probabilità).
- A causa dell'incertezza intrinseca, gli architetti software devono spesso lavorare e prendere decisioni iterativamente. Per fare questo, devono sistematicamente ottenere un feedback dagli altri stakeholder.

OA 1-7: Differenziare gli obiettivi a breve e lungo termine (R1)

Gli architetti software possono:

- Spiegare i requisiti di qualità di lungo termine e le loro differenze dagli obiettivi di progetto di breve termine.
- Spiegare potenziali conflitti tra obiettivi a breve termine e obiettivi a lungo termine, per trovare una soluzione adatta per tutti gli stakeholder.
- Identificare e specificare i requisiti di qualità.

OA 1-8: Distinguere tra affermazioni esplicite e assunzioni implicite (R1)

Gli architetti software:

- Dovrebbero esplicitamente presentare assunzioni o prerequisiti, pertanto evitando assunzioni implicite.
- Sanno che le assunzioni implicite possono causare potenziali incomprensioni tra gli stakeholder.
- Possono formulare implicitamente, se è appropriato in un dato contesto.

OA 1-9: Responsabilità degli architetti software all'interno del più ampio contesto architetturale (R3)

Il Focus di iSAQB CPSA Foundation Level è sulle strutture e sui concetti di singoli sistemi software.

Inoltre, gli architetti software sono familiari con altri domini architetturali, come ad esempio:

- Architettura IT Enterprise: struttura di ambienti applicativi.
- Architettura di Business e di Processo (Business and Process Architecture): struttura, tra le altre cose, dei processi di business.
- Architettura informativa: struttura cross-system e utilizzo di informazioni e dati.
- Architettura dell'infrastruttura o della tecnologia: struttura dell'infrastruttura tecnica, struttura hardware, reti, ecc.
- Architettura hardware o di processore (per sistemi relativi ad hardware).

Questi domini architetturali non sono il focus della certificazione CPSA-F.

OA 1-10: Distinguere i tipi di sistemi IT (R3)

Gli architetti software conoscono diversi tipi di sistemi IT, ad esempio:

- Sistemi informativi.
- Sistemi di supporto a decisioni, Data-Warehouse o sistemi di Business Intelligence.
- Sistemi cellulari.

- Processi o sistemi batch.
- Sistemi hardware-related; in questo caso, gli architetti software comprendono la necessità di progettazione del codice hardware/software (dipendenze temporali e relative al contenuto della progettazione hardware e software).

OA 1-11: Sfide dei sistemi distribuiti (R3)

Gli architetti software sono in grado di:

- Identificare la distribuzione all'interno di una architettura software data.
- Analizzare i criteri di coerenza all'interno di un problema settoriale dato.
- Spiegare la causalità degli eventi in un sistema distribuito.

Gli architetti software sanno:

- Che la comunicazione potrebbe non riuscire in un sistema distribuito.
- Che ci sonolimitazioni per quanto riguarda la coerenza nei database del mondo reale.
- Cos'è il problema dello "split-brain" e perché è difficile da risolvere.
- Che è impossibile determinare l'ordine temporale esatto degli eventi in un sistema distribuito.

Riferimenti Bibliografici

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

2. Progettazione e Sviluppo delle Architetture Software

Durata: 330 min.	Durata esercitazione: 90 min.

Termini Importanti

Progettazione; procedura di progettazione; decisione di progettazione; viste (view); interfacce; technical e cross-cutting concepts; stereotipi; pattern (modelli) architetturali; linguaggi di modellazione; principi di progettazione; dipendenze; accoppiamento; coesione; architetture funzionali e tecniche; approcci Top-down e Bottom-Up; progettazione model-based (basata sul modello); progettazione iterativa/incrementale; progettazione Domain-Driven (guidata dal dominio)

Obiettivi di Apprendimento

OA 2-1: Selezionare e utilizzare approcci ed euristiche per lo sviluppo dell'architettura (R1, R3)

Gli architetti software sono in grado di menzionare, spiegare e usare gli approcci di base dello sviluppo dell'architettura, ad esempio:

- Approcci Top-down e Bottom-Up di progettazione.
- Sviluppo dell'architettura view-based (basata su viste).
- Progettazione iterativa e incrementale.
  - Necessità di iterazioni, in particolare in caso di incertezza nel prendere decisioni.
  - Necessità di riscontri su decisioni di progettazione.
- Domain-Driven Design (progettazione guidata dal dominio), consulta [Evans 2004] (R3).
- Evolutionäre Architektur (architettura evolutiva), consulta [Ford 2017] (R3).
- Globale Analyse (analisi globale), consulta [Hofmeister et. al 1999] (R3).
- Architettura guidata dai modelli (R3).

OA 2-2: Progettare architetture software (R1)

Gli architetti software sono in grado di:

- Progettare, comunicare e documentare in modo appropriato architetture software sulla base di noti requisiti funzionali e di qualità per sistemi software che non sono safety-critical o business-critical.
- Prendere decisioni importanti per la struttura sulla decomposizione di sistema e sulla struttura a componenti, progettando volontariamente dipendenze tra gli elementi costitutivi.
- Riconoscere e giustificare le interdipendenze e i trade-off di decisioni di progettazione.
- Spiegare e applicare in modo appropriato i termini *Black-Box* e *White-Box*.
- Applicare un raffinamento secondo un approccio graduale, e specificare gli elementi costitutivi.
- Progettare architectural view (viste architetturali), in particolare building-build view (vista degli elementi costitutivi), runtime view (vista runtime) e deployment view (vista deployment).
- Spiegare le conseguenze di queste decisioni sul corrispondente codice sorgente.
- Separare gli elementi architetturali relativi al dominio da quelli tecnici e giustificare queste decisioni.

- Identificare i rischi relativi alle decisioni architetturali.

OA 2-3: Identificare e considerare i fattori che influenzano l'architettura software (R1-R3)

Gli architetti del software sono in grado di tenere in considerazione e rielaborare quei fattori di influenza (condizioni limite) che vincolerebbero la libertà di progettazione. Sono a conoscenza che le loro decisioni impongono ulteriori requisiti e vincoli sia per il sistema da progettare, sia per la sua architettura o per il processo di sviluppo.

Riconoscono e considerano l'influenza di:

- Fattori legati al prodotto come (R1)
  - Requisiti funzionali
  - Requisiti e obiettivi di qualità
  - Fattori aggiuntivi come i costi del prodotto, il modello di licenza previsto o il modello di business del sistema
- Fattori tecnici come (R1-R3)
  - Decisioni tecniche e concezioni del committente esterno (R1).
  - Infrastruttura hardware e software preesistente o pianificata (R1).
  - Limitazioni tecnologiche per strutture di dati e interfacce (R2).
  - Architetture di referenza, librerie, componenti e framework (R1).
  - Linguaggi di programmazione (R3).
- Fattori organizzativi come
  - Struttura organizzativa del team di sviluppo e del cliente (R1).
  - Cultura aziendale e di squadra (R3).
  - Partnership e collaborazioni (R2).
  - Norme, linee guida e modelli di processo (per esempio processi di approvazione e rilascio) (R2).
  - Disponibilità di risorse come budget, tempo e personale (R1).
  - Disponibilità, qualificazione e coinvolgimento dei dipendenti (R1).
- Fattori regolatori come (R2)
  - Vincoli legali locali e internazionali.
  - Questioni contrattuali e di responsabilità.
  - Leggi sulla protezione dei dati e sulla privacy.
  - Problemi di conformità o obblighi di onere della prova.
- Tendenze come (R3)
  - Tendenze di mercato.
  - Tendenze tecnologiche (p.e. blockchain, microservizi).
  - Tendenze metodologiche (p. e. agile).
  - Impatto (potenziale) dovuto sia a ulteriori interessi delle parti in causa, sia decisioni di progettazione già date o determinate dall'esterno (R3).

OA 2-4: Progettare e implementare cross-cutting concepts(R1)

Gli architetti software sono in grado di:

- Spiegare il significato dei cross-cutting concepts.
- Decidere su e progettare cross-cutting concepts, ad esempio persistenza, comunicazione, GUI, gestione degli errori, concorrenza.
- Identificare e valutare potenziali interdipendenze tra queste decisioni.

Gli architetti software sanno che tali cross-cutting concepts possono essere riutilizzati in tutto il sistema.

OA 2-5: Descrivere, spiegare e applicare in modo appropriato importanti pattern architetturali (R1, R3)

Gli architetti del software conoscono:

- Diversi pattern architetturali (vedi sotto) e possono applicarli quando richiesto.
- I pattern sono un modo per raggiungere determinate qualità per problemi e requisiti dati all'interno dei contesti presi in considerazione.
- Ci sono diverse categorie di pattern (R3).
- Esistono fonti aggiuntive di pattern che si riferiscono al loro specifico dominio tecnico o applicativo (R3).

Gli architetti del software possono spiegare e fornire esempi per i seguenti pattern (R1):

- Gli Strati (Layers):
  - Gli strati di astrazione nascondono i dettagli, per esempio: Strato di rete ISO/OSI o "Strato di astrazione dell'hardware". Consultare: https://en.wikipedia.org/wiki/Hardware_abstraction.
  - Un'altra interpretazione sono gli strati per la separazione (fisica) di funzionalità o responsabilità, consultare: https://en.wikipedia.org/wiki/Multitier_architecture.
- Pipes-and-filters: sono rappresentativi per i pattern di flusso di dati, i quali separano l'elaborazione passo dopo passo in un insieme di attività di elaborazione ("filtri") e trasporto/buffer associati ("pipes").
- I microservizi dividono le applicazioni in servizi eseguibili separati che comunicano in rete (in remoto).
- L'iniezione di dipendenza viene vista come possibile soluzione al principio di inversione delle dipendenze.

Gli architetti del software possono spiegare alcuni dei seguenti modelli, spiegare la loro rilevanza per sistemi concreti e fornirne esempi (R3):

- Lavagna: gestire problemi che non possono essere risolti da algoritmi deterministici ma che richiedono conoscenze diverse.

- Broker (intermediario): responsabile del coordinamento della comunicazione tra il fornitore (o i fornitori) e il consumatore (o i consumatori), applicato nei sistemi distribuiti. Responsabile dell'invio delle richieste e/o della comunicazione dei risultati, degli errori e delle eccezioni.
- Combinator (sinonimo: Closure of Operations), per oggetti di dominio di tipo T, ricerca le operazioni con input e output di tipo T. Consulta [Yorgey 2012].
- CQRS (Command-Query-Responsibility-Segregation): separazione delle operazioni di lettura e scrittura nei sistemi informatici. Richiede una conoscenza approfondita della tecnologia concreta di database/persistenza per capire le diverse caratteristiche e requisiti delle operazioni di "lettura" e "scrittura".
- Event sourcing: gestione delle operazioni sui dati attraverso una sequenza di eventi, ognuno dei quali viene registrato in un archivio di appendici.
- Interpreti: rappresentano l'oggetto di dominio o DSL come sintassi, offrono una funzione che implementa un'interpretazione semantica dell'oggetto di dominio separata dall'oggetto di dominio stesso.
- Modelli di integrazione o di messaggistica (per esempio da [Hohpe+2004]).
- La famiglia di modelli MVC, MVVM, MV Update e PAC che separano la rappresentazione esterna (vista) dei dati dai servizi operativi e dal loro coordinamento.
- Modelli di interfaccia come adapter, facade, proxy. Tali modelli aiutano a integrare i sottosistemi e/o a semplificare le dipendenze. Gli architetti dovrebbero sapere che questi modelli possono essere usati indipendentemente dalla tecnologia (degli oggetti).
    - Adattatore: disaccoppiamento di consumatore e fornitore - quando l'interfaccia del fornitore non corrisponde esattamente a quella del consumatore.
    - Facade: semplifica l'uso di un fornitore (provider) per il consumatore (o i consumatori) grazie a accesso semplificato.
    - Proxy: un intermediario/agente tra il consumatore e il fornitore che permette, per esempio, il disaccoppiamento temporale, il caching dei risultati o il controllo dell'accesso al fornitore.
- Osservatore: un produttore di valori notifica un intermediario centrale presso il quale le parti interessate (consumatori) possono registrarsi per essere a loro volta avvisati dei cambiamenti.
- Plug-in: estende il comportamento di un componente.
- Ports&Adapters (syn. Onion Architecture, Hexagonal Architecture): concentrano la logica di dominio al centro del sistema e possiedono connessioni con il mondo esterno (database, UI) solo ai bordi. Le dipendenze sono dall'esterno all'interno (outside-in), mai dall'interno all'esterno (inside-out).
- Remote Procedure Call: fa eseguire una funzione o un algoritmo in un altro spazio di indirizzo.

- SOA: architettura orientata al servizio. Un approccio per fornire agli utenti del sistema servizi astratti piuttosto che implementazioni concrete, con lo scopo di promuovere il riutilizzo dei servizi tra i dipartimenti e tra le aziende.
- Modelli e strategia: rendere flessibili algoritmi specifici attraverso l'incapsulamento.
- Visitatore: separa l'attraversamento delle strutture di dati dall'elaborazione specifica.

Gli architetti del software devono conoscere le fonti essenziali dei modelli architettonici, come la letteratura POSA (per esempio [Buschmann+ 1996]) e PoEAA ([Fowler 2002]) (per i sistemi informativi) (R3).

OA 2-6: Spiegare e utilizzare principi di progettazione (R1-R3)

Gli architetti del software sono in grado di spiegare cosa sono i principi di progettazione. Per quanto riguarda l'architettura del software, possono delinearne gli obiettivi di base e l'applicazione (R2).

Con rilevanza di controllo, dipendente dal relativo principio specifico elencato qui sotto, gli architetti del software sono in grado di:

- Spiegare i principi di progettazione elencati di seguito, mostrandoli con esempi.
- Spiegare come questi principi dovrebbero essere applicati.
- Dimostrare come i requisiti di qualità influenzino l'applicazione di questi principi.
- Spiegare le implicazioni dei principi di progettazione per l'implementazione.
- Analizzare il codice sorgente e i progetti architettonici per valutare se questi principi di progettazione sono stati o dovrebbero essere applicati.

Astrazione (R1)

- Inteso una procedura per sviluppare generalizzazioni appropriate.
- Come costrutto di progettazione in cui i blocchi di costruzione dipendono da astrazioni piuttosto che da implementazioni.
- Interfacce come astrazioni.

Modularizzazione (R1)

- Principio di segretezza (Information Hiding) e incapsulamento (R1).
- Separazione delle aree di competenza (Separation of Concerns - SoC) (R1).
- Accoppiamento lasco ma funzionalmente sufficiente (R1) dei blocchi di costruzione, vedi OA 2-7.
- Alta coesione (R1).
- Principi SOLID (R1-R3), nella misura in cui sono rilevanti a livello architettonico:
  - S: Principio di responsabilità unica (R1) e la sua relazione con SoC.
  - O: Principio aperto/chiuso (R1).
  - L: Il principio di sostituzione di Liskov (R3) inteso come modalità per ottenere coerenza e integrità concettuale nella progettazione orientata agli oggetti.
  - I: Principio di segregazione dell'interfaccia (R2) e la sua relazione con l'obiettivo di

apprendimento 2-9 "Progettare e specificare le interfacce".

- D: Principio di inversione delle dipendenze (R1) - Inversione delle dipendenze (R1) attraverso interfacce o astrazioni simili.

Integrità concettuale (R2)

- Significa raggiungere l'uniformità (omogeneità, coerenza) delle soluzioni per problemi simili (R2).
- Inteso come mezzo per raggiungere il principio della minima sorpresa (R3).

Semplicità (R1)

- Come mezzo per ridurre la complessità (R1).
- Come motivo dei principi KISS (R1) e YAGNI (R2).

Aspettarsi errori (R1-R2)

- Come mezzo per progettare sistemi robusti e resilienti (R1).
- Come generalizzazione del principio di robustezza (legge di Postel) (R2).

OA 2-7: Gestire le dipendenze tra gli elementi costitutivi (R1)

Gli architetti software comprendono le dipendenze e l'accoppiamento tra elementi costitutivi e possono applicarli in modo mirato. Essi:

- Conoscono e comprendono diversi tipi di accoppiamento tra elementi costitutivi (ad esempio accoppiamento strutturale attraverso l'uso/delega, messaggi/eventi, composizione, generazione, ereditarietà, accoppiamento temporale, accoppiamento, accoppiamento tramite dati, tipi di dati o hardware).
- Capiscono come le dipendenze vadano ad ampliare l'accoppiamento.
- Sono in grado di usare in modo mirato tali tipi di accoppiamento e valutare le conseguenze di tali dipendenze.
- Conoscono e sono in grado di applicare le possibilità di eliminazione o riduzione dell'accoppiamento, ad esempio:
  - Pattern (si veda OA 2-5).
  - Principi di base della progettazione (si veda OA 2-6).
  - Esternalizzazione delle dipendenze, ovvero definire dipendenze concrete in fase di installazione o runtime, ad esempio utilizzando la tecnica di "Dependency Injection".

OA 2-8: Soddisfare i requisiti di qualità con approcci e tecniche appropriate (R1)

Gli architetti software comprendono e considerano la forte influenza dei requisiti di qualità nelle decisioni architetturali e di progettazione, ad esempio per:

- Efficienza / Prestazioni.
- Disponibilità.
- Manutenibilità, modificabilità, estendibilità, adattabilità.

Sono in grado di:

- Spiegare e applicare opzioni di soluzione, Architectural Tactics, pratiche idonee nonché possibilità tecniche per soddisfare i requisiti di qualità importanti dei sistemi software (differenti per sistemi integrati o per sistemi informativi).

- Identificare e comunicare possibili trade-off tra tali soluzioni e relativi rischi associati.

OA 2-9: Progettare e definire interfacce (R1-R3)

Gli architetti software conoscono l'importanza delle interfacce. Sono in grado di progettare o specificare interfacce tra elementi costitutivi architetturali nonché interfacce esterne tra il sistema e gli elementi al di fuori del sistema.

Essi conoscono:

- Caratteristiche desiderate delle interfacce e sono in grado di utilizzarle nella progettazione:
  - Semplicità di apprendimento, semplicità di utilizzo, semplicità di estensione.
  - Difficoltà di utilizzarlo in modo improprio.
  - Completezza funzionale dalla prospettiva degli utenti o degli elementi costitutivi utilizzati.
- Necessità di trattare diversamente le interfacce interne ed esterne.
- Approcci differenti per l'implementazione delle interfacce (R3):
  - Approccio orientato alle risorse (REST, REpresentational State Transfer).
  - Approccio orientato al servizio (si veda servizi web basati su WS-*/SOAP).

Riferimenti Bibliografici

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

3. Specifica e Comunicazione delle Architetture Software

Durata: 180 min.	Durata esercitazione: 60 min.

Termini Importanti

Architectural view (Viste architetturali); strutture; concetti (tecnici); documentazione; comunicazione; descrizione; stakeholder-oriented; meta-strutture e template per la descrizione e comunicazione; contesto di sistema; elementi costitutivi; building-block view (vista dell'elemento costitutivo); runtime view (vista runtime); deployment view (vista deployment); nodo; canale; artefatto di deployment; mapping degli elementi costitutivi in artefatti di deployment; descrizione delle interfacce e decisioni di progettazione; UML; strumenti di documentazione

Obiettivi di Apprendimento

OA 3-1: Spiegare e prendere in considerazione la qualità della documentazione tecnica (R1)

Gli architetti software conoscono i requisiti di qualità della documentazione tecnica e sono in grado di considerarli e soddisfarli quando documentano i sistemi:

- Comprensibilità, correttezza, efficienza, appropriatezza, manutenibilità.
- Forma, contenuto e livello di dettaglio adattato agli stakeholder.

Essi sanno che l'audience target è in grado di valutare la comprensibilità della documentazione tecnica.

OA 3-2: Descrivere e comunicare le architetture software (R1, R3)

Gli architetti software sono in grado di:

- Documentare e comunicare le architetture per i corrispondenti stakeholder, rivolgendosi quindi a diversi gruppi target, ad es. management, team di sviluppo, Quality Assurance, altri architetti software e potenzialmente stakeholder aggiuntivi.
- Consolidare e armonizzare lo stile e il contenuto di contributi da diversi gruppi di autori.
- Conoscere i benefici della documentazione template-based (basata su template).

Gli architetti del software sanno che varie proprietà della documentazione dipendono dalle specificità del sistema, dai suoi requisiti, dai rischi, dall'approccio di sviluppo, dall'organizzazione o da altri fattori.

Questi fattori influenzano, per esempio:

- Se si debba dare priorità alla comunicazione scritta o orale.
- La portata e il livello di dettaglio della documentazione richiesta in ogni fase dello sviluppo.
- Il formato della documentazione.
- L'accessibilità della documentazione.
- Formalità della documentazione (per esempio diagrammi corrispondenti a un meta-modello o semplici disegni).
- Revisioni formali e processi di approvazione della documentazione.

Gli architetti del software sono consapevoli di questi fattori e possono adattare le caratteristiche della documentazione in base alla situazione.

OA 3-3: Spiegare e applicare notazioni/modelli per la descrizione dell'architettura software (R2-R3)

Gli architetti software conoscono almeno i seguenti diagrammi UML per descrivere le architectural view:

- Class diagram, package diagram, component diagram (R2) e composition-structure diagram (R3).
- Deployment diagram.
- Sequence diagram e activity diagram.
- State diagram.

Gli architetti software conoscono le alternative ai diagrammi UML, per esempio (R3).

- Archimate, consulta [Archimate].
- Per le visualizzazioni di runtime, per esempio, diagrammi di flusso, liste numerate o notazione di modellazione dei processi aziendali (BPMN).

OA 3-4: Spiegare e usare le architectural view (R1)

Gli architetti software sono in grado di applicare le seguenti architectural view:

- Context view (vista del contesto).
- Component view o building-block view (composizione degli elementi costitutivi software).
- Runtime view (vista dinamica, interazione tra elementi costitutivi software durante il runtime, macchine a stati).
- Deployment view (infrastruttura hardware e tecnica nonché mapping degli elementi costitutivi software sull'infrastruttura).

OA 3-5: Spiegare e applicare la context view dei sistemi (R1)

Gli architetti del software sono in grado di:

- Rappresentare il contesto di sistemi, ad es. sotto forma di context diagram con spiegazioni.
- Rappresentare interfacce esterne di sistemi nel context view.
- Differenziare il contesto commerciale e tecnico.

OA 3-6: Documentare e comunicare cross-cutting concepts (R1)

Gli architetti software sono in grado di documentare e comunicare adeguatamente i tipici cross-cutting concepts (sinonimi: principi e aspetti), ad es. persistenza, gestione dei workflow, UI (User Interface), deployment/integration, logging.

OA 3-7: Descrivere le interfacce (R1)

Gli architetti software sono in grado di descrivere e specificare entrambe le interfacce interne ed esterne.

OA 3-8: Spiegare e documentare le decisioni architetturali (R1-R2)

Gli architetti software sono in grado di:

- Prendere sistematicamente decisioni architetturali, ustificarle, comunicarle e documentarle.
- Identificare, comunicare e documentare interdipendenze tra decisioni di progettazione.

Gli architetti del software conoscono gli Architecture Decision Records (ADR, consulta [Nygard 2011]) e possono usarli per la documentazione delle decisioni (R2).

OA 3-9: Usare la documentazione come comunicazione scritta (R2)

Gli architetti software usano la documentazione per supportare la progettazione, l'implementazione e l'ulteriore sviluppo (detto anche *Manutenzione* o *Evoluzione*) dei sistemi.

OA 3-10: Conoscere risorse e strumenti aggiuntivi per la documentazione (R3)

Gli architetti software conoscono:

- I principi base di differenti framework pubblicati per la descrizione delle architetture software, ad esempio:
  - ISO/IEEE-42010 (in precedenza1471), [ISO 42010]
  - arc42, [arc42]
  - C4, [Brown]
  - FMC, [FMC]
- Idee ed esempi di checklist per la realizzazione, documentazione e testing delle architetture di software.
- Possibili strumenti per la realizzazione e la gestione della documentazione architetturale.

Riferimenti Bibliografici

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

4. Architettura Software e Qualità

Durata: 60 min.	Durata esercitazione: 60 min.

Termini Importanti

Qualità; caratteristiche di qualità (chiamate anche attributi di qualità); DIN/ISO 25010; scenari di qualità; albero di qualità; trade-off tra caratteristiche di qualità; valutazione qualitativa dell'architettura; metriche e valutazione quantitativa

Obiettivi di Apprendimento

OA 4-1: Discutere modelli di qualità e caratteristiche di qualità (R1)

Gli architetti software possono spiegare:

- Il concetto di qualità (in base a DIN/ISO 25010, in precedenza 9126) e di caratteristiche di qualità.
- Modelli di qualità generici (come DIN/ISO 25010).
- Le correlazioni e i trade-off delle caratteristiche di qualità, ad esempio:
  - Configurabilità vs. affidabilità.
  - Requisiti di memoria vs. efficienza delle prestazioni.
  - Sicurezza vs. usabilità.
  - Flessibilità in runtime vs. manutenibilità.

OA 4-2: Chiarire i requisiti di qualità per le architetture software (R1)

Gli architetti software possono:

- Chiarire e definireconcretamente specifici requisiti di qualità per il software da sviluppare e le loro architetture, per esempio sotto forma di scenari e diagrammi ad albero di qualità.
- Spiegare e fare uso di scenari e diagrammi ad albero di qualità.

OA 4-3: Analisi qualitativa e valutazione delle architetture software (R2-R3)

Gli architetti software:

- Conoscono approcci metodici per l'analisi e la valutazione qualitativa delle architetture software (R2), ad esempio come specificato da ATAM (R3).
- Possono analizzare e valutare qualitativamente sistemi più piccoli (R2).
- Sanno che le seguenti fonti informative possono supportare l'analisi e la valutazione qualitativa delle architetture (R2):
  - Requisiti di qualità, ad esempio sotto forma di alberi e scenari di qualità.
  - Documentazione dell'architettura.
  - Modelli architetturali e di progettazione.

- Codice sorgente.
- Metriche.
- Altra documentazione di sistema, come documentazione sui requisiti, operativa o di test.

OA 4-4: Valutazione quantitativa delle architetture software (R2)

Gli architetti software conoscono gli approcci per un'analisi e una valutazione quantitativa (misurazione) del software.

Essi sanno che:

- La valutazione quantitativa può aiutare a identificare parti critiche nei sistemi.
- Ulteriori informazioni possono essere utili per la valutazione delle architetture, ad esempio:
  - Documentazione dei requisiti e architetturale.
  - Codice sorgente e relative metriche come linee di codice, complessità (ciclomatica), dipendenze in input e in output.
  - Errori conosciuti nel codice sorgente, in particolare cluster di errori.
  - Test casi e risultati di test.

Riferimenti Bibliografici

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

5. Esempi di Architetture Software

Durata: 90 min.	Durata esercitazione: nessuna

Questa sezione non è rilevante ai fini dell'esame.

Obiettivi di Apprendimento

OA 5-1: Conoscere la relazione tra requisiti, vincoli e soluzioni (R3)

Gli architetti del software dovrebbero riconoscere e comprendere la correlazione tra requisiti e vincoli e le soluzioni scelte utilizzando almeno un esempio.

OA 5-2: Conoscere il razionale dell'implementazione tecnica di una soluzione (R3)

Gli architetti software comprendono la realizzazione tecnica (implementazione, concetti tecnici, prodotti utilizzati, decisioni architetturali, strategie risolutive) di almeno una soluzione.

Riferimenti Bibliografici

- [arc42] arc42, the open-source template for software architecture communication, online: https://arc42.org. Maintained on https://github.com/arc42.
- [Archimate] The ArchiMate® Enterprise Architecture Modeling Language, online: https://www.opengroup.org/archimate-forum/archimate-overview.
- [Bass+2012] Len Bass, Paul Clements, Rick Kazman: Software Architecture in Practice. 3rd edition, Addison Wesley, 2012.
- [Brown] Simon Brown: Brown, Simon: The C4 model for visualising software architecture. https://c4model.com https://www.infoq.com/articles/C4-architecture-model.
- [Buschmann+1996] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal: Pattern-Oriented Software Architecture (POSA): A System of Patterns. Wiley, 1996.
- [Buschmann+2007] Frank Buschmann, Kevlin Henney, Douglas C. Schmidt: Pattern-Oriented Software Architecture (POSA): A Pattern Language for Distributed Computing, Wiley, 2007.
- [Clements+2002] Paul Clements, Rick Kazman, Mark Klein: Evaluating Software Architectures. Methods and Case Studies. Addison Wesley, 2002.
- [Clements+2010] Paul Clements, Felix Bachmann, Len Bass, David Garlan, David, James Ivers, Reed Little, Paulo Merson and Robert Nord. Documenting Software Architectures: Views and Beyond, second edition, Addison Wesley, 2010.
- [Eilebrecht+2019] Karl Eilebrecht, Gernot Starke: Patterns kompakt: Entwurfsmuster für effektive Software-Entwicklung (in German). 5th Edition Springer Verlag, 2019.
- [Evans 2004] Eric Evans: Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley, 2004.
- [FMC] Siegfried Wendt: Fundamental Modeling Concepts, online: http://www.fmc-modeling.org/.
- [Ford 2017] Neil Ford, Rebecca Parsons, Patrick Kua: Building Evolutionary Architectures: Support Constant Change. O'Reilly, 2017.
- [Fowler 2002] Martin Fowler: Patterns of Enterprise Application Architecture. (PoEAA) Addison- Wesley, 2002.
- [Gamma+94] Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1994.
- [Geirhosk 2015] Matthias Geirhos. Entwurfsmuster: Das umfassende Handbuch (in German). Rheinwerk Computing Verlag, 2015 ISBN: 9783836227629
- [Gharbi+2020] Mahbouba Gharbi, Arne Koschel, Andreas Rausch, Gernot Starke: Basiswissen Softwarearchitektur. 4th edition, published by dpunkt, Heidelberg 2020.
- [Goll 2014] Joachim Goll: Architektur- und Entwurfsmuster der Softwaretechnik: Mit lauffähigen Beispielen in Java. Published by Springer-Vieweg, second edition 2014.
- [Hofmeister et. al 1999] Christine Hofmeister, Robert Nord, Dilip Soni: Applied Software Architecture, Addison-Wesley, 1999.
- [ISO 42010] ISO/IEC/IEEE 42010:2011, Systems and software engineering — Architecture description, online: http://www.iso-architecture.org/ieee-1471/.
- [iSAQB Working Groups] Gernot Starke et. al. Annotated collection of Software Architecture References, for Foundation and Advanced Level Curricula. Available for free on https://leanpub.com/isaqbreferences.

- [Keeling 2017] Michael Keeling. Design It!: From Programmer to Software Architect. Pragmatic Programmer. ISBN 978-1680502091.
- [Lilienthal 2018] Carola Lilienthal: Langlebige Softwarearchitekuren. Second edition, published by dpunkt Verlag, 2018.
- [Lilienthal 2019] Carola Lilienthal: Sustainable Software Architecture: Analyze and Reduce Technical Debt. dpunkt Verlag, 2019.
- [Martin 2003] Robert Martin: Agile Software Development. Principles, Patterns, and Practices. Prentice Hall, 2003.
- [Martin 2017] Robert Martin. Clean Architecture: A craftsman's guide to software structure and design. MITP.
- [Miller et. al] Heather Miller, Nat Dempkowski, James Larisch, Christopher Meiklejohn: Distributed Programming (to appear, but content-complete) https://github.com/heathermiller/dist-prog-book.
- [Newman 2015] Sam Newman. Building Microservices: Designing Fine-Grained Systems. O'Reilly, 2015. ISBN 9781491950357.
- [Nygard 2011] Michael Nygard: Documenting Architecture Decision. https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions. See also https://adr.github.io/.
- [Pethuru 2017] Raj Pethuru et. al: Architectural Patterns. Packt, 2017.
- [Starke 2020] Gernot Starke: Effektive Softwarearchitekturen - Ein praktischer Leitfaden (in tedesco). 9th edition, published by Carl Hanser, 2020. Sito web: https://esabuch.de
- [UML] The UML reading room, collection of UML resources https://www.omg.org/technology/readingroom/UML.htm. See also https://www.uml-diagrams.org/.
- [van Steen+Tanenbaum] Andrew Tanenbaum, Maarten van Steen: Distributed Systems, Principles and Paradigms. https://www.distributed-systems.net/.
- [Yorgey 2012] Brent A. Yorgey, Proceedings of the 2012 Haskell Symposium, September 2012 https://doi.org/10.1145/2364506.2364520.
- [Zörner 2015] Stefan Zörner: Softwarearchitekturen dokumentieren und kommunizieren. second edition, published by Carl Hanser, 2015.