Lezione 2 03/10/2024

Architettura del software

L'**architettura del software** ha a che fare con le strutture e la qualità dei sistemi software. Ha a che fare con elementi architetturali, a come questi elementi **interagiscono** tra loro, e ai loro **vincoli** di modo che possano **soddisfare** i requisiti sistema.

Si parla quindi progettazione, di design. Si va oltre ad algoritmi e strutture dati, mi preoccupo dell'**organizzazione complessiva**.

Ci sono tante definizioni di architettura del software. Quella più comunemente accettata è la seguente:

L'**architettura del software** di un sistema è l'**insieme delle strutture** necessarie per poter ragionare sul sistema che verrà creato. Questa struttura comprende **elementi software** (ma non solo), le loro **relazioni** e le **proprietà** di entrambi.

Elementi

Una struttura include **elementi**. Un elemento deve essere un **elemento architetturale**, perchè ci dà **informazioni** rilevanti del sistema nella sua complessità. Permette quindi di ragionare a livello di **requisiti** funzionali e requisiti di qualità.

Per esempio la struttura che scelgo per memorizzare i dati è un elemento architetturale, perchè il tipo di databse che scelgo ha degli impatti importanti sulle performance.

Le strutture possono contenere elementi che non sono software (moduli, processi, componenti software, servizi, database), come l'hardware di deployment o i team che lavoreranno su ciascun modulo software.

Bisogna quindi definire bene le **responsabilità** di ciascun elemento (cosa fa). Per fare le sue azioni, ciascun componente richiede un'**interfaccia** ben definita:

- I servizi che l'elemento fornisce o richiede da altri elementi.
- I livelli di qualità di ciascun servizio fornito.

Quindi non andiamo a dire esattamente come sarà sviluppato, ma andiamo a definire delle tattiche su come verrà garantita la qualità, non andiamo a definire per esempio le classi nel linguaggio ad oggetti.

Strutture

L'architettura è un insieme di strutture, ciascuna struttura include elementi.

Abbiamo più strutture perchè devo vedere l'architettura rispetto a più punti di vista, che dipendono da chi sono gli stakeholder. Di solito bisogna descrivere:

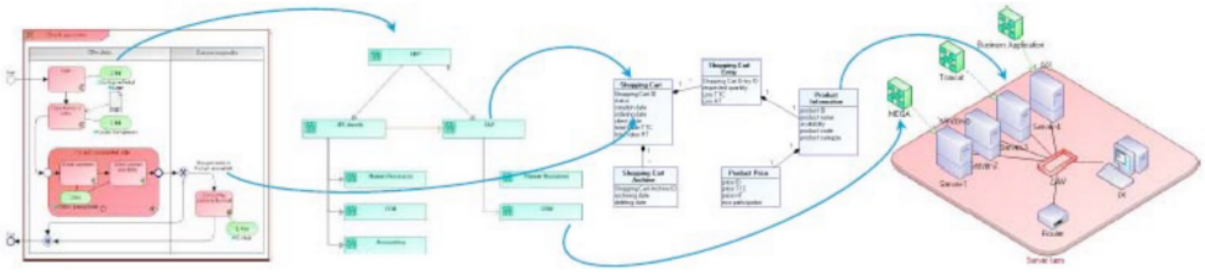
- **Struttura statica**: quali sono i moduli e che relazioni hanno
- **Struttura dinamica**: rappresenta come i componenti software (ovvero le istanze dei moduli a runtime) interagiscono tra di loro
- **Struttura di deployment**: devo decidere quale usare, e quali componenti vanno in quali nodi computazionali (virtuali o fisici)
- **Struttura di sviluppo**: quale team dovrà sviluppare ciascun modulo

Posso avere anche multiple strutture per tipo, dipende dall'obiettivo che devo raggiungere, quanto complesso o grande è il progetto.

Ogni struttura ha l'obiettivo di catturare uno o più interessi (punti di vista) di uno o più stakeholder.

Ogni struttura include solo alcuni elementi e definisce solo alcune relazioni tra questi elementi. Una struttura singola non può descrivere l'intera architettura di un sistema complesso. Diverse strutture contengono diverse informazioni. Ciascuna struttura descrive diversi interessi, diversi aspetti.

Le strutture sono descritte attraverso **viste**.



Le strutture sono divise in **categorie** (accenni):

- **Struttura a moduli**: partiziona il sistema in unità chiamate moduli, alloca responsabilità e definisce interfacce che ciascun modulo deve fornire e richiedere. Modellano il sistema nella sua **staticità**, i requisiti funzionali sono strutturati in moduli.
- **Struttura Component-and-Connector**: Sono delle strutture che modellano la **dinamicità** del sistema, ovvero come gli elementi interagiscono a runtime.
- **Strutture di allocazione**: mappano elementi software negli ambienti del sistema (organizzativi, di sviluppo, allocativi, ...).

Relazioni

All'interno della **stessa struttura**, le **relazioni** definiscono come gli elementi interagiscono tra di loro.

Tra diverse **strutture invece**, le **relazioni** definiscono come diversi elementi di diverse strutture interagiscono.

Proprietà

Ci sono due tipi di **proprietà degli elementi**:

- **Comportamento** (funzionalità): ciò che l'elemento dichiara di fare
- **Qualità**: con che livello di qualità viene erogato il servizio (per esempio anche la reliability).

Io sono in grado di capire come funziona il sistema nella sua complessità se sono in grado di definire in maniera chiara la struttura interna del sistema.

L'architettura del software si focalizza nel capire come la struttura interna del sistema e la sua organizzazione, contribuisce alle **proprietà generali dell'intero sistema**.

Quindi tutte le definizioni messe insieme devono descrivere il sistema, con i livelli di qualità che si aspettano gli stakeholders.

Strutturo le responsabilità a livelli, facendo una comunicazione da un livello verso l'altro, senza la comunicazione nell'altro verso.

Posso avere delle **tattiche** (esempi):

- se il sistema richiede di una tolleranza ai fallimenti, allora potrei replicare alcuni elementi.
- se il sistema richiede di essere modificato, allora le responsabilità devono essere assegnate di modo che i cambiamenti impattino il numero minore possibile di elementi.
- se il sistema deve essere molto sicuro, allora la comunicazione tra elementi deve essere gestita e protetta in modo appropriato.
- se la performance e la scalabilità sono importanti, allora il workload deve essere diviso tra elementi multipli che possono essere replicati.

L'architettura **omette** certe informazioni sugli elementi che non sono utili per ragionare sul sistema. Quindi **non** pensiamo alle implementazioni.

Un'**architettura è un'astrazione** che ci permette di guardare al sistema in termini dei suoi elementi, come sono organizzati, come interagiscono, come sono composti e quali sono le loro proprietà.

! Nei diagrammi non vuole vedere i casi anomali. I diagrammi dovrebbero essere semplicemente delle sequenze di azione-dato-azione-dato. Per esempio se modello il processo di login, non voglio andare a gestire in queste viste (viste funzionali) se l'utente inserisce la password sbagliata. C'è una vista apposita delle eccezioni che viene fatta quando è rilevante farla.

Altri concetti

Stakeholders: sono individui, gruppi o organizzazioni che hanno interesse nel sistema che sarà sviluppato. Chiunque, anche chi è il team di sviluppo che progetta i test, il programmatore che implementa l'applicazione, il committente del sistema, l'utente finale.

Un **interesse (concern)** su un sistema è definito come un requisito (funzionale o non funzionale), un vincolo, un obiettivo, un'intenzione o un'aspirazione che uno stakeholder ha su un sistema (o architettura. è una definizione ampia.

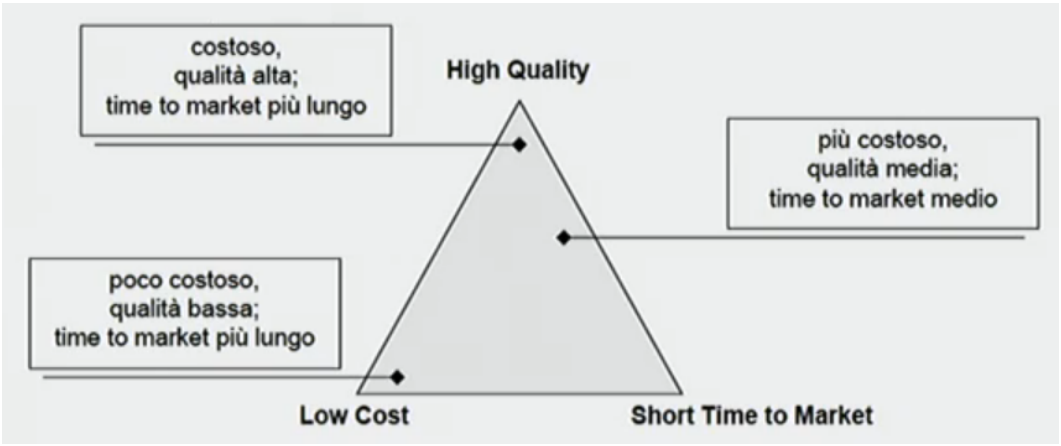
L'**ASR (Architecturally significant requirement)** è un requisito (funzionale o non) che è così importante che se non viene soddisfatto, rischio che il sistema finale non soddisfi uno o più interessi (concern) degli stakeholders.

L'**architectural definition process** include una serie di aspetti come il capire cosa dobbiamo fare, quali sono i requirements, quali sono i constraints. Quindi in questo processo:

- Bisogna capire le preoccupazioni e i bisogni degli stakeholders
- L'architettura deve avere un design che mira a considerare queste preoccupazioni
- L'architettura deve essere chiara e descritta in modo non ambiguo tramite una descrizione architetturale

Quality e trade-offs

Alcuni concetti possono essere in conflitto l'un l'altro (per esempio se voglio un progetto veloce, sarà di bassa qualità).



Difficilmente si riesce ad ottenere tutti e 3. Tante decisioni architetturali richiedono tradeoff appropriati tra gli attributi di qualità. è anche comune che i requisiti di qualità vengono rinegoziati durante il processo di definizione architetturale.

Una **descrizione architetturale** consiste in un insieme di prodotti che la documentano:

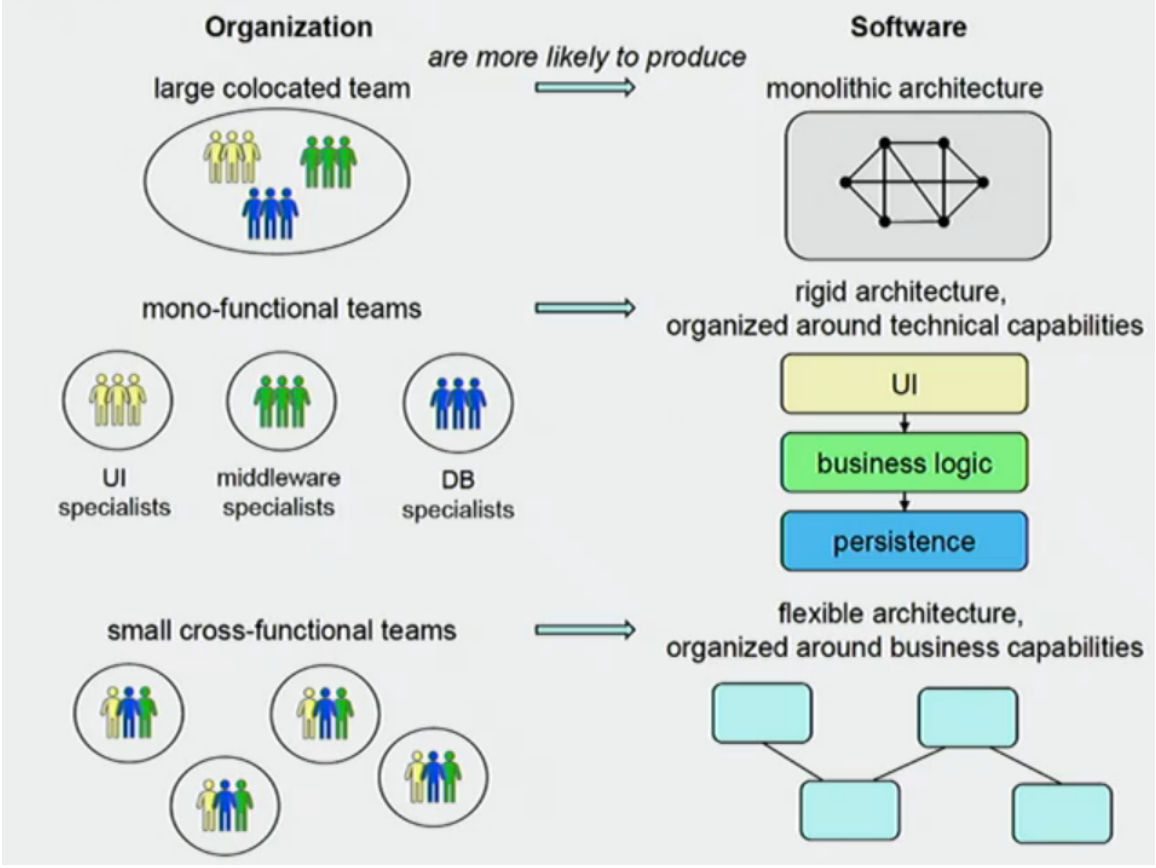
- Modelli architetturali (**views**)
- Descrizione di concerns, constraints, principi e scelte di design rilevanti
- Una giustificazione logica dell'architettura

Questo supporta anche la comunicazione con gli stakeholder e serve come guida per lo sviluppo e l'evoluzione del sistema.

L'**architetto** deve essere in grado di fare molte cose: identificare gli stakeholder, capire i suoi interessi, scegliere i trade-offs, fare decisioni, comunicare le decisione con gli stakeholder (come anche il project manager). L'architetto ha anche dei doveri come lo stare al passo con i tempi.

Il **team** è importante, deve essere **coordinato**, soprattutto se i componenti sono collegati in posti diversi o anche tra i diversi team. La comunicazione tra team diversi è più complicata e costosa.

La legge di Conway dice che le organizzazioni che progettano sistemi producono sistemi che riflettono le strutture interne di comunicazione.



Cosa rende un'architettura buona?

non esiste LA buona architettura, ma esistono tante architetture che sono buone rispetto al problema che si sta trattando.

L'importante è che l'architettura permetta di realizzare un sistema software che soddisfi tutti i concerns di tutti gli stakeholders coinvolti.

Process Recommendations

Ci sono delle **pratiche** ben consolidate che dovrebbero essere seguite per arrivare alla descrizione di un'architettura sensata:

- L'architettura dovrebbe essere il prodotto di un **singolo architetto**, o di un piccolo gruppo di architetti con un leader. Questo permette di avere consistenza nelle scelte.
- L'architetto dovrebbe basare l'architettura su una **lista di requisiti di qualità ben specificati**. L'architetto deve quindi trovarli e deve dare a ciascuno una priorità a ciascuno, per rendere più sensata la scelta dei tradeoff in futuro.
- L'architettura deve essere **documentata attraverso viste**. In questo modo si riesce a comunicare con gli stakeholder, ragionando.
- L'architettura deve essere **valutata** sia in corso di definizione che quando si arriva a quella finale (facendo anche valutazioni esterne).
- L'architettura dovrebbe essere fatta in maniera incrementale, anche per andare a provare tecnologie ed impararle. Una tecnologia che funziona o non funzione potrebbe far cambiare le scelte architetturale, oppure ci possono essere tagli di budget. Bisogna quindi creare lo scheletro del sistema dove i canali di comunicazione sono usati ma dove all'inizio si ha funzionalità minima.

Structural Recommendations

- I moduli devono essere ben definiti, ognuno deve fare una cosa sola, e che non rendano chiaro all'esterno come facciano le cose (per esempio i protocolli usati internamente). bisogna quindi nascondere le tecniche implementative. Questi moduli che nascondono le informazioni dovrebbero incapsulare le cose che potrebbero cambiare (come i protocolli), ciascuno con interfacce che incapsulano questi aspetti che possono cambiare.

- Gli attributi di qualità dovrebbero essere raggiunti usando pattern architetturali e tattiche ben conosciute, per ogni singolo attributo.
- L'architettura non dovrebbe mai dipendere da una particolare versione di un prodotto commerciale o tool. Se deve per forza usarli, allora deve rendere semplice e non costoso il cambio di versione.
- I moduli che producono dati dovrebbero essere separati da quelli che consumano dati. In questo modo se in futuro cambio dei sensori che prendono i dati, devo solo cambiare il singolo modulo.
- Non bisogna aspettarsi una corrispondenza uno a uno tra moduli e componenti
- Tutti i processi dovrebbero essere scritto di modo che siano assegnati ad un processore che può essere cambiato facilmente, volendo anche a runtime.
- L'architettura dovrebbe avere un piccolo numero di modi in cui i componenti possono interagire. Il sistema dovrebbe fare le stesse cose nello stesso modo. Questo aiuta a capire facilmente, a ridurre il tempo di sviluppo e ad aumentare la reliability e la modificabilità.
- L'architettura dovrebbe contenete uno specifico e piccolo numero di "resource contention areas", la risoluzione delle quali deve essere ben specificata e mantenuta.