Progettazione software(9)

Progettare prima di produrre. Si deve avere un approccio industriale con un metodo ingegneristico, ossia si deve perseguire la **correttezza per costruzione** e non la correttezza per correzione.
Bisogna cercare di ridurre al minimo lo sforzo della verifica, altrimenti pago un costo doppio. Per cercare di mitigare la verifica uso il principio di riduzione della complessità. Progettare per:

1. Governare la complessità del prodotto;
2. Organizzare e ripartire le responsabilità di realizzazione;
3. Produrre in economia (efficienza);
4. Garantire controllo di qualità (efficacia);

L'approccio dell'analista è detto *approccio investigativo*. Tutto non è necessariamente esplicito. Il lavoro del progettista è l'esatto opposto, deve riportare a sintesi i requisiti spezzettati e proporre una delle possibili soluzioni al problema, argomentando il valore di quella soluzione. L'analista ha fatto un buon lavoro se i requisiti sono tutti **tracciabili** e **verificabili**.
Dijkstra afferma che per soddisfare i nostri bisogni:

- **Stabilire le proprietà** di quella cosa in virtù delle quali soddisfo le proprietà attese e i bisogni. Questo è il compito dell'analista.
- **Fare quella cosa** in modo tale che le proprietà attese ci siano. E' quello che fa il progettista.

Uno dei compiti che aiuta il progettista è **fissare un'architettura**, cioè il modo in cui affrontiamo la struttura della soluzione.
Un progettista che lavora strettamente sul progetto fa **scelte tattiche** sul breve periodo, mentre un architetto ha **governance**, ha una visione sul lungo periodo. Vogliamo che il progettista si avvicini il più possibile all'architetto. Architettura sw comprende:

- Una collezione di software e componenti di sistema, regole e vincoli;
- Una collezione di bisogni per gli stakeholders;
- Una motivazione per cui la prima collezione soddisfa tutti i bisogni della seconda.

Prima di avere componenti, connessioni e vincoli bisogna avere l'idea di come sono le parti, dobbiamo avere un principio costruttivo. Sapere che forma devono avere le parti per ottenere le caratteristiche che mi servono. Esponendo un'interfaccia mostro che cosa offro. Ogni architettura ha uno **stile** architetturale riconoscibile. Secondo ISO/IEC/IEEE 42010-2011:

- L'architettura è un modo per distinguere le parti (*divide*);
- Quelle parti sono organizzate (*impera*);
- Per poter avere un'organizzazione di parti bisogna avere delle interfacce che facilitino l'organizzazione;
- Paradigma di composizione, il criterio con cui metto insieme queste parti, regole, criteri, vincoli che hanno impatto sulla manutenzione futura.

Assunto di aver capito questo, cerchiamo quali sono le qualità da perseguire in un'architettura:

- **Sufficienza:** soddisfa tutti i requisiti;

- **Comprensibilità:** comprensibile hai portatori d'interesse;
- **Modularità:** fatta di parti facilmente riproducibili e distinte;
- **Robustezza:** sopporta ingressi diversi *(giusti, sbagliati, tanti pochi)*;
- **Flessibilità:** permette modifiche a costo contenuto al variare dei requisiti;
- **Riusabilità:** architettura riadattabile a molti altri sistemi, sia nell'insieme che nelle parti;
- **Efficienza:** nel tempo, nello spazio, nelle comunicazioni. Consumare il meno possibile.
- **Affidabilità:** nessun *effetto sorpresa*, riesco a fare ciò che è atteso;
- **Disponibilità:** necessita di poco o nullo tempo di manutenzione *fuori linea*;
- **Sicurezza rispetto a intrusioni**;
- **Sicurezza rispetto al funzionamento**;
- **Semplicità:** ogni parte contiene il necessario e niente di *superfluo*, utilizzo il principio di *Occam*, elimino tutto quello che è superfluo;
- **Incapsulamento:** (information hiding), nascondo il dettaglio, mostro solo il necessario, in oltre i cresce anche la manutenibilità. Le componenti sono *Black box*, i clienti conoscono solamente l'interfaccia mentre la loro specifica nasconde gli algoritmi e le strutture dati usati all'interno;
- **Coesione:** le parti che stanno insieme hanno gli stessi obiettivi, questa aiuta sia a decomporre che a mettere un limite inferiore alla decomposizione. Maggiore manutenibilità e riusabilità, minore interdipendenza, maggiore comprensione dell'architettura;
- **Basso accoppiamento:** una modifica ha impatto minimo con gli altri. Le modifiche locali non devono avere effetto sul globale, questo è misurabile ed e composto da due valori, utilità e bisogno. Per calcolarlo si usa la metrica fan-in, fan-out, la prima da massimizzare e la seconda da minimizzare.

All'inizio della progettazione devo avere sicuramente uno stile architetturale che può essere o **top-down** (decompone i problemi) o **bottom-up** (compone le soluzioni). L'approccio normalmente utilizzato è un **compromesso** fra queste due tecniche, **meet-in-the-middle**, un approccio intermedio.
Un **pattern** lo possiamo intendere come soluzioni fattorizzate per problemi ricorrenti.
I pattern architetturali possono essere molti:

- Architettura a livelli (*Three-tier*) che comprende lo stato della presentazione, quello della logica operativa, e quello dell'organizzazione dei dati;
- Architettura produttore-consumatore;
- Architettura cliente-servente, ci sono due sottovarianti, con *fat client* e con *thin client*;
- Architettura *peer-to-peer*, interconnessione di scambio senza server di mezzo.
- Architettura di relazione, MVC, il **Model** incorpora il modello dei dati e le operazioni su di essi, la **View** incorpora la logica di presentazione, riceve in input e mostra in output l'esito delle azioni svolte dal controller sul model, **Controller** incorpora la logica di controllo del sistema;

E' importante capire che l'architettura dev'essere valutata per la sua **qualità**. Cercheremo metodi utili e ambienti di progettazione che questi metodi importano. Questo perchè perseguiamo *qualità by construction.* Vogliamo che la **baseline** sia pulita e quindi bisogna verificare che l'architettura lo sia. La progettazione la distingueremo in due fasi (lassi di tempo contigui e continui):

- una di **alto livello** in cui fissiamo l'architettura e i pattern;
- una di **dettaglio** in cui riempiamo i buchi per agevolare il lavoro del programmatore. Definizione delle unità realizzative **moduli**.In questa fase in oltre si assegnano le attività ai componenti, si produce la documentazione necessaria perchè la programmazione possa procedere in modo più disciplinato e certo possibile.

Stati di processo per SEMAT:

- **Architecture selected:** dobbiamo scegliere l'architettura e dobbiamo spiegare perché essa è adatta al sistema. Selezione delle tecnologie necessarie, posso quindi fare stime di costo sensate. Decisione su **buy**, **build**, **make**. In questo stadio di avanzamento non c'è nemmeno una linea di codice, niente di realizzato;
- **Demonstrable:** dimostrazione delle principali caratteristiche del sistema agli stakeholder. Decisioni su interfacce e configurazioni di sistema. Ho fatto e ho completato la progettazione ad alto livello ed eventualmente ho dei prototipi, ma non ho ancora implementato nulla;
- **Usable:** il sistema è usabile e ha le caratteristiche desiderate. Non è completamente finito, ho ancora difetti, ma essi sono accettabili. Quindi possiamo *sperare* di iniziare a fare la revisione di collaudo;
- **Ready:** il prodotto è così maturo che posso iniziare a scrivere il *manuale utente*, la documentazione per l'utente è pronta, gli stakeholder hanno accettato il prodotto e vogliono che diventi operativo.