Lezione #18

Martedì, 19 Novembre 2013

Tullio Vardanega, 9:30-11:30

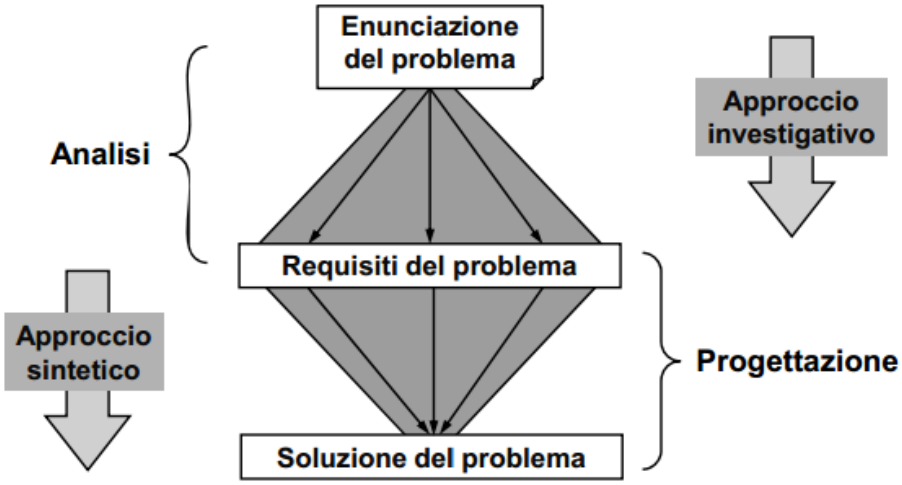
Luca De Franceschi

Progettazione Software

Progettare prima di produrre. Finora nella nostra vita abbiamo progettato relativamente poco. La progettazione è un'attività molto importante e complicata. Lo studio è oscurato perché nel mondo veloce dell'informatica conta quello che vediamo e non quello che *sta sotto*. Siamo attenti alla superficie e non alla sostanza. Il software è normalmente poco visibile. Ci sono due tipi di sforzo:

- Correttezza tramite correzione;
- Raggiungere la correttezza per costruzione (molto meglio).

L'intento è fare scelte buone, cossichè venga ridotto al minimo lo sforzo della verifica (che è costosa), altrimenti pago un costo doppio. Principio di riduzione della complessità. La prima cosa che devo fare in un progetto sw è spezzare i problemi in parti più piccole per poterle governare insieme (*divide et impera*). La progettazione e l'analisi sono due imbuti rovesciati tra loro.



C'è un'apertura molto grande fatta per frazionamento (*approccio investigativo*) tipico dell'analista. 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. Si torna da un punto singolare ad un punto singolare. L'analista ha fatto un buon lavoro se i requisiti sono tutti **tracciabili** e **verificabili**.
Dijkstra afferma che il compito che sta a un progettista di fare in modo che soddisfi i nostri bisogni è fatto di due parti:

- **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. Ci sono diversi punti di vista che devono avere tutti la stessa soluzione:

- **Committente**, che ha la visione del prodotto;
- **Fornitore**, confini del lavoro commissionato;

- **Analista**, che fissa vincoli e rischi tecnologici;
- **Progettista** che ha il compito di portare alla sintesi;
- **Architetto**

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 come un mezzo** per raggiungere un fine. L'architettura ha una soluzione che è costruttiva e si può fare, è fatta di scelte costruttive che si amalgamano molto bene insieme. "*L'arte è un fine, l'architettura un mezzo*" (Wells). Prima della fine degli anni '80 l'architettura veniva. L applicata solamente al sistema fisico. La nozione di architettura software appare per la prima volta nel 1992. Architettura sw fatta di:

- **Elementi costruttivi**;
- **Forma di queste parti**;
- **Rational**, con una giustificazione.

Dobbiamo cercare queste tre cose. Architettura sw come insieme di **componenti**, **connessioni** e **vincoli**.

- Un insieme di espressioni di bisogni che vengono dagli stakeholder (requisiti);
- Una spiegazione esplicita che giustifica le scelte e che sia razionale.

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:

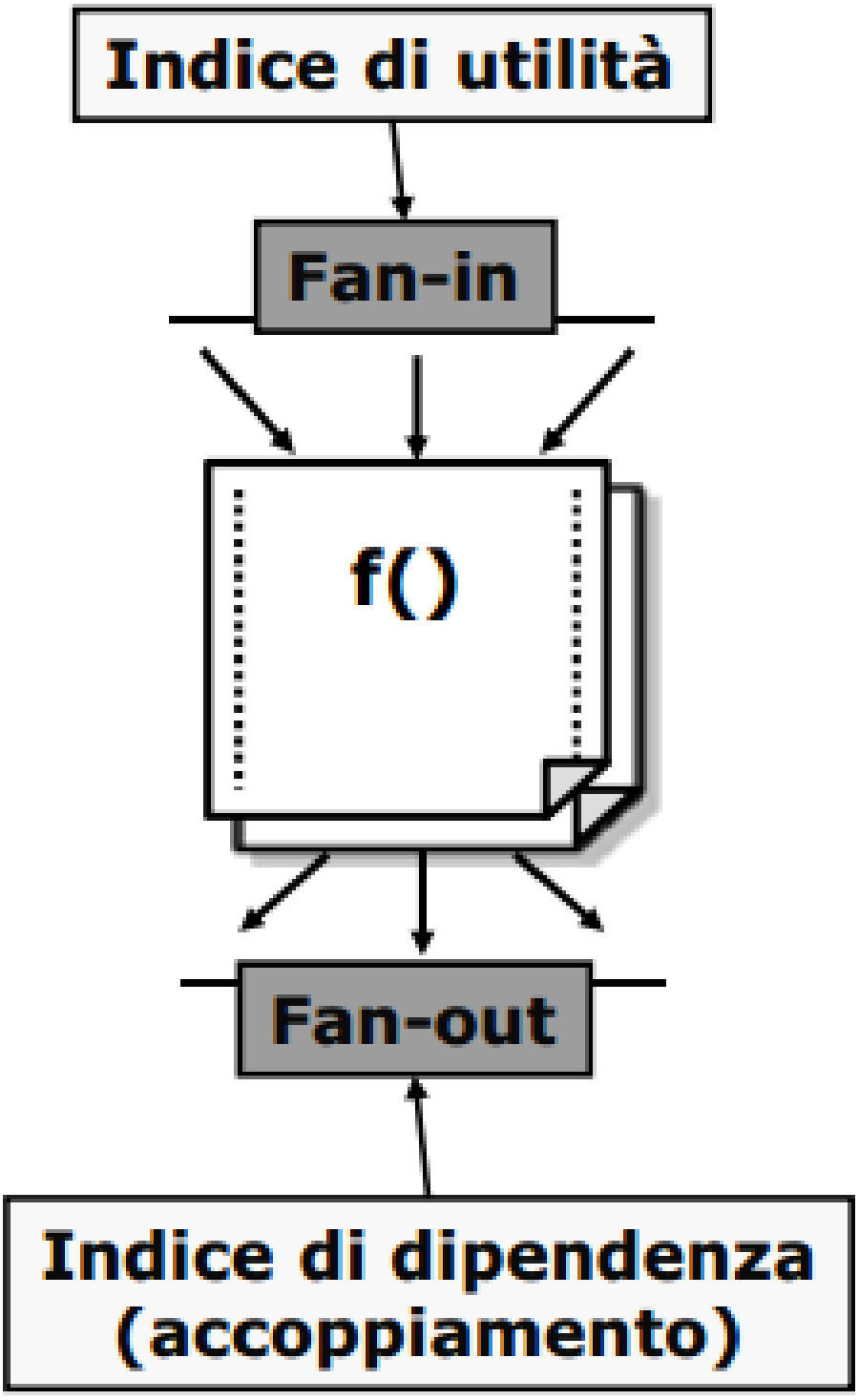
- L'architettura è un modo per distinguere le parti (*divide*);
- Quelle parti sono organizzate (*impera*);
- Per poter avere un'organizzazione di parti bisogna avere delle interfacce che facilitino l'organizzazione;
- Paradigma di composizione, il criterio con cui metto insieme queste parti, regole,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

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

- **Sufficienza**, ho tutto ciò che mi serve;
- **Comprensibilità**, non dev'essere astratto ma comprensibile agli stakeholder, tutto deve risultare chiaro;
- **Modularità**, fatta di parti facilmente riproducibili e distinte;
- **Robustezza**, se per caso un modulo fallisce non deve cadere l'intero sistema, non possiamo assumere che tutti i programmatori non commettano errori;
- **Flessibilità**, il maggior valore di un prodotto è **per quanto esso è in uso**. L'uso lungo richiederà manutenzione. Bisognerà avere una struttura che permette facilmente l'adattamento,;
- **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*;
- **Incapsulamento** (information hiding), nascondo il dettaglio, mostro solo il necessario. Tutto ciò che non è rilevante lo nascondo;
- **Coesione**, le parti che stanno insieme hanno gli stessi obiettivi;
- **Basso accoppiamento**, una modifica ha impatto minimo con gli altri. Le modifiche locali non devono avere effetto sul globale.

Soffermiamoci sulla **semplicità**. "*La mplteplicità non è da assumere senza la necessità*", principio di rimozione, devo togliere tutto ciò che non è necessario. Principio noto come "*il rasoio di Occam*". "*Ogni cosa dovrebbe essere il più semplice possibile, ma non di più*" (Albert Einstein).
L'incapsulamento è un grande beneficio per la manutenzione, perchè i cambi locali hanno poca influenza sul globale. Questo è il principio del *black box*".
L'accoppiamento è il processo per il quale io dipendo dall'esterno, e questo è misurabile. E' fatto da due cose: **utilità** e **bisogno**. Massimo di utilità e minimo di indipendenza.



All'inizio della progettazione devo avere sicuramente uno stile architetturale, ma dopo devo scegliere un *approccio progettuale* al sistema. Decomporre il sistema e identificarne le parti (**top-down**); ma questo non è lo stile *object-oriented* che invece è **bottom-up**. Inizia da parti già esistenti e le specializza (principio del **riuso**). Per fare progettazione bottom-up serve molta esperienza, l'alternativa è spezzare in parti più semplici. L'approccio normalmente utilizzato è un **compromesso** fra queste due tecniche, **meet-in-the-middle**, un approccio intermedio, ed è quello più frequentemente utilizzato.

Il riuso ha due forme:

- **Riuso interno**, uso cose che ho fatto io;
- **Riuso esterno**, uso cose già fatte da altri.

Il primo atteggiamento è guardare fuori se ci sono cose utili e abbastanza solide. E' delirante pensare di progettare in proprio. Oggi è sempre più inevitabile/intelligente utilizzare il riuso, soprattutto quello esterno.

Un **framework** è il "quadro di lavoro", un insieme integrato di componenti software prefabbricate. Impongo un'organizzazione, ciascuno si deve adattare ad esso, usare il suo stile architetturale. Nel mondo *pre-oo* erano chiamate **librerie**. Mi aiuta a fare delle cose e mi impone uno stile che è stato definito **buono**. Un framework diventa buono se è giudicato tale dalla comunità dei suoi utenti (presumibilmente utenti esperti).

Un **pattern** lo possiamo intendere come un *motivo*, uno *stile ricorrente* che sappiamo riconoscere e che ha un ruolo preciso. Vogliamo portare un pattern dentro una progettazione architetturale. E' molto importante affidarci ad un pattern. Questo concetto è stato donato al mondo del software da *C. Alexander* nel 1979. Al giorno d'oggi un buon progettista software deve assolutamente conoscere i pattern e la loro utilità.