

Lezione 4

Lunedì 14 Ottobre 2013
Luca De Franceschi

Ciclo di vita di un software → conoscere il ciclo di vita ci aiuta a capire la swe. Il sw va inteso in un senso molto più ampio di “*programma*”, serve per soddisfare i requisiti di qualcuno. Queste cose evolvono nel tempo e raggiungono **stati** tramite **transizioni** scatenate da attività che hanno il fine di far *avanzare* il sw. Si divide in 4 fasi:

- **Concezione** → quando qualcuno pensa che ci sia (o abbia) bisogno di qualcosa.
- **Sviluppo**
- **Utilizzo**
- **Ritiro**

Questi stati cambiano attraverso attività specifiche, derivanti da processi di cicli di vita.

Fase → è un periodo di tempo *contiguo* con un inizio ed una fine. Un prodotto sw si spalma sul tempo. Immaginare il ciclo di vita su un asse temporale, le fasi sono segmenti tra uno stato e l'altro.

C'è un processo che si chiama **documentazione**. Serve a produrre documenti che evidenziano lo stato di maturità di un prodotto sw. Questo processo va utilizzato **sempre** (tranne che nel ritiro).

E' importante avere un ciclo di vita chiaro in testa, perché averlo ci fa capire quello che dobbiamo fare e se siamo in grado di farlo. Il compito fondamentale è utilizzare un approccio quantificabile, in modo da poter misurare gli obiettivi e le aspettative. Cosa devo fare? Perché? Sono capace in questo modo di puntare a una qualità misurabile.

Conformità → un grado di corrispondenza ad aspettative (efficacia). Misurare quanto soddisfiamo (siamo conformi) le aspettative.

Maturità → è il grado di stabilità con il quale siamo in grado di fare le cose. Siamo maturi quando abbiamo acquisito il “*modus operandi*”. Essere sistematici attraverso un processo di maturità.

Modello di ciclo → astrazione del modo nel quale vediamo gli stati e il loro avanzamento.

Non modello → “*code and fix*”, raggiungere la correttezza “*by correction*”. Non so perché ce l'ho fatta. Prima faccio, poi penso → insieme negativo delle cose da fare. La programmazione è una delle attività meno importanti, deve essere la conseguenza.

I modelli devono essere **organizzati**. Si identificano 6 modelli organizzativi:

- 1) **Ragionamento storico** → il modo in cui si fanno attività organizzate nasce dall'ambito industriale, modello sequenziale definito nel 1970 da Wiston W. Royce. L'unico modo per non produrre “*code and fix*” è fare le cose in modo rigidamente sequenziale. **Modello Waterfall**, “*a cascata*”. Non è ammesso un ritorno ad una fase già visitata così come non è possibile risalire una cascata. Fatto una volta il codice deve *essere apposto*. *Document driven*, documenti che spiegano la fiducia nelle scelte. Prima la documentazione poi il sw. Senza i documenti non si va avanti, è una *pre-condizione*. La *post-condizione* è avere un documento approvato. Le fasi sono distinte e non si sovrappongono, tutto è allineato in una

linea temporale. Ancora oggi è un modello molto utilizzato. La frustrazione più grande è che il codice arriva tardi. Si programma molto dopo e rispetto alla nostra abitudine è frustrante. Non arrivano inoltre prototipi, bisogna immaginarseli. Questo si fa solo se tutti gli stakeholder sono d'accordo sulle *pre* e le *post*. E' una cascata senza ritorno. Funziona in un mercato non troppo competitivo. Tipicamente negli appalti pubblici funziona esattamente così.

Si può rendere più flessibile introducendo *incrementi* e/o *iterazioni* (incremento è additivo e potenzialmente distruttivo); per arrivare a una soluzione ci arriviamo con passaggi più fini (piccoli scalini). Quando ho dubbi residui il modello sequenziale va corretto, si cerca di far capire meglio il problema agli stakeholder; questi aiuti possono essere incrementali o iterativi.

- 2) Un **modello di tipo incrementale** prevede ritorni. E' difficile in questo modello definire una fase. Fisso un quadro generale, inizio a sviluppare la base, poi torno in ciclo e aggiungo delle cose. Ogni realizzazione aggiunge un pezzo. Esempio: sviluppo separato di interfaccia grafica utente e base di dati. Procedendo a cicli di incremento so esattamente quanti cicli farò. In ogni progetto devo dire quando finirò. C'è un servizio di cui si conosce esattamente la durata (*service level agreement*).
- 3) Un **modello iterativo** farebbe un passo più indietro. Se non ritorno in analisi il problema è completamente definito e non corro il rischio di buttare via cose. Nel modello iterativo torno indietro da *realizzazione* ad *analisi*. Si **riprova**. In questo modello non so dire con certezza quante iterazioni farò. Il modello iterativo lo uso quando gli stakeholder non sanno esattamente cosa vogliono. Si va *per tentativi*. E' un modello molto rischioso. Correggere il modello iterativo attraverso un modello che produce più
- 4) Nel **modello evolutivo** si ragiona su un'analisi preliminare (*schizzo*, un'idea non precisa) poi si inizia a fare una versione. Insegue il futuro non ritirando il passato, ho tante attività concorrenti. Se le cose che voglio fare non le so all'inizio una nuova versione "*asfalta*" molto del passato. Si fa su un prodotto che ha la capacità di assorbire molte versioni sul mercato (esempio i browser). Questo modello lo attua chi può sostenere molte versioni in parallelo (e quindi ha buone capacità finanziarie).
- 5) **Modello a spirale** → il problema è il *rischio non studiato*. Bisogna usare un modello fortemente consapevole dei rischi. A partire dall'inizio ci sono molti cicli ripartiti per capire meglio i rischi e solo dopo la spirale si apre. Per riuscire a trovare una via di uscita si utilizzano i *prototipi*. Servono a capire se il rischio si può risolvere, confermano se ciò che ho fatto è ragionevole. Questo modello aspira a togliere i rischi; ha avuto molto uso in ambienti *risk driven*, in progetti di carattere "*sistema*".
- 6) **Modello a componenti** → nasce sull'osservazione che molto di quello che ci serve esiste già. Pensare che rifare da capo sia molto probabilmente fallimentare. Ragionare per *riuso*. Componenti riusabili. La progettazione confronta il problema con le realtà esistenti. Adattare i requisiti alle disponibilità. Si negozia con lo stakeholder. Adatto i requisiti al panorama delle possibilità e poi progetto riusando le possibilità. La creatività sta nell'integrare e nell'usare bene ciò che esiste già. Richiede però di **studiare** il dominio.

Metodi agili → opposizione al modello sequenziale. 4 principi su cui si basa:

1. Mettere in primo piano persone e iterazioni piuttosto che processi e strumenti. Gli individui sono importanti ed è importante il modo in cui interagiscono tra loro.

2. “*Dei documenti me ne frego, basta che funzioni*”. La documentazione non sempre corrisponde a sw funzionante. E' importante che *funzioni*. Ma nel lungo periodo farà soffrire chi dovrà prendere in mano il sw.
3. Avere un buon rapporto con il *customer*, coinvolgerlo, non ingessare il rapporto.
4. Reattività piuttosto che pianificazione. Capacità di adattamento a cambiamenti delle situazioni.

Il modello agile è maturato in qualcosa di plausibile, in alcune tecniche che funzionano.

Modello agile educato → tutto ruota intorno al “*document user story*”, è l'espressione precisa di quello che l'utente vuole in quel momento. Associato ad ogni *user story* produrre una strategia che sia in grado di soddisfare quei bisogni. *User story* associato a un insieme di cose da fare che dimostrino al *customer* che c'è una soluzione (**product backlog**). Quello che si ottiene va verso il *customer* o in *feedback*. C'è un supervisore che decide l'*urgenza strategica* (**sprint**). Tutti fanno un pezzo e poi lo consegnano. Ogni iterazione è rapida e va molto verso le aspettative del cliente.

Scrum daily → mucchio giornaliero, un contatto giornaliero dell'avanzamento. Tecnica ragionevole per chi ha poca esperienza.