

UNIVERSITÀ DEGLI STUDI DI PADOVA

LAUREA IN INFORMATICA

---

Riassunto di fine corso

---

*Studente:*  
Carlo Sindico

*Matricola:* 1069322

18/02/2017

# 1 Lezione 1 Introduzione al corso

- Il software, se utile, ha vita lunga. Il ruolo di un informatico é "al controno della programmazione", in modo da non produrre codice usa e getta; deve **eseguire un processo di manutenzione del software e del codice**. Tutte le attività hanno un costo e non possono essere buttate al vento.
- **Engineering**, ingegnere, scienziato; l'ingegnere é colui che dai principi scientifici ne trae una finalità concreta, pratica. La scienza deve scoprire questi principi, prepararli. Crea qualcosa che sia mantenibile nel tempo. (piramidi, ponti, statue)
- **Software**, é un terreno molto giovane, nato durante la seconda guerra mondiale. Il SWE molto dopo, non é un ramo della computer science, ma é **una disciplina ingegneristica che mette insieme elementi e conoscenze**.
- **In assenza di principi ingegneristici, aimé si fanno solo "dei pasticci"**. Molte conoscenze vengono da campi che non sono l'informatica:
  - Informatica, si vuole che un buon ingegnere del software conosca tutte le competenze informatiche;
  - Matematica, scienza di base che aiuta a risolvere problemi;
  - Scienze gestionali ed economia, é un attività di gruppo, correlazionale, capire come gestire risorse, tempo, denaro, cogestire;
  - Ingegneria, é un pezzo di un sistema complesso che passa informazioni.
  - Psicologia, il software é rilasciato con interfacce molto orientate alle persone, bisogna intercettare le aspettative di chi usa il software.
- **Ciclo di vita di un software**, per regola il software nasce e ha una vita che termina con il ritiro. Dal momento in cui nasce esso passa in diversi stati, che lo fanno transire. Il software spende la maggior parte del suo tempo in uno stato che si chiama manutenzione, in cui si possono correggere degli errori e cambiare lo stato. Noi vorremmo una manutenzione che sia il meno invasiva possibile.
- Esistono svariate forme di manutenzione:
  - **Corretiva**, per correggere difetti eventualmente rilevati
  - **Adattiva**, per adattare il sistema alla variazione dei requisiti
  - **Evolutiva**, per aggiungere funzionalità al sistema
  - **NOTA: il software muore quando non cé più nessuna manutenzione da nessun developer o da nessuna comunità - utenza.**

- **Efficienza**, quante risorse ho impiegato per fare ciò che ho richiesto; misura il consumo e cresce al diminuire del consumo. La massima efficienza è il consumo zero, quindi da sola non basta. Le risorse che si consumano sono persone, tempo, denaro, materiale.
- **Efficacia**, è una misura della conformità, il raggiungere l'obiettivo atteso. Si è efficaci se si raggiunge con rapidità l'obiettivo; non misura le risorse. Bisogna cercare l'ottimo tra efficienza ed efficacia, trovare il massimo equilibrio, massimizzare gli obiettivi che sono tra l'efficienza e l'efficacia. Questi due termini sono in contrasto tra loro.
- **Progetto, assignment (incarico)**, sono progetti, un incarico contrattualizzato fra parti e non più negoziabile, mentre tutto il resto è negoziabile. Verrà enfatizzato molto l'aspetto contrattuale. Assignment + commitment (impegno inderogabile); il lavoro viene dato da un assignment e trasformato in commitment.
- **Engagement**, essere impegnati formalmente, impegno dal quale non si può fallire. A volte i progetti falliscono clamorosamente, per un sacco di motivi che il sve dovrebbe cercare alla radice. Il progetto potrebbe essere obsoleto dalla nascita, incapacità di chi ha l'impegno, oppure esaurimento di tempo e/o finanziamenti.

In questo caso vogliamo imparare come si fa a non fallire, a soddisfare gli obiettivi entro i tempi e i costi noti a priori. Quante ore produttive mi servono e in quante ore di calendario posso soddisfare determinati obiettivi (a prescindere dalle persone). Per fare ciò devo applicare principi ingegneristici. Per imbarcarsi in un progetto devo sapere che ce la posso fare.

- **Best practice**, il miglior modo di fare le cose, la selezione di ciò che è meglio fare in una certa professione.
- **Stakeholder**, è una persona che conta, un portatore di interesse, che ha influenza sul prodotto o sul processo.
- **DEFINIZIONE DI INGEGNERIA DEL SOFTWARE:**
- **approccio sistematico, disciplinato e quantificabile allo sviluppo, uso, manutenzione e ritiro del software.**
  - **Sistematico**, agisco secondo un sistema, affronto il medesimo problema sempre nello stesso modo (nel modo giusto); senza adottare approcci improvvisati o creativi.
  - **Disciplinato**, ciascuno fa il suo e nessuno fallisce, perché il software engineering è un'attività collaborativa. Seguire in modo rigoroso la disciplina è nostro dovere, e produce affidabilità.

- **Quantificabile**, esprimere una quantità; se si sistematici e disciplinati, si può calcolare a priori quanto tempo é necessario per svolgere quella determinata cosa. Si può essere quanti cabili solo se si é sistematici e disciplinati.
- Si cerca un approccio con queste caratteristiche (4P):
- **People**, insieme delle persone che commissionano e ricevono;
- **Product**, il software e' parte di un sistema complesso;
- **Process**, l'insieme di attività che svolgiamo ad un particolare stato;
- **Project**, attività specifi che svolte a fronte di un assignment che diventa un commitment.
- Tre strati:
- **Customer**, il cliente, quello che ha un bisogno espresso o inespresso;
- **Solution**, dove esiste il customer cé qualcuno che trova una soluzione, fatta di requisiti (di cosa cé bisogno);
- **Endeavor**, l'impegno per realizzare il software, il team interagisce con gli stakeholder; il software engineering é un'attività strettamente collaborativa, ho bisogno di un metodo di lavoro per il team. Way of working.  
SWE != PROGRAMMING, la programmazione é solo un elemento, e anche il meno importante. Il programmatore deve fare solo quello che viene chiesto (da un membro del team stesso). Il programmatore deve obbedire, non può essere creativo.
- Principi etici:
- **Considerare la qualità come primo obiettivo**
- **Produrre sw di qualità é possibile**
- **Aiutare il cliente a comprendere i suoi veri bisogni**
- **Adottare i processi più adatti al progetto**
- **Ridurre la distanza intellettuale tra il software e il problema da risolvere**
- **Essere proattivi nel cercare e rimuovere gli errori**
- **Motivare, formare, far crescere le persone**

## 2 Lezione 2 Processi Software

- La tecnologia evolve ad una velocità incontrollabile diventa obsoleta rapidamente. é necessario capire la differenza tra le cose essenziali e le cose accidentali. Tra le cose essenziali vi é la disciplina, mentre tra le cose accidentali vi é la tecnica e gli strumenti. é importante concentrarsi sugli aspetti fondamentali e non accidentali. Occorre avere capacità di impegno concettuale, astrazione, di analisi e rigore (best practice). Vogliamo utilizzare la traccia di chi ha fatto le cose in passato e le a fatte bene. Nessuno scopo può essere raggiunto senza impegno.
- **Progetto didattico**, ci sono due approcci ed uno spazio intermedio. Come si può notare dal grafico, le **skills** (competenze) servono per affrontare **challenges** sempre più alte.
- **Cosa non é un progetto**, aggregare informazioni e attività e processi per ottenere un prodotto che sembra funzionale, ma non é il "basta che respiri", bisogna sostituire il principio di "by correction", con il principio di "by construction" (costruire sapendo che funzionerà). Niente correzioni in fase di sviluppo.
- **Cosa é un progetto**:
  - Pianificazione, pianifica chi sa cosa vuole fare, é l'essenza per controllarsi, per sapere se stiamo convergendo o divergendo. Ogni attività inizia con la pianificazione
  - Analisi dei requisiti: si analizza ciò di cui si ha bisogno. L'analisi ha un'importanza decisiva, bisogna capire il problema
  - Progettazione, si decide la forma della soluzione
  - Realizzazione, dove sta anche la programmazione, che deve aderire al 100% alla progettazione. Nella realizzazione attuo, ma non sono ancora sicuro che il risultato soddisfi il cliente
  - Verica e validazione
  - Manutenzione, per la maggior parte della sua vita il prodotto resta in manutenzione. Non esiste vita operativa senza manutenzione, non esiste software perfetto quindi esso deve essere mantenuto.
  - Qualità, uno dei principi su cui punteremo, perché la qualità é possibile. Vorremmo quantificare la qualità in modo oggettivo.
- **Processi software**, attività coordinate, processi di ciclo di vita per far evolvere il software da uno stato all'altro.  
Il software é una macchina a stati:
- Concezione

- Sviluppo
- Utilizzo
- Ritiro

Nella fase di ritiro il software cessa di esistere nel senso che non c'è più alcun tipo di supporto per quel prodotto. Le transizioni sono strettamente e formalmente regolate.

- **Modelli di ciclo di vita**

- **Iterazione ed incremento**, iterazione significa che faccio la stessa operazione più volte

, incremento significa aggiungere, è additivo. Sono incrementale solo se aggiungo, è un obiettivo molto importante da raggiungere (e potenzialmente distruttivo, potrebbe far perdere tempo). Iterare significa riprovare, non ripetere. Non si può togliere quando si è incrementali.

- **Prototipo**, serve per imparare, è tipicamente usa e getta. Viene diviso in due categorie: quelli rivolti al cliente, per fargli capire un esempio di risultato finale, e quelli rivolti verso di noi per aiutarci a trovare una soluzione. Quelli rivolti verso di noi sono un costo, mentre quelli rivolti verso il cliente sono un valore aggiunto. Il primo impatto tra l'utente e il prodotto è l'interfaccia. I prototipi possono essere usa e getta" ma costano tempo.

- **Riuso**, il software che già esiste è la maggior parte del nostro prodotto. L'informatico deve insegnare a riusare in modo sistematico e non opportunistico (nessun copia incolla). Il riuso è cosa saggia se so approvvigionarmi da un fornitore intelligente.

- La **manutenzione** richiede gestione della storia, versionamento. Bisogna avere una storia del proprio prodotto. Bisogna spiegare e documentare la scelta ed avere una tecnica che salvi la storia (repository).

- **L'efficienza** si vede dove vedo il consumo di risorse. L'efficacia si misura guardando i prodotti e vedendo se sono buoni o cattivi rispetto alla produzione. Un processo è un insieme di attività coordinate e coese (tutti hanno bisogno di tutti).

Standard di riferimento: ISO/IEC 12207.

### 3 Lezione 3 Processi Software

- Il rischio è tutto ciò che mina le nostre certezze, ciò che non riusciamo a governare. Bisogna convivere con il rischio e mitigarlo, cercare di prevenire il danno.

**Risk mitigation**, ma prima bisogna capire quali sono i rischi. Bisogna affrontarli in modo che siano "addomesticati".

- Un progetto non inizia se non con un'attività importante che faccia emergere i requisiti, la maggior parte dei quali sono inespressi. (Gli stakeholder non hanno la capacità o la pazienza di scrivere tutto ciò di cui hanno bisogno). Un fornitore di un prodotto deve intuire queste necessità. Questo si fa con analisi ed emersioni predefinite.
- **Verifica e validazione**, insieme li chiamiamo **qualifica** ma sono due cose distinte ed occupano un sacco di tempo. (Consapevolezza di consegnare al customer un prodotto valido. Non dobbiamo far fare a lui il debug).  
 é irragionevole sviluppare software da zero (from scrutch).é necessario puntare sul riuso, e non sul coding la quale é un'attività frazionaria con un notevole rischio, quindi da evitare.
- **Un processo** é un insieme di attività complesse. Le attività sono suddividibili in parti più piccole, chiamate task. é importante concentrarsi sulla Pianificazione, suddividendo il tempo. Quanto più piccolo é il compito da svolgere tanto più piccolo é il rischio. Interesse decisivo é spezzare le attività ad una grana fine per ridurre i tempi, costi rischi. E' importante capire come frantumare le attività.
- Un compito lo si assume avendo acquisito consapevolezza degli strumenti e delle tecniche per portarlo a termine. Bisogna scegliere uno strumento che sia collaborativo.
- Un software non é  
 fine a se stesso ma é parte di un sistema. Il sistema lo compongono tutti coloro che sono chiamati a usarlo.
- **Organizzazione** é un aggregato di persone che agiscono secondo regole, sono sistematici, disciplinati e quantificabili.
- Un'impresa certamente funziona a "pipeline". L'organizzazione é divisa in settori, ciascuno con un proprio compito, che sono  
 finalizzati a un "flusso". Ogni settore risponde alle esigenze di attività trasversali. **Le regole sono i processi.**
- La conoscenza su come il software funziona deve essere scritta. I processi vanno documentati.( Norme di progetto, che dicono come un ruolo va svolto).
- I processi, che sono l'organizzazione, hanno bisogno di manutenzione, non possono durare per sempre.
- I processi produttivi devono avere un ciclo interno atto a migliorarli costantemente. Il ciclo interno di miglioramento é indicato con l'acronimo PCDA (o principio di Deming). Questo ciclo é fatto di 4 attività a ciclo:
  - **Plan**, non é il piano delle attività di processo, ma il piano di miglioramento; pianifico ciò che produce efficienza ed efficacia;

- **Do**, stretta attuazione di ciò che ho pianificato. Devo sapere rispetto a cosa migliorare;
- **Check**, guardo l'esito migliorativo ottenute dalle modi che fatte rispetto al piano;
- **Act**, ottenuto il risultato porto questo esito a migliorare il processo. Porto miglioramenti o, se fallisco, ragiono su come portare miglioramenti.
- **Il PDCA serve a migliorare l'efficienza e l'efficacia attraverso degli obiettivi di miglioramento.**  
Un ciclo di vita sono gli stati di maturità che ha un software dalla sua nascita al suo ritiro.

## 4 Lezione 4 Ciclo di vita del software

- **Ciclo di vita di un software**, conoscere il ciclo di vita di un software ci aiuta a capire la software engineering.
- Il software va inteso in un senso molto più ampio di "programma", serve per soddisfare i requisiti dei clienti. il software evolve nel tempo e raggiunge stati tramite transizioni scatenate da attività che hanno il fine di far avanzare il software stesso. 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 software si spalma sul tempo.  
Possiamo Immaginare il ciclo di vita su un asse temporale, dove le fasi sono segmenti tra uno stato e l'altro.
- **documentazione**. Serve a produrre documenti che evidenziano lo stato di maturità di un prodotto software. 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.



- **Conformit ,** un grado di corrispondenza ad aspettative (efficacia). Misurare quanto soddisfiamo (siamo conformi) alle aspettative.
- **Maturit ,**   il grado di stabilit  con il quale siamo in grado di svolgere le attivit . Siamo maturi quando abbiamo acquisito il "modus operandi". Ossia essere sistematici attraverso un processo di maturit .
- **Modello di ciclo,** astrazione del modo nel quale vediamo gli stati e il loro avanzamento.
- **Non modello,** ad esempio il "code and x", raggiungere la correttezza "by correction", nel quale le azioni sono eseguite senza organizzazione preordinata. "Non so perch  ce l'ho fatta".
- **I modelli devono essere organizzati.** Si identificano 6 modelli organizzativi:
  - \* **Modello Waterfall, "a cascata".** Non   ammesso un ritorno ad una fase gi  visitata cos  come non   possibile risalire una cascata. Fatto una volta il codice deve essere corretto. **Document driven,** documenti che spiegano la fiducia nelle scelte. Prima la documentazione poi il software. Senza i documenti non si va avanti,   una pre-condizione. La post-condizione   avere un documento approvato. Le fasi sono distinte e non si sovrappongono, tutto   allineato in una linea temporale. Ancora oggi   un modello molto utilizzato. La frustrazione pi  grande   che il codice arriva tardi. Si programma molto dopo e rispetto alla nostra abitudine   frustrante. Non si realizzano prototipi, bisogna immaginarseli. Questo si fa solo se tutti gli stakeholder sono d'accordo sulle pre e le post. Questo modello funziona in un mercato non troppo competitivo, tipicamente negli appalti pubblici.
  - \* **Modello "incrementale",** 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 far .
  - \* **Modello "Iterativo",** un modello iterativo farebbe un passo pi  indietro. Se non ritorno in analisi il problema   completamente definito e non corro il rischio di buttare via cose. Nel modello iterativo torno indietro da realizzazione ad analisi (fasi). Si riprova. In questo modello non so dire con certezza quante iterazioni far . Il modello iterativo lo uso quando gli stakeholder non sanno esattamente cosa vogliono. Si va per tentativi. E' un modello molto rischioso;

- \* **Modello "Evolutivo"**, 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);
- \* **Modello "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, confermando se ciò che si è fatto è ragionevole. Questo modello aspira a togliere i rischi; ha avuto molto uso in ambienti risk driven, in progetti di carattere "sistema";
- \* **Modello "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:
  - 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
  - "Dei documenti me ne frego, basta che funzioni". La documentazione non sempre corrisponde a software funzionante. E' importante che funzioni. Ma nel lungo periodo farà soffrire chi dovrà prendere in mano il software
  - Avere un buon rapporto con il customer, coinvolgerlo, non ingessare il rapporto
  - 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", e 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.