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 attivitá hanno un costo e non possono essere buttate al vento.
- **Engineering**, ingegnere, scienziato; l'ingegnere é colui che dai principi scientifici ne trae una finalitá 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 attivitá 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 aspet- tative di chi usa il software.
- **Ciclio 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 funzionalitá al sistema
  - **NOTA: il software muore quando non cé piú nessuna manutenzione da nessun developer o da nessuna comunitá - utenza.**

- **Efficienza**, quante risorse ho impiegato per fare ció che ho richiesto; misura il consumo e cresce al diminuire del consumo. La massima efficienza é il consumo zero, quindi da sola non basta. Le risorse che si consumano sono persone, tempo, denaro, materiale.

- **Efficacia**, é una misura della conformitá, il raggiungere l'obiettivo atteso. Si é efficaci se si raggiunge con rapiditá 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 piú negoziabile, mentre tutto il resto é negoziabile. Verrá enfatizzato molto l'aspetto contrattuale. Assignment + commitment (impegno inderogabile); il lavoro viene dati da un assignement e trasformato in commitment.

- **Engagement**, essere impegnati formalmente, impegno dal quale non si puó fallire. A volte i progetti falliscono clamorosamente, per un sacco di motivi che il swe dovrebbe cercare alla radice. Il progetto potrebbe essere obsoleto dalla nascita, incapacitá di chi ha l'impegno, oppure esaurimento di tempo e/o fi

  nanziamenti.

  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 ció 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 ció che é meglio fare in una certa professione.

- **Stakeholder**, é 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 quanti cabile 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 é un'attivitá collaborativa. Seguire in modo rigoroso la disciplina é nostro dovere, e produce affdabilitá.

  - **Quantificabile**, esprimere una quantitá; se si śistematici e disciplinati, si puó calcolare a priori quanto tempo é necessario per svolgere quella determinata cosa. Si puó 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 attivitá che svolgiamo ad un particolare stato;
- **Project**, attivitá 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'attivitá 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 puó essere creativo.

- Principi etici:
- **Considerare la qualitá come primo obiettivo**
- **Produrre sw di qualitá é possibile**
- **Aiutare il cliente a comprendere i suoi veri bisogni**
- **Adottare i processi piú 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 velocitá incontrollabile diventa obsoleta rapidamente. é necessario capire la differenza tra le cose essenziali e le cose aaccidentali. 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 capacitá 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 puó essere raggiunto senza impegno.

- **Progetto didattico**, ci sono due approcci ed uno spazio intermedio. Come si puó notare dal grafico, le **skills** (competenze) servono per affrontare **challenges** sempre piú alte.

- **Cosa non é un progetto**, aggregare informazioni e attivitá 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 funzionerá). Niente correzioni in fase di sviluppo.

- **Cosa é un progetto**:

  - Pianificazione, pianfica chi sa cosa vuole fare, é l'essenza per controllarsi, per sapere se stiamo convergendo o divergendo. Ogni attivitá inizia con la pianif

    cazione

  - Analisi dei requisiti: si analizza ció 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.
  - Qualitá, uno dei principi su cui punteremo, perché la qualitá é possibile. Vorremmo quantificare la qualitá in modo oggettivo.

- **Proccessi software**, attivitá 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'é piú 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 piú volte

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

- **Prototipo**, serve per imparare, é tipicamente usa e getta. Viene diviso in due categorie: quelli rivolti al cliente, per fargli capire unésempio 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 é l'interfaccia. I prototipi possono essere usa e getta" ma costano tempo.
- **Riuso**, il software che giá esiste é la maggior parte del nostro prodotto. L'informatico deve insegnare a riusare in modo sistematico e non opportunistico(nessun copia incolla). Il riuso é 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 é un insieme di attivitá coordinate e coese (tutti hanno bisogno di tutti).

  Standard di riferimento: ISO/IEC 12207.

# 3 Lezione 3 Processi Software

- Il rischio é tutto ció che mina le nostre certezze, ció 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'attivitá importante che faccia emergere i requisiti, la maggior parte dei quali sono inespressi. (Gli stakeholder non hanno la capacitá o la pazienza di scrivere tutto ció di cui hanno bisogno). Un fornitore di un prodotto deve intuire queste necessitá. 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'attivitá frazionaria con un notevole rischio, quindi da evitare.

- **Un processo** é un insieme di attivitá complesse. Le attivitá sono suddividibili in parti piú piccole, chiamate task. é importante concentrarsi sulla Pianificazione, suddividendo il tempo. Quanto piú piccolo é il compito da svolgere tanto piú piccolo é il rischio. Interesse decisivo é spezzare le attivitá ad una grana fine per ridurre i tempi, costi rischi. E' importante capire come frantumare le attivitá.

- 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 attivitá 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 attivitá a ciclo:

  - **Plan**, non é il piano delle attivitá di processo, ma il piano di miglioramento; pianifico ció che produce efficienza ed efficacia;

- **Do**, stretta attuazione di ció 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 maturitá che ha un software dalla sua nascita al suo ritiro.

# 4 Lezione 4 Ciclo di vita del software

- **Coclio 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 piú ampio di "programma", serve per soddisfare i requisiti dei clienti. il software evolve nel tempo e raggiunge stati tramite transizioni scatenate da attivitá 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 attivitá specifiche, derivanti da processi di cicli di vita.
- **Fase**, é 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 maturitá 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 é 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 qualitá misurabile.

- **Conformitá**, un grado di corrispondenza ad aspettative (efficacia). Misurare quanto soddisfiamo (siamo conformi) alle aspettative.
- **Maturitá**, é il grado di stabilitá con il quale siamo in grado di svolgere le attivitá. Siamo maturi quando abbiamo acquisito il "modus operandi".Ossia essere sistematici attraverso un processo di maturitá.
- **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 perché ce l'ho fatta".
- **I modelli devono essere organizzati**. Si identificano 6 modelli organizzativi:
  * **Modello Waterfall, "a cascata"**. Non é ammesso un ritorno ad una fase giá visitata cosí come non é 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, é una pre-condizione. La post-condizione é avere un documento approvato. Le fasi sono distinte e non si sovrappongono, tutto é allineato in una linea temporale. Ancora oggi é un modello molto utilizzato. La frustrazione piú grande é che il codice arriva tardi. Si programma molto dopo e rispetto alla nostra abitudine é 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.
  * textbfModello "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 faró.
  * **Modello "Iterativo"**,un modello iterativo farebbe un passo piú indietro. Se non ritorno in analisi il problema é 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 faró. 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 attivitá 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 capacitá di assorbire molte versioni sul mercato (esempio i browser). Questo modello lo attua chi puó sostenere molte versioni in parallelo (e quindi ha buone capacitá finanziarie);
* **Modello "spirale"**, il problema é 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 puó risolvere, confermandoo se ció che si é fatto é 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 giá. Pensare che rifare da capo sia molto probabilmente fallimentare. Ragionare per riuso. Componenti riusabili. La progettazione confronta il problema con le realtá esistenti. Adattare i requisiti alle disponibilitá. Si negozia con lo stakeholder. Adatto i requisiti al panorama delle possibilitá e poi progetto riusando le possibilitá. La creativitá sta nell'integrare e nell'usare bene ció che esiste giá. Richiede peró 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 é 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 fará soffrire chi dovrá prendere in mano il software
  - Avere un buon rapporto con il customer, coinvolgerlo, non ingessare il rapporto
  - Reattivitá piuttosto che pianificazione. Capacitá di adattamento a cambiamenti delle situazioni. Il modello agile é 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 cé una soluzione (product backlog). Quello che si ottiene va verso il customer o in feedback. Cé un supervisore che decide l'urgenza strategica (sprint). Tutti fanno un pezzo e poi lo consegnano. Ogni iterazione é 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.

# 5 Lezione 5 Gestione di progetto

- I requisiti sono l'elemento fondamentale nella maturazione di un progetto; spesso non sono ben espressi o capiti dal cliente o da noi.
- Vi sono vari passi (stadi di maturitá):
  - textbfConceived, i requisiti vanno concepiti, nascono nel luogo dove stanno gli stakeholder, ovvero tutti coloro che danno forma ad un prodotto. Non significa "completamente formato", i requisiti vanno fatti emergere;
  - **Bounded**, recintati, evitare che i requisiti si "espandano come il gas", cercare di "inserire i requisiti dentro un recinto";
  - **Coherent**, devo guardare i requisiti e devono appartenere tutti allo stesso ambito. Anche qui gli stakeholder sono gli interlocutori;
  - **Acceptable**, tutti gli stakeholder possono dire "yes! "; designa lo stato di maturitá e dice che i requisiti sono uguali per tutti e tutti sono concordi;
  - **Addressed**, ho la soluzione per i requisiti marcati come acceptable, e sono capace di dimostrare che soddisfo tali requisiti. Questo va dimostrato nella progettazione. "Sappiamo che abbiamo la soluzione";
  - **Fullfilled**, "compiere". Ho un prodotto che fa esattamente ció che mi aspettavo.
  - **Seeded**, si comincia a identificare di chi ho bisogno, non come persone ma come competenze. Dobbiamo spersonalizzare, importa il ruolo.
  - **Formed**, le persone cé le ho e posso lavorare; istituisco dunque tecniche e strumenti per collaborare;
  - **Collaborating**, le forme di collaborazione vanno studiate per evitare perdite di tempo;
  - **Performed**, una volta acquisite le tecniche siamo efficienti ed efficaci;

- **Il project manager** si occupa della assegnazione delle risorse e di stimare il problema in modo corretto, senza sovrastimarlo o sottostimarlo, concentrandosi sui problemi per cui lo spazio dei rischi sia conosciuto. Tiene conto del fatto che il personale non sará mai a sua completa disposizione, e si occupa di pianificare rispetto agli obiettivi, eseguendo una pianificazione adattativa.
- I rischi latenti provocano problemi alla gestione di progetto:
  * –¿Variante nel personale, nella disponibilitá e nella composizione del team. Le persone possono "sparire". A quel punto bisogna trovare un rimpiazzo. Questa cosa va gestita, devo fare un piano;
  * –¿Tecnologia, non sta mai ferma; due tecniche: chi usa solo tecnologie consolidate (esempio uso di cobol nelle banche) perché cambiare produce rischio, e chi usa tecnologia innovativa (per motivi di competizione) ma fortemente instabile. Questa variabilitá é un grande rischio.
- Alcuni rischi sono evitabili:
  * Cambio di requisiti;
  * Ritardo nelle specifiche. Gestione di rischi, vanno identificati all'inizio e in corso d'opera, perché possono variare nel tempo;
  * Identificazione, capire quali sono;
  * Pianificazione, cosa posso fare per evitare i rischi; Un rischio si previene o lo si mitiga.
- Il project manager vive costantemente sul risk management. Il team necessita di ruoli, che identificano capacitá e compiti. Dentro un'organizzazione produttiva ci sono raggruppamenti di ruoli. Un ruolo é attivo su un progetto, la competenza di istanza si chiama funzione aziendale.
- **Qualitá**, ció che ci consente di puntare al miglioramento continuo di efficienza ed efficacia. Qualitá di prodotto é diverso da qualitá di processo. E' piú importante la qualitá di processo;
- **Sviluppo**, insieme delle competenze per la parte di attivitá tecnica e realizzativa del prodotto;
- **Direzione**, fa sí che l'organizzazione possa stare in piedi;
- **Amministrazione**, ("service manager") erogano/gestiscono l'infrastruttura che aiuta a fare il proprio lavoro (es. manutenzione e sicurezza). Se esiste una buona infrastruttura si lavora meglio.
- Competenze allo sviluppo:
  * **Analista**, colui che serve per fare analisi dei requisiti; aiuta l'avanzamento di maturitá dei requisiti; é molto importante e occorre avere competenze su molti fronti, sapere ascoltare gli stakeholder, scrivere requisiti bounded e coherent, ragionevoli, utili, realizzabili e verificabili. L'analista pensa al problema, non alla soluzione;

* **Progettista**, pensa alla soluzione. Deve avere competenze tecnologiche e tecniche. Ricevuto il problema tira fuori la migliore soluzione possibile nel rispetto dei vincoli finanziari e temporali. Competenze tecnologiche senza pregiudizi, ma in relazione alle necessitá. Deve sapere di architettura, mettere insieme le parti della soluzione (divide et impera). Rende il problema piccolo e dominabile. Le parti devono essere organizzate in modo da essere governabili. Queste parti devono funzionare in un aggregato coerente. Un solo progettista in un team;
* **Programmatori**, il programmatore non inventa niente, deve fare ció che ha detto il progettista. Deve scrivere in modo non ambiguo. Il passaggio di consegna tra progettista e programmatore deve essere chiaro. Permette il massimo parallelismo.
* **Verificatori**, la validazione si fa sul prodotto
finito. Impiega almeno 1/3 del tempo. E' un ruolo attivo fi da subito. Serve per creare l'ambiente giusto, a rendere puliti i requisiti.
* **Responsabile**, ce ne sará 1. Rappresenta il progetto presso il fornitore e presso il committente. Svolge il ruolo di intermediario, che dice "come siamo messi"; pianifica, gestisce risorse e rischi, coordina. Responsabilitá su relazioni esterne. Deve sapere ció di cui parla.
* **Amministratore**, ruolo molto utile, prepara l'ambiente di lavoro, strumento di collaborazione e controllo di avanzamento. Gestione della documentazione di progetto. Risoluzione di problemi legati alla gestione di processi. Assegnazione di tickets.

# 6 Lezione 6 Gestione di progetto

- **Gestione qualitá** la possiamo guardare da due punti di vista: qualitá sul prodotto e qualitá su ció che si fa (qualitá di processo). E' una competenza strategica, una funzione aziendale e non un ruolo di progetto. Serve come indicatore che stiamo lavorando bene".
- **Pianificazione di progetto** serve a sapere cosa dobbiamo fare nell'assicurarci che le nostre attivitá producano esiti che aiutino la nostra collaborazione; regolare l'avanzamento. Cominciamo ad identificare cosa "cé da fare", pianifichiamo le attivitá.
  Chi lo puó fare? Quanto mi costa?
- Le attivitá hanno un flusso, hanno un inizio ed un'uscita. Si distendono su un asse temporale.
- Capite le attivitá mi devo chiedere quali ruoli assegnare. Attivitá che hanno dei vincoli sugli estremi ed eventualmente in mezzo. Devo capire quanto mi servirá per svolgere un'attivitá ad esempio: data, tempo, persone, denaro.

- Le ore vanno intese come ore di calendario: tempo/persona, quanto tempo uso tali persone sapendo che ciascuna percepisce un costo orario preciso. In base a questo posso capire quanto mi costerá un progetto.
- **Strumenti per la pianificazione**:
  * **Diagrammi di Gantt** strumento intellettuale semplice per mettere in relazione attivitá.
    Prima cosa da fare:
  * determino le attivitá da svolgere su tutta la durata o su periodi piú piccoli. L'orizzonte é dato in parte dal modello di ciclo di vita che adotto.
  * determinare le dipendenze, che consumano tempo e non producono.
  * dire quanto tempo e persone mi serviranno per una certa attivita;
  * Una volta decise le risorse, alloco persone alle attivitá. Poi mettiamo a diagramma le cose e vediamo se tornano, altrimenti si torna indietro (ciclo).
  * Il diagramma descrive, sull'asse orizzontale il tempo segmentato per unitá di lavoro (es. 1gg, una settimana, un mese..). Una buona scelta é utilizzare come unitá di misura un giorno (8 ore lavorative);
    Sull'asse verticale, ci sono tutte le attivitá che devo svolgere e archi che rappresentano le transizioni tra una e l'altra.
  * **PERT** (Program Evaluation and Review Technique), serve a calcolare il rischio di calendario che consiste nell'avere un realistico margine tra un input e un output (slack). Se misuro lo slack e mi accorgo che é 0 o negativo ho una criticitá.. Non si puó avere nemmeno slack troppo grande(slack=differenza tra due date). Diagrammi specifici collocano gli slack e rappresentano il rischio. Il diagramma di Pert indica inoltre l'ordine delle attivitá e la loro durata e indica la prima data in cui posso iniziare un'attivitá che abbia tutti gli input necessari. Calcolando il cammino piú lungo tra l'inizio e la
    fine del progetto posso calcolare l'attivitá che finisce prima. Questo diagramma legge il Gantt rispetto allo slack e calcola i cammini critici.