SWE Theory

Indice

Elenco delle figure

1 L'ingegneria del software

L'ingegneria del software ha come scopo quello di soddisfare gli obbiettivi dati entro limiti accettabili di tempo e di sforzo, però non è facile applicare principi ingegneristici al software. Essa ha strette relazioni con svariate discipline sia informatiche che non (linguaggi di programmazione, architetture di sistemi operativi e di basi di dati, scienze gestionali..)
Un principio cardine dell'ingegneria del software fu illustrato da P. Brooks nel 1987 (*"No Silver Bullet. Essence and Accidents of Software engineering"*) distinguendo tra:

- **Problematiche essenziali**: da usare come atteggiamento fondamentale nell'approccio al software, esse sono invasanti nel tempo della progettazione software. Esempi: specifica, realizzazione, verifica di prodotti software.
- **Problematiche accidentali**: strumenti e tecniche per la rappresentazione e la verifica di accuratezza di rappresentazione delle problematiche essenziali. Esse mutano nel tempo e non sono sostanziali nello sviluppo software. Esempi: linguaggio di programmazione varia nel tempo.

Le problematiche accidentali possono essere rese sempre più agevoli dall'evoluzione tecnica e tecnologica ma il problema di analisi, di rigore, di astrazione fondamentali per poter risolvere le problematiche essenziali non potranno mai venire annullati.

1.1 Definizioni di ingegneria del Software

Definizione IEEE: L'approccio sistematico allo sviluppo, all'operatività, alla manutenzione e al ritiro del software –> il software è un prodotto con un proprio ciclo di vita.
Definizione di Fairley (1985): La disciplina tecnologica e gestionale per la produzione sistematica e la manutenzione di prodotti software sviluppati e modificati con tempi e costi preventivati –> Controllo della qualità. È una complicata disciplina che racchiude varie tematiche tecnologiche connesse a tematiche economiche e manageriali. Essa affronta 3 tematiche principali:

- **Realizzazione di sistemi software**: attraverso strategie di analisi e progettazione;
- **Processo software**: attraverso un'organizzazione e gestione dei progetti, l'individuazione di un ciclo di vita del software e l'applicazione di modelli astratti del processo di sviluppo;
- **Qualità del software**: vengono previsti metodi di verifica e controllo della qualità secondo metriche fissate che ne indicano la qualità.

1.2 Software Engineer

Il software engineer non è un programmatore ma una figura professionale che realizza parte di un sistema complesso che potrà essere usato, completato e modificato da altri. Egli deve guardare e comprendere in generale il quadro nel quale il suo sistema si colloca (che include il software) e saper operare compromessi tra visioni e spinte opposte (come costi, qualità, risorse..).

1.3 Tipologie del software

Le tipologie di prodotti software da produrre possono essere:

- Software su commessa (per specifici scopi);
- Pacchetti software;
- Componenti software (moduli integrati in altri software più complessi);
- Servizi su sistemi e dati.

Molte volte però i progetti software subiscono forti ritardi o addirittura fallimenti dovuti o alle sbagliate analisi dei requisiti, cambi di tecnologie, esaurimento dei fondi, obsolescenza prematura.

1.4 Manutenzione

Tipi di manutenzione:

- **Correttiva**: per correggere difetti eventualmente rilevati;
- **Adattativa**: per adattare il sistema a requisiti modificati (anche in corso d'opera);
- **Evolutiva**: per aggiungere funzionalità al sistema (per migliorarlo);

2 Il processo software

Definizione ISO: insieme di attività correlate che trasformano ingressi in uscite:

Requisiti, problemi –> processo –> soluzioni Esempio: processo di codifica Analisi dei requisiti Norme di codifica –> Codifica –> Codice Strumenti, programmatore I processi software si dividono in 3 categorie principali:

- **Processi standard**: riferimenti di base (generici) usati come stile comune per lo svolgimento delle funzioni aziendali, pensati per una collettività di casi (modelli come un template);
- **Processi definiti**: specializzazione del processo standard necessaria per renderlo adatto ad esigenze specifiche di progetto (specializzazione del modello ad uno specifico problema);
- **Processi secondo standard aziendali**: istanza di un processo definito che utilizza risorse aziendali per raggiungere obbiettivi prefissati (il processo viene calato nella realtà aziendale).

Per la risoluzione di problemi diversi non è corretto partire continuamente con nuove istanze di processi standard, meglio piuttosto cercare un processo definito che risolva al meglio il problema, tali processi costituiscono un bagaglio di processi definiti che possono essere mappati su problemi.
L'organizzazione aziendale si basa sul riconoscimento e il supporto dei suoi processi rendendo nota la sua struttura a partire dal conoscimento dei suoi processi (lavorativi, di sviluppo..). Ci saranno elementi "verticali" fortemente orientati alla specializzazione (settori) ed elementi orizzontali (processi) che "abbracciano" più settori specializzati. L'attività di un'entità produttiva (azienda, gruppo di progetto) è regolata dall'insieme dei suoi processi che determinano le sue prestazioni.

2.1 Processi del ciclo di vita del software (12207)

Processi di ciclo di vita: definizione di ciò che occorre fare al prodotto nelle sue varie fasi di vita.
Standard ISO/IEC 12207:1995: è un modello ad alto livello che definisce i processi del ciclo di vita del software identificando i processi dello sviluppo, la struttura modulare, le entità responsabili. Esso richiede una definizione istanziata alla realtà che decide di adottarlo. Secondo questo standard i processi sono relazionati tra loro in modo chiaro e distinto (modularità) e i rispettivi compiti sono ben definiti e delineati (coesione). Esso distingue tra 3 categorie di processi:

- **Processi primari**: necessari fondamentali
  - Acquisizione: origine, fatta da parte del committente (atto di acquisto);
  - Fornitura: controparte dell'acquisizione;
  - Sviluppo dei prodotti (anche appalto esterno);
  - Gestione operativa: utilizzo del prodotto consegnato secondo regole;
  - Manutenzione: correttiva, di adattamento, evolutiva.
- **Processi di supporto** ci si può o meno appoggiare, a seconda delle decisioni
  - Documentazione del prodotto;
  - Gestione delle versioni e delle configurazioni Accertamento della qualità;
  - Verifica;
  - Validazione.
- **Processi organizzativi**: definisce una struttura operativa a prescindere dal prodotto
  - Gestione dei progetti;
  - Gestione delle infrastrutture (idoneità a fare quanto si propone) Miglioramento del processo;
  - Formazione del personale.

2.2 Il modello di ciclo di vita del software

Modello di ciclo di vita: descrizione di come i vari processi si correlano nel tempo e del flusso informativo e di controllo tra essi. Descrive l'evoluzione di un prodotto software dalla sua origine al suo ritiro. È un'astrazione delle fasi che un prodotto software attraversa da quando nasce a quando sparisce. Fornisce la base concettuale sulla quale pianificare, organizzare, eseguire e controllare lo svolgimento delle attività necessarie.
Il modello di ciclo di vita è uno strumento di pianificazione e gestione dei progetti. Non è un insieme di metodi e strumenti di sviluppo software. È una base concettuale, non esprime la tecnica specifica da utilizzare.
Esistono molti modelli, i principali sono:

- **Modello SEQUENZIALE , A CASCATA:** il ciclo di vita del prodotto è diviso e procede attraverso una sequenza ordinata di fasi (segmento del ciclo di vita). Ogni fase è caratterizzata da pre-condizioni di ingresso e post-condizioni in uscita ed ognuna di

queste fasi è distinta dalle altre, non sono ammesse sovrapposizioni tra fasi diverse. Tale modello risulta adatto allo sviluppo di progetti complessi e il suo costo totale è dato dalla somma dei costi delle singole fasi.

- **Modello INCREMENTALE:** consente che una stessa fase sia attraversata più di una volta in tempi diversi per consentire la realizzazione di approssimazioni del prodotto finale. Infatti esso procede per approssimazioni successive e a priori non si sa di quante approssimazioni si abbia bisogno prima di ottenere il giusto prodotto finale avendo un rischio di non convergenza con il prodotto finale.

- **Modello PER EVOLUZIONI SUCCESSIVE:** permette di soddisfare l'esigenza di dover rispondere a bisogni non preventivati né preventivabili. Sancisce inizialmente il numero di quante saranno le evoluzioni (versioni). Esso comporta il ri-attraversamento di fasi precedenti in tempi successivi. Ad ogni versione si attribuisce un insieme diverso di funzionalità.

- **Modello A SPIRALE:** ha come obbiettivo primario il controllo dei rischi di progetto. Esso prevede cicli interni ripetuti e rapidi dedicati a sviluppi prototipali. Per governare tutta la difficoltà iniziale servono molte iterazioni veloci di analisi.

2.3 Prototipazione e riuso

In tutti i modelli è prevista o richiesta prototipazione che può essere sviluppata secondo obbiettivi diversi per la gestione e il rilascio del prototipo. Essa può essere una versione di tipo interna e "usa e getta", una versione interna formale (baseline) oppure una versione esterna con manutenzione.
In qualsiasi modello è inerente una certa dose di riuso, che può essere occasionale (opportunistico) con poco impatto, oppure sistematico (per progetto, per prodotto) con maggiore impatto. Il riuso è fortemente agevolato dalla presenza o meno di controllo della configurazione.

2.4 Relazione tra processi e modelli

Una definizione di processi non implica necessariamente un modello di ciclo di vita. Il livello di coinvolgimento del cliente determina natura, funzione e sequenza dei processi di revisione necessari.

2.5 Fattori determinanti del ciclo di vita

Ecco alcuni fattori che influenzano la determinazione del ciclo di vita software:

- *Politica di acquisizione e sviluppo adottata a livello sistema*: logica, modalità di acquisizione, versione unica / multipla, dipendenze richieste / attese da altre componenti;

- *Natura, funzione e sequenza dei processi di revisione richiesti*: interne / esterne, bloccanti / non bloccanti, eventuali effetti sanzionatori;

- *Necessità o meno di fornire evidenza di fattibilità*: sviluppi prototipali (usa e getta, da mantenere, da evolvere), studi ed analisi preliminari;

- L'evoluzione del sistema e dei suoi requisiti.

2.6 L'organizzazione di processo

L'organizzazione interna di ogni processo si basa sul principio del PDCA[1]

- **Pianifica** (plain): definire attività, scadenze, responsabilità. Un processo esiste solo se è stato definito un piano;
- **Esegui** (do): esecuzione delle attività secondo il piano;
- **Valuta** (check): verificare internamente l'esito del processo e delle sue attività (attraverso processi di revisione);
- **Agisci** (act) correzione dei problemi identificati (feedback).

2.7 IEEE/EIA 12207 – Processi del ciclo di vita del software

(seminario J.W.Moore – 1998)

2.7.1 Natura dello standard 12207

Catalogazione in un quadro di riferimento (framework) di termini e loro significati. Si parla di processi, non di procedure e di processi del ciclo di vita, non di modelli di ciclo di vita.
Fornisce gli elementi astratti per i quali una singola parte degli aderenti al progetto aderisce alle attese (2 parti coinvolte in un progetto) e può avere legami con altri standard utili a raggiungere gli obbiettivi.
Lo standard è pensato per essere adottato da realtà aziendali, non per progetto singolo, ed è stato pensato affinché ci sia un'attuazione volontaria di esso.

2.7.2 Software Engineering (definizione IEEE)

L'applicazione di un approccio sistematico, disciplinato e quantificabile allo sviluppo, operatività e manutenzione del software che è l'applicazione dell'ingegneria al software.

- **Sistematico:** con struttura ripetitiva;
- **Disciplinato:** regolato da norme non soggettive che consentono relazioni interpersonali;
- **Quantificabile:** capacità di sapere a priori il consumo speso per ottenere l'uscita secondo misure e pianificazione in base alle misure.

2.7.3 Standard dell'ingegneria del software

Importanza: consolida la tecnologia esistente in una ferma base per l'introduzione di nuove tecnologie, protegge gli affari, gli acquirenti e migliora i prodotti; **Obbiettivi organizzativi**: miglioramento e valutazione delle competenze software, struttura per accordare le due parti interessate, valutazione dei prodotti software, garanzia di alto livello di integrità del prodotto software.

[1] Parlare anche del ciclo di demming?

2.7.4 Gerarchia

All'interno del 12207, i processi sono divisi in attività coese che a loro volta sono suddivise in task (compiti). Attività coese: contenenti solo ciò che è rilevante ai fini dell'attività e scarta tutto ciò che non è inerente. I task si possono si possono vedere come la specificazione per l'esecuzione di un'attività[2].
L'identificazione dei processi è basata su 2 principi:

- **Modularità**: i processi dovrebbero essere coesi ma dovrebbero avere un accoppiamento debole l'un l'altro;
- **Responsabilità**: ogni processo dovrebbe essere eseguibile da una "singola parte" (responsabile).

Le attività e i task all'interno del 12207 prevedono una continuità di responsabilità per tutta la durata dei processi.

2.7.5 Considerazioni

Lo standard non specifica un modello di ciclo di vita (ad esempio a cascata, sequenziale, a spirale), e inoltre non mette le dipendenze (di ordine o di tempo) sui task perché questi varieranno a seconda della scelta del modello di ciclo di vita e dal piano di progetto.
Lo standard distingue tra item (elemento) e configuration item. Una baseline costituisce una versione approvata di un configuration item[3].

2.8 Il ciclo di vita del software

Ciclo di vita: evoluzione di un prodotto dalla sua concezione al suo ritiro Concetto di ciclo di vita: concezione–>sviluppo–>utilizzo –>ritiro.
Dato un ciclo di vita, chiunque se ne occupi deve identificare le attività coese che devono essere organizzate secondo un ordinamento e dei criteri oggettivi per verificare lo stato di avanzamento e completamento.
Una volta definito il modello di ciclo di vita e di processo generico posso avere il processo definito.

2.8.1 Evoluzione dei modelli di ciclo di vita

Code-n-Fix: "NON modello": attività casuali non organizzate e può provocare progetti caotici non gestibili

2.8.1.1 A cascata:

nato agli inizi della disciplina quando c'erano progetti complicati, serviva qualcosa per gestirli. Individua fasi distinte e ordinate nelle quali si decompone il progetto, non consente ritorno a fasi precedenti ed eventi eccezionali fanno ripartire il progetto dall'inizio. Il passaggio da una fase alla successiva è basato sulla documentazione poiché ogni fase produce documenti che la concretizzano che devono essere approvati per il passaggio alla fase successiva.

[2] Inserire la definizione di task
[3] Inserire la definizione di Configuration item, milestone e baseline

- **Analisi:** fase in cui si analizzano le parti del problema senza presentare alcuna soluzione;
- **Progettazione:** emettere il progetto di realizzazione. Si possono distinguere progettazione architetturale: riguardo il sistema e progettazione di dettaglio: all'interno delle parti più piccole del sistema;
- **Realizzazione:** qui inizia il codice (codifica) a partire dai dettagli usciti dalla progettazione (moduli). Si applica in termini di codice ciò che la progettazione di dettaglio dice;
- **Manutenzione:** correttiva, adattativa, di miglioramento.

Modello a cascata in ISO 12207: ogni attività interna è indicata in attività che 12207 riconosce.12207 distingue 2 problematiche: il sistema, il software del sistema quindi le 2 cose devono essere accompagnate. Difetti principali del modello a cascata: troppa rigidità, possibile stallo nella fase di analisi, stretta sequenzialità tra fasi, non ammette modifiche dei requisiti in corso d'opera, richiede molta manutenzione, versione troppo burocratica.
Varianti:

- **Modello a cascata con prototipazione:** alcuni ritorni dovuti alla prototipazione;
- **Modello a cascata con ritorni:** divisione ulteriore in fasi.

Da dei modelli rigidi si deve passare a dei modelli che consentono ritorni, detti modelli iterativi.

2.8.1.2 Modello INCREMENTALE

Cerca di arrivare ad avere una soluzione per approssimazioni. C'è una separazione di metodo tra il modo in cui si fa analisi-progettazione e il modo in cui si fa realizzazione. I requisiti e il progetto sono fissati una sola volta. I passi della realizzazione incrementale sono pianificati a priori. Al risultato ci si arriva per incrementi successivi. Si avranno più versioni rilasciate (release) che costituiscono una baseline pubblica.

2.8.1.3 Modello EVOLUTIVO

In questo modello si sa che nel tempo ci sarà un percorso nel quale posso diluire le fasi per la risoluzione di esso. Strategia basata all'uso da parte di terze parti dei prodotti, quindi evoluzioni successive, partendo da un nucleo essenziale per poi evolverlo con aggiunte fino al prodotto finale (passi realizzativi –> evoluzioni). L'analisi preliminare non si evolve ma presenta il problema come una massima (non specifica). Questo modello equivale al sequenziale racchiuso su se stesso con in più la fase di istanziazione.

2.8.1.4 Modello A SPIRALE

(Boehm 1988) Esso riconosce 4 attività principali:

- *Definizione degli obbiettivi*: per rendere chiaro ciò che il cliente vuole;
- *Analisi dei rischi*: sicurezza e valutazione del rischio di raggiungere o meno gli obbiettivi fissati (rischi tecnici, di tempo, di costo, di soddisfazione);
- *Sviluppo e validazione*: realizzazione del prodotto;

- *Pianificazione*: governo di tutte le fasi.

Questo modello va istanziato nello specifico. Modello orientato al contenimento dei rischi che pone grande attenzione ai problemi gestionali. Nelle fasi vicino al centro si avranno molte iterazioni di sessioni col cliente per capire gli scopi e sessioni col fornitore per capire i mezzi, poi altre sessioni sviluppo e una pianificazione del nuovo ciclo. Quanto più ci si allontana dal centro tanto più si diventa uguali ad un altro modello. Questo modello non fa altro che governare in maniera diversa le iterazioni degli altri modelli. Esso inoltre attribuisce ruoli fondamentali ai clienti e ai fornitori:

- obbiettivi e ripetizioni della sessione –> clienti corretto
- sviluppo –> fornitori
- rischi –> clienti e fornitori (diversi ovviamente)

3 La gestione di progetto

Si è visto che c'è un'esigenza di doppia istanziazione perché abbiamo un modello astratto (12207) e una volta fatta la prima istanziazione del modello di processo ve ne è una seconda che gestisce il singolo progetto. Per fare un'istanza corretta bisogna fare una corretta stima di risorse, di attività, di responsabilità. Lo standard 12207 dice che se esiste un'attività allora esiste un responsabile per quell'attività.

3.1 Funzioni e ruoli

Funzione: riguarda l'azienda **Ruolo**: risorse che assumono una certa responsabilità all'interno del progetto. Alle funzioni seguono i ruoli:

- Sviluppo –> aspetti tecnologici;
- Direzione –> responsabilità decisionali;
- Amministrazione –> gestione dei processi;
- Controllo –> gestione del sistema qualità.

Funzione o ruolo? Funzione in aziende strutturate con progetti simili, ruolo in progetti diversificati.

3.2 Profilo professionale

Insieme di competenze (tecniche e tecnologiche) ed esperienze (temporali o quantitative) proprie di una persona che fanno da requisiti per l'assunzione di un ruolo in un progetto.
Differenza tra tecnico e tecnologico:

- **Tecnico**: modo in cui si usa uno strumento (neutro rispetto la tecnologia);
- **Tecnologico**: strumento sul quale si opera (ad esempio un linguaggio di programmazione).

3.2.1 Analista

Deve fare l'operazione di traduzione da bisogno del cliente a specifica utile per trovare una soluzione (da fornire al progettista). Deve essere in grado di capire il dominio nel quale lavora il cliente e molto dipende dalla buona riuscita del suo lavoro poiché ha un forte impatto sul successo dei progetti. Sono pochi e solitamente non seguono il progetto fino alla fine (passano al successivo).

3.2.2 Progettista

Esperto professionale, tecnico e tecnologico Deve indicare la tecnologia più idonea per risolvere il problema indicato dall'analista. Deve dire, rispetto al dominio applicativo e alla tecnologia, come usare gli strumenti. Il progettista rimane legato al progetto, le sue scelte (soggettive) fanno dipendere il corso del progetto e spesso si assume la responsabilità di gestione del progetto.

3.2.3 Programmatore

A basso costo perché ce ne possono essere più di uno e ha competenze specifiche, visione e responsabilità circoscritte. Il suo lavoro si può parallelizzare alla verifica. Sta a lungo sul progetto perché può essere coinvolto nella manutenzione.

3.2.4 Verificatore

Attività noiosa ma fondamentale. Ruolo di breve durata ma che più stanzia sul progetto. Attribuibile a più persone contemporaneamente. Esso deve capire le norme sulle quali viene fatta la verifica (tecniche di verifica).

3.2.5 Responsabile di progetto

Dovrà allocare le persone giuste ai posti giusti e incitare coordinamento e controllo qualitativo. Esso deve inoltre avere conoscenze e capacità tecniche. Si accentra responsabilità di scelta ed approvazione. La responsabilità dovrà avere continuità storica.

3.2.6 Amministratore di progetto

Deve assicurarsi che ad ogni istante della vita del progetto le risorse intese come umane, materiali, economiche, strutturali, siano presenti e operanti senza necessariamente essere un tecnico. Esso deve inoltre garantire un'infrastruttura funzionale. Il suo ruolo è più "orizzontale" rispetto agli altri che si occupano di specifici settori.

3.2.7 Controllo della qualità

Funzione di recente introduzione che accerta la qualità dei prodotti. Questa attività serve all'azienda stessa (per essere sicura di ciò che produce) e al cliente (garanzie sul prodotto e sull'azienda).

3.3 Attività di pianificazione del progetto

Tecnica che consente di pianificare lo svolgimento del progetto e controllarne l'attuazione, serve per avere una base su cui gestire l'allocazione delle risorse e per stimare e controllare scadenze e costi. Essa produrrà un piano di progetto che dovrà essere vivo per tutto il corso del progetto. In esso di decide "chi fa cosa", quando e per quanto tempo rispetto agli obblighi e alle scadenze.
La pianificazione comporta anche un controllo che si stia effettivamente procedendo secondo i piani, correggendo eventuali errori in corso d'opera. Non ha senso fare un'unica pianificazione di progetto, ma sarà utile strutturarla in moduli per poter gestire meglio le attività ed arginare gli errori. Va detto che le attività non possono deviare dalla pianifica.

3.4 Strumenti per la pianificazione

- **Work breakdown structure** (strutturazione interna del lavoro): consente di separare il lavoro in moduli strutturati. È una struttura statica che divide le attività in sotto attività fortemente coese e accoppiate in modo lasco (dipendenze controllabili con altre attività). Esse sono unicamente identificate secondo un ordine numerico: si va in profondità fino a quanto serve in modo che quando individuo una sotto attività posso risalire alla struttura interna (gerarchia).

- **Diagrammi di Gantt**: (ideologia capitalista) dislocazione temporale delle attività per rappresentarne la durata, la sequenzialità e il parallelismo. Il collocamento delle attività è arbitrario, l'importante è il collocamento preciso nell'asse del tempo per avere chiare inizio e fine (piano preventivo). Può succedere che l'attività abbia uno sviluppo diverso dalla pianificazione e con i diagrammi si può notare questo fatto anche visivamente. Essi infatti possono fungere da strumento visivo per lo studio di attività (parallelismo, ordine, ritardi, dipendenze, slittamenti).

- **Diagrammi di Pert**: unificazione delle 2 tecniche precedenti dove viene fornito in modo numerico (non visivo come in Gantt) la collocazione delle attività (attraverso identificatore) e lo spazio temporale (attraverso le date) in cui si colloca. I diagrammi riportano inoltre la dipendenza "da" e "verso", e il margine (slack) che le attività hanno tra di loro.

3.5 Allocazione delle risorse

Fatta la pianificazione per astratto delle attività bisogna allocare le risorse, cioè assegnare attività a ruoli e ruoli a persone. Ci sono 2 tendenze nel fare queste operazioni:

- Sovrastimare –> molto margine di cautela (spreco tempo)

- Sottostimare –> inganno, causa pianificazione senza margini, a rischio di slittamento a cascata.

Per mitigare i rischi, un'azienda assegna le persone a più progetti ma diventa molto costoso il cambio di contesto dei progetti.

3.6 Stima dei costi di progetto

Tecnica analitica per pianificare quante persone mi servono in un progetto, per organizzare le attività ed evidenziare le criticità. L'unità di misura è il tempo/persona che rappresenta il tempo nel quale una persona è impegnata in un progetto. Tale stima può essere o meno influenzata da diversi fattori:

- Dimensione del progetto (maggiore dimensione, più tempo/persona richiesto);
- Esperienza nel dominio;
- Tecnologie adottate e ambiente di sviluppo;
- Qualità richiesta ai processi;
- Fattore umano.

Legge di Parkinson: "*datemi del tempo per svolgere un lavoro e ho garanzie che il lavoro riempie quel tempo.*" Osservazione di realtà fino ad oggi vera. Le attività che si svolgono aumentano con l'aumentare dell'aspettativa su quel prodotto.

3.6.1 CoCoMo - Constrictive Cost Model

Tecnica per la stima del tempo necessario per la realizzazione di un progetto, *Constructive*: costruttivo perché si basa sull'esperienza e calibra il modello sulla base di essa, *Cost Model* spiegazione della metrica. Funzione matematica che produce in uscita un valore in tempo/persona.

$$\text{Lavoro (in mesi/persona): } \frac{M}{P} = CxPM^{S}xM$$

I fattori statici che compaiono identificano:

- **C**: complessità del progetto. Valore soggettivo che decresce con l'esperienza (valore basso, modesta complessità; valore alto, alta complessità);
- **PM**: dimensione stimata del prodotto (diversa dalla complessità) anch'essa dipendente dall'esperienza;
- **KDSI**: Kilo Delivered Source Instruction, unità di misura in cui si esprime la dimensione del prodotto che rappresenta il peso del codice sorgente consegnato richiesto per il progetto;
- **M**: dice che altri elementi del progetto (anche esterni) incidono nella dimensione del progetto (ad esempio una collaborazione remota tra gli sviluppatori..).

Criteri per definire la complessità di progetto:

- **Bassa complessità**: tutti gli sviluppatori coinvolti sul progetto hanno una buona visione dei contenuti del progetto, associata a valori consigliati dei parametri C=2,4; S=1,05; M=1;
- **Media complessità**: per capire tutto bisogna tornare al "divide et impera", capendo le parti singole ma nel complesso un po' meno. Valori consigliati C=3; S=1,12; M=1;

- **Elevata complessità** (embedded): il prodotto interagisce con componenti esterne che non sono parte del progetto sulle quali noi abbiamo zero influenza. Valori consigliati C=3,6; S=1,2; M=1.

CoCoMo va applicato su una base di esperienza alle spalle e di solito viene usato per fare delle stime a posteriori (per dedurre la complessità di progetto una volta che esso è terminato)

Una strategia per ridurre il tempo necessario allo sviluppo di un progetto è quella di aggiungere risorse, ma questa tecnica non funziona sempre in quanto alcune attività non sono parallelizzabili oppure non si ha a disposizione del buget ulteriore, dato che l'aumento di risorse provoca anche un aumento dei costi di progetto.

3.7 Rischi di progetto

Chaos: rapporto stilato dallo Studio Standish Group che si interroga sui progetti falliti degli anni precedenti, fornendo una serie di indicatori che servono a far aumentare l'esperienza. In base ai rischi, i progetti si possono dividere in categorie:

- **Progetti di successo**: consegnati in tempo, senza costi aggiuntivi, con soddisfazione del cliente (16,2%)
- **Progetti a rischio**: fallimenti sui criteri fondamentali (fuori tempo, o con costi aggiuntivi, o con prodotto difettoso – 52,7% con media sovra-costi del 189% rispetto le stime iniziali);
- **Fallimenti**: progetti cancellati prima della conclusione (31,1

Fattori incidenti sul successo dei progetti:

- *Coinvolgimento del cliente*: 15,9% sistematico, ragionevole, poiché molto spesso non è chiaro ciò che vuole;
- *Supporto della direzione esecutiva*: 13,9% l'azienda supporta il progetto disponendo strutture e risorse;
- *Definizione chiara dei requisiti*: 13% responsabilità del fornitore;
- *Pianificazione corretta*: 9,6%.

Fattori incidenti sul fallimento dei progetti:

- Requisiti incompleti: 13.1%;
- Mancato coinvolgimento del cliente: 12.4%;
- Mancanza di risorse: 10.6

4 Amministrazione di progetto

Lo scopo dell'amministrare un progetto è quello di evitare conflitti che si manifestano quando ci sono sovrapposizioni di ruoli e responsabilità. L'amministratore non dirige, ma ha il compito di fare in modo che l'infrastruttura di lavoro sia operante. Si assicura che una volta fatte le scelte esse vengano seguite dagli altri. Egli fa in modo che le regole ci siano (su incarico ed approvazione del responsabile) e fa in modo che vengano rispettate mantenendole in base alla realtà. Non ha responsabilità sulle risorse personali ma su tutte le altre (ambiente, infrastrutture, strumenti, prodotti, documenti) delle quali se ne deve occupare sapendo che l'obbiettivo è ottenere la massima produttività con il minor sforzo.

4.1 Documentazione di progetto

È la chiave per rendere il progetto gestibile, controllabile e ripetibile, essa raccoglie tutto ciò che documenta le attività, cioè tutto ciò che descrive le attività coinvolte nel progetto.
La documentazione può essere:

- `Documentazione di sviluppo`: quella fornita dal cliente, diagrammi di progettazione, codice, piani di qualifica e risultati delle prove, documentazione di accompagnamento del prodotto;
- **Documentazione di gestione di progetto**: documenti contrattuali, piani e consuntivi delle attività, piani di qualità.

La disponibilità e la diffusione dei documenti deve essere regolata secondo diverse visibilità (interni, esterni..) fissate da norme. Ogni documento ha una propria lista di distribuzione, nel caso questa mancasse, allora il documento è pubblico. Inoltre tutti i documenti devono essere chiaramente identificati, corretti nei contenuti, aggiornati, verificati ed approvati. Per ogni documento c'è un ciclo sistematico di verifica ed approvazione. La versione *i-esima* del documento, è la versione *(i-1)esima* con alcune modifiche annotate nel "diario delle modifiche" contenuto nel documento.

4.2 Ambiente di lavoro

È necessario al processo di produzione, la sua qualità influisce sulla qualità del processo e del prodotto. Le caratteristiche di qualità di ambiente possono essere:

- **Completo**: deve offrire tutto il necessario;
- **Ordinato**: deve essere facile trovarvi ciò che si cerca;
- **Aggiornato**: il materiale obsoleto non deve intralciare.

4.3 Infrastruttura

È costituita da tutte le risorse hardware (server, mainframe, rete locale, connettività, stazioni di lavoro e da tutte le risorse software (ambienti di sviluppo, di prova e di lavoro, server, rete intranet..).

4.4 Strumenti di sviluppo

Per lo sviluppo di un progetto non servono solamente compilatori, ma servono strumenti come editori di testo, verificatori, debugger, strumenti per il versionamento, per la configurazione e anche ambienti di supporto come:

- **CASE - Computer Aided Software Engineering**: insieme di strumenti che consente di agevolare le attività di un ingegnere del software con l'aiuto del computer (insieme di editor, compilatori, debugger..). Enfasi sul ciclo programmazione-verifica (non analisi);
- **Ambienti Wizard**: formati di librerie per un ciclo rapido delle operazioni (più legati alla piattaforma) –> *RAD – Rapid Application Development*;
- **CAST – Computer Aided Software Test**: enfasi orientata per agevolare le prove;

- **IDE – Integrated Development Environment**: evoluzione dell'ambiente CASE, enfasi sull'integrazione di strumenti per fare sviluppo. Fortemente integrati alla specifica piattaforma.

4.5 Strumenti di processo

Serve qualcosa per fare pianificazione, strumenti professionali per la gestione del progetto. L'analisi e la progettazione racchiudono in se stesse le attività più delicate sul progetto, quindi serve un linguaggio che aiuta ad esprimere in modo non ambiguo un'analisi, per fare un corretto tracciamento dei requisiti perché sarà utile in seguito riportare ogni volta e in che modo viene soddisfatto un requisito fatto in analisi secondo evidenza documentale. Anche sulla codifica-programmazione si applicano regole rigide poiché il codice dovrà essere il più portabile possibile e quindi dovrà essere capibile da altri, esterni al progetto. Serve quindi un qualcosa che fornisca una "misura" del codice.

4.6 Norme di processo

Si necessita di linee guida per le attività di sviluppo che al loro interno contengono norme di codifica, organizzazione ed uso delle risorse di sviluppo, convenzioni sull'uso degli strumenti di sviluppo, organizzazione della comunicazione e della cooperazione.
Come nasce una norma? Di solito in un gruppo di persone, ci si da delle convenzioni utili, secondo studi, esperienze, che devono essere rispettate. Le norme, se sottoposte ad un controllo della loro applicazione diventano regole alle quali si riconosce necessità e convenienza e per questo ne è richiesto ed accertato il rispetto.
Norme:

- controllo –> regole
- senza controllo –> raccomandazioni, consigli

Anche il contesto e le esigenze definiscono la portata della norma. Tutte le norme dovranno essere accertate dall'amministratore.
Norme di codifica: hanno come obbiettivo quello di fare in modo che il codice sorgente sia leggibile nel tempo. Da esse possono essere tratti degli standard di codifica che consentono al codice sorgente di comunicare perché ha in se cose chiare, indipendentemente dal tipo di linguaggio utilizzato.
Convenzioni su:

- **Nomi**: non solo nel codice ma si possono trovare anche all'interno di un progetti. Scelte comuni su tipi, variabili, costanti, funzioni, nomi dei file.. per aiutarci a capire che un certo nome si riferisce a una cosa specifica. Non è un'attività banale e trascurabile;
- **Indentazione**: per evidenziare visivamente la struttura del programma, pensata per rendere il flusso del programma visivamente comprensibile;
- **Intestazione**: ogni file conterrà un'intestazione obbligatoria, un preambolo a struttura fissa che contiene le caratteristiche del file, la responsabilità delle modifiche e la sua storia.

4.7 Uso del linguaggio

Serve che qualcuno fissi quali parti del linguaggio possono essere usate e quali evitate. È una strategia per costringere i programmatori a lavorare come si conviene (compilazione senza errori, uso chiaro dei costrutti del linguaggio..)

4.8 Leggibilità del codice

Leggere il codice dovrebbe essere un'attività formativa ma spesso non è così. L'ispezione del codice è un'attività con la quale si fa sostegno all'amministratore. La sintassi di un linguaggio e il compilatore sono strumenti neutri rispetto la chiarezza.

5 Requisiti

5.1 Relazione cliente – fornitore

Decide molti aspetti del progetto, determinando parte dei bisogni ed imponendo certe scelte sul prodotto. Secondo lo standard 12207 vengono riconosciuti come processi primari l'acquisizione e la fornitura. L'acquisizione è iniziata dal cliente e richiede adempimenti formali (definizione dei bisogni, gara d'appalto, valutazione e selezione delle offerte..), essa solitamente assegna la responsabilità del progetto ad un solo fornitore. La fornitura è un processo che può avere inizio prima o dopo la sigla di un contratto, ed esso richiede diverse attività di tipo PDCA (pianificazione, esecuzione e controllo, revisione e valutazione, consegna e completamento).
Si possono fissare diversi tipi di contratti col cliente, i principali sono:

- **Contratto a prezzo fisso** dove l'iterazione cliente-fornitore è cruciale per la stima del costo, la definizione e l'accettazione del prodotto, con la possibilità di esonerare il cliente da ogni intervento durante la fase di sviluppo (quindi l'accettazione diventa molto critica);
- **Contratto a rimborso dei costi** che prevede frequenti iterazioni cliente-fornitore e si presta bene per attività aperte, prototipali.

5.1.1 L'utente

L'utente non è necessariamente il cliente, spesso vengono delegate agenzie per l'acquisizione di prodotti per conto di utenti finali, quindi servono più competenze tecniche su aspetti specifici dello sviluppo richiesto.

5.1.2 Manutenzione

Una volta che il cliente ha accettato il prodotto, si entra nella fase di sviluppo operativo. Questa fase è sotto responsabilità dell'utente e può prevedere o necessitare di un processo di manutenzione. Tale processo è regolato contrattualmente e l'esecutore della manutenzione può non essere lo stesso del processo di sviluppo (consigliato che le due persone non siano la stessa).

5.1.3 Organizzazione di un sistema

Un sistema complesso può essere variamente organizzato e la definizione dell'organizzazione è responsabilità del cliente, con il coinvolgimento dell'utente. Il fornitore è chiamato ad aderire all'organizzazione data. La definizione di un sistema è inerentemente gerarchica dove ogni

livello propone una relazione cliente-fornitore. In una visione dall'alto dell'organizzazione a livelli, ciascuna componente di sistema a ciascun livello è il prodotto di uno specifico processo di acquisizione (il fornitore ad un dato livello può necessitare l'acquisizione di uno o più prodotti al livello inferiore)

5.1.4 Ruoli e responsabilità

Livello 0 (più alto):

- **Cliente** è corresponsabile nella definizione dei bisogni, coadiuvato dall'utente. Ha responsabilità sulla definizione dei vincoli di progetto e dei requisiti (specificati dal fornitore). Esprime accordo con la qualifica di prodotto dichiarata dal fornitore.
- **Fornitore** può venire consultato riguardo la definizione dei bisogni e dei vincoli di progetto e concorda con il cliente la specifica dei requisiti da soddisfare. Il fornitore si assume la responsabilità della qualifica e della fornitura del prodotto. Può relazionarsi come cliente con fornitori di livello inferiore.

5.1.5 Relazione tra processi

Hey! Tu! Si dico a te! Ho bisogno di un'immagine, vedi pagina 14 della fonte.

5.2 Analisi dei requisiti

L'attività di analisi prevede di portare ad un accordo tra cliente e fornitore quando le due visioni del problema si incontrano, in modo da identificare il prodotto da commissionare, capire cosa deve essere realizzato e per definire completamente gli accordi committente/fornitore. Occorre inoltre capire le implicazioni economiche da parte del fornitore (analista) e da parte del cliente c'è solo l'interesse che siano soddisfatti tutti i requisiti, che oltre ad espliciti possono essere impliciti o derivati. Da tener presente, dal rapporto Chaos 1995: la prima causa dei fallimenti dei progetti sono i requisiti incompleti.

5.2.1 Analisi tradizionale e analisi moderna

L'attività di analisi secondo la prassi di tipo tradizionale prevede uno studio di fattibilità che è lo studio del capitolato d'appalto per una comprensione immediata e per capire se vale la pena entrare nel progetto. Nel caso lo studio di fattibilità dia esito positivo si passa all'analisi dei requisiti che definisce il problema all'interno del dominio, nel quale certi termini, conoscenze devono essere imparate perché sono specifiche del dominio (glossario). L'analisi dei requisiti è aiutata dall'uso di linguaggi formali o semi-formali (UML,diagrammi..).
Una volta finita l'analisi si passa alla stesura della specifica (progettazione architetturale, spettante al progettista) dove si inizia a trovare la soluzione al problema fornito dall'analista. Non è detto che un'analisi dei requisiti abbia una sola soluzione, è compito del progettista trovare la migliore. L'enfasi di questa fase è sulla funzionalità e sul modo di ottenere risultati (in base a vincoli). Dopo la specifica segue la progettazione di dettaglio (top-down) che parte senza avere elementi di soluzione prefissati. Si inizia dal problema, lo si spezza in sottoproblemi e si risolve ogni singola parte. Questa fase aggiunge dettaglio agli elementi della soluzione.

Studio di fattibilità $\rightarrow$ Analisi dei requisiti $\rightarrow$ Specifica $\rightarrow$ Progettazione top-down $\rightarrow$ realizzazione

L'analisi di tipo moderno invece è principalmente orientata agli oggetti (Object Oriented). Anch'essa prevede uno studio di fattibilità, un'analisi di tipo orientata agli oggetti che viene supportata da formalismi grafici (es. diagrammi "use case") ed ha uno stretto collegamento con la progettazione perché già individua le classi. Successivamente all'analisi si ha la progettazione che è sempre di tipo object oriented. Essa è coadiuvata dall'utilizzo di componenti prefabbricati e realizza componenti riusabili. Dopo la progettazione si ha la programmazione ad oggetti.

Studio fattibilità → Analisi orientata agli oggetti → Progettazione OO → Programmazione ad oggetti

5.2.2 Lo studio di fattibilità

L'obbiettivo principale di questa fase è quello di fornire informazioni necessarie alla decisione sull'effettuazione di un progetto, valutando la fattibilità tecnica ed organizzativa (strumenti, soluzioni architetturali, attrezzature,...). Tale studio non richiede conoscenze aggiuntive, ma è un'attività molto breve che guarda l'esistente, valutando costi e benefici, individuando i rischi legati alla realizzazione del progetto e considerando le scadenze temporali che il progetto impone. Durante tale fase possono emergere delle alternative di tipo architetturale (modello o tipo di sistema da adottare...) o di tipo realizzativo (possibili subappalti).

5.2.3 Comprensione del dominio

Consente di acquisire le competenze di dominio per fare una corretta analisi dei requisiti. Tali competenze possono essere acquisite tramite la documentazione esistente, oppure tramite la conduzione di interviste agli utenti (clienti) interessati, oppure tramite lo studio di soluzioni esistenti. Si deve arrivare ad un glossario in cui ci siano tutti (per completezza) e soli (per sinteticità) i termini chiave del dominio. Il glossario si può ottenere in 2 modi:

- mettendo una sezione "glossario" in ogni documento;
- unificare il glossario evolvendolo durante il progetto.

5.2.4 I requisiti

Si possono dividere in 2 categorie:

- **Funzionali**: tradizionalmente i requisiti a cui è dato maggio risalto poiché il prodotto è visto come un insieme di funzionalità.
- **Caratteristiche di qualità**: secondo una visione più ampia e moderna legata ad efficienza, affidabilità, secondo modelli della qualità del software e regolata da norme per identificare le caratteristiche di qualità.

L'attività di analisi dei requisiti ha come obbiettivo quello di raccogliere i requisiti, redarre il documento Analisi dei Requisiti, valicare i requisiti, rinegoziare i requisiti non chiari o rischiosi, gestire i requisiti.

5.2.5 Il documento dei requisiti e la sua validazione

Tradizionalmente i requisiti possono essere estratti da interviste ai clienti/utenti, da questionari scritti, da osservazioni di futuri utenti al lavoro o dallo studio di documenti. Negli ultimi tempi tale estrazione si è spostata sulla produzione di prototipi o su incontri periodici tra clienti e realizzatori (JAD – Join Application Development). I requisiti devono poi essere riportati nel "Documento dei requisiti" del quale il verificatore si deve accertare di completezza, struttura e correttezza del linguaggio. Tale documento non deve fornire una soluzione al problema. Il documento dei requisiti deve successivamente venire validato controllando la correttezza dell'associazione con l'analisi dei requisiti, indipendentemente dalla forma. Per fare tale validazione si possono usare alcune tecniche:

- **Ispezione**: analisi mirata ad alcune parti del prodotto;
- **Walkthrough**: analisi esaustiva (completa);
- **Matrice delle dipendenze**: legata ai requisiti.

Solitamente l'analisi dei requisiti identifica troppi requisiti, quindi serve un negoziato col cliente per modificare la lista dei requisiti individuando quelli obbligatori, quelli desiderabili (non necessari, ma utili) e quelli opzionali. Tutti i requisiti devono essere classificati e identificabili, quindi dovranno essere registrati in strutture dati apposite e dovranno essere identificati da un numero sequenziale basato sulla struttura del documento. Inoltre occorre anche tenere traccia di eventuali cambiamenti dei requisiti.
Il documento di analisi dei requisiti è destinato soprattutto ai progettisti e agli sviluppatori. I prodotti dell'attività di analisi dei requisiti sono spesso documenti scritti in linguaggio naturale con evidente rischio di ambiguità interpretativa. Servono quindi alcune linee guida per evitare espressioni ambigue. Esso può contenere quindi un linguaggio di tipo semi-formale (grafici, diagrammi) o di tipo formale.
La struttura di tale documento prevede un'introduzione che spiega lo scopo del documento, del prodotto, eventuali definizioni, acronimi e riferimenti. Ci sarà poi una descrizione generale del progetto che specifica il contesto del prodotto, le sue funzioni, le caratteristiche dell'utente utilizzatore ed eventuali vincoli o limiti del prodotto.
Sarà anche presente un glossario per i termini non di uso comune, quelli specifici del dominio del prodotto. Ovviamente si avrà la lista dei requisiti che specificherà quelli funzionali, quelli di qualità e di interfaccia. Sarà anche definito l'ambiente di esecuzione del sistema e se necessario i vincoli sul formato dei dati. Si può inoltre prevedere un appendice e un indice (molto utile).

5.3 Dai requisiti al progetto architetturale

Definizione: Un requisito è una proprietà (attributo) che occorre possedere per soddisfare un determinato bisogno.
Le attività primarie richieste sono:

- **Analisi dei bisogni**: serve a definire dei requisiti a livello di sistema (a responsabilità del cliente) e di specificare dei requisiti software;
- **Partizionamento del sistema in componenti**: decomposizione del problema in parti di sistema;
- **Attribuzione dei requisiti ai componenti**: ogni componente soddisfa qualche requisito.

L'unione dei componenti dà l'architettura.

5.3.1 Classificazione dei requisiti

Occorre distinguere tra attributi di prodotto ed attributi di processo: Gli **attributi di prodotto** definiscono le caratteristiche richieste al sistema da sviluppare (esempio: specifica di una funzione da calcolare). Esprimono requisiti funzionali determinando le capacità computazionale richieste al sistema e requisiti non funzionali riducendo i gradi di libertà disponibili nella definizione della soluzione.

Gli **attributi di processo** pongono vincoli sulla conduzione e sulle uscite delle attività previste dal processo. Esempio: imposizione di una particolare tecnologia di sviluppo (un linguaggio, uno strumento).

Tutti i requisiti devono essere verificabili poiché chi impone un requisito deve sapere come accertarne il soddisfacimento e chi è chiamato a soddisfare un requisito deve poterne stimare il costo di verifica. Alcuni requisiti derivano implicitamente da attributi di prodotto e/o di processo assegnati dal cliente o decisi dal fornitore

IMMAGINE!!! Schema della classificazione dei requisiti

Alcuni requisiti di primo livello (di sistema) possono non essere soddisfacibili perché possono essere:

- Tecnicamente impossibili (ad esempio integrare parti software scritte in linguaggi incompatibili tra loro);
- Possibili, ma di implementazione troppo costosa (come considerare un componente senza possederne i sorgenti);
- Possibili, ma mutuamente esclusivi tra loro (come usare componenti standard e avere un limite sul sistema).

L'analisi dei requisiti deve accertare la soddisfacibilità dei requisiti rispetto ai vincoli esistenti sui processi del progetto. Al termine dell'analisi i requisiti confermati devono essere tutti necessari e sufficienti: nessun bisogno trascurato e nessuna caratteristica superflua. Una priorità relativa può essere assegnata ai requisiti confermati.

5.3.1.1 Progettazione architetturale

Essa segue l'attività di analisi e può essere influenzata da esigenze od opportunità di riuso (meglio se sistematico) di componenti che possono essere di tipo aziendale, commerciale o imposte dal cliente. Le componenti riusabili che possono includere codice sorgente od eseguibile, specifiche di interfaccia, modelli architetturali.

5.3.2 L'ingegneria dei requisiti

Ingegneria dei requisiti: termine che denota l'insieme delle attività necessarie per il trattamento sistematico dei requisiti. I requisiti software sono uno dei prodotti del relativo processo.

L'ingegneria dei requisiti deve essere vista come un ciclo PDCA: da formalizzare e pianificare (modello di processo, piano di attività), da eseguire e gestire (responsabilità primarie, organizzative, di supporto), da verificare e migliorare (a livello di efficienza di processo e di qualità del prodotto).

Tutto ciò richiede responsabilità con competenze di ingegneria del processo. Si possono inoltre individuare altre attività e competenze richieste dal processo (già visti):

- **Analisi dei requisiti**: analisi delle fonti, classificazione, modellazione concettuale, decomposizione del sistema, allocazione, negoziazione;
- **Verifica e validazione**: tramite revisione interna e/o esterna, prototipazione, analisi del modello concettuale;
- **Produzione (dei documenti di specifica)**: Studio di Fattibilità, Analisi dei Requisiti, Specifica Tecnica;
- **Gestione e manutenzione dei prodotti**: tracciamento delle attribuzioni, gestione dei cambiamenti.

Tecniche di analisi delle fonti:

- Interviste con il cliente;
- Generazione ed analisi di scenari - Prototipazione che può essere: interna (per il fornitore) oppure esterna (per il cliente);
- Discussioni creative "Brainstorming" (approccio maieutico);
- Osservazione dei comportamenti e dei bisogni.

Definizioni:
Verifica: intende accertare che l'esecuzione di un dato processo non abbia introdotto errori. è principalmente rivolta al processo, ma applica anche ai prodotti di processi intermedi.

$\rightarrow$ *Did I build the system right?*

Validazione: intende accertare che l'uscita dell'insieme di processi eseguiti sia il prodotto atteso.

$\rightarrow$ *Did I build the right system?*

6 Qualità del software

Dallo standard ISO 9000

Qualità: La capacità di un insieme di caratteristiche di un prodotto, sistema o processo, di soddisfare le esigenze dei clienti e degli altri portatori di interesse.
La percezione della qualità in campo software è arrivata molto tardi nonostante essa contribuisca valore alle attività. La qualità non è un concetto molto chiaro poiché può essere vista sotto aspetti differenti come il corretto soddisfacimento dei requisiti, l'idoneità all'uso del prodotto, la soddisfazione del cliente...
Gestione della qualità: L'insieme delle attività coordinate per dirigere e controllare un'organizzazione rispetto alla qualità.
La qualità può essere gestita secondo diverse componenti quali la politica e gli obbiettivi di qualità, la pianificazione della qualità, il suo controllo ed accertamento, il suo miglioramento.
Se l'attività di gestione della qualità è ben pensata costituisce una politica che prevede una pianificazione a priori e un monitoraggio continuo, secondo il paradigma PDCA (processo che tende a migliorare con auto-verifica).

Pianificazione della qualità: Le attività della gestione della qualità mirate a definire gli obbiettivi della qualità ed i processi e le risorse necessarie per conseguirli.
La pianificazione della qualità è una premessa al controllo di essa secondo politiche corrette, scelte strategiche, strumenti di controllo e di gestione della qualità.

Controllo della qualità: Le attività della gestione della qualità messe in atto affinché il prodotto soddisfi i requisiti. Il controllo può prevedere diverse modalità per la sua attuazione come il collaudo e la verifica oppure l'analisi e la conoscenza del dominio. L'enfasi in questa attività è legata alla gestione (vedo se quanto stiamo facendo è uguale a quanto pianificato). Fare controllo di qualità solo sul prodotto finito può essere molto rischioso quindi bisogna definire una tecnologia che dipani il controllo in corso d'opera, in momenti strategici, in modo da avere un controllo globale e continuo, per fare in modo di risolvere eventuali problemi.

Accertamento della qualità: Le attività della gestione della qualità messe in atto per accertare che i requisiti siano soddisfatti.

L'accertamento è mirato al soddisfacimento degli obbiettivi. La percezione del livello di soddisfacimento può essere:

- *Interna* –> da parte dell'azienda;
- *Esterna* –> da parte del cliente.

Si può parlare di qualità riferita a:

- prodotto (visione verticale): bene o servizio;
- sistema: insieme degli elementi in cui il prodotto si colloca;
- processi (visione orizzontale): attività correlate finalizzate alla realizzazione degli obbiettivi;
- organizzazione: sulla struttura e l'amministrazione, a maggior ritorno economico.

La qualità di prodotto è legata alla qualità dei processi e non è mai rinchiusa all'interno del perimetro del prodotto (coinvolge anche attività esterne ad esso). Da tenere presente che se un processo ha qualità allora anche i prodotti hanno qualità. (non viceversa). Requisiti di qualità: Un esigenza od un'aspettativa dichiarata, comunemente intesa come implicita oppure obbligatoria (ISO 9000)

6.1 Per avere buona qualità

Definire bene cosa deve essere realizzato, come si controllerà
Controllare

- Per conoscere ed intervenire
- Per dare/avere confidenza
- Per migliorare i risultati

La qualità deve essere certificata con Norme per i prodotti, per tutelare il cliente rispetto all'uso od al valore di prodotti:

- FCC (Federal Communications Commission);
- CE (Consumer Electronics);
- OEM (Original Equipment Manufacturer);
- DOC;
- Carte dei servizi.

Norme per i processi:

- Requisiti di una funzione aziendale;
- Ad es.: ISO 9001 per il sistema di qualità aziendale.

La norma esprime requisiti comuni.
Quali strumenti per garantire qualità?

- Seguire la ricetta (limitare la libertà creativa) definendo bene cosa fare (P-D) e controllare (C-A);
- Analisi e definizione dei requisiti attraverso modelli per la qualità del software, strumenti per la definizione dei sistemi, metriche per definire livelli qualitativi;
- Controllo continuo del progetto: rispetto dei vincoli contrattuali, controllo e verifica delle attività e dei risultati.

6.2 Modelli della qualità software

Valutazione dei prodotti: Visione dell'utente (problemi d'uso), visione dello sviluppatore (problemi tecnici), visione della direzione (problemi di costi) Serve un solo modello per committenti e fornitori per uniformare la percezione della qualità e per uniformare la valutazione della qualità

6.3 Modelli della qualità

Strategia tipica: definizione di caratteristiche e loro organizzazione in una struttura logica **Modello ISO/IEC 9126:2001** Software engineering - Product quality - Part1: Quality model 7 caratteristiche principali – 31 sottocaratteristiche Strumento di definizione e valutazione che prevede organizzazione gerarchica delle caratteristiche e definizione di metriche Visioni della qualità: interna, esterna, in uso -7 caratteristiche principali visione del cliente: funzionalità, affidabilità, usabilità, efficienza, qualità in uso visione del fornitore: manutenibilità, portabilità Il processo di valutazione:

7 Qualità del processo software

Lo scopo è quello di

- Definire il processo per controllarlo (e farlo controllare) meglio, per raccontarlo in maniera più convincente;
- Controllare il processo per migliorarlo;

- Efficacia: prodotti rispondenti ai requisiti.

Efficienza: minori costi a pari qualità di prodotto erogata –> Esperienza: apprendere dall'esperienza (anche degli altri)

7.1 Le norme ISO 9000

Certificazione ISO 9001 (2a metà anni '90) per valutare, per controllare, non per scegliere La famiglia delle norme:

- 9000 Fondamenti e glossario;
- 9001 Sistema di Gestione della Qualità (SGQ) – requisiti;
- 9004 Guida al miglioramento dei risultati.

7.1.1 Documentazione del Sistema Gestione di Qualità (SGQ)

Manuale della qualità: Il documento che definisce il sistema di gestione della qualità di un'organizzazione (ISO 9000) *Caratteristiche richieste*: completo rispetto ai requisiti, deve collegarsi al resto della documentazione del SGQ e delle procedure aziendali, deve relazionare gli obiettivi di qualità alle strategie per ottenerli (esprimere la politica aziendale rispetto alla qualità).
Piano della qualità: Il documento che definisce gli elementi del SGQ e le risorse che devono essere applicati in uno specifico caso (prodotto, processo, progetto) (ISO 9000). È una concretizzazione specifica del Manuale della Qualità, ha spesso valenza contrattuale. *In pratica*: accerta la disponibilità di analisi dei requisiti, architettura e soluzioni tecniche, pianificazione delle verifiche e delle prove, risultati delle verifiche e delle prove. Fornisce modelli dei documenti, accertare la tracciabilità di soluzioni a requisiti e pianifica le attività **Valutazione di un processo: modello CMM (CapabilityMaturityModel)** Limiti del modello: una stessa realtà aziendale può adottare pratiche poste a livelli diversi e se non applica tutte le pratiche di un dato livello non può avanzare al livello superiore; modello discreto e non continuo incapace di differenziare tra L- ed L+; troppo focalizzato sulle pratiche (cosa si fa e come); insufficiente attenzione agli obiettivi (perché lo si fa) Obiettivi di una valutazione

- I portatori d'interesse: i destinatari dei risultati, i responsabili dei processi valutati, i responsabili delle attività di valutazione;
- Valutazione o miglioramento: risultato esterno o interno, valutazione formale o no (self-assessment);
- Definizione della portata: dei processi inclusi nella valutazione e degli indicatori di valutazione.

8 Progettazione software

Perché progettare: per far fronte alla complessità del prodotto, per organizzazione e ripartizione delle responsabilità, per l'economia di produzione, per il controllo di qualità.
Progettare non è pianificare

- Analisi: qual è la cosa giusta da fare?

- Progetto: come farla giusta?

Attraverso una descrizione di una soluzione che soddisfi tutti i portatori di interesse.
Il codice non esiste ancora → i prodotti di questa fase sono l'architettura, il modello logico, ...
Si ha un approccio sintetico con valutazione delle possibili alternative

8.1 Obiettivi della progettazione

Soddisfare i requisiti di qualità fissati dal committente e dal fornitore. Definire l'architettura del prodotto impiegando componenti con specifica chiara e coesa, realizzabili con risorse date e costi fissati e con struttura che faciliti cambiamenti futuri dovuti a modifica od evoluzione dei requisiti.
L'architettura non è il fine, ma lo strumento per il raggiungimento degli obiettivi di progetto.
Definizione di architettura: la decomposizione del sistema software in componenti costitutive, l'organizzazione di tali componenti (ruoli, responsabilità ed interazioni), le interfacce necessarie all'interazione tra tali componenti, i modelli di composizione delle componenti.

8.2 Problemi di progettazione

- Problemi strutturali: architettura interna (p.es.: sistema distribuito vs. sistema centralizzato);
- Problemi infrastrutturali: architettura esterna (p.es.: piattaforma di sistema operativo; supporto DBMS; sistema di comunicazione e trasporto dati; sistema di interfaccia utente);
- Problemi tecnologici (p.es.: linguaggi di programmazione e strumenti di sviluppo associati)
- Problemi realizzativi (p.es.: componenti acquistabili, riusabili o da sviluppare ex-novo)
- Problemi tecnici (p.es.: scelte algoritmiche)
- Problemi organizzativi : come la quantificazione del lavoro (p.es.: stimata mediante COCOMO), l'uso delle risorse e ripartizione dei compiti (p.es.: piano di progetto), l'allestimento dell'ambiente di lavoro (p.es.: sviluppo, test, versionamento, configurazione)

8.3 Strumenti di progettazione

- Strumenti concettuali : definizione del "sistema software" (visione concettuale (astratta), distinta dalla realizzazione concreta), delimitazione dei problemi e delle loro soluzioni, isolamento e decomposizione(+coesione,-accoppiamento), progettazione orientata agli oggetti.
- Strumenti metodologici : progettazione architetturale del sistema, progettazione di dettaglio delle sue parti (sottosistemi)

8.4 Progettazione architetturale

Al fine di dominare la complessità del sistema, organizzando il sistema in componenti di bassa complessità secondo la logica del "divide et impera", per ridurre le difficoltà di comprensione e realizzazione, identificare schemi realizzativi e componenti riusabili.

È importante anche riconoscere le componenti terminali che non necessitano di ulteriori trasformazioni e cercare un buon bilanciamento: più semplici le componenti più complessa la loro interazione. Strategie di decomposizione : top-down (decomposizione di problemi), bottom-up (composizione di soluzioni), sandwich (più frequente) Linguaggi di descrizione architetturale

- Descrizione degli elementi: componenti, porte e connettori (p.es. diagramma delle classi in UML)
- Descrizione dei protocolli di interazione: tra componenti, tramite connettori
- Supporto ad analisi: di consistenza (analisi statica ad alto livello), di conformità ad attributi di qualità, di comparazione tra soluzioni architetturali diverse

8.5 Progettazione di dettaglio:

Attività: definizione delle unità realizzative (moduli) cioè un carico di lavoro realizzabile dal singolo programmatore, un "sottosistema" definito (una componente terminale od un aggregato di esse); un insieme di funzionalità affini (un modulo package); specifica delle unità; definizione delle caratteristiche significative dal nulla o tramite specializzazione di componenti esistenti. **Obbiettivi**: assegnare unità logiche a componenti fisiche anche per organizzare il lavoro di programmazione; produrre la documentazione necessaria perché la programmazione possa procedere senza bisogno di ulteriori informazioni, per attribuire i requisiti alle unità (tracciamento), per definire le configurazioni ammissibili del sistema; definire gli strumenti per le prove di unità: casi di prova e componenti ausiliarie per le prove e l'integrazione.

8.6 Riuso

Capitalizzare i sottosistemi già realizzati impiegandoli più volte per la realizzazione di altri prodotti ottenendo minor costo realizzativi e minor costo di verifica. Problemi: progettare diventa un problema aperto perché occorre anticipare bisogni futuri ed è raro riusare al 100% e modificare è difficile e rischioso ma un investimento $\rightarrow$ risparmio a lungo termine.

8.6.1 Qualità della progettazione

- Progettazione architetturale: astrazione <–> dettaglio, ricerca del "miglior equilibrio"
- Progettazione di dettaglio: incapsulazione (Information hiding) <–> coesione <–> accoppiamento.

Incapsulazione: componenti "blackbox" :semplici fornitori e clienti di funzionalità (relazione d'uso), è nota solo la loro interfaccia (dichiarazione dei servizi). Sono mantenuti nascosti gli algoritmi usati e le strutture dati interne.
Vantaggi di un alto grado di incapsulazione: nessuna assunzione sul funzionamento della componente, maggiore manutenibilità, maggiori opportunità di riuso.
Coesione: proprietà endogena di componente, funzionalità "vicine" devono stare nello stesso componente, vicinanza per tipologia, algoritmi, dati in ingresso ed in uscita. Vantaggi di un elevato grado di coesione migliora la manutenibilità e facilita il riuso, riduce il grado di dipendenza fra componenti e facilita la comprensione dell'architettura del sistema. **Accoppiamento**: proprietà esogena di componenti: quanto le componenti si usano fra di loro $\rightarrow$ U = M x M massimo accoppiamento, U = 0 minimo accoppiamento. Metriche: *Fan-in* e `fan-out` strutturale: SFIN

è indice di riuso (utilità), SFOUT è indice di accoppiamento (dipendenza). Buona progettazione si ottiene con SFIN delle componenti SFOUT del sistema

9 Documentazione

Perché documentare: processo di supporto secondo ISO/IEC 12207.
Cosa documentare: attività e prodotti da pianificare, eseguire, verificare, correggere.
Come documentare: contenuti attesi ai fini di revisione, contenuti rilevanti ai fini di pianificazione ed esecuzione.
Cosa misurare: non serve misurare tutto indistintamente, bisogna focalizzarsi su quanto che serva il processo organizzativo di miglioramento secondo obiettivi strutturali, secondo priorità assegnate dall'organizzazione (esulano dal prodotto). Si ha una misurazione per obiettivi (ad hoc): processi, prodotti e risorse posseggono attributi misurabili.

9.1 Metriche

- Dimensione del prodotto: ISO/IEC 14143 Software engineering
- Struttura del prodotto: flusso di controllo, flusso dei dati, annidamento, modularità ed interazione
- Uso delle risorse: risorse tecniche (strumenti), risorse fisiche e logiche (spazio di memoria, tempo d'esecuzione), risorse umane (personale)
- Qualità del prodotto: ISO/IEC 9126 Software product quality(1999-2001)

9.2 Cosa documentare

Modello software: descrizione semplificata del sistema realizzato mediante uso di simboli e notazioni organizzate secondo una convenzione fissata e coerente (UML) costruito mediante metodi e strumenti standard, standard aziendale, di fatto, internazionale (meglio). Usato per ragionare sul software da sviluppare e anche sull'esito dello sviluppo **Architettura logica** –> Specifica Tecnica: prodotta al termine della fase di ingegneria dei requisiti, fissa linee e strategie di realizzazione, avvia la fase realizzativa (ingegneria di progetto), non fissa gli aspetti realizzativi concreti, mostra ciò che il sistema deve fare. È organizzata gerarchicamente attraverso livelli di astrazione (o decomposizione) successivi e consente di stabilire relazioni tra cause ed effetti offrendo una visione d'insieme della soluzione proposta al problema complessivo.
Specifica Tecnica –> progetto (design) architetturale: specifica per ogni componente del sistema la funzione svolta (strutture dati utilizzate, flussi di controllo impiegati), dati in ingresso (tipo), dati in uscita (tipo), risorse logiche e fisiche necessarie.
Architettura fisica –> Definizione di prodotto: procede dall'architettura logica e consente sviluppo parallelo ed indipendente dei componenti terminali (di basso livello), consente di stimare lo sforzo (costo, tempi) di realizzazione. Ha qualità valutabile mediante precise metriche: coesione, accoppiamento, utilità (fan-in), dipendenza (fan- out), complessità.
Definizione di prodotto –> progetto (design) di dettaglio: procede dal progetto architetturale, decompone le componenti architetturali in moduli a grana più fine finché ogni modulo ha dimensione, complessità, coesione ed accoppiamento adeguati. è influenzato da esigenze ed opportunità di riuso e la natura dei "moduli" è fissata dal supporto offerto dal linguaggio di

programmazione selezionato per la codifica (modulo è diverso file).

9.3 Tracciamento dei requisiti

Fissa la relazione tra i prodotti del processo di sviluppo:

- in avanti (forward)–>completezza: ciascun ingresso ad una fase deve essere messo in relazione con una specifica uscita di quella fase mediante matrici di tracciabilità (una sorta di base dati) che evidenziano incompletezza e duplicazione
- all'indietro (backward)–>necessità: ciascuna uscita di una fase deve essere messa in relazione con uno specifico ingresso a quella fase mediante matrici di tracciabilità dove le componenti non tracciate o non tracciabili sono superflue e da eliminare (a meno di omissioni all'ingresso)

9.3.1 Tracciamenti necessari

- Requisiti utente (capitolato) $\leftrightarrow$ requisiti software (AR)
- Requisiti software (AR) $\leftrightarrow$ descrizione di componenti (ST)
- Test di unità $\leftrightarrow$ moduli di disegno di dettaglio (DP)
- Test di integrazione $\leftrightarrow$ componenti architetturali (ST)
- Test di sistema $\leftrightarrow$ requisiti software (AR)
- Test di accettazione $\leftrightarrow$ requisiti utente(capitolato)

9.3.2 Architettura della documentazione

10 Metodiche standard di sviluppo industriale

- Segmento di ciclo di vita coinvolto compreso nell'intervallo temporale tra la prima e l'ultima revisione esterna
- Modello di ciclo di vita interno : adottato dal fornitore entro tale segmento determinando il piano d'uso delle risorse disponibili

10.1 Pianificazione

Relazione tra compiti, persone ed impegno necessario: analisi delle componenti di impegno non riducibili, aggiungere risorse ad un gruppo di progetto. Una buona pianificazione – insieme a buona analisi e buona progettazione architetturale costa sforzo a monte ma risparmia sforzo a valle.

10.2 Modello di ciclo di vita interno

Non tutti i modelli si adattano allo stesso modo agli adempimenti esterni richiesti dal progetto, la scelta interna è libera, ma comporta oneri variabili, la scelta è spesso per prodotto (per progetto) indipendentemente dall'organizzazione di appartenenza

10.2.1 Sequenziale

- Vantaggi: impone disciplina al progetto, comporta verifiche rigorose sul completamento di ogni fase.
- Svantaggi: richiede un notevole sforzo di documentazione, allontana la percezione del prodotto (analisi e progetto) dalla sua realizzazione (codifica e qualifica).
- Si presta bene a: progetti a rischio contenuto, con poche dipendenze dall'esterno e limitati impatti sull'esterno

10.2.2 Incrementale

- Vantaggi: agevola il dialogo cliente-fornitore approssimando progressivamente il prodotto finale, tollera bene una limitata fluttuazione (e maturazione) di requisiti.
- Svantaggi: degenera facilmente nel circolo vizioso "code 'n fix", è difficile decidere a priori quante versioni intermedie siano utili o necessarie.
- Si presta bene a: progetti nei quali l'integrazione del prodotto finale ha elementi di incertezza tecnica e finanziaria

10.2.3 Evolutivo

- Vantaggi: quelli del modello incrementale, ma all'interno di un contesto più formale
- Svantaggi: aggiornamento dei documenti di specifica e definizione (AR, ST, DP) per ogni versione successiva
- Si presta bene a: progetti nei quali la definizione dei bisogni è particolarmente laboriosa

10.2.4 A spirale

- Vantaggi: favorisce l'esplorazione di alternative e promuove pratiche di riuso
- Svantaggi: l'analisi delle alternative e dei rischi ed il contenimento dei costi di prototipazione richiedono esperienza
- Si presta bene a: progetti nei quali l'analisi dei rischi beneficia e coinvolge anche il cliente

10.3 Progettazione software

Comprende specifica (ST) e definizione (DP) del prodotto.
Definizione IEEE: Il processo di definizione dell'architettura, delle componenti, delle interfacce e delle altre caratteristiche di un sistema o di una sua componente.
Processo interno al processo di sviluppo che procede dall'analisi dei requisiti producendo una descrizione della struttura interna e dell'organizzazione del sistema.

- **Attività 1**: Progetto (design) architetturale: definisce la struttura e l'organizzazione del sistema secondo una visione ad alto livello (identifica le componenti, entità funzionalmente coese e suscettibili di implementazione mediante ulteriore decomposizione)

- **Attività 2**: Progetto di dettaglio: ciascuna componente è descritta ad un livello sufficiente per determinarne la codifica

Tecniche:

- Astrazione: dimenticare informazione (attributi specifici) ad un certo livello per applicare operazioni uguali ad entità diverse (Ad ogni astrazione corrisponde una concretizzazione)
- Accoppiamento: misura dell'intensità della relazione tra moduli (inter) [se è forte –>scarsa modularità]
- Coesione: misura dell'intensità della relazione tra i costituenti di un modulo (intra) [se forte–>buona caratterizzazione]
- Decomposizione modulare: una buona decomposizione architetturale identifica componenti tra loro indipendenti
- Incapsulazione (information hiding): separare l'astrazione dal dettaglio realizzativo
- Sufficienza: la definizione dell'astrazione è sufficiente a caratterizzare l'entità desiderata
- Completezza: l'astrazione esibisce tutte le caratteristiche richieste
- Atomicità: la definizione dell'astrazione non può essere convenientemente decomposta in astrazioni più primitive

Problematiche della progettazione software

- Concorrenza : se e come decomporre il sistema in entità attive concorrenti (processo, task, thread) assicurando efficienza di esecuzione, atomicità di azione, consistenza ed integrità di dati condivisi, semantica precisa di comunicazione e sincronizzazione, predicibilità di ordinamento (scheduling)
- Controllo e gestione degli eventi: evento relativo al flusso dei dati o relativo al flusso di controllo o relativo al trascorrere del tempo. La trattazione di tutti questi.
- Distribuzione : se e come componenti softwaresono disseminate su più nodi di elaborazione, come tali componenti comunicano fra loro
- Trattamento degli errori e delle eccezioni: come prevenire, gestire e tollerare eventi anomali (guasti, difetti interni, errori d'uso)

10.4 Integrità concettuale

Desiderabile in ogni architettura di sistema, procede da una definizione unitaria (non unilaterale, perché passa al vaglio dei membri del progetto) richiede osservanza (ai costruttori) e vigilanza (all'architetto), nozione aristocratica piuttosto che democratica. è distinta dalla realizzazione concreta, consente più percorsi implementativi, permette parallelismo con l'implementazione

10.5 Sistemi reattivi - interattivi

10.5.1 Sistemi reattivi

Non possono essere adeguatamente descritti in termini puramente funzionali o relazionali. Agiscono permanentemente su (oppure interagiscono con) un ambiente circostante. Soggetto a vincoli di tempo reale. Il flusso di controllo all'interno del programma è regolato dall'arrivo di eventi che richiedono un trattamento (di controllo, di risposta) da completarsi entro un preciso limite temporale (deadline)

10.5.2 Sistemi interattivi

Il software interattivo è generalmente soggetto a vincoli temporali laschi od inesistenti. Limitate conseguenze da tempi di risposta variabili od anche del tutto impredicibili. Non gestisce la piattaforma hardware del sistema. Opera in stretto contatto con (in risposta ai comandi di un) operatore

10.5.3 Sistemi a tempo reale

Si definisce "sistema a tempo reale" qualunque sistema nel quale l'istante temporale nel quale i valori di uscita sono prodotti è significativo. Il tempo di reazione agli stimoli provenienti dall'ambiente (incluso il trascorrere del tempo) è determinato dalle caratteristiche ed i requisiti dell'ambiente. Interagiscono con il loro ambiente e lo tengono sotto controllo. Il tempo di rilevazione ed il tempo di modifica sono estremamente importanti. Devono avere correttezza funzionale e temporale da dimostrare mediante validazione.

10.6 Definizioni

Sicurezza (safety): i requisiti di sicurezza tendono a rendere il sistema incapace di produrre danni catastrofici. Non è importante prevenire ogni guasto, ma assicurare che quelli che avvengano abbiano conseguenze tollerabili. Un linguaggio adatto per la realizzazione di sistemi con caratteristiche di sicurezza possiede almeno i seguenti attributi: è definito mediante uno standard internazionale (p.es. ISO), ciò consente di accertare l'adeguatezza (conformance) del compilatore; ha struttura modulare e supporta progettazione e codifica strutturate (facilita l'astrazione ed il riuso di componenti verificate ed affidabili); facilita la rilevazione di errori specialmente a livello di compilazione; consente accesso logico e strutturato agli elementi hardware del sistema; fornisce controllo rigoroso sulla visibilità di tipi, operazioni e dati ai moduli del programmi `Affidabilità (reliability)`: riguarda la prevenzione di ogni tipo di errore che possa condurre ad un guasto di sistema. Certificazione di sicurezza di applicazione: processi fortemente orientati alla produzione di documentazione di supporto perché la documentazione fornisce evidenza di certificazione.