

Ingegneria dei Sistemi Software

20 ottobre 2013

1 Introduzione

Software Engineering: l'applicazione di un approccio sistematico, quantificabile, disciplinato allo sviluppo e al mantenimento del software. Riguarda tutti gli aspetti del software, dalle fasi iniziali fino alla conclusione.

È un processo ingegneristico di innovazione, riuso e condivisione delle informazioni.

Cos'è un software? Programma di computer con associata una documentazione.

Computer science si focalizza più sulla teoria, l'ingegneria del software invece riguarda tutti gli aspetti pratici dello sviluppo del software. Il system engineering riguarda tutti gli aspetti del computer, quindi software, hardware e processi; comprende anche il software engineering.

I costi del software sono la parte principale della produzione di un sistema, molto superiori ai costi hardware. Esiste una legge che dice che il costo di sviluppo è quadratico rispetto alle dimensioni del software (Berra-Meo).

Gli Attributi di un buon SW:

- **Manutenibilità:** il sw deve poter evolvere per stare al passo al cambiamento dei bisogni dei clienti.
- **Affidabilità e sicurezza:** un sistema affidabile non dovrebbe causare danni fisici o economici quando accade un system failure. Utenti malintenzionati non dovrebbero accedere o danneggiare il sistema.
- **Efficienza:** non sprecare risorse ma essere tempestivo, buon utilizzo di memoria ecc. Parametri: responsiveness, memory allocation, CPU utilization, etc.
- **Accettabilità:** il sw deve essere accettato dai clienti su cui è stato designato. Usabile e compatibile con i sistemi proprietari e altri sw. Usabilità in base al target di utenza.

Stadi di avanzamento del SW process:

SW Specification: Clienti ed Ingegneri definiscono il SW da produrre specificando funzionalità e constraints dello stesso.

SW Development: Realizzazione del prodotto software.

SW Validation: Testing del prodotto SW per garantire le funzionalità richieste e i constraints fissati.

SW Evolution: Modifica ed arricchimento del prodotto dopo la messa in funzione per:

- Correggere bug o anomalie.
- Aggiungere funzionalità.
- Adattamento a nuovi bisogni/leggi.

I processi di progettazione e sviluppo devono essere **standardizzati** e **documentati**.

Note utili:

- Aggiungere forza lavoro ad un progetto in ritardo, **aumenta** il ritardo.
- Cambiare i requisiti in corso di sviluppo `e **molto costoso**.
 - La manutenzione del SW mal progettato ha un costo **paragonabile** al costo della realizzazione del SW stesso.

Consigli:

- Anticipare i cambiamenti
- Utilizzo di formalismi.
- Requisiti formulati in modo **non ambiguo**.
- Requisiti **pienamente discussi** con il cliente.
 - Modularità`.
 - Astrazione → flessibilità del prodotto.
 - Estendibilità`.

La correttezza del sw soddisfa le specifiche, e specifiche formali permettono un base formale per la correttezza.

L'ACM/IEEE è il codice etico degli ingegneri.

2 Requirements Engineering

Cosa sono i requisiti:

- Asserzioni **astratte** ad alto livello delle funzionalità e dei constraints.

- Spesso soggetti ad interpretazioni, possono creare incomprensioni tra cliente ed ingegneri.

Analisi dei requisiti:

Definisce i vincoli, interfacce, funzionalità, performance e ogni altra caratteristica che il cliente chiede.

Non include: la definizione di come il software viene sviluppato o costruito.

- Definire in modo **univoco** ed **non ambiguo** per entrambi (cliente e fornitore) le caratteristiche del SW da produrre.
- Le funzionalità devono soddisfare i bisogni dei clienti.
- Review prima dell'inizio del development.
- Stima dei costi di progettazione e sviluppo.
- Produrre la baseline per la valutazione ed il testing.
- Facilità di trasferimento: un buon documento aiuta il trasferimento del prodotto a nuovi utenti o macchine.

I tipi di requisiti:

User: Sono quei requisiti che interessano direttamente l'utente finale, sono descritti in linguaggio naturale e specificati mediante **use-case diagram** e **sequenze diagram**. Scritti per il cliente, manager, sistemisti, ingegneri.

System: Sono quei requisiti che non hanno un impatto diretto sull'utente ma sono riferiti alla realizzazione del sistema (nascosti). Definiscono funzionalità interne e constraints del sistema (file da supportare, vincoli di prestazioni, . . .). Scritti per gli sviluppatori di sw.

Funzionali:

- Stati in cui si può o non si può trovare il sistema.
- Comportamenti che deve/non deve avere.
- Descrizione dei servizi svolti/necessari dal SW, delle funzionalità.
- Quelli di sistema descrivono i servizi nel dettaglio.

Non-Funzionali:

- Constraints dei requisiti funzionali, di comportamento e di prestazioni.
- Tipo di SW da sviluppare.
- Standard da utilizzare per la progettazione e/o sviluppo.
- Potrebbero generare requisiti funzionali.

Di dominio:

- Vincoli del sistema nel dominio di operazione.

2.1 Software Requirements Specification (SRS)

Questo documento contiene tutte le specifiche dei requisiti: funzionale, non funzionali utente e di sistema che il SW deve possedere, inoltre indica i constraints da seguire ed consigli sulla metodologia di sviluppo del SW.

SRS `e di fatto un contratto tra cliente e progettisti.

1- Natura dell'SRS:

- Funzionalità: cosa il sw fa.
- External Interface: come il sw interagisce con le persone e le altre componenti del computer.
- Performance: responsetime, velocità.
- Attributi: portabilità, correttezza, manutenibilità, sicurezza, ecc.
- Desing Constraints imposed on an implementation: politiche di integrità database, limiti di risorse.

2- Environment of the SRS:

dovrebbe correttamente definire tutti i requisiti sw. Non specifica particolari design e non impone vincoli addizionali al sw.

3- Attributi di qualità del SRS:

Non ambiguit`a: Tutte le informazioni contenute non devono essere ambigue quindi si devono specificare oltre che in linguaggio naturale anche per mezzo di forme tabellari, diagrammi e use-case.

Correttezza: Tutto ciò che `e contenuto nel SRS deve essere corretto.

Completezza: Deve contenere tutti i requisiti voluti dal cliente.

Verificabilit`a: Deve consentire la verifica dei requisiti.

Tracciabilit`a: Deve consentire la tracciabilita` tra requisiti e funzionalità a sviluppate (codice).

Consistenza: Non deve contenere requisiti in contraddizione.

Modificabilit`a: Deve prevedere la possibilità a di modifica con le relative metodologie.

Ranked: Deve prevedere una valutazione dei requisiti per definirne criticita` e priorit`a.

Problemi:

- Clienti non sanno cosa vogliono.
- Clienti non spiegano in modo corretto i loro bisogni.
- Clienti omettono informazioni.

- Progetti multi-cliente potrebbero produrre requisiti in conflitto.

- Requisiti cambiano durante l'analisi.
- Necessità di adeguarsi a constraints politici, organizzativi, di immagine.

Gli **strumenti** per la redazione del SRS in modo univoco e non ambiguo sono:

- Utilizzo di logiche formali.
- Utilizzo di definizioni tabellari.
- Use-Case diagram.
- Sequence diagram.
- Diagrammi delle classi.

4- Joint preparation of the SRS:

clienti e fornitori dovrebbero lavorare insieme per produrre un capibile SRS.

5- SRS Evolution:

potrebbe cambiare nel corso del tempo. I requisiti devono essere specificati ogni volta completamente. Revisionare le parti sostituite.

6- Requirements Validation:

Bisogna verificare che effettivamente il sistema produca i requisiti del cliente. Le metodologie per la **validazione** più importanti sono:

Manual review: Revisione manuale del SRS da parte di vari team di verifica.

Prototipi: Produzione di un prototipo di massima per la verifica della corretta definizione dei requisiti. La validazione ha un basso costo e rileva molti errori.

Testing: Generazione del programma di testing. (Non sicuro che sia qua).

Proprietà da verificare: validità, consistenza, completezza, realistica.

IEEE Std 1028 for sw Review.

- Definire una verifica sistematica applicabile al sw di acquisizione, fornitura, sviluppo, operazioni e mantenimento. Supporto alla validazione e verifica del sw. Questo standard descrive come effettuare una revisione.
- In questo modello i requisiti e gli attributi di qualità sono parametri di input e sono imposti dal Caller. Quando la review è finita ritorna al Caller per una nuova azione. L'uscita è di solito una lista di anomalie.

Definizioni:

- Review: processo di approvazione del sw da parte di un team

- Management review:
- Technical review: team qualificato che controlla le discrepanze tra il prodotto e le specifiche scritte nel documento.
- Walk-through: tecnica di analisi nel quale un programmatore guida gli sviluppatori o altre parti interessate, verso il prodotto sw. I partecipanti domandano e fanno commenti su soluzioni o possibili errori.
- Anomalie
- Audit (revisione): un indipendente esame del prodotto sw per vedere se rispetta il contratto, requisiti, specifiche.
- Inspection: esame del sw per scovare anomalie, includendo errori sulle specifiche.

7- Embedding design in SRS

Un design descrive un particolare sottocomponente di un sistema e le sue interfacce. Srs deve distinguere tra i vincoli di design e la progettazione di un design specifico.

In casi speciali alcuni requisiti potrebbero limitare il design, come la sicurezza.

I requisiti devono essere indicati da un punto di vista esterno.

8- Embedding project requirements in the SRS

SRS dovrebbe indirizzare il prodotto sw, non il processo di produzione. Requisiti di progetto come: costo, schedulers, reporting, metodi, validation, quality, procedure; non dovrebbero essere inclusi nel SRS. Questi sono descritti in altri documenti.

Classificazione dei requisiti Non-Funzionali

- Product Requirements: requisiti del prodotto come velocità di esecuzione ecc.
- Organisational Requirements: requisiti che sono una conseguenza delle politiche di organizzazione e procedure come quali standard sono usati ecc.
- External Requirements: nati da fattori esterni come requisiti legislativi ecc.

I requisiti non funzionali possono essere molto difficili da fare e quelli imprecisi non aiutano nello sviluppo e nella verifica.

l'obiettivo: un intenzione generale dell'utente come la facilità di utilizzo.

Per verificarli si usano delle misure con cui si possono testare i requisiti.

Esempio pag 27 L2

Metriche: velocità, size, ease to use, affidabilità, robustezza, portabilità.

Il documento dei requisiti sw è il rapporto ufficiale di quello che è richiesto dagli sviluppatori. Contiene i requisiti. Gli utenti che producono questi documenti sono ingegneri, manager, clienti, tester

La struttura di un buon documento è composto da: Preface, introduction, glossary, user requirements definition, system architecture, system requirements specification, system models, system evolution, appendice, indice.

È importante che sia completo quanto più possibile.

I requisiti potrebbero essere parte di un contratto scritti in linguaggio naturale. si potrebbe usare anche un approccio di scrittura più strutturato oppure tabellare.

I requisiti dovrebbero descrivere cosa il sistema fa, ed il design come fa a realizzarli. In pratica i due concetti sono legati. Un'architettura può essere progettata per strutturare i requisiti oppure usare una specifica architettura per soddisfare certi requisiti.

Linee guida per scrivere requisiti: inventare un formato standard da usare per tutti, linguaggio in maniera consistente, evidenziare punti chiave, evitare l'uso del gergo informatico, includere una spiegazione del perché è necessario quel requisito.

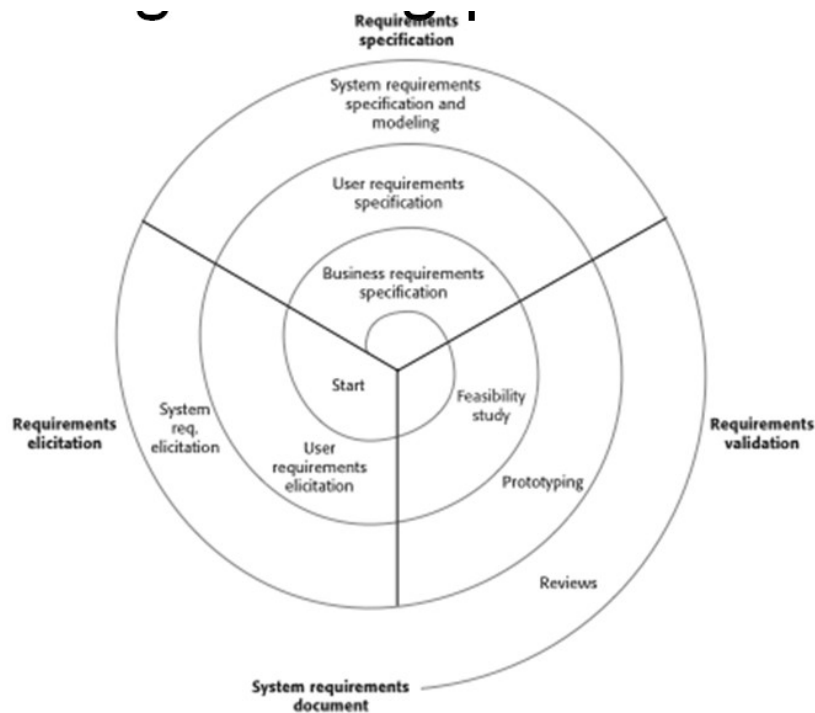
Requirements engineering processes

Alcune attività sono comuni a tutti i processi:

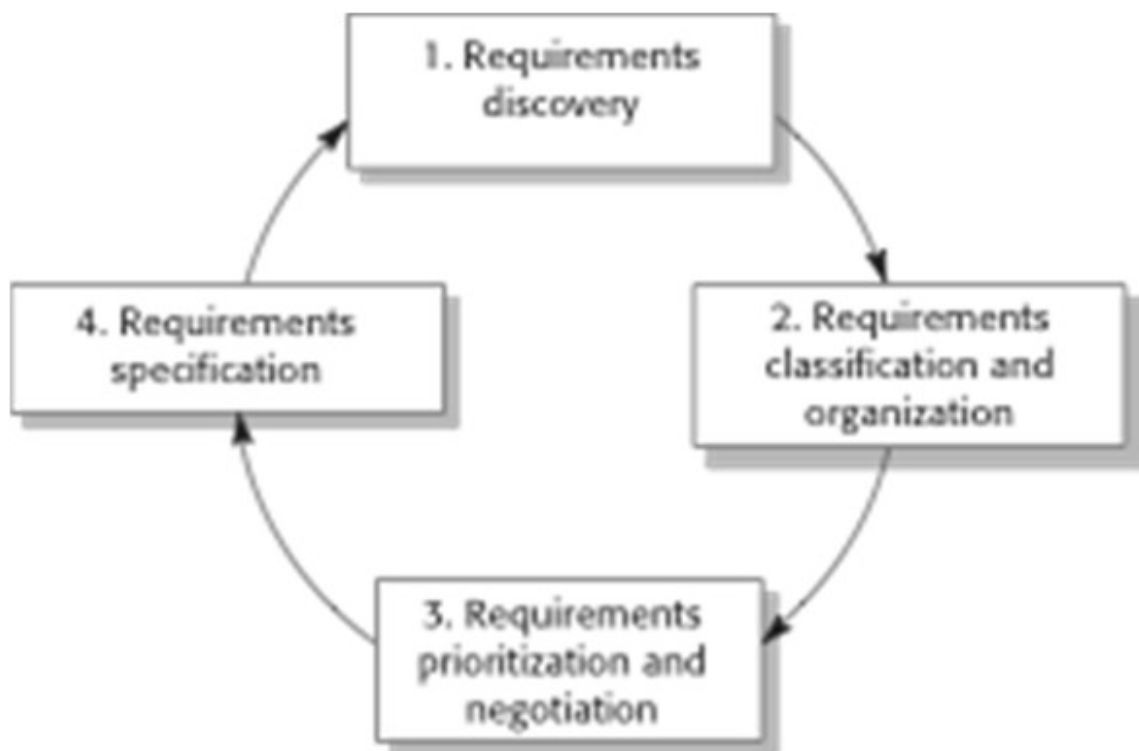
1. Requirements elicitation
2. Requirements analysis
3. Requirements validation
4. Requirements management

RE è una attività iterativa nella quale si susseguono queste fasi in maniera non contigua.

Vista a spirale del processo RE



Le fasi 1 e 2 vengono fatte con i seguenti stage



Nella fase di discovery bisogna parlare con i clienti, fare interviste o fare scenari. Si possono poi usare use-cases, sequence chart ecc.

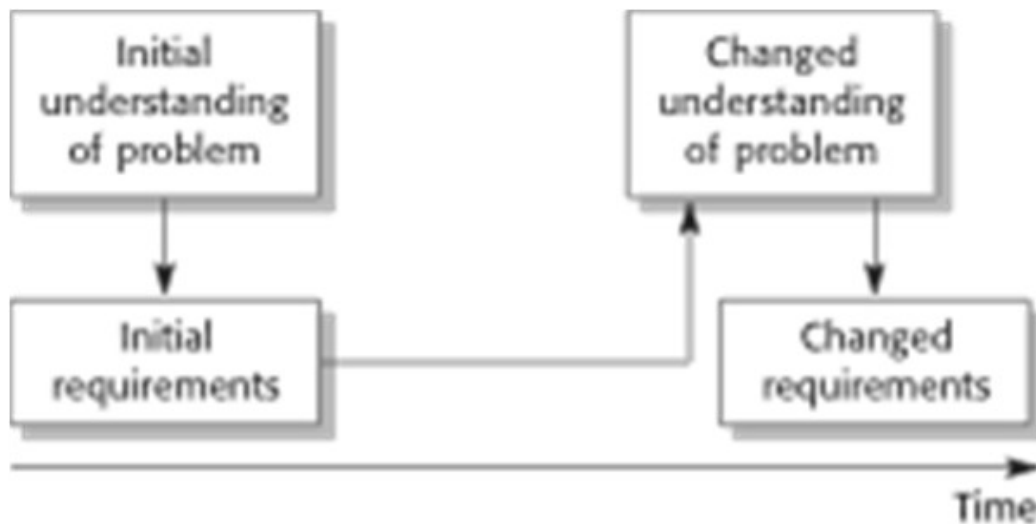
Fase 3: errori nei requisiti portano ad un costo di risoluzione molto elevato. Fare una validazione a questo punto è molto importante. Validità (Il sistema fornisce le funzioni che meglio supportano le esigenze del cliente?), consistenza (c'è qualche conflitto tra i requisiti?), completezza (ci sono tutte le funzioni che il cliente ha chiesto?), realismo (si possono implementare con la tecnologia attuale e costi?), verificabilità (si possono testare?).

Si usano revisioni manuali, con prototipi o casi di test.

Fase 4: è il processo di gestione dei cambiamenti dei requisiti durante lo sviluppo. Nuovi requisiti possono emergere via via. È necessario tenere traccia delle singole esigenze e di mantenere i collegamenti tra i requisiti che dipendono in modo da poter valutare l'impatto dei cambiamenti dei requisiti. È necessario stabilire un processo formale per fare proposte di cambiamento e collegarle al sistema dei requisiti.

C'è necessità di introdurre nuovo hw, nuove funzionalità ecc. bisogna controllare che tutto vada bene.

Evoluzione dei requisiti:



Requirement management planning: bisogna stabilire il livello di dettaglio del requirements management richiesto.

Requirements management decisions:

- Requirements identification: ogni requisito deve essere univocamente identificato.
- A change management process: attività che stimano l'impatto dei costi di cambiamento.
- Politiche di tracciabilità: relazioni tra ogni requisito e tra requisiti e design del sistema.
- Tool support.

Dobbiamo decidere se il cambiamento del requisito può essere accettato, superando le seguenti fasi:



3 Tipologie di ciclo di vita

Che cosa è: Caratterizzazione descrittiva o prescrittiva di come un sistema software dovrebbe essere sviluppato. Linee guida per organizzare, pianificare, dimensionare personale, assegnare budget, schedulare attività.

3.1 Code&Fix

Attività:

- 1.1. Costruire una prima versione.
- 1.2. Ciclo: Modifica fino a che il cliente `e soddisfatto.
- 1.3. Prodotto operativo (Per la manutenzione tornare al punto 2).
- 1.4. Ritiro del prodotto.

Conseguenza:

- Attività non organizzata.
- Produzione caotica, non tracciabile.
- Aumento della complessità esponenziale.
- Qualità del prodotto scarsa.

3.2 Waterfall

Modello sequenziale dove ogni fase deve essere completata prima dell'inizio della successiva. Ogni fase comprende attività omogenee e per passare alla fase successiva la fase corrente deve essere completata e deve aver prodotto dei documenti. La fine di ogni fase è detta milestone.

Fasi:

- 2.1. **Analisi dei requisiti:** Analisi dei requisiti del SW produzione del documento SRS. Documentazione prodotta:

- Definizione del problema, proposta ed obiettivi.
- SRS.
- Budget per il progetto.
- Piano di massima.

- Studio di fattibilità`.
- Piano di testing.

2.2. Progettazione: Partendo dal documento SRS si progetta la struttura del SW tramite diagrammi delle classi (UML), specifica di alcuni dettagli implementativi e design architetturale.

2.3. Development: Realizzazione (coding) del prodotto SW, validazione e testing dello stesso. Documento di testing con scenari e risultati.

2.4. Manutenzione: Correzione dei bug, modifiche, estensioni, adattamenti. Documentazione aggiornata, version control.

Vantaggi e vantaggi:

Pro:

- Terminata una fase si hanno documentazione e linee guida per la fase successiva.
- Visione a compartimenti.
- Concentrazione dei tipi di azioni.

Contro:

- Non è possibile tornare alla fase precedente.
- Completata una fase un cambiamento rilevante comporta il ritorno alla fase iniziale di analisi dei requisiti.
- Installazione possibile solo quando SW completato.
- Fasi rigidamente separate.

3.3 Modelli Do it twice

Consiste nella prima implementazione di un sistema rozzo (di prova), utile per la stesura dei **requisiti** e la verifica di fattibilità`. Dopo di che si utilizza il modello Waterfall per costruire la seconda versione (definitiva).

Modello con pilota:

- Modello completo ma "rough to the edges".
- Esplorativo.
- Non orientato alla produzione (non ottimizzato).

Modello con prototipo:

- Costruito in maniera molto veloce.

- Indicazioni sulle migliorie introducibili e sulle criticità..
- Supporto alla definizione dettagliata dei requisiti utente.

- Da eliminare una volta ultimata la progettazione. Esistono due tipi di prototipo:

Mock-ups: Realizzazione completa delle interfacce senza dare peso alle funzionalità, molto utile per l'analisi dei requisiti utente. Si può subito valutare quanto sia produttivo e utile un sw.

Bread-boards: Implementazione dell'insieme delle sotto funzioni critiche del sistema senza le interfacce.

Problemi:

- Generatori di applicazioni.
- Vaste librerie di componenti.
- Ambienti avanzati di sviluppo.
- Spesso il prototipo diventa il prodotto finito.

3.4 Modelli iterativi

Sono modelli che rispondono bene ai cambiamenti in corso di progettazione, sia di requisiti che tecnologici. Soluzione generale:

- Decomporre la realizzazione del sistema
- Differire la realizzazione delle componenti critiche
- Le iterazioni sono pianificate

3.5 Modelli evolutivi

Il software deve evolvere, superare un limite dei modelli precedenti e permette di produrre presto qualcosa di utile.

Fase di EVOLUZIONE:

- Analisi dell'esperienza ed utilizzazione della maggiore conoscenza per definire nuovi obiettivi
- Nuove esigenze e nuove funzionalità emerse o non coperte in precedenza

Fase di Riciclo:

Sulla base dei risultati della fase di Evoluzione, si riprende il ciclo dalla definizione dei requisiti alla messa in esercizio e manutenzione del nuovo sistema software nato dall'arricchimento e dalla evoluzione del precedente.

3.6 Modello ad implementazione incrementale

Accoglie i principi del modello evolutivo e realizza un sw con modello incrementale.

- Le fasi ad alto livello del WM sono tutte fatte,

- Il sistema viene decomposto in sotto sistemi che vengono progettati e realizzati in tempi diversi
- Fase di integrazione dei pezzi di sw

3.7 Modelli a prototipazione evolutiva

1. Produzione del prototipo
2. Trasformazione del prototipo nel sistema finale.

3.8 Modello a spirale

Il modello a spirale `e un modello di ciclo di vita di un sistema, `e un modello di tipo risk-driven applicabile solo a sistemi "large scale ed `e sviluppato per uno sviluppo interno (sviluppatore e organizzatore appartengono alla stessa organizzazione). Fasi del modello a spirale:

- definizione degli obiettivi
- analisi dei rischi
- sviluppo e validazione
- pianificazione: decisione circa il proseguimento, pianificazione del ciclo.

Caratteristiche del modello a spirale:

- **Rapid prototyping:** finalizzato alla minimizzazione dei rischi
- **Risk Analysis:** `e un task in ogni fase di sviluppo
- **Misure, stime, valutazioni e servono modelli per fare ci`o**

Il modello a spirale evidenzia gli aspetti gestionali, evidenzia i ruoli di committente e fornitore ed `e applicabile ai cicli tradizionali

- **Raggio Spirale:** Costi Accumulati.
- **Angolo:** Stato di avanzamento del processo.
- La **manutenzione** `e un altro ciclo della spirale.

3.9 Modello Microsoft: Synchronize-and-stabilise

Lascia spazio all'inventiva degli sviluppatori. Il progetto viene suddiviso in varie build parallele, le build vengono assegnate a un team che le sviluppa e le testa molte volte. La particolarita` di questo modello `e la frequente sincronizzazione che viene fatta tra i team, giornalmente vengono prodotte delle daily build e testate.

1. Analisi dei requisiti
2. Specifiche
3. Builds
4. Team Work

Ciclo a 3 fasi:

1. Planning Phase
2. Development
3. Stabilization: testing, prodotto finale.

3.10 Modello Trasformatzionale

Lo sviluppo del Software viene visto come una sequenza di passi che trasformano formalmente una specifica in una implementazione:

- Primo passo: analisi dei requisiti informali e loro trasformazione in specifiche formali
- Ciclo: dalle specifiche formali a specifiche formali di più basso livello
- Fino al: codice eseguibile

Ad ogni livello di specifica formale corrisponde un processore astratto in grado di eseguirla

- Trasformazioni Manuali
- Trasformazioni interattive Macchina-Ing. del Software
- Trasformazioni Automatiche

I prodotti sono 3: Specifiche formali, sistema da rilasciare, diario di sviluppo

Contro:

- Ancora in fase di ricerca
- Usato per piccoli sw

3.11 Modello Reuse

Utilizza componenti software già esistenti. Si può fare un riuso organizzato e sistematico.

Problemi:

- Trovare componenti riusabili
- Avere supporti per l'archiviazione e ricerca
- Ambienti di supporto alla interoperabilità

4 Software process

Uno strutturato set di attività per lo sviluppo del sw. Esistono molti tipi di SW process ma tutti utilizzano queste fasi:

- **Specification:** Definire cosa deve fare il sistema.
- **Desing & Implementation:** Definire l'organizzazione del sistema e la sua implementazione.
- **Validation:** Controllo che il prodotto corrisponda alle richieste del cliente.
- **Evolution:** Cambiare il sistema in risposta alle esigenze del cliente.

Plan-driven processes: sono processi dove tutte le attività sono pianificate in maniera rigida. Viene misurato lo stato di avanzamento.

Agile processes: la pianificazione è incrementale ed è più facile cambiare processi che riflettono il cambiamento dei requisiti.

I principali SW process sono:

Waterfall: Plan-driven model, fasi separate rigidamente.

Incremental Dev: Specification, development e validation sono intervallate per maggiore agilità.

Reuse-oriented SW Eng: Assemblare il progetto da componenti già esistenti.

4.1 Modelli sequenziali

Hanno una struttura molto rigida e fasi che si susseguono in modo sequenziale.

Waterfall

Arriviti a:

- 1.1. Analisi e definizione dei requisiti.
- 1.2. System e SW design.
- 1.3. Implementazione e unit test.
- 1.4. Integration test e system test.
- 1.5. Messa in opera e manutenzione.

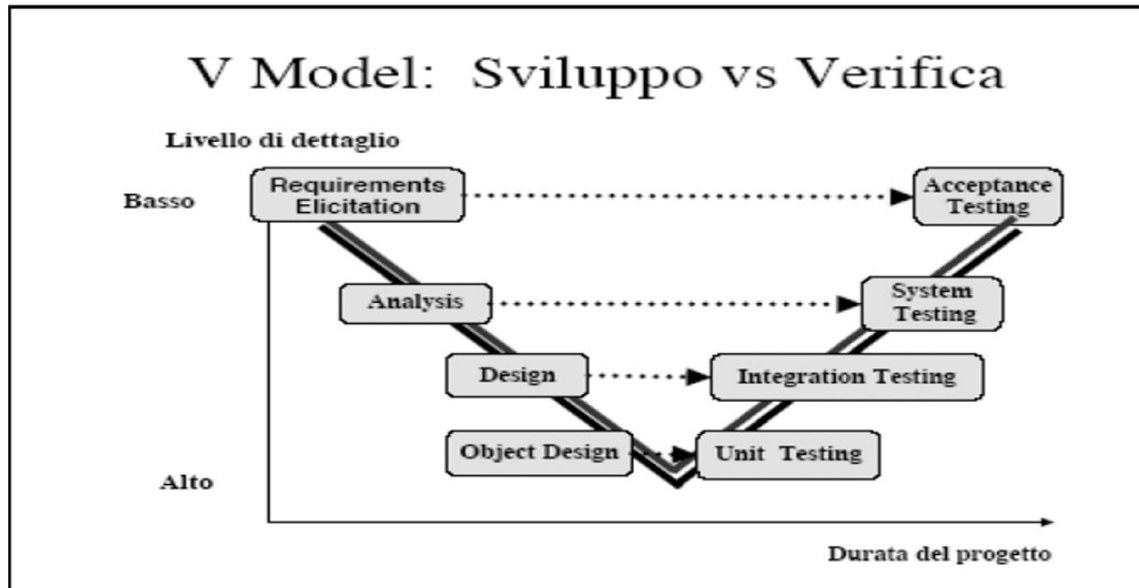
Pro:

- In sistemi molto grandi e sviluppati in sedi diverse la struttura plan-driven aiuta a coordinare il lavoro.
- Appropriato quando i requisiti sono ben conosciuti ed i cambiamenti saranno limitati durante la progettazione e sviluppo.

Contro:

- La struttura rigida implica una scarsa flessibilità al cambiamento.
- Pochi sistemi hanno requisiti fissi.
- Soggetto a rischi.

V Model



Questo modello `e diviso in due parti la parte discendente che passa da un livello di dettaglio basso ad uno alto nella quale si procede alla progettazione e realizzazione del prodotto ed `e formata da:

1. Raccolta dei requisiti.
2. Analisi dei requisiti.
3. Design.
4. Object design.



Mentre la seconda parte `e ascendente da un livello di dettaglio alto ad uno basso si effettua il test:

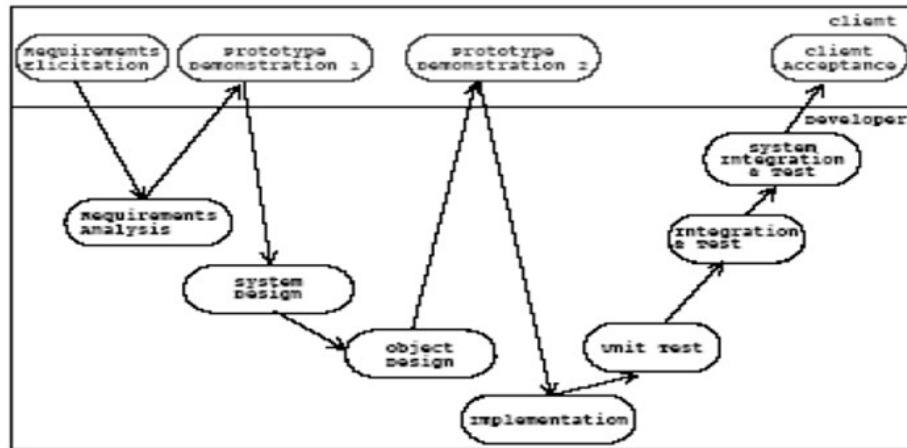
1. Unit testing
2. Integration testing.
3. System testing.
4. Acceptance testing.

Ogni fase del primo gruppo ha la sua corrispondente nel secondo, di fatto si effettua una costruzione in un senso (alto → basso) e si valida il sistema nel senso opposto (basso → alto).




Altri modelli sequenziali

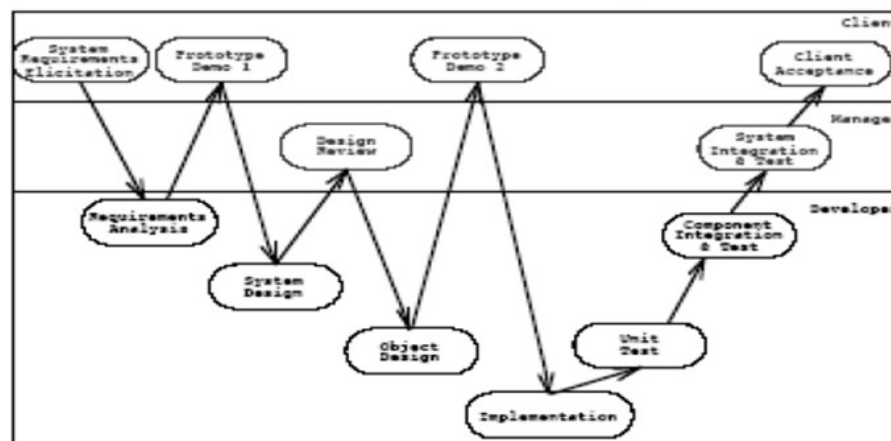
Sawtooth Model

 Client's Understanding
 Developer's Understanding

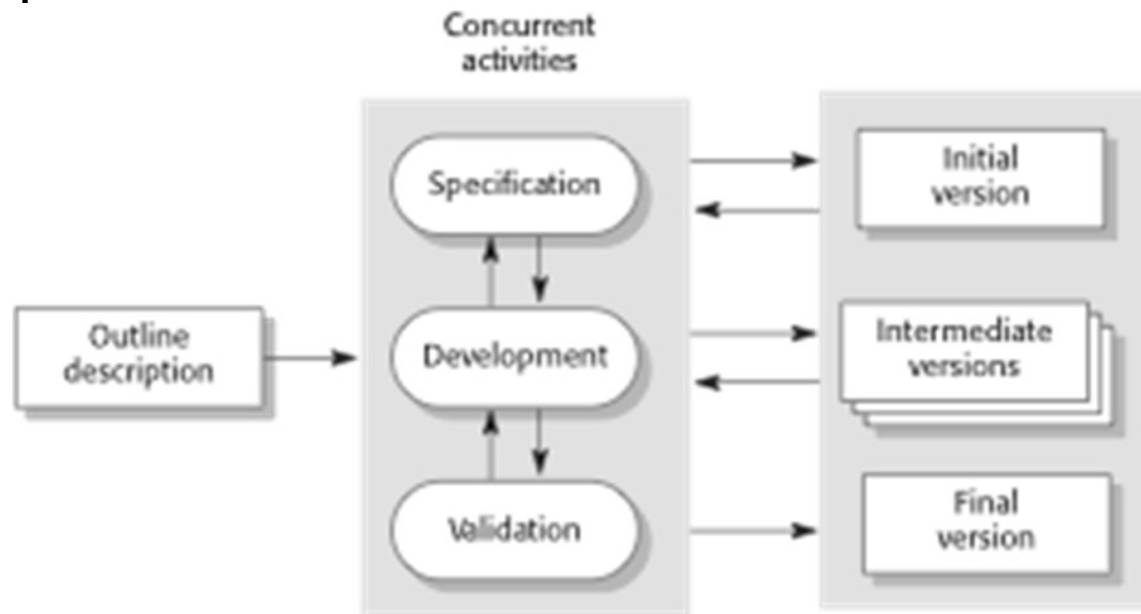


Sharktooth Model

 User's Understanding
 Manager's Understanding
 Developer's Understanding



4.2 Modello incrementale



Benefici:

- Partizionamento dei requisiti per priorit  e criticita .
- Flessibilit  al cambiamento, con conseguente riduzione dei costi per modifiche ed adattamenti.
- Un piu  facile e veloce feedback tra cliente e fornitore sul SW prodotto.
- Installazione pi  rapida.

Problemi:

- Il processo non   visibile.
- Il manager ha bisogno di documentazione regolare per controllare lo stato di avanzamento del processo.
- I cambiamenti tendono a peggiorare l'efficienza del SW ed   necessario tempo per la ottimizzazione del codice.
- L'integrazione dei cambiamenti diventa sempre piu  difficile e complessa con l'aumentare degli stessi, e porta ad un maggior costo.

4.3 Modello Reuse-oriented

È un modello che si basa sul forte riutilizzo di componenti già pronti in casa o soluzioni SW commerciali di terze parti, il processo è articolato nelle seguenti parti:

- 3.1. Specifica dei requisiti.
- 3.2. Analisi dei componenti.
- 3.3. Modifica dei requisiti.
- 3.4. Design del sistema per il riutilizzo.
- 3.5. Sviluppo ed integrazione.
- 3.6. Validazione.

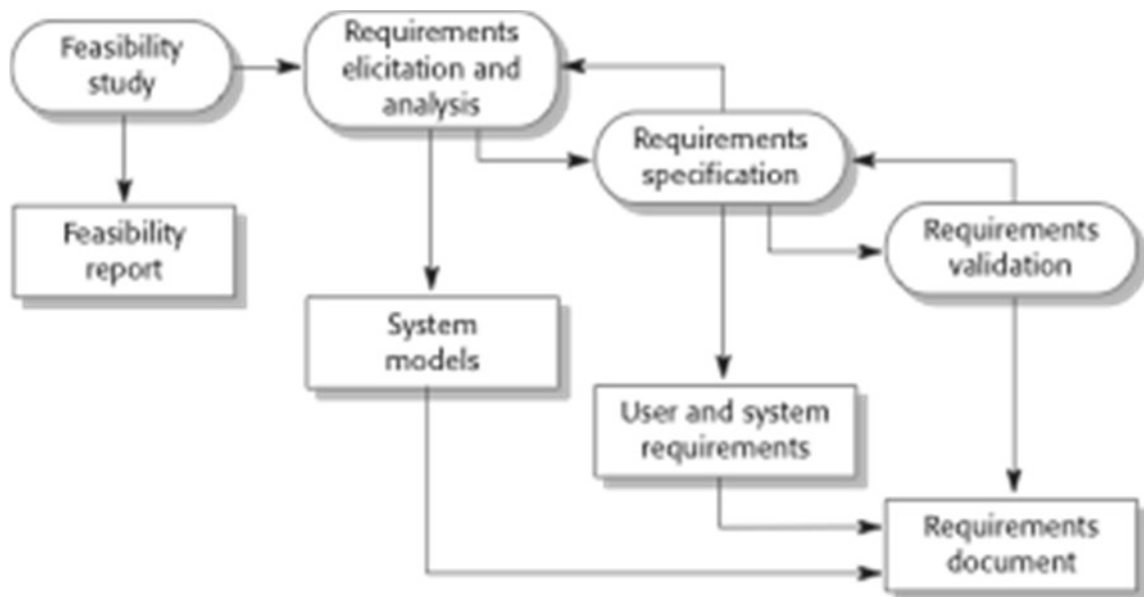
Sistemi Web che si basano fortemente su questo tipo di processo sono tutti servizi in modo da aderire a standard comuni e perché è possibile l'invocazione remota.

4.4 Definizioni e processi

SW Specification: Questa parte provvede a stabilire quali servizi sono richiesti e quali vincoli debbano essere rispettati. Utilizza il:

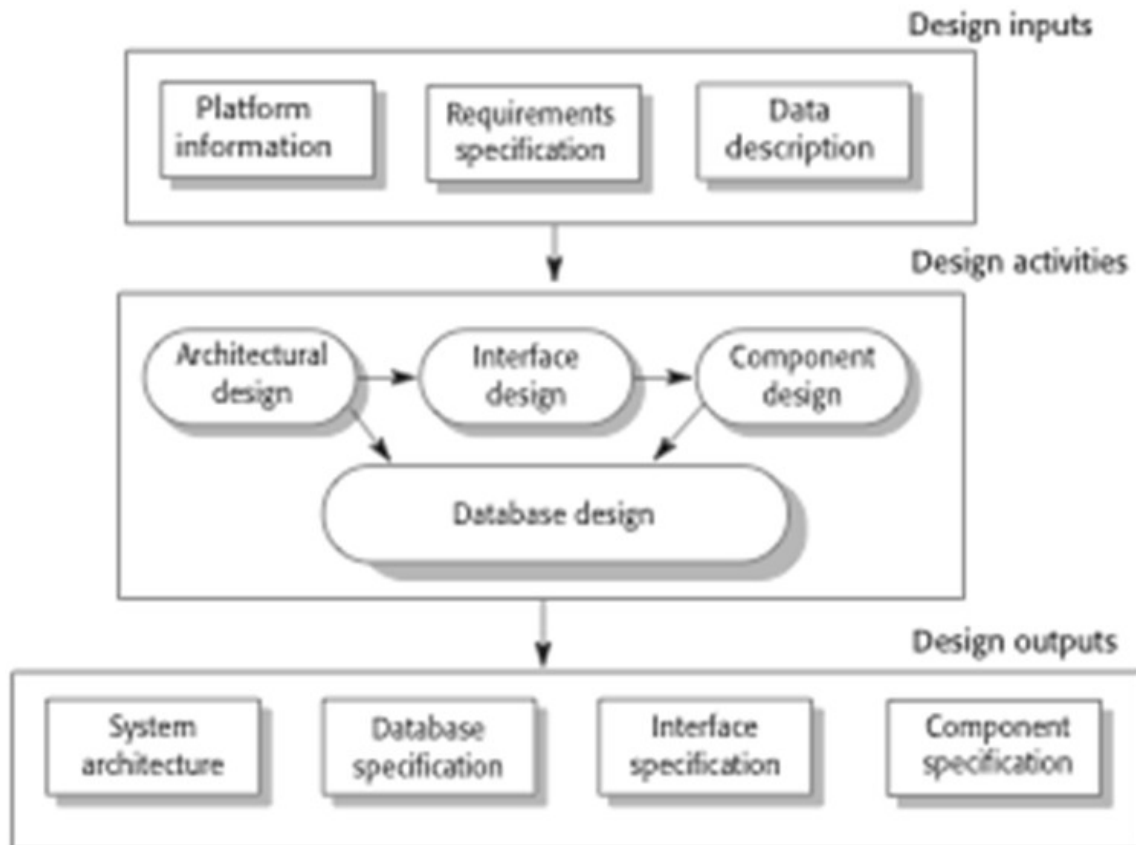
Requirement eng. process:

1. Studio di fattibilità.
2. Raccolta ed analisi dei requisiti.
3. Specifica dei requisiti.
4. Validazione dei requisiti.



Design: Design della struttura software che realizza le specifiche.

- Architetturale.
- Interfacce.
- Component.
- Database.



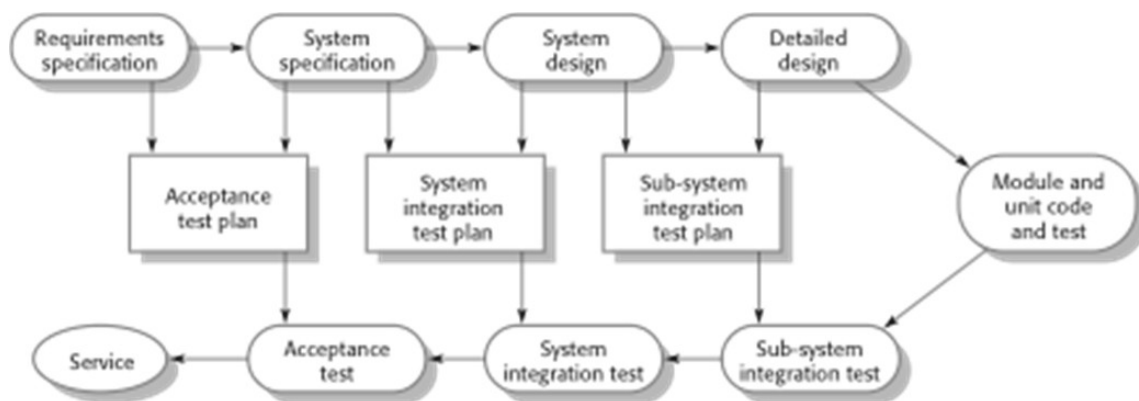
Implementation: Traduzione del design in un programma eseguibile.

Software Validation: Fase che dimostri che i requisiti e i vincoli specificati nel SRS sono rispettati. Validation & Verification (V&V). Si fa uso del test.

Testing:

1. Unit/Component testing: parti testate individualmente.
2. System testing: test sull'intero sistema.
3. Acceptance testing: si testa il sistema con i dati del cliente per vedere se sono soddisfatti i requisiti.

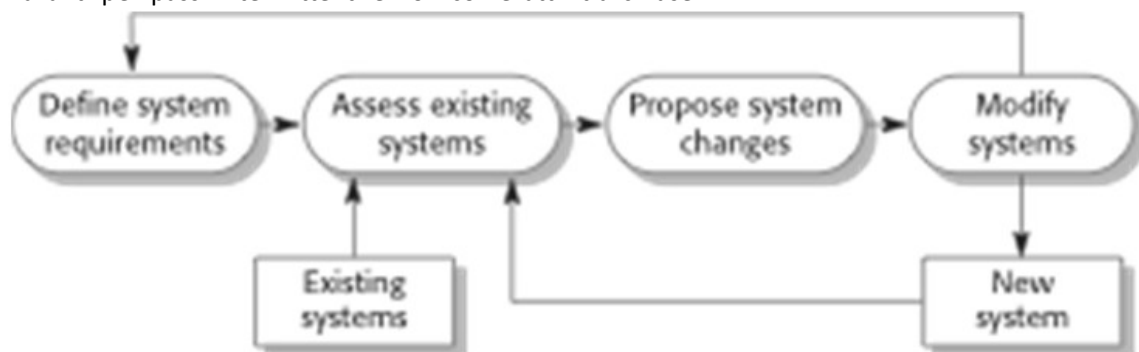




4.5 Software Evolution

Modifica dei requisiti, adattamento ad nuove tecnologie, migliorie e correzione errori. Il sw è flessibile e può cambiare.

Le azioni di **design** ed **implementazione** sono strettamente correlate e possono essere portate avanti per passi intermittenti e non come attività a chiuse.



4.6 Cambiamento

I cambiamenti possono essere frequenti e bisogna evitare di perdere tempo e soldi lavorando su parti già fatte per applicare i cambiamenti.

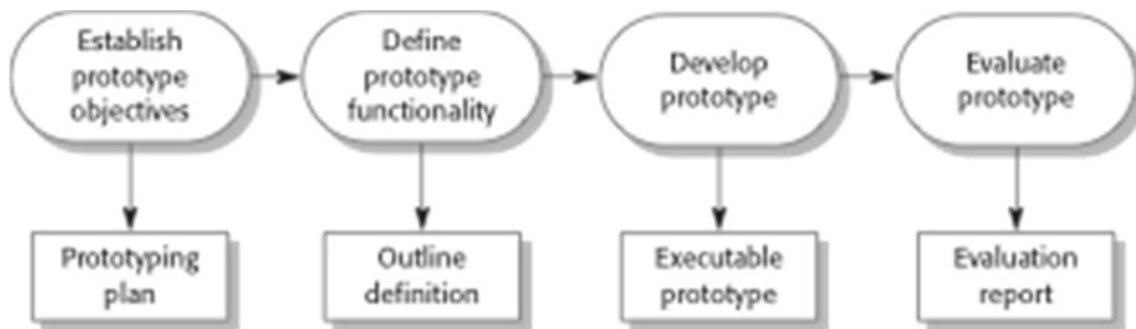
Bisogna cercare di includere attività che possono anticipare i cambiamenti (come l'uso di prototipi), oppure usare processi tolleranti al cambiamento per limitare i costi.

4.7 Software prototyping

Il prototipo è una versione iniziale del sistema per mostrare e testare i requisiti o parti del sw critiche.

Benefici:

- Aumenta l'usabilità
- Molto vicino ai bisogni reali dell'utente
- Aumenta la qualità del design
- Aumenta la manutenibilità



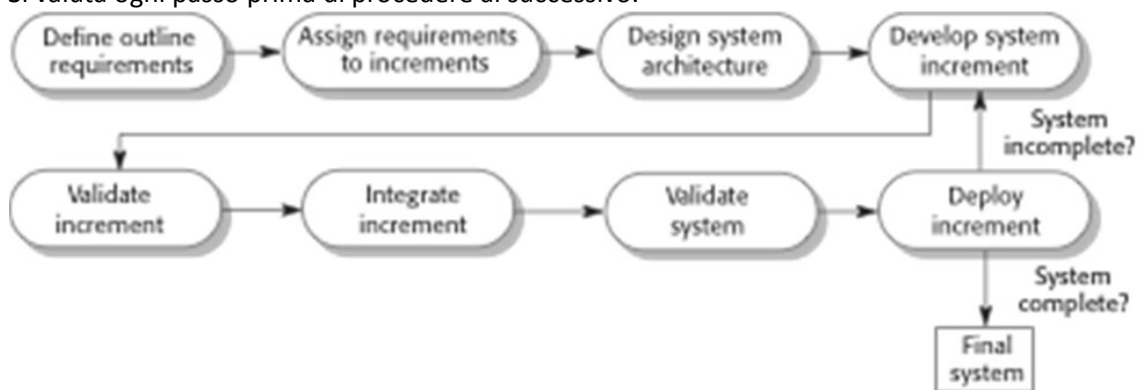
I prototipi dovrebbero essere buttati dopo la fase iniziale, non usarli come punto di partenza per il prodotto finito.

- Non hanno documentazione
- Non rispetta standard di qualità
- È difficile implementare requisiti non funzionali

4.8 Incremental development e delivery

Lo sviluppo è diviso in passi incrementali, ogni passo soddisfa un requisito funzionale. I requisiti ad alta priorità sono fatti nei passi iniziali. Una volta che lo sviluppo è di un passo è iniziato i requisiti sono congelati, anche se i requisiti dei passi successivi possono evolvere.

Si valuta ogni passo prima di procedere al successivo.



Vantaggi:

- Sw distribuito più velocemente
- Minor rischio di fallimento del progetto complessivo.
- Le priorità più alte sono testate prima e molte volte.

Svantaggi:

- I sistemi hanno delle caratteristiche che possono essere servite da più parti differenti del sistema
- La specifica è fatta congiuntamente con il sw

4.9 Modello a spirale

Questo modello è **risk-oriented**, utilizzabile per progetti di **grosse dimensioni** e per lo **sviluppo interno** cioè quando cliente e progettista sono la stessa organizzazione. La **risk analysis** è un task in ogni parte di progettazione.

9.1. Definizione degli obiettivi.

- Requisiti.
- Rischi.
- Piano di gestione.

9.2. Analisi dei rischi.

- Studio di alternative.
- Valutazione dei costi.

9.3. Sviluppo e validazione.

- Realizzazione del prodotto.
- Validazione e testing.

9.4. Pianificazione del ciclo e del proseguimento.

Caratteristiche:

Raggio della spirale: È il costo cumulato fino a quel momento dal progetto in tutte le varie fasi.

Angolo: È lo stato di avanzamento del progetto in quel ciclo.

Aspetti gestionali

- Pianificazione delle fasi.
- Analisi dei rischi.

I Ruoli Committente:

- Definizione degli obiettivi.
- Pianificazione.
- Analisi dei rischi.

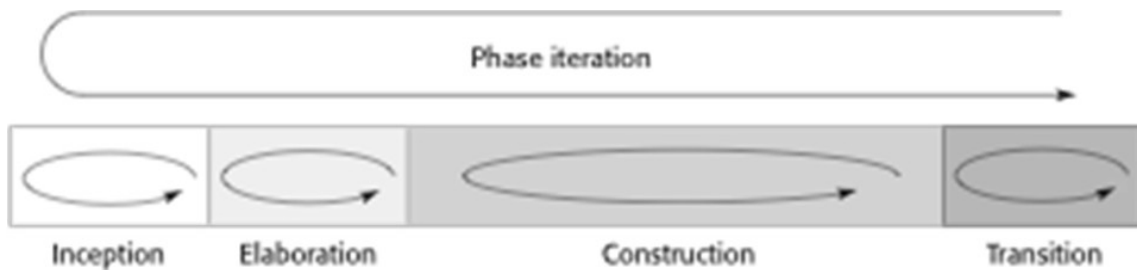
Fornitore:

- Sviluppo.
- Validazione.
- Analisi dei rischi.

Figura 1: Modello a Spirale

4.10 The Rational Unified Process (RUP)

È un moderno modello di progettazione che si ispira ai tre grandi modelli: **Waterfall**, **Incrementale/Evolutivo**, **Spirale**.



Le cui fasi sono:

- 1 **Inizio:** Stabilisce il caso da sviluppare per il sistema.
- 2 **Elaborazione:** Sviluppo del problema e dell'architettura del sistema.
- 3 **Costruzione:** System design, implementazione e testing.
- 4 **Transizione:** Installazione del sistema nel ambiente operativo.

Le interazioni in RUP sono di due tipi, **in-phase** e **cross-phase**. Nella prima ogni fase è iterativa e ad ogni passo si incrementa il risultato, la seconda consiste nel ripartire il ciclo dalla prima fase incrementando il passo.

Buone pratiche:

- Sviluppare il sw iterativamente.
- Gestire i requisiti.
- Usare architetture basate su componenti per incentivare il riuso.
- Usare UML
- Verificare la qualità del sw
- Controllare i cambiamenti del sw.

I punti chiave:

- Sviluppo del SW iterativo.
- Pianificazione degli incrementi in base alle richieste del cliente su base prioritaria.
- Uso di una architettura component-based.
- Organizzazione dell'architettura SW in modo che si riusabile.
- Utilizzo di linguaggi visuali per la progettazione/programmazione (UML)
- Facilit'a nella verifica della qualita` del SW
- Gestione delle modifiche.
- Possibilit'a di sviluppo di prototipi per l'analisi dei requisiti o delle criticita`.

5 Project management & Project Planning

5.1 Cost estimate

L'unità di misura di un progetto software è uomo/mese. I fattori che influenzano il costo sono: fattori umani, dimensione del sw, stabilità dei requisiti, complessità del programma

Ci sono due tecniche per la stima dei costi, una basta sulla **esperienza** l'altra attraverso degli **algoritmi** di **cost modeling**. La prima `e guidata dalle passate esperienze di progetti gi'a realizzati, puo' essere utilizzata per un analisi sommaria ma non ´e molto precisa, vengono valutati i moduli SW e la documentazione da produrre per valutare il costo.

Algorithmic cost modeling

Il costo `e calcolato da una funzione matematica che tiene conto dei seguenti fattori, stimata dal project manager:

$$\text{Effort} = A \times P_s \times C$$

- A: costante sull'organizzazione dei dipendenti
- S: riflette lo sforzo sproporzionato per i grandi progetti
- C: multiplo che indica attributi del processo, delle persone e del prodotto

- P: grandezza del codice.

Una metrica molto affidabile `e la valutazione delle **Line of Code** (LoC) del prodotto finito a partire dall'analisi delle funzionalita`, **Functional Point** (FP), previste e dal tipo di linguaggio consigliato (tipicamente quello con l'indice LoC per FP minore). Standard di riferimento ISO/IEC 20926:2003.

- Productivity $P = LoC / M$ (M totale di uomo/mesi)
- Quality $Q = E / LoC$ (E errori)
- Cost per instruction $C = \$ / LoC$
- Documentation level $D = PD / LoC$ (PD numero totale di pagine dei documenti prodotti)

Functional Point

È un indice indiretto della dimensione del sw derivato sulla base delle funzionalità del sw. Gli FP sono calcolati a partire da 5 indicatori:

FN: Functional Name.

EI: External Input.

Sono quelle funzioni che permettono all'utente il mantenimento dei file (*ILFs*), la loro aggiunta, modifica e cancellazione.

EO: External Output.

Sono quelle funzioni che permettono all'utente di generare output dal sistema.

ILF: Internal Logical Files.

Sono quelle funzioni che permettono all'utente di utilizzare i dati per i quali `e responsabili per quanto riguarda il mantenimento.

EIF: External Interface Files.

Sono quelle funzioni con le quali il sistema rende disponibili all'utente alcuni dati per i quali non `e responsabile per quanto riguarda il mantenimento.

EQ: External Inquiries.

Sono quelle funzioni che permettono all'utente di poter selezionare specifiche informazioni dai file del sistema.

Da valutare per ogni funzione identificata nel nostro progetto, i valori di questi indicatori possono essere **simple** (S), **average** (A) oppure **complex** (C) a seconda della complessita` della funzione. Il totale viene calcolando facendo la somma pesata di ogni valore nella tabella, dopo di che si effettua un aggiustamento seconda dei gradi di influenza di alcuni parametri tipo:

- Performance.
- Coesione del team.
- Esperienza nel tipo di progetto.

- Riutilizzabilità del codice.
- Operabilità.
- Tipi di comunicazioni.

Possible values of the adjustment factors

F_i	Degree of influence
0	None
1	Insignificant
2	Moderate
2.5	Intermediate
3	Average
4	Significant
5	Strong

17

È possibile dare una valutazione dell'eventuale numero di LoC di un dato programma dati i suoi functional point.

5.2 CoCoMo: Constructive Cost Model

Questo metodo può calcolare la quantità di mesi/uomo necessari ad un progetto, il tempo necessario al completamento e il costo necessario. È basato sull'analisi statistica di 63 progetti reali, inoltre vengono sempre aggiornati i coefficienti di costo. (Dal 1997 c'è il CoCoMo II). È possibile effettuare oltre che un'analisi dei costi, un'analisi di riduzione dei costi.

$$M = C \cdot P^S \cdot A$$

- **M:** Costo stimato.
- **C:** Fattore di complessità del progetto.
- **P:** Stima della dimensione del progetto.

- **S:** Fattore per tenere conto della natura non lineare del rapporto tra dimensione del progetto e costo (per grossi progetti).
- **A:** Altri attributi.

Le tipologie di progetto sono:

1. Simple
2. Moderate
3. Embedded

Queste tipologie sono in ordine crescente di complessità e hanno triplette **C, S, A**. Le curve hanno andature che tendono ad essere fortemente non lineari al crescere delle LoC e al cambio della tipologia.

Cocomo ha una gerarchia di modelli che cresce all'aumentare della complessità: modello base, intermedio, avanzato.

Produce stime con scostamenti inferiori al 20% nei seguenti casi:

- **Base:** 25% dei casi.
- **Medio:** 68% dei casi.
- **Avanzato:** 70% dei casi.

Dimensionamento dello staff

Si può utilizzare il modello di Putman o il Cocomo

Putnam ha ipotizzato che la curva di Rayleigh-Norden potesse descrivere in modo appropriato l'andamento del numero di risorse necessarie (numero di persone) nel corso del progetto (ipotesi verificata in circa 200 progetti reali).

- t_d = mese in cui il lavoro ha il suo picco.
- K = Quantità di lavoro richiesta in mesi-uomo.

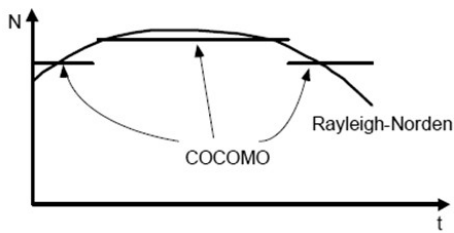
$$M = \frac{K}{t_d^2} t e^{-t^2/2 t_d^2}$$

Il COCOMO è in grado di valutare i valori di M e T per ogni fase del progetto: da ciò è semplice ricavare il valore N del numero di persone richiesto in ogni fase.

$$- N = M / T$$

Nel calcolo dei mesi/uomo e del tempo totale CoCoMo approssima, con una funzione costante a tratti, molto bene la curva di **Putman** che è risultata accurata per più di 200 progetti reali.

Il COCOMO approssima la curva di Putnam per intervalli discreti.



Esponente

L'esponente dipende da 5 fattori. La loro sum/100 è aggiunta a 1.01. Tiene conto di caratteristiche del progetto come:

- Precedenti progetti.
- Flessibilità di progettazione e sviluppo.
- Rischi e soluzioni.
- Coesione del team.
- Maturità del processo.

I fattori moltiplicativi, invece, aspetti più tecnici come:

- Capacità del team.
- Esperienza con la piattaforma di progettazione e sviluppo.
- Requisiti non funzionali.

Moltiplicatori

- Attributi di prodotto
- Attributi di computer
- Attributi del personale
- Attributi di progetto

Riflettono la capacità degli sviluppatori, dei requisiti non funzionali, ecc.

Cost drivers

Cocomo ha un ampio set di cost drivers per rifinire la stima.

Reliability, complexity, memory constraints, tool, schedule. Si aggiunge la stima uomo/mese.

Ridurre il costo

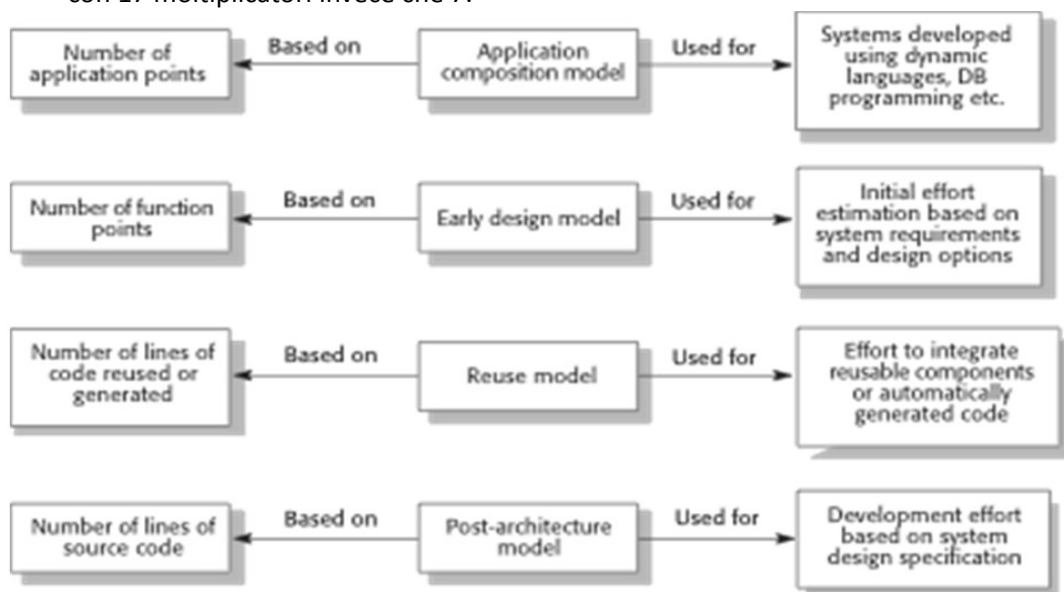
Dall'analisi del cocom il manager agisce sui parametri per diminuire il costo. Ridurre il costo del personale non sempre porta a costi più bassi, non si utilizzano programmatori senior. Usare hw più economico. Se si hanno problemi ancora, si può ridurre la dimensione del progetto e ridurre

l'affidabilità (si aumenta però la manutenzione). Un'altra soluzione efficace è aumentare la qualità degli sviluppatori, se si riesce ad assumere questa tipologia di persone.

5.3 CoCoMo2

Modello empirico basato su esperienze di progetto. Incorpora sotto modelli per la stima del costo che sono:

- Application composition model. Used when software is composed from existing parts.
- Early design model. Used when requirements are available but design has not yet started. Based on the standard formula: $A = 2.94$ in initial calibration, P in KLOC, S varies from 1.05 to 1.24
- Reuse model. Used to compute the effort of integrating reusable components. Ci sono 2 approcci, black-box (il codice non viene modificato) e white-box.
- Post-architecture model. Used once the system architecture has been designed and more information about the system is available. Usa la stessa formula del cocomo ma con 17 moltiplicatori invece che 7.



5.4 Project management

Riguarda tutte le attività che assicurano la consegna in tempo del sw, l'organizzazione dello sviluppo, i costi, e che soddisfino i requisiti del cliente.

Attività del Management:

Project planning: Pianificazione ed assegnazione del lavoro, stima dei costi e scheduling dello sviluppo.

Reporting: Comunicazione con clienti e produttori del software.

Risk management: Analisi dei rischi e piano di azione.

People Management: scegliere le persone del team e motivarli.

Proposal Writing: descrive gli obiettivi del progetto.

I compiti di un Project Manager (PM):

- Applicare le regole dell'organizzazione.
- Non prende decisioni tecnologiche o economiche.
- Controlla l'ambiente di lavoro.
- Organizza le persone.
- Gestisce la documentazione.

Risk management process

1. Identificazione: rischi di progetto, business o di prodotto. Si misura quindi il tipo di rischio
2. Analisi delle possibili conseguenze (Probabilità & Conseguenze). Si misurano con un valore di probabilità di avvenimento, e quanto misura il suo effetto.
3. Stesura di un piano di azione per gestire i rischi o evitarli/ridurli (Strategie). Considera ogni rischio e sviluppa una strategia.
4. Monitoraggio dei rischi, tramite indicatori.



Gestire le persone

Fattori:

- Consistenza
- Inclusione
- Rispetto
- Onestà

La motivazione è un fattore psicologico molto complesso.



La motivazione deve tenere conto anche della tipologia delle persone che lavorano: task-oriented, self-oriented, interaction-oriented. La motivazione è un complesso bilanciamento di fattori individuali e fattori esterni e culturali.

Lavorare in team ed avere un team coeso sarà fondamentale per il lavoro.

Il manager deve:

- Motivare persone e team.
- Coesione del team.
- Selezione dei membri di un team, cercando di mettere insieme persone che hanno le stesse motivazioni, ed è molto complesso a causa delle differenti tipologie di persone. Un team deve avere un bilanciamento di tutte queste tipologie.
- Organizzazione del team.
- Comunicazione intra/extra team.

I team possono essere organizzati rigidamente oppure in gruppi informali.

5.5 Project Planning

La pianificazione ha le seguenti fasi:

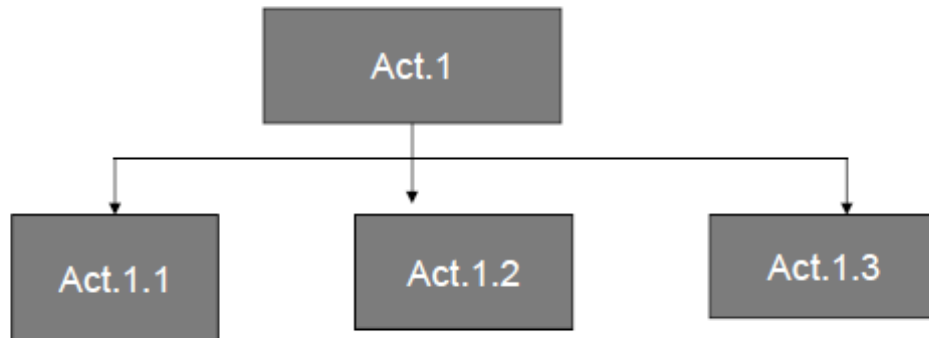
- Definizione degli obiettivi
- Definizione delle strategie e delle politiche di gestione
- Piano per la gestione del rischio
- Definizione del budget
- Definizione delle procedure e regole standard
- Scelte operative (strumenti da usare).

Piano operativo:

- Definizione delle attività: durata, precedenze, tempo, risorse,
- Milestones.
- Tools per far questo: work breakdown structure, Gant, Pert

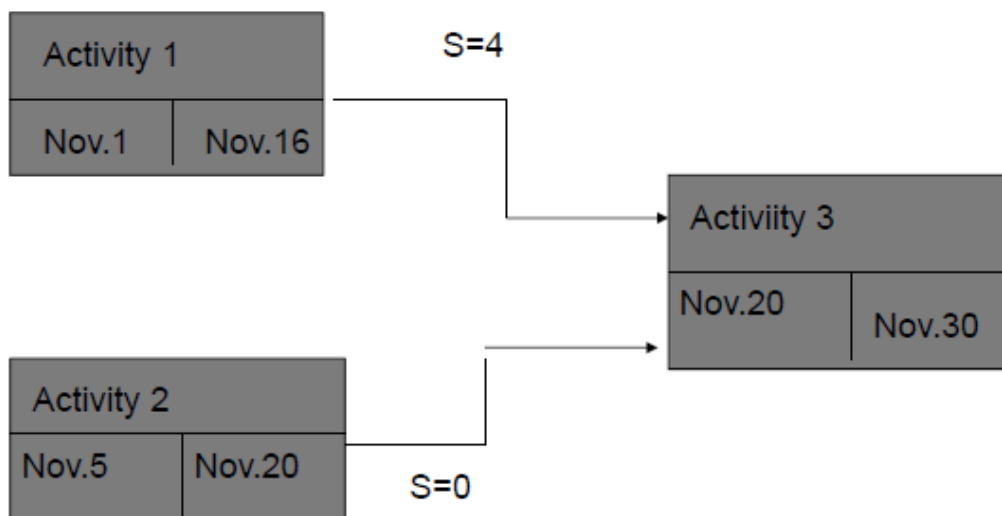
Work Breakdown structure

- Struttura gerarchica delle attività
- Ogni attività è partizionata in sotto attività
- Attività sequenziali o parallele



Diagrammi di Pert

Ci sono dipendenze temporali tra le attività



- Slack time: due differenti attività con durata diversa controllano l'inizio di una terza attività
- Critical path: sequenza di attività con $S=0$
- Free slack: ritardo massimo nel tempo di attivazione di una attività senza causare ritardo ad altre dipendenti da essa.

- Total slack: ritardo massimo nel tempo di attivazione di una attività senza causare ritardo della fine del progetto.

Diagramma di Gant

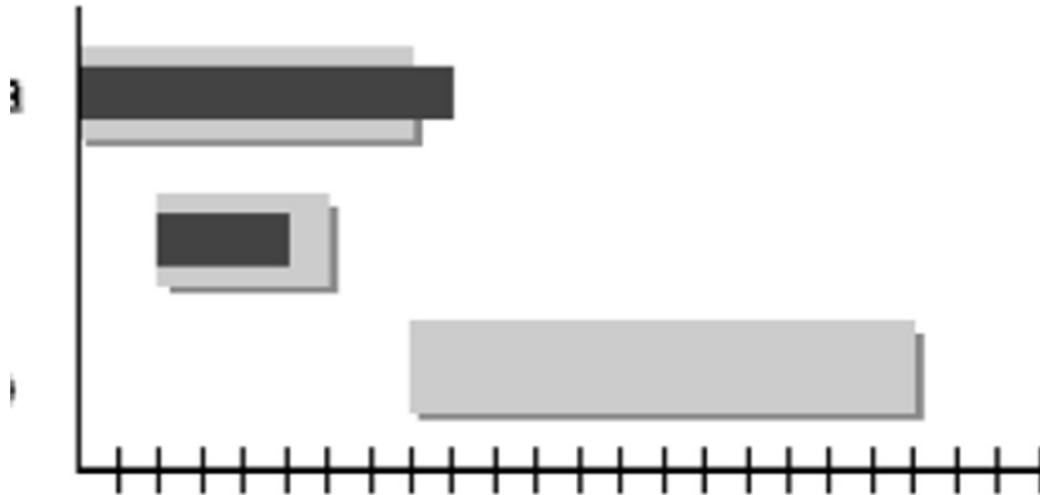


Diagramma per schematizzare l'allocazione di risorse a disponibilità limitata: scheduling dell'orario di lavoro, ossia distribuire il tempo della risorsa (in percentuale, in giorni, totale ecc.). L'allocazione delle attività viene fatta su un asse temporale, con data di inizio e fine. Un'attività può essere inserita come: ASAP, ALAP, Fixed.

Conflitti di risorse: una risorsa è destinata ad attività che richiedono più tempo del suo tempo di lavoro standard. Si deve cercare di porre rimedio, allocando più risorse, mettendo dei ritardi, partizionando le attività.

Organizzazione del lavoro:

- Progetto:
 - Organizzazione di tipo funzionale: niente parallelismi
 - Organizzazione per progetto: utilizzo poco efficiente delle risorse
 - Organizzazione matriciale: ogni persona ha 2 capi.
- Team:
 - Gruppo democratico: capo a rotazione
 - Gruppo con capo programmatore
 - Gruppo a controllo decentralizzato: ogni senior coordina un po' di junior

Gestione del rischio nei progetti:

- Valutazione dei rischi:
 - Identificazione
 - Analisi
 - Valutazione
- Controllo dei rischi:
 - Pianificazione dei controlli
 - Soluzione dei rischi
 - Monitoraggio

Fattori di fallimento: requisiti incompleti, non c'è supporto dal management, requisiti instabili, esclusione dei clienti.

Fattori di successo: staff qualificato e tutto il contrario di quello scritto sopra.

5.6 Amministrazione

Si occupa della gestione del progetto e del prodotto

- Documentazione:
 - Sviluppo
 - Gestione
 - Diffusione dei documenti
- Ambiente di lavoro
 - Qualità dell'ambiente
- Infrastrutture
 - Risorse hw
 - Risorse sw
- Strumenti
 - Ambienti di sistema
 - Tools, ecc.
- Norme di progetto
- Controllo della qualità

6 Quality management

I modelli di gestione della qualità non sono modelli operativi, si propongono come template di definizioni di riferimento ma non rispondono né sul piano modellistica né su quello metodologico alle problematiche di valutazione della qualità dei processi produttivi reali.

Attività:

- Provvede ad un controllo indipendente sullo sviluppo del SW.
- Controlla che i deliverables siano consistenti gli obiettivi e gli standard scelti.

- Il team preposto deve essere indipendente da quello di sviluppo, in modo che possa avere una visione obbiettiva e che non sia influenzato da problemi di sviluppo.

Struttura del piano di qualità

- Introduzione del prodotto.
- Piani di produzione.
- Descrizione del processo.
- Obiettivi di qualità.
- Rischi e gestione dei rischi.

Dovrebbe essere un documento corto da leggere.

Attributi per il sw di qualità: portabilità, riusabilità, complessità, robustezza, sicurezza, affidabilità, adattabilità, usabilità.

Bisogna dare delle priorità perché un sw non potrà essere ottimizzato per tutti gli attributi.

La qualità del prodotto è influenzata dalla qualità del processo di sviluppo.



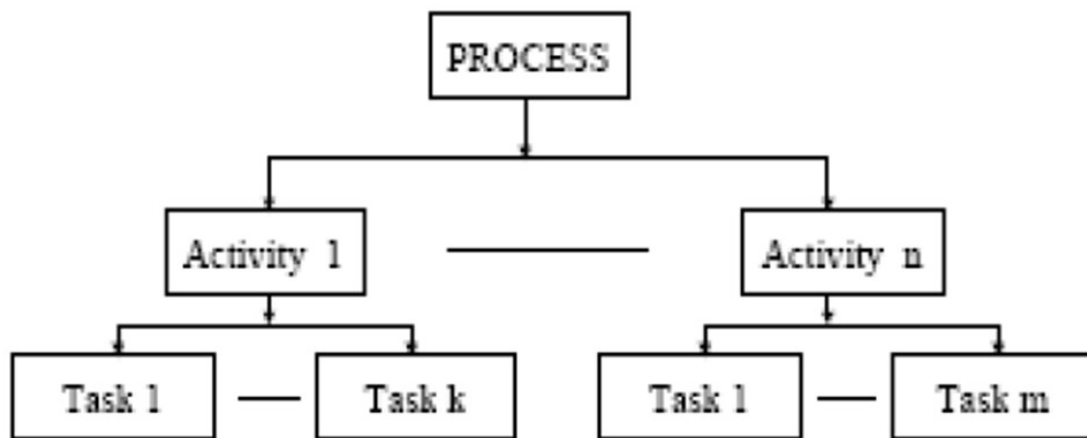
Gli standard sono caratteristiche che un prodotto dovrebbe avere. Hanno molta importanza gli standard perché derivati da anni di raffinamenti e di sviluppo nel campo SW, i più importanti sono **ISO 9001** per lo sviluppo del sw (ultima versione del 2008) e **ISO/IEC 12207** del 1995 per la gestione del ciclo di vita di un prodotto.

6.1 ISO 12207

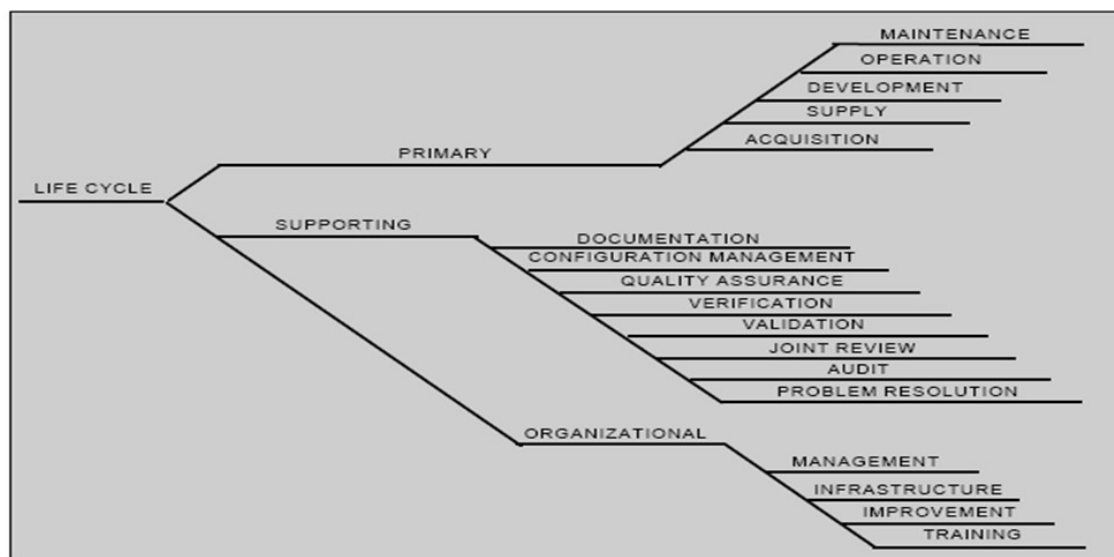
L'architettura è costruita con un set di processi interconnessi tra di loro.

Le caratteristiche dei processi sono la modularità e la responsabilità.

Mentre per la struttura, ogni processo è disegnato con le sue attività che lo costituiscono. Un'attività è un insieme di task coeso. Il task è un'azione atomica.



I processi sono raggruppati secondo questo schema:



Il tailoring process è la cancellazione di tutti quei processi (attività, task) che sono inapplicabili o che non provocano nessun effetto.

Processi primari

- Acquisition: processo che deve essere sviluppato da chi acquisisce uno o più beni o servizi sw. Deve essere coniugato con le leggi vigenti e con le regole aziendali.
 - Inizializzazione
 - Preparazione richiesta di acquisizione
 - Preparazione e update del contratto
 - Monitoraggio del fornitore

- Accettazione e completamento
- Supply: fornitura dei servizi.
 - Inizializzazione
 - Preparazione risposta a proposta di acquisizione
 - Contratto
 - Planning
 - Esecuzione e controllo
 - Revisione e valutazione
 - Rilascio e completamento.
- Development process: analisi requisiti, design architettura, code testing, integrazione.
- Operation process: erogazione di un servizio o di gestione operativa
 - Implementazione del processo
 - Testing operazionali
 - System operation
 - Supporto utente
- Maintenance process: manutenzione del sw
 - Process implementation
 - Problem and modification analysis
 - Modification implementation
 - Maintenance review and acceptance
 - Migration
 - Ritiro del sw.

Esempio L5.27

Applicabilità della norma

Applicabilità della norma ISO 12207

Campi di applicazione della ISO 12207

- aiuta gli acquirenti di servizi IT a definire nei documenti contrattuali le specifiche dei processi da richiedere ai fornitori (>affidabilità)
- permette di comparare le prestazioni di più fornitori rispetto ad un modello di riferimento comune (> efficienza)
- i fornitori vi trovano a loro volta gli elementi per definire i contenuti di una Offerta Tecnica basata su una terminologia comune con gli acquirenti

- La norma non specifica nei dettagli cosa deve essere fatto per eseguire i processi definiti nel modello di CVS
- Non fornisce una guida allo assessment del livello di maturità/qualità dei processi presso una organizzazione
- Non prescrive una particolare metodologia di sviluppo od un ciclo di vita (a cascata, incrementale, prototipale etc...)
- Non si applica a prodotti off-the-shelf

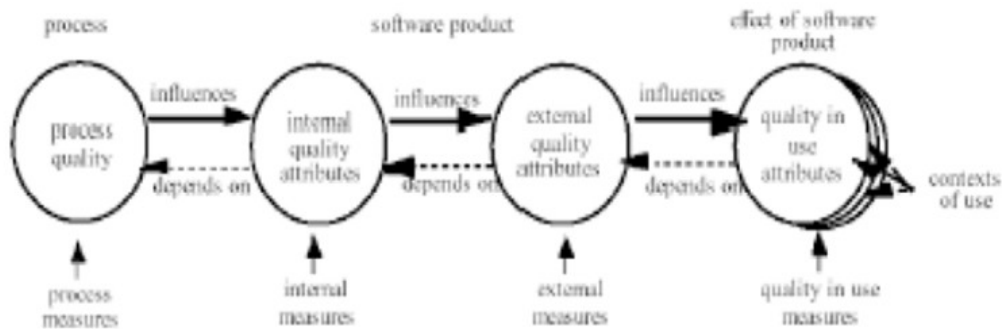
6.2 ISO 9126

La qualità può essere vista dal punto di vista del prodotto e dal punto di vista dei processi. Ci sono standard per entrambi gli aspetti.

Primo modello

McCall 1977 si basa su tre usi di un prodotto sw: da utente, da sviluppatore, da controllore.

Modello attuale



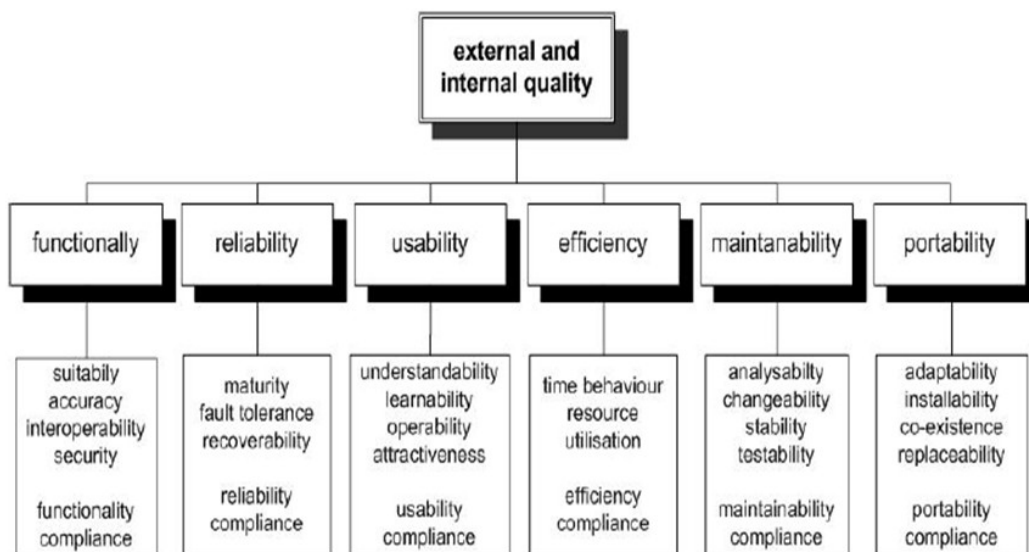
Approccio:

- La qualità del processo migliora la qualità del prodotto
- La qualità del prodotto migliora la qualità d'uso
- Controllare e migliorare il processo
- Controllare e migliorare il prodotto

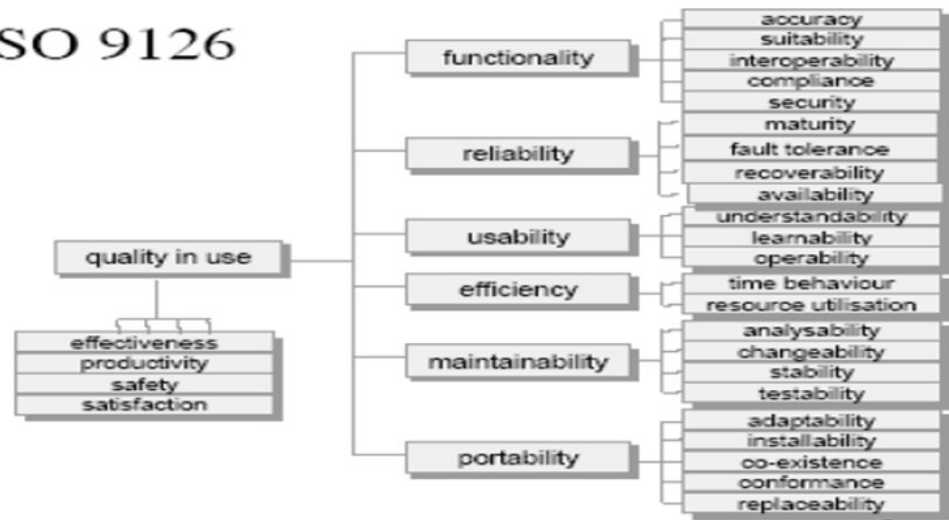
Schema:

- Attributi:
 - Caratteristiche esterne (viste dall'utente)
 - Funzionalità: capacità di fornire funzioni che soddisfano funzioni esplicite o implicite quando il sw è usato sotto certe condizioni
 - Appropriatezza
 - Accuratezza
 - Interoperabilità
 - Sicurezza
 - Conformità
 - Affidabilità: mantiene determinate prestazioni
 - Maturità
 - Tolleranza all'errore
 - Recuperabilità
 - Conformità
 - Usabilità: capito, appreso da un utente
 - Comprensibilità
 - Apprendibilità
 - Operabilità
 - Attrattività
 - Conformità

- Manutenibilità: capacità di essere modificato
 - Analizzabilità
 - Modificabilità
 - Testabilità
 - Stabilità
 - Conformità
 - Efficienza: appropriate prestazioni relative alla quantità di risorse utilizzate
 - Comportamento rispetto al tempo
 - Utilizzo delle risorse
 - Conformità
 - Portabilità: trasferimento da un ambiente ad un altro
 - Adattabilità
 - Installabilità
 - Coesistenza
 - Sostituibilità
 - Conformità
- Caratteristiche esterne
- 3 livelli del modello:
 - Caratteristiche
 - Sottocaratteristiche: ognuna di loro deve essere associata ad una metrica e valutata quantitativamente.
 - Metriche



ISO 9126



Qualità in uso: capacità di abilitare specificati utenti ad ottenere specificati obiettivi con efficacia e sicurezza e soddisfazione d'uso

- Efficacia
- Produttività
- Sicurezza
- Soddisfazione

Internal characteristics

- Internal characteristics are 38 and are bind to the external sub-characteristics
- Completeness
- Access control
- Informativeness
- Self-descriptiveness
- Instrumentability
- Expressiveness
- Data-commonality
- Self-containedness
- Communication-commonality
- Well-equipmentness
- Traceability
- Timeliness
- Robustness
- Integrity
- Modularity
- Simplicity
- Coherency
- Accessibility
- Uniformity
- Accuracy
- Hierarchieness
- Consistency
- Metaphorability
- Attractiveness
- Access audit
- Memorability
- Conciseness
- Choosability
- Guideability

6.3 Revisione

Un gruppo esamina tutto (o una parte) del processo o del sistema e documenta potenziali problemi. Ci sono differenti tipi di revisione:

- Ispezione per la rimozione dei difetti (prodotto)
- Revisione per la valutazione del progresso (prodotto e processo)
- Revisione di qualità (prodotto e standard)

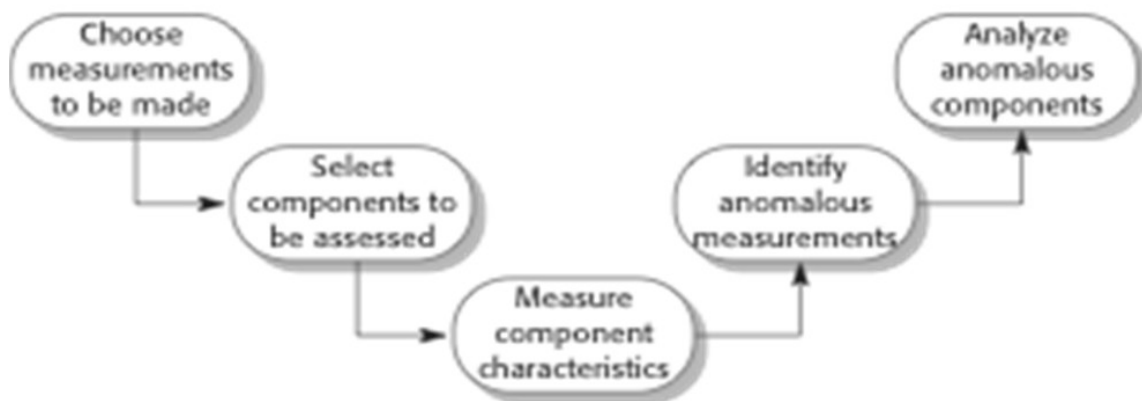
Il processo di revisione in uno sviluppo di sw agile è di solito informale. Nell'extreme programming invece si assicura che il codice sia costantemente esaminato da un altro team.

Le ispezioni sono come le revisioni, dove gli ingegneri esaminano il codice di un sistema per scoprire anomalie/difetti. Non richiede l'esecuzione del programma. Si usa una checklist dei più comuni errori che si è solito fare. (inizializzazione, loop, naming, array ecc.)

Si utilizzano le versioni del sw per comparare gli obiettivi e per quantificare il processo sw. Oltre alla versione si possono usare altre misure per assicurare la qualità del sw. Le metriche del prodotto si possono suddividere nelle seguenti classi:

- Metriche dinamiche dove sono collezionate le misure fatte su un programma in esecuzione. Aiutano la valutazione dell'efficienza e dell'affidabilità
- Metriche statiche dove sono collezionate le misure fatte sulla rappresentazione del sistema. Aiutano a valutare la complessità, manutenibilità e se è capibile un programma.

Le metriche dinamiche sono vicine agli attributi di qualità di un sw, le metriche statiche hanno una indiretta relazione invece.



Riducendo il numero di errori si porta ad incrementare il numero di chiamate all'help desk! Il programma è più affidabile e aumenta quote di mercato.

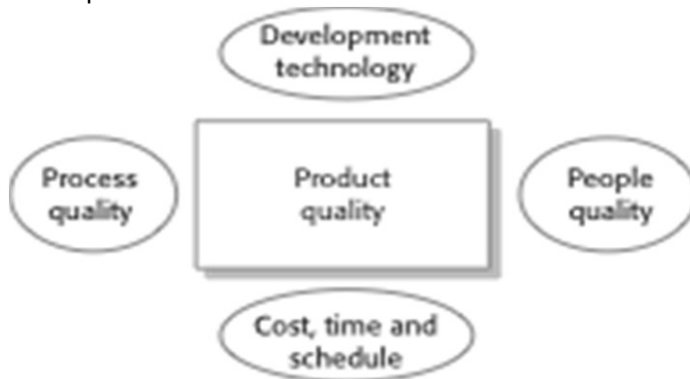
6.4 La qualità totale

- Qualità richiesta: contenuta nelle specifiche del cliente
- Qualità attesa: quella che il cliente considera scontata.
- Qualità latente: quella che il cliente non ha chiesto e non si aspetta, va aldilà delle altre 2.
- Qualità percepita: percezione complessiva da parte de cliente.

La qualita` totale si estende al Controllo di Qualita` di tutti i processi aziendali, la mentalita` aziendale si sposta dalla ricerca del profitto alla ricerca della qualita` nei singoli processi, visione a lungo termine. L'azienda si prefigge di garantire e controllare la qualita` del prodotto che verra` sviluppato. Tale controllo viene fatto tramite la *Quality Assurance* e la *Quality Control* che hanno rispettivamente l'obiettivo di assicurare che tutti i processi di qualita` vengano applicati durante la produzione del prodotto e controllare che il prodotto finito rispetti quanto riportato nel quality plan redatto. Per ottenere la qualita` totale molte aziende si sono adoperate nell'utilizzare dei processi per il miglioramento.

6.5 Process improvement

Molte compagnia hanno capito che devono migliorare i loro processi per fare un prodotto di qualità. Significa capire i processi esistenti e apportare dei cambiamenti. Ci sono degli approcci per fare questo.



Per piccoli progetti un fattore di qualità è la bravura degli sviluppatori. Per progetti grossi un fattore è il processo di sviluppo. Non esiste una cosa standard che vada bene per tutte le tipologie di aziende.

Bisogna considerare anche i singoli aspetti di un processo che si vuole andare a migliorare.

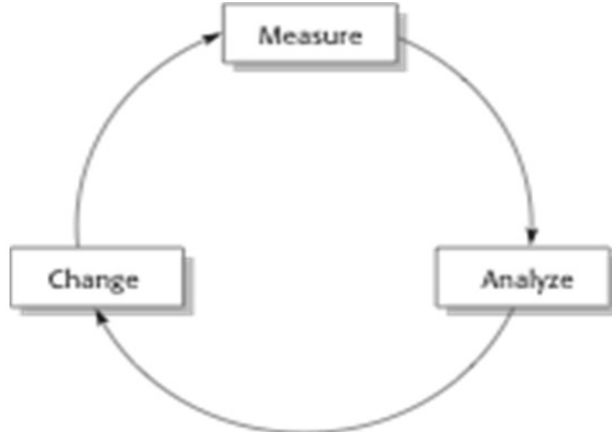
Potremmo migliorare molte cose diverse e avere vari obiettivi

Attributi di processo:

- Accettabilità
- Affidabilità
- Robustezza
- Manutenibilità
- Rapidità
- Non-comprensione
- Standardizzazione
- Visibilità
- Misurabilità
- Supportabilità

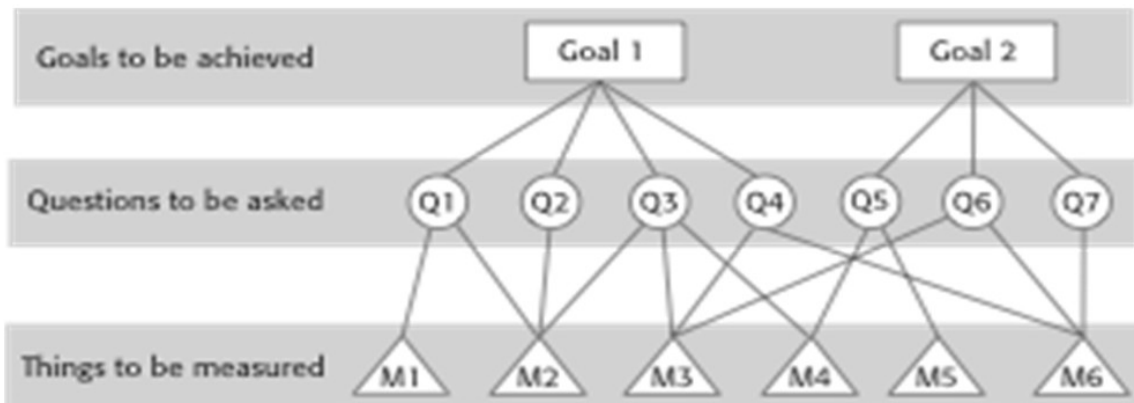
Stage:

- Process measurement: gli attributi del processo corrente che sono misurati. Collezionare dati quantitativi.
- Process analysis: valutazione del processo corrente e identificare i colli di bottiglia
- Process change: cambiare il processo dopo l'analisi.



Metriche: il tempo richiesto dal processo per completare le attività, le risorse necessarie, numero di occorrenze.

Paradigma Goal-questions-metrics



Per l'analisi dei processi usare le seguenti tecniche:

- Pubblicare i modelli di processo e gli standard.
- Questionari e interviste
- Analisi etnografica

Il processo di cambiamento dei processi.



I cambiamenti portano problemi perché ci sono delle resistenze ai cambiamenti da parte delle persone. Molto spesso i cambiamenti iniziali non vengono applicati e si ritorna al punto di partenza.

Capability Maturity Model (CMM)

Un modello di maturità può essere descritto come una strutturata collezione di elementi che descrivono certi aspetti della maturità in una organizzazione.

È un approccio al miglioramento dei processi il cui obiettivo è di aiutare un'organizzazione a migliorare le sue prestazioni. È il modello di riferimento dei processi all'interno di un progetto, una divisione o un'intera organizzazione. Il CMM può essere usato per guidare il miglioramento che definisce il livello di qualità di un insieme di processi aziendali. In alcuni ambiti è richiesta una certa categoria di certificazione per poter essere presi in considerazione per un determinato lavoro.

Un processo è caratterizzato da 3 attributi:

- **Capability:** insieme dei risultati che un processo consente di conseguire.
- **Performance:** misura dei risultati effettivi ottenuti nell'applicazione del processo.
- **Maturity:** misura dell'efficacia del processo e della estensione e precisione con cui le sue fasi e attività sono esplicitamente definite, gestite, misurate e controllate.

Struttura del modello:

- **Maturity levels:** sono 5. Ogni livello è composto da un insieme di aree chiave.
- **Key process areas:** è un set di attività che quando vengono eseguite collettivamente, sbloccano una serie di obiettivi considerati importanti. Ciascuna area è organizzata in 5 sezioni dette "caratteristiche comuni". Le caratteristiche identificano le "pratiche chiave", sono quelle che vengono eseguite.
 - Obiettivi
 - Commitment

- Ability
- Measurement
- Verification

Level 1 — Initial (Chaotic)

It is characteristic of processes at this level that they are (typically) undocumented and in a state of dynamic change, tending to be driven in an ad hoc, uncontrolled and reactive manner by users or events. This provides a chaotic or unstable environment for the processes.

Level 2 — Repeatable

Processes and their outputs could be visible to management at defined points, but results may not always be consistent. Some basic processes are established to track cost, schedule, and functionality, and a degree of process discipline is in place to repeat earlier successes on projects with similar applications and scope; nevertheless there could still be a significant risk of exceeding cost and time estimates

Level 3 — Defined

It is characteristic of processes at this level that there are sets of defined and documented standard processes established and subject to some degree of improvement over time. These standard processes are in place (i.e., they are the AS-IS processes) and used to establish consistency of process performance across the organization.

Level 4 — Managed

It is characteristic of processes at this level that, using process metrics, management can effectively control the AS-IS process (e.g., for software development). In particular, management can identify ways to adjust and adapt the process to particular projects without measurable losses of quality or deviations from specifications. Process Capability is established from this level.

Level 5 — Optimizing

It is a characteristic of processes at this level that the focus is on continually improving process performance through both incremental and innovative technological changes/improvements.

Ogni livello è associato con una performance attesa, e da tools che possono essere usati durante il processo.

CMMI

Evoluzione del CMM. Sono state aggiunte nuove aree di processo e un “implementation goal” ad ogni area di processo.

Componenti:

- Process areas
- Goals
- Practices

Valutazione:

Ssamina i processi e valuta la loro maturità in ogni area di processo. È basato su 6 punti:

- Not performed
- Performed
- Managed
- Defined
- Quantitatively managed
- Optimizing



5	Optimized	Process Improvement is institutionalized
4	Managed	Products and processes are quantitatively controlled
3	Defined	Software engineering and management processes defined and integrated
2	Repeteable	Project management system in place; performance is repeateable
1	Initial	Process is informal and ad hoc; performance is unpredictable

Modalità di accertamento

L'accertamento è condotto mediante una analisi dell'azienda (documenti organizzativi, procedure aziendali) e realizzando interviste attraverso appositi questionari. Il SEI¹ (Software Engineering Institute) ha definito un questionario da utilizzare che va adattato e tarato alle varie realtà. Il questionario è suddiviso in sezioni, quali:

- Organizzazione.
- Risorse, Personale e formazione.
- Gestione della tecnologia.
- standard e procedure.
- Metriche di processo.
- Raccolta ed analisi dei dati.
- Controllo processo.

Le domande sono divise in due sezioni *a* e *b* e sono a risposta chiusa (si/no) ognuna è associata ad un livello da 2 a 5. La maturità di un processo è di un livello *i*, se si risponde sì ad almeno all'80% delle domande di tipo *a* e al 90% delle domande di tipo *b* associate ai livelli da 2 ad *i*.

Un'altra metodologia agile ed un approccio all'ingegneria del software è l'**Extreme Programming**, i cui aspetti caratteristici sono la programmazione a più mani (generalmente in coppia), la verifica continua del programma durante lo sviluppo per mezzo di programmi di test e la frequente reingegnerizzazione del software, di solito in piccoli passi incrementali, senza dover rispettare fasi di sviluppo particolari.

- Feedback a scala fine.
- Processo continuo.
- Comprensione condivisa.

Considerazione finali sul CMM

La massima valutazione cmm non necessariamente garantisce il massimo risparmio per il cliente. Le indagini confermano che livelli cmm più alti sono correlati con una minore difettosità del sw. Il modello indica ventidue Aree di processi aziendali (Process area) strutturate su cinque livelli, ognuna con i propri obiettivi generici (Generic Goal) e specifici (Specific Goal). Gli obiettivi generici e specifici sono implementati da una sequenza temporale di attività generiche (Generic practice) e specifiche (specific practice), che hanno determinate tipologie di output (typical work product).

5 System modeling

Il system modeling `e il processo di sviluppo di un modello astratto del sistema, attraverso linguaggi grafici (UML). E' molto utile alla visione delle funzionalita' da fornire al sistema, alla loro interazione e comunicazione. Viene utilizzato anche come metodo di comunicazione con i clienti.

¹www.sei.cmu.edu

Sistemi esistenti: I modelli in questo caso sono molto utili per analizzare i punti di forza e di debolezza di un sistema, agevolano l'analisi dei requisiti per l'integrazione con altri sistemi.

Nuovi sistemi: I modelli chiarifica agli stakeholders l'incontro tra i requisiti richiesti e il progetto. Aiuta gli ingegneri nell'implementazione e nella creazione della documentazione.

I punti di vista su di un sistema

External perspective: Modello del contesto e dell'ambiente in cui si collocherà un sistema. Si identifica il contesto di applicazione e i "confini" dell'sistema.

Interaction perspective: Modello dell'interazione tra sistema e l'ambiente o tra moduli appartenenti al sistema. Tipi di interazione:

User: utili per la chiarificazione degli user requirements.

System-to-system: che descrivono le comunicazioni tra diversi sistemi.

Component: che descrivono le comunicazioni tra i componenti all'interno di un sistema.

Structural perspective: Modello dell'organizzazione dei dati che il sistema processa. Può essere un modello **statico**, il quale mostra il design della struttura del sistema, oppure o **dinamico**, il quale descrive l'organizzazione del sistema quando è in esecuzione.

Behavioral perspective: Modello del comportamento dinamico e della risposta agli eventi del sistema.

Data: Dati che arrivano dall'esterno e devono essere processati dal sistema, dati che vengono forniti dal sistema all'esterno.

Event: Eventi che pilotano l'andamento del sistema, che lo portano in uno stato o che attivano dei processi.

Process perspective: Modello dei processi che compongono il sistema.

Strumenti utilizzati per la descrizione

User-case diagram: Sono specificati in linguaggio visuale UML, descrivono la sequenza di azioni che un **attore** deve fare per portare a termine un **task**.

Sequence diagram: Sono specificati in linguaggio visuale UML, descrivono le interazioni tra **attori** ed **oggetti** del sistema. Mostrano le interazioni che si hanno tra vari **use-case**.

State diagram: Sono utilizzati per descrivere il comportamento di un sistema in risposta agli stimoli interni ed esterni. Sono utilizzati per l'analisi del comportamento del sistema a run-time.

Class diagram: È usato quando si sviluppa un sistema object-oriented ed evidenzia le relazioni tra le varie classi di oggetti (astrazione, derivazione, cardinalità, etc.). Gli oggetti rappresentano "qualcosa" nel mondo reale. Permette di astrarre e di ignorare i dettagli implementativi, prescinde dal linguaggio utilizzato per lo sviluppo.

Activity diagram: Sono utilizzati per modellizzare il processing dei dati dove ogni attività rappresenta un passo del processo.

Alcuni tipi di modelli

Data-driven model:

È un modello che viene "guidato" dai dati, cioè le sue azioni e stati sono funzione dei dati di input e gli output generati dagli stessi. È molto utile durante l'analisi dei requisiti perché permette una facile visualizzazione del processo end-to-end.

Event-driven model:

È un modello che viene "guidato" dagli eventi, dove gli stati del sistema sono determinati in funzione degli eventi che avvengono nell'ambiente circostante o sul sistema stesso (es. real-time system). Solitamente si assume che esistono un numero finito di transizioni di stato che portano da uno stato di partenza al corrispondente stato finale.

6 Design & Implementation

Questa fase è dove viene sviluppato il SW eseguibile.

Design:

È l'attività creativa per mezzo della quale si identificano i componenti SW e le loro relazioni, basandosi sui requisiti SW.

Implementation:

È il processo per mezzo del quale si realizza il SW di cui si è fatto il design.

Architectural Design

Dopo aver compreso le interazioni tra l'ambiente ed il sistema si usano queste informazioni per fare il design dell'architettura del sistema.

1. Si identificano i maggiori componenti del sistema e le loro interazioni

2. Si organizza il sistema secondo un pattern (es. client-server). I vantaggi di un modello architettuale:

- Supporto alla comunicazione con gli stakeholders.
- Supporto all'analisi del sistema e dei requisiti non funzionali.
- Riutilizzo dell'architettura.
- Migliore sviluppo di sistemi reused-oriented.

Generic layered

- Usata per la modellazione delle interfacce dei sotto sistemi.
- Organizza il sistema in una set di strati (macchine astratte).
- Supporta lo sviluppo incrementale.
- Crea una separazione tra gli strati diminuendo la complessità delle modifiche/adattamenti.
- Usata in molti campi per esempio OS e web-application.

Repository: Modello utile quando una grossa quantità di dati deve essere condivisa.

Client-server: Molto diffusa tra i sistemi distribuiti, suddivide il sistema in diversi servizi erogati da diverse macchine (server) a cui i client possono chiedere l'utilizzo di un servizio.

Pipe & filter: Utile per sistemi di tipo batch e transaction-based application (non per interactive system). I dati sono organizzati (filter) in base al tipo di data- transformation che formano il data flow (pipe) tra vari componenti.

Open source licensing

Il principio fondamentale dell'open source è che il codice dovrebbe essere di libero accesso e che ognuno possa fare ciò che vuole di quel codice.

- Legalmente lo sviluppatore del codice continua ad esserne il proprietario ed è sia facoltà aggiungere restrizioni o pagamento per l'utilizzo, anche successivamente.
- Alcuni credono che se del codice open source è utilizzato per sviluppare un certo sistema allora l'intero sistema dovrebbe essere open source.

I tipi di licenze:

GNU General Public License (GPL):

È anche chiamata *licenza reciproca* perché il SW che integra il codice sotto GPL deve aderire alla GPL.

GNU Lesser General Public License (LGPL):

È una variante della GPL che permette di utilizzare codice sotto LGPL senza dover pubblicare il codice dell'applicazione creata.

Berkley Standard Distribution (BSD): Non ha valore reciproco, questo permette di includere codice sotto BSD license in una propria applicazione e vendere il prodotto.

7 Software testing

Dopo aver sviluppato il codice la fase di testing `e essenziale per verificare la correttezza del software e il raggiungimento degli obiettivi rispetto a requisiti e qualita`.

Verification: Il software corrisponde alle specifiche.

Validation: Il software corrisponde a quello che si aspetta il cliente.

Questa operazione del **V&V** comincia con lo sviluppo del software, continua nella fase di messa in operativita` e manutenzione, termina con la fine dell'utilizzo del SW. Deve essere fornito il livello di **dependability** per il quale si ritiene soddisfatto in certo aspetto, inoltre il progettista deve essere in grado di determinare quando questo valore `e stato raggiunto.

Aviability: Misura il la QoS in relazione al tempo in cui il sistema `e funzionante e il tempo in cui non lo `e.

Reliability: E la ragione di operazioni terminate con successo rispetto a tutte quelle tentate.

Failure: Il presentarsi di un evento che l'utente vede come anomalo/errato.

Fault: La causa della **Failure**. **Error:** Ha

due significati

1. La differenza tra il risultato ottenuto e quello corretto.
2. Riferimento all'azione di un attore che causa un

Fault. Gli obiettivi del testing sono:

- Dimostrare ai clienti e sviluppatori che il SW rispetta le specifiche.
- Scoprire i casi per cui il SW si comporta in modo scorretto o se non `e conforme alle come specifiche.
- Scoprire i difetti come esecuzioni errati, deadlock, crash, corruzione dei dati.

7.1 Tipi di testing

La differenza con il classico testing ingegneristico `e molto grande, infatti i programmi non hanno un comportamento lineare quindi verificarne la correttezza per un valore molto elevato non implica la correttezza per i valori inferiori (come per il carico ammissibile di un ponte.) Il total test nei software non `e realizzabile perch`e il numero di configurazioni possibile cresce in maniera estremamente rapida.

Input-output model: Questo tipo di testing prevede di fornire un certo numero di dati di input al sistema ed analizzare i risultati di output per verificarne la correttezza..

Inspection model: Comporta l'ispezione manuale da parte dei progettisti/sviluppatori del codice prodotto con il supporto dei modelli di alto livello in UML. Non necessita l'esecuzione del prodotto.

- Durante il testing on-line errori possono coprire altri errori, cos'è che in particolari scenari nonostante siano presenti due o più errori non ne vengono rilevati nemmeno uno.
- Il testing può essere fatto anche su porzioni di applicazioni senza che il prodotto completo sia disponibile e funzionante.
- Questo tipo di testing può verificare solo l'aderenza del software ai requisiti (Verification), non può verificare il soddisfacimento dei reali requisiti utente (Validation).
- Non si possono valutare i requisiti non funzionali come le performance e l'usabilità

Unit testing: viene testato un singolo componenti SW.

Integration testing: vengono testate le comunicazioni tra moduli diversi.

Big bang: Tutti i moduli vengono integrati in un solo colpo.

Incrementale: Si aggiungono i componenti a passi ed ad ogni passo si effettua l'integrazione testing.

System testing: viene testato il sistema nella sua interezza per verificare che il sistema aderisca alle specifiche, eseguito da un team indipendente a quello di sviluppo.

Acceptance testing: viene testato il sistema nell'ambiente dove dovrà lavorare per verificarne le specifiche utente.

Configuration test: verifica di tutti i comandi per cambiare le relazioni fisiche e logiche dei componenti HW.

Recovery test: Capacità del sistema di reazione ai fault. **Stress test:**

Affidabilità nelle condizioni di carico elevato. **Security test:**

Invulnerabilità rispetto ad accesso non autorizzati. **Safety test:** Sicurezza

di cose e/o persone che utilizzano il SW.

Use-case test: Verifica della correttezza di tutti gli use-case previsti.

Test-driven development (TDD): Si effettua il development di pari passo con il testing.

- Il risultato del test è un indicatore molto importante dell'avanzamento dello sviluppo.

- Non si procede al successivo incremento di sviluppo fino a che non viene superato i test.
- Usato nell'extreme programming e nel plan-driven development.
- Ottima code coverage.
- Debugging semplificato.

Requirement testing: Si basa sui requisiti e crea uno o più casi di test per verificarli.

User testing: Viene testato il sistema dal punto di vista utente (interfacce, processi attivati, dati modificati).

Alpha testing: Utente e sviluppatori lavorano insieme al testing.

Beta testing: Viene rilasciata una release del prodotto con cui l'utente può interagire più o meno completamente e segnalare problemi agli sviluppatori.

Acceptance: Il SW viene considerato pronto e viene testato dopo l'installazione presso il cliente.

7.2 Tecniche di testing

White-box testing

Il tester sa come è fatto il componente e svolge un test basato sulla copertura di questi aspetti:

- Codice.
- Condizioni.
- Percorsi.
- Flusso dati.

Black-box testing

Il tester non sa come è fatto il componente e svolge un test basato attuando questi passi:

1. Decomposizione del sistema in un insieme di funzioni indipendenti.
2. Individuazione dei casi speciali, normali ed errati in cui si può portare il sistema.
3. Considera i limiti di combinazione dei valori.

Lo standard di riferimento per il testing `e **IEEE Standard for Software testing documentation (829-1998)**

Design:

Plan: Pianificazione del testing.

Design specification: Scelta del tipo dei tipi di test da utilizzare.

Case specification: Definizione degli scenari di test.

Procedure: Procedure da attuare nel test.

Item transmittal report: Definizione dei deliverables da produrre nei test.

Execution:

- Log.
- Incident report.

End: Summary report.

Cleanroom

- Mira prevenzione dei difetti piuttosto che alla loro correzione.
- Sviluppo SW tramite metodi formali.
- Verifica delle specifiche tramite "team review".
- Sviluppo incrementale per poter effettuare dei "micro-test" e tenere bassa la complessita` per ogni passo.
- Analisi statistica degli input/output.

Extreme programming

Metodologia agile che considera naturale cambiamenti ed aggiunta di requisiti in corso d'opera (piu` naturale della definizione dei requisiti subito all'inizio). Ha come scopo la riduzione dei costi dei cambiamenti. I passi dell' extreme programming:

1. **Generare storie d'uso:** Fase ciclica con il cliente. puo` essere eseguita in parallelo parallela alle altre. E` necessaria per iniziare ma
2. Selezione delle storie prioritarie.
3. Generare test di unita`.
4. Programmazione di correzione.
5. Integrazione, si passa al passo 2 fino per il completamente del programma.

6. Test di accettazione.

7. Piccole release incrementali.

L'utilizzo dei grafi di controllo (CFG — Control Flow Graph) avviene mediante la loro creazione partendo dal codice e descrivendone tutti i possibili percorsi, a seconda delle varie categorie di istruzioni **if-else**, **loops**, **return**.

Cammino linearmente indipendente

Si definisce tale un cammino che introduca almeno un nuovo insieme di istruzioni o una nuova condizione. In un **CFG** un cammino si dice linearmente indipendente se attraversa almeno un arco non ancora attraversato. Il numero di cammini linearmente indipendenti è pari al numero cicломatico di *McCabe* (metrica del SW basata sull'analisi della complessità del flusso di controllo).

$$V(G) = E - N + 2$$

E: numero di archi in G.

N: numero di nodi in G

$$V(G) = P + 1$$

P: numero di predicati in G.

V(G): numero di regioni chiuse in G + 1

Test case esercitanti questi cammini garantiscono l'esecuzione di ciascuna istruzione almeno una volta

7.3 Testing strutturale

È una tecnica di testing basata sui metodi di copertura del codice, cioè si cerca di fare in modo che i test eseguiti eseguano almeno una volta (meglio se molte volte) tutte le istruzioni, **nodi**, e tutti gli **archi decisionali** del SW. Parametri di copertura:

- Codice.
- Condizioni.
- Percorsi.
- Flusso dati.

Note:

- La copertura di tutte le condizioni non implica la copertura di tutte le decisioni.

- Non basta progettare un test che testi ogni valore di ogni condizione almeno una volta (nested-condition).
- La copertura di tutte le combinazioni delle condizioni implica la copertura di tutte le decisioni.

8 SW Maintainance

La SW maintenance `e quella parte del ciclo di vita che si occupa di apportare modifiche, correzioni ed adattamenti ad un prodotto SW dopo il suo rilascio. Standard di riferimento ISO/IEC 14764.

Sistemi SW Legacy: Sistemi critici ereditati dal passato che non possono venire modificati efficientemente o `e troppo difficile farlo. Sono legati a vecchie tecnologie, vecchi requisiti.

- Sistemi grandi.
- Monolitici.
- Difficile modifica e/o sostituzione.

Technical computer-based system: Sistemi costituiti da HW e SW, ma operatori e processi non sono considerate parte del sistema. Es. editor di testo usato per scrivere un libro.

Socio-technical system: Sono sistemi che includono in essi anche i operatori e processi. Es. Un sistema di pubblicazione per produrre un libro.

- Emergent properties, proprietà che dipendono dai componenti del sistema e dalle loro relazioni.
- Non deterministici, non producono i soliti output presentando gli stessi input. Comportamento del sistema parzialmente condizionato dal comportamento degli utenti.

Lehman's Laws

Continuing change: Un sistema deve necessariamente cambiare ed evolvere per mantenere la sua utilità perché cambia l'ambiente in cui opera.

Large program evolution: Se modifiche critiche hanno dimensione paragonabile al SW `e tempo di progettare una nuova versione.

Organizational stability: Durante il ciclo di vita di un programma la sua velocità di sviluppo `e all'incirca costante ed indipendente dalle risorse dedicate allo sviluppo del sistema stesso.

Increasing complexity: risorse aggiuntive devono essere impiegate per preservare la semantica del sistema e semplificare la struttura.

Conservation of familiarity: Durante il ciclo di vita di un prodotto SW il tasso di cambiamento di ogni versione è all'incirca costante.

Esempi di costi della manutenzione:

- Corrective: 20%.
- Adaptive: 25%.
- Perfective: 55%.

Complessità ciclomatica

La complessità ciclomatica di McCabe stima la manutenibilità di un programma tramite il numero di cammini indipendenti. Tale complessità viene calcolata come :

$$M C = e - n + 2 * p$$

e: numero di archi.

n: numero di nodi.

p: numero di sottografi disconnessi.

La complessità ciclomatica di McCabe viene messa in relazione con la probabilità di errori all'interno del codice, inoltre viene utilizzata anche per dimensionare il numero di test da portare avanti su di un determinato programma. Una bassa complessità (es. fino a 10) ciclomatica corrisponde a un programma con pochi errori e di facile comprensione.

Refactoring

Processo con il quale si modifica la struttura interna di un programma senza modificarne il comportamento esterno o le funzionalità esistenti. applicato per migliorare le proprietà non funzionali di un SW:

- Leggibilità e struttura del codice.
- Grado di manutenibilità.
- Esentendibilità.
- Prestazioni.

Caratteristiche dei sistemi distribuiti

- Resource sharing — Condivisione di risorse SW e HW.
- Openness — Utilizzo di HW e SW provenienti da diversi fornitori.
- Concurrency — Esecuzione concorrente per aumentare le performance.
- Scalability — incremento del throughput aggiungendo risorse.
- Fault tolerance — L'abilit  a di continuare l'esecuzione nonostante un fault.
- Security — I dati ed i servizi sono distanti tra loro fisicamente,   quindi necessario provvedere ad una comunicazione sicura tra i nodi del sistema.
- QoS.

Tipi di architetture:

Client-server: Esistono dei noti **server** che forniscono servizi e nodi **client** che richiedono l'utilizzo di questi servizi (web-based application).

Master-slave: C'  un'unit  centrale **master** che impartisce gli ordini agli altri **slave** (real-time application).

Peer-to-peer: Architettura decentralizzata, tutti sono sia client che server, ogni nodo possiede parte dei servizi, richiede quelli che gli mancano e fornisce quelli di cui   fornito. Nell'insieme di tutti i nodi si avranno molti servizi duplicati.

9 Service-Oriented Architecture (SOA)

Software as a service (SaaS):   la visione di un intero SW come servizio remoto attraverso internet. Il SW   posseduto e gestito dal fornitore del servizio che si occupa anche del HW

Service-Oriented Architecture (SOA):   l'approccio architetturale per l'implementazione dei **SaaS**. In modo da provvedere ad una separazione e distribuzione delle richieste di servizio

10 Testing Strutturale

[L7.93]

Il testing strutturale si basa sull'adozione di metodi di copertura degli oggetti che compongono la struttura dei programmi. Per copertura si intende la definizione di un insieme di casi di test, in particolare di input, in modo tale che gli oggetti di una definita classe (es. strutture di controllo, istruzioni..etc.) siano attivati almeno una volta nell'esecuzione dei test.

Esistono diversi criteri di copertura basati sul flusso del controllo :

- **Statement Coverage:** copertura delle istruzioni
- **Edge Coverage:** copertura delle decisioni
 - **Condition Coverage :** tutti i possibili valori delle componenti di condizioni composte sono provate almeno una volta
- **Path Coverage:** deve essere attivato ogni cammino del grafo
- **Data flow coverage**

Si definisce un insieme di casi di test che coprono ogni tipologia di coverage, ad esempio andremo a definire per il coverage delle istruzioni un insieme di test che prevede l'esecuzione di ogni istruzione del programma. Per la copertura delle istruzioni e per la copertura delle decisioni andiamo a definire il TER: *Test Effectiveness Ratio* che mette in relazione il numero di istruzioni/decisioni coperte con il numero di istruzioni/decisioni totali.

Per quanto riguarda il path coverage (copertura dei cammini) si presentano diversi problemi:

1. il numero di cammini è generalmente infinito
2. infeasible path : cammino non eseguibile

Le possibili soluzioni sono quella di scegliere un numero finito ed eseguibile di cammini tramite metodi fondati sui grafi di controllo o metodi di tipo data flow

11 Verifica Statica, esecuzione simbolica

[L7.10] [L7.147]

L'analisi statica permette di controllare le proprietà del codice, è una tecnica che fa parte del validation e verifying del software. I programmi vengono eseguiti in modo simbolico (o astratto), lungo i cammini si propagano dei valori simbolici delle variabili e gli statement si considerano eseguibili solo se gli input soddisfano determinate condizioni. Le condizioni vengono indicate tramite le path condition, una per un determinato statement, che indicano le condizioni che gli input devono soddisfare affinché un'esecuzione percorra un cammino lungo cui lo statement sia eseguito.

Una path condition `e un'espressione Booleana sugli input simbolici di un programma; all'inizio dell'esecuzione simbolica essa assume il valore vero. Per ogni condizione che si incontrer`a lungo l'esecuzione la pc assumer`a differenti valori a seconda dei differenti casi relativi ai diversi cammini dell'esecuzione.

12 Manutenibilit`a, complessit`a ciclomatica

[L9]

La manutenzione di un software `e una fase molto importante della vita del SW stesso e rappresenta una percentuale di costo elevata. Esistono diversi tipi di manutenzione:

- **Manutenzione preventiva** : fatta prima di fare la delivery del SW
- **Manutenzione correttiva** : fatta dopo che si sono scoperti degli errori.
- **Manutenzione adattiva** : fatta per mantenere il SW usabile in un contesto che `e mutato.
- **Manutenzione perfective** : manutenzione fatta per aumentare le prestazioni del SW.

E ` importante definire il costo della manutenzione del SW, per fare questo possono essere utilizzate diverse metriche:

- Line Of Code
- Mc Cabe complexity
- Halstead's complexity
- Structural metric

La complessit`a ciclomatica di McCabe stima la manutenibilit`a di un programma tramite il numero di cammini indipendenti. Tale complessit`a viene calcolato come :

$$MC = e - n + 2 * p$$

dove:

e: numero di archi n

: numero di nodi

p : numero di sottografi disconnessi

La complessit`a ciclomatica di McCabe viene messa in relazione con la probabilit`a di errori all'interno del codice, inoltre viene utilizzata anche per dimensionare il numero di test da portare avanti su di un determinato programma. Una bassa complessit`a (es. fino a 10) ciclomatica corrisponde a un programma con pochi errori e di facile comprensione.

13 Modelli per il miglioramento

[L5.62]

I modelli per il miglioramento prendono in esame gli attuali processi e li cambiano in modo da ottenere un incremento della qualità e/o una riduzione del costo e del tempo di sviluppo. Approcci al miglioramento:

- **maturity approach:** introduce buone pratiche per quello che riguarda l'ingegneria del SW.
- **agile approach:** tende a concentrarsi sullo sviluppo iterativo cercando di diminuire gli overheads nel processo SW. Si tende a essere molto veloci nel produrre qualcosa e essere rapidi nell'apportare cambiamenti ai requisiti.

Per scegliere il metodo di miglioramento è necessaria una attenta analisi della situazione iniziale e delle caratteristiche che voglio migliorare del mio processo. Ad esempio potrei voler migliorare la qualità del mio prodotto oppure potrei voler migliorare il tempo di sviluppo cercando di diminuirlo.

14 Test di integrazione

Si parla di test di integrazione quando più unità già singolarmente testate vengono unite e testate come un'unica entità. Tale test può essere portato avanti in vario modo:

- **Big Bang:** Le unità vengono riunite tutte insieme in una singola volta e vengono testate.
- **Incrementale:** Le unità vengono unite tra di loro in maniera successiva in modo da poter scovare il prima possibile eventuali difetti.

Questo tipo di testing mira ad accertare la correttezza dello scambio di informazioni tra le interfacce, dopo questo testing viene condotto il system testing, il test globale del sistema. È possibile applicare una politica del tipo bottom-up con la quale vengono unite per prime le unità più piccole in modo tale che il processo di testing può essere iniziato anche se le unità più complesse non sono state ancora portate a termine.

15 White Box Testing, Black Box Testing

Lo WBT prevede il testing da parte di un utente che è a conoscenza della struttura interna del prodotto da testare, è importante garantire il coverage dell'intero programma per aver testato bene l'unità. Il BBT non prevede la conoscenza della struttura interna del prodotto sottoposto a test, il tester conosce solo le funzioni portate a termine dall'unità,

`e indicato per l'integration testing.

A Definizioni

Sistemi SW Legacy : Sistemi critici ereditati dal passato che non possono venire modificati efficientemente o `e troppo difficile farlo.