Ingegneria dei Sistemi Software

Alessandro Puccetti & Alessandro Donati

12 febbraio 2013

Schema per ripasso

Disclaimer

Questo documento va considerato **esclusivamente** come uno strumento per il ripasso, **non è assolutamente esaustivo e completo**, non sostituisce in alcun modo il materiale didattico ufficiale. Inoltre nessuno degli autori si prende alcuna responsabilità per la veridicità e affidabilità delle informazioni. Si chiede scusa in anticipo per errori di scrittura o discorsi poco chiari, questo materiale è nato per uso e consumo personale.

Indice

1 Introduzione

Gli Attributi di un buon SW:

Manutenibilità: Il Sw deve essere scritto in modo da consentire un agile evoluzione, modifica, estensione e correzione.

Dependability: È la caratteristica di un SW di mostrarsi affidabile.

- Non deve causare danni a cose e/o persone.
- Non deve causare danni economici.
- Sicurezza:
  - Evitare che utenti malevoli possano creare infiltrasi nel sistema.
  - Mitigare la possibilità di errori da parte degli utenti del sistema.

Efficency:

- Utilizzo corretto di risorse come memoria, CPU, Disco, etc.
- Parametri: responsivness, memory allocation, CPU utilization, etc.

Acceptability:

- Usabilità in relazione al target di utenza.
- Creazione del sistema in funzione dell'utente finale.

Stadi di di avanzamento del SW process:

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

SW Development: Realizzazione del prodotto software.

SW Validation: Testing del prodotto SW per garantire le funzionalità richieste e i constrains 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 è **molto costoso**.

- La manutenzione del SW mal progettato ha un costo **paragonabile** al costo della realizzazione del SW stesso.

Consigli:

- Utilizzo di formalismi.
- Requisiti formulati in modo **non ambiguo**.
- Requisiti **pienamente discussi** con il cliente.
- Modularità.
- Astrazione $\rightarrow$ flessibilità del prodotto.
- Estendibilità.

2 Requirements Engeneering

Cosa sono i requisiti:

- Asserzioni **astratte** ad alto livello delle funzionalità e dei constrains.
- Spesso soggetti ad interpretazioni, possono creare incomprensioni tra cliente ed ingegneri.

Obiettivi dell'analisi dei requisiti:

- Definire in modo **univoco** ed **non ambiguo** per entrambi (ciente e fornitore) le caratteristiche del SW da produrre.
- Le funzionalità devono soddisfare i bisogni del clienti.
- Review prima dell'inizio del development.
- Stima dei costi di progettazione e sviluppo.
- Produrre la vaseline per la valutazione ed il testing.

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**.

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, ...).

Funzionali:

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

Non-Funzionali:

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

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 è di fatto un contratto tra cliente e progettisti.

Caratteristiche del SRS:

Non ambiguità: 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 è contenuto nel SRS deve essere corretto.

Completezza: Deve contenere tutti i requisiti voluti dal cliente.

Verificabilità: Deve consentire la verifica dei requisiti.

Tracciabilità Deve consentire la tracciabilità tra requisiti e funzionalit a sviluppate (codice).

Consistenza: Non deve contenere requisiti in contraddizione.

Modificabilità: Deve prevedere la possibilit a di modifica con le relative metodologie.

Ranked: Deve prevedere una valutazione dei requisiti per definirne criticità e priorità.

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 constrains 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.

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 del requisiti.

Testing: Generazione del programma di testing.

3 Tipologie di ciclo di vita

3.1 Code&Fix

Attività:

1. Costruire una prima versione.
2. Ciclo: Modifica fino a che il cliente è soddisfatto.
3. Prodotto operativo (Per la manutenzione tornare al punto 2).
4. Ritiro del prodotto.

Conseguenza:

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

3.2 Waterfall

Modello a cascata dove ogni fase deve essere completata prima dell'inizio della successiva. Fasi:

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. **Progettazione:** Partendo dal documento SRS si progetta la struttura del SW tramite diagrammi delle classi (UML), specifica di alcuni dettagli implementativi e design architetturale.
3. **Development:** Realizzaizone (coding) del prodotto SW, validazione e testing dello stesso. Documento di testing con scenari e risultati.
4. **Manutenzione:** Correzione dei bug, modifiche, estensioni, adattamenti. Documentazione aggiornata, version control.

Vantaggi e svantaggi:

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 Do it twice model

Consiste nella prima implementazione di un sistema rozzo, utile per la stesura dei **requisiti** e la verifica di textbffattibilià. Dopo di che si utilizza il modello Waterfall per costruire la seconda versione (definitiva).
Modello con pilota/prototipo:

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

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

Problemi:

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

3.4 Mopdelli iterativi/evolutivi

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

- Modello ciclico, si effettuano tutti i passi e si torna all'inizio **riciclando** il meta-prodotto realizzato.
- Produrre presto qualcosa di utile, installabile ed operativo.
- Progettare e sviluppare in modo che il prodotto SW possa agilmente evolversi ed adattarsi.
- Evoluzione da uno stato base a prodotto completo.
- Decomposizione in sotto sistemi.

4 Software process

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.

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

Arrività:

1. Analisi e definizione dei requisiti.
2. System e SW design.
3. Implementazione e unit test.
4. Integration test e system test.
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 durande 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 è 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 è formata da:

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

Mentre la seconda parte è ascendente da un livello di dettaglio alto ad uno passo 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 $\rightarrow$ basso) e si valida il sistema nel senso opposto (basso $\rightarrow$ alto).

4.2 Modello incrementale

Benefici:

- Partizionamento dei requisiti per priorità e criticità.
- Flessibilità al cambiamento, con conseguente riduzione dei costi per modifiche ed adattamenti.
- Un più facile feedback tra cliente e fornitore sul SW prodotto.
- Installazione più rapida presso il cliente di versioni intermedie, funzionanti ed operative.

Problemi:

- Il processo non è 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 è necessario tempo per la ottimizzazione del codice.
- L'integrazione dei cambiamenti diventa sempre più difficile e complessa con l'aumentare degli stessi.

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:

1. Specifica dei requisiti.
2. Analisi dei componenti.
3. Modifica dei requisiti.
4. Design del sistema per il riutilizzo.
5. Sviluppo ed integrazione.
6. Validazione.

Sistemi 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.

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

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.

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

Implementation: Traduzione del design in un programma eseguibile.

Validation: Fase che dimostri che i requisiti e i vincoli specificati nel SRS sono rispettati.

Testing:

1. Unit/Component testing.
2. Integration teting.
3. System testing.

4. Acceptance testing.

Evolution: Modifica dei requisiti, adattamento ad nuove tecnologie, migliorie e correzione errori.

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

4.4 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.

1. Definizione degli obiettivi.
    - Requisiti.
    - Rischi.
    - Piano di gestione.
2. Analisi dei rischi.
    - Studio di alternative.
    - Valutazione dei costi.
3. Sviluppo e validazione.
    - Realizzazione del prodotto.
    - Validazione e testing.
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 gesionali

- Pianificazione delle fasi.
- Analisi dei rischi.

I Ruoli

Committente:

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

Fornitore:

- Sviluppo.
- Validazione.
- Analisi dei rischi.

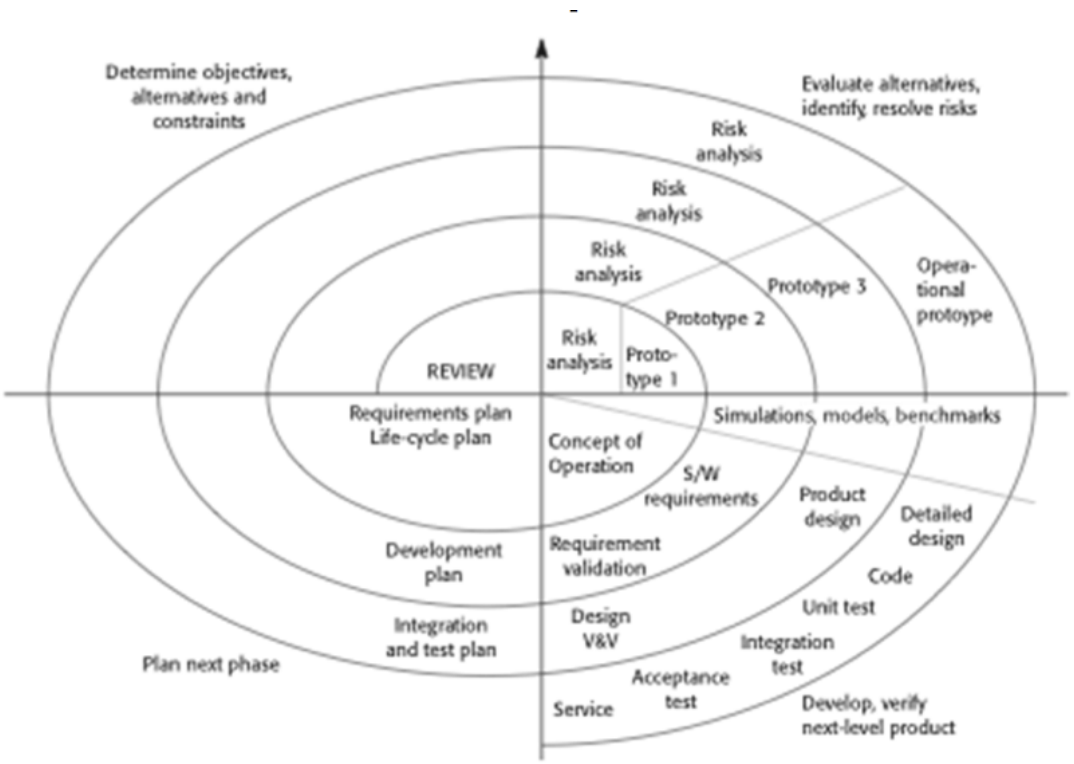

Figura 1: Modello a Spirale

4.5 The Rational Unified Process (RUP)

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

Inizio: Stabilisce il caso da sviluppare per il sistema.

Elaborazione: Sviluppo del problema e dell'achitettura del sistema.

Costruzione: System design, implementazione e testing.

Transizione: Installazione del sistema nel ambiente operativo.

Le interazioni in RUP sono di due tipi, **in-phase** e **cross-pahse**. La prima è iterativa ad ogni passo si incrementa il risultato, la seconda consiste nel ripartire il ciclo dalla prima fase.
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)
- Facilità nella verifica della qualità del SW
- Gestione delle modifiche.
- Possibilità di sviluppo di prototipi per l'analisi dei requisiti o delle criticità.

5 Project management & Project Planning

5.1 Cost estimate

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

Algoritmic cost modeling

Il costo è calcolato da una funzione matematica che tiene conto dei seguenti fattori:

- Dimensione.
- La natura non proporzionale tra la dimensione del progetto e la complessità finale.
- Tipologia.

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

Functional Point

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 aggiunda, modifica e cancellazione.

EO: External Output.
Sono quell 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 è 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 quli non è responsabile per quanto riguarda i 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), **averange** (A) oppure **complex** (C) a seconda della complessità della funzione. Il totale viene calcolando facendo la somma pesata di ogni valore nella tabella, dopo diché si effettua un aggiustamento seconda dei gradi di influenza di alcuni parametri tipo:

- Performance.
- Coesione del team.
- Esperienza nel tipo di progetto.
- Riutilivabilità del codice.
- Operabilità.
- Tipi di comunicazioni.

CoCoMo: Constructive Cost Model

Questo metodo può calcolare la quantit a 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 una analisi di riduzione dei costi.

$$M = C \cdot P^{\mathrm{S}} \cdot A$$

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 complessità dell'algoritmo utilizzato e produce stime con scostamenti inferiori al 20% nei seguenti casi:

Base: 25% dei casi.

Medio: 68% dei casi.

Avanzato: 70% dei casi.

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 pi'u di 200 progetti reali.

Significato dei coefficienti

L'esponente 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 si progettazione e sviluppo.
- Requisiti non funzionali.

5.2 Project management

Attività:

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.

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.
2. Analisi delle possibili conseguenze (Probabilità & Conseguenze).
3. Pianificazione delle delle misure per minimizzare i rischi.
4. Stesura di un piano di azione per gestire i rischi (Strategie).
5. Monitoraggio.

Gestire le persone

- Motivare persone e team.
- Coesione del team.
- Selezione dei membri di un team.
- Organizzazione del team.
- Comunicazione intra/extra team.

6 Quality management

I modelli di gestione della qualità non sono modelli operativi, si propongono come templare 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 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.

Hanno molta importanza gli stancar perchè derivati da anni di raffinamenti e di sviluppo nel campo SW, i più importanti sono **ISO 9001** (ultima versione del 2008) e **ISO/IEC 12207** del 1995.
Il documento della qualità deve contenere inoltre:

- Le metriche di valutazione e la loro spiegazione
- Gli obiettivi descritti in modo non ambiguo (eventualmente portati in forma numerica).

- Trattazione su:
  - Affidabilità.
  - Usabilità.
  - Manutenibilità.
  - Efficienza.
  - Portabilità.

Capability Maturity Model (CMM)

È un approccio al miglioramento dei processi il cui obiettivo è di aiutare un'organizzazione a migliorare le sue prestazioni. Il CMM può essere usato per guidare il miglioramento dei processi all'interno di un progetto, una divisione o un'intera organizzazione. È il riferimento 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.

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
It is characteristic of processes at this level that some processes are repeatable, possibly with consistent results. Process discipline is unlikely to be rigorous, but where it exists it may help to ensure that existing processes are maintained during times of stress.

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.

Modalità di accertamento

L'accertamento è condotto mediante una analisi dell'azienda (documenti organizzativi, procedure aziendali) e realizzando interviste attraverso appositi questionari. Il SEI[1] (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 si 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.

7 System modeling

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

[1] www.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 modello 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 interzione:

- **User:** utili per la chiarificazione degli user requirements.
- **System-to-system:** che descrivono le comunicazioni tra diversi sitemi.
- **Comonent:** 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.

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

- **Data:** Dati che arrivano dall'esterno de 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, cardinalià, 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 perchè 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.

8 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 dichi si è fatto il design.

Architectural Design

Dopo aver compreso le interazioni tra l'ambiente ed i sistema di 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 architetturale:

- 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 quantià di dati doveà 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 system di typo batch e transaction-based application (non per interactive system). I dati sono organizzati (filter) in base al tipo di data-trasformation 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.

9 Software testing

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

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 operatività 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 è stato raggiunto.

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

Reliability: È la razione 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 è conforme alle come specifiche.
- Scoprire i difetti come esecuzioni errati, deadlock, crash, corruzione dei dati.

9.1 Tipi di testing

La differenza con il classico testing ingegneristico è 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 è realizzabile perché il numero di configurazioni possibile cresce in maniera estremamente rapida.

Input-output model: Qesto tipo di testing prevede di fornire una 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, così 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'usablità

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'integraiton 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 dovà 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: Affitabilità nelle condizioni di carico elevato.

Security test: Invulerabilità rispetto ad accesso non autorizati.

Safety test: Sicurezza di cose e/o persone che utilizzano il SW.

Use-case test: Verifica della correttezza di tutti gli usa-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 coverance.
- Debugging semplificato.

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

User testing: Viene testato i 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.

9.2 Tecniche di testing

White-box testing

Il tester sa come è fatto il componente e svolge un 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 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 è **IEEE Standard for Software testing documentation (829-1998)**

Deign:

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 attuale 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 complessità per ogni passo.
- Analisi statistica degli input/output.

Extreme programming

Metodologia agile che considera naturale cambiamenti ed aggiunta di requisiti in corso d'opera (più 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. È necessaria per iniziare ma può essere eseguita in parallelo parallela alle altre.
2. Selezione delle storie prioritarie.
3. Generare test di unità.
4. Programmazione di correzione.
5. Integrazione, si passa al passo 2 fino per il completamente del programma.

6. Test di accettaizone.

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 istruzoni **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 ciclomatico di *McCabe* (metrica del SW basata basata sull'analisi della complessità del flusso di controllo).

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

E: numero di archi in G.

N: numero di nodi inG

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

P: numero di predicati in G.

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

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

9.3 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.

10 SW Maintainance

La SW maintenance è 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 è 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 operatore operational process non sono considerate part del sistema. Es. editor di test usato per scrivere un libro.

Soco-tenchnical system: Sono sistemi che includono in essi anche i operatoti e operational process. Es. Un sistema di pubblicazione per produrre un libro.

- Emergent proprerties, 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 system deve necessariamente cambiare ed evolvere per mantenere la sua utiliyà perché cambia l'ambiente in cui opera.

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

Organizational stability: Durante il ciclo di vita di un programma la sua velocità di sviluppo è 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 semplicizzare 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%.
- Perfectice: 55%.

Complessità ciclomatica

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

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

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 truttura del codice.
- Grado di manutenibilità.
- Esntendibilità.
- 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, è 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'utilizo di questi servizi (web-based application).

Master-slave: Cè un'unità 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 è fornito. Nell'insieme di tutti i nodi si avranno molti servizi duplicati.

11 Service-Oriented Architecture (SOA)

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

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

12 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 contrololo, 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 Effectivenes 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

13 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 verifing del software. I programmi vengono eseguiti in modo simbolico (o astratto), lungo i cammini si progagano 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 affinchè un'esecuzione percorre un cammino lungo cui lo statement sia eseguito.

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

14 Manutenibilità, complessità ciclomatica

[L9]

La manutenzione di un sotware è 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 è mutato.
- **Manutenzione perfective :** manutenzione fatta per aumentare le prestazioni del SW.

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

MC = e - n + 2*p

dove:
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.

15 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.

16 Modello a Spirale

[L3.11]

Il modello a spirale è un modello di ciclo di vita di un sistema, è un modello di tipo risk-driven applicabile solo a sistemi "large scale ed è 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

Caratteristiche del modello a spirale:

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

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

17 Qualità totale, Qualità latente

[L5.57]

La qualità totale si estende al controllo di qualità di tutti i processi aziendali, la mentalità aziendale si sposta dalla ricerca del profitto alla ricerca della qualità nei singoli processi. L'azienda si prefigge di garantire e controllare la qualità del prodotto che verrà 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 qualità vengano applicati durante la produzione del prodotto e controllare che il prodotto finito rispetti quanto riportato nel quality plan redatto.

Per ottenere la qualità totale molte aziende si sono adoperate nell'utilizzare dei processi per il miglioramento.

18 Metodi di pianificazione

I modelli di processo utilizzabili sono i seguenti:

- **Fix & Code:** si costruisce la prima versione e si modifica fino a che il cliente non è soddisfatto del risultato finale, tale processo risulta molto caotico e poco preciso
- **Waterfall :** composto da fasi a compartimenti stagni, una volta finita una fase non si può tornare indietro. È un modello organizzato molto bene che produce per ogni sua fase un documento che traccia le linee guida per la fase successiva, se si vuole modificare il progetto è necessario andare a riscrivere i requisiti.
- **Do it Twice:** viene prodotto inizialmente un sistema grezzo per analizzare i costi e la fattibilità del progetto dopo di che viene iniziata la progettazione vera e propria con la finalità di sviluppare il prodotto, di solito viene seguito in seconda battuta il *Waterfall Model*. È possibile produrre un prototipo che può essere di due tipi : *Mock-up* o *Breadboard* rispettivamente riproducono le interfaccie e le funzioni principali del prodotto finito.
- **Modello iterativo evolutivo:** Viene fatta l'analisi dei requisiti e vengono definite in modo preciso le funzionalità del prodotto dopo di che viene prodotto un prototipo. Tale prototipo viene aggiornato per produrre il prodotto finito.
- **Modello Microsoft:** lascia spazio all'inventiva degli sviluppatori. Il progetto viene suddiviso in varie build, le build vengono assegnate a un team che le sviluppa e le testa molte volte. La particolarità di questo modello è la frequente sincronizzazione che viene fatta tra i team, giornalmente vengono prodotte delle daily build e testate.
- **Modello Trasformazionale:** Si parte dal requisito e si arriva al codice mediante trasformazioni.
- **Reuse Model:** prevede l'utilizzo di codice preesistente.

19 Modelli di processo

- Waterfall model
- Incremental Development
- Reuse
- V model

20 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 interfaccie, 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.

21 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, è indicato per l'integration testing.

A Definizioni

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