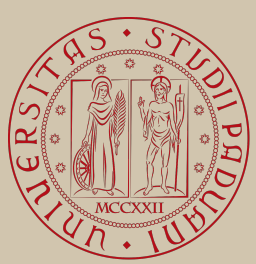
APPUNTI INGENGERIA DEL SOFTWARE 2024-25



Indice

Lista della immagini

Lista delle tabelle

1 Introduzione

1.1 Ingegneria del Software

L’ingegneria del software è l’applicazione di principi matematici e fisici per scopi pratici. Tali scopi sono spesso di interesse sociale e civile e non solo di consumo. E’ l’approccio sistematico, disciplinato e quantificabile allo sviluppo, mantenimento e ritiro software. Sistematico e disciplinato in quanto segue degli standard prefissati. Quantificabile perché si vuole che il costo di un processo sia noto a priori.

SwE è strettamente correlata alle seguenti discipline:

- Informatica
- Matematica
- Economia e Management
- Psicologia e sociologia

L’ingegnere del software deve assicurare la qualità del prodotto richiesta dai requisiti, massimizzando efficacia dei processi, contenendo i costi e massimizzando anche l’efficienza, minimizzando l’uso di risorse.

Ci sono più tipi di prodotti software, come i prodotti generici, stand alone prodotti da aziende e messi apertamente sul mercato, o prodotti specifici, commissionati da un particolare cliente.

Ogni prodotto ha un **ciclo di vita** che rappresenta tutti gli stati assunti dal prodotto nel corso del tempo. Durante questo ciclo, i prodotti sono soggetti a manutenzione che può essere:

- **correttiva:** corregge difetti
- **Adattiva:** adatta il prodotto a nuovi ambienti
- **Evolutiva:** aggiunge nuove funzionalità

1.2 Software Engeneer

Implementa parti di un sistema complesso conscio del fatto che altre persone potranno in futuro usarlo o modificarlo. E’ necessario per lui comprendere il contesto in cui è posto tale sistema.

E’ guidato da alcuni principi etici:

- Considerare la qualità come obiettivo primario;
- Scegliere i processi più adatti al progetto

- Aiutare il cliente a comprendere ciò di cui ha veramente bisogno
- Ridurre la distanza intellettuale tra il software e il problema da risolvere
- Essere proattivo nella ricerca degli errori
- Motivare, formare e far crescere le persone

1.3 Persone

Con il termine **stakeholder** si individua un soggetto influente nei confronti di un iniziativa economica. Un progetto software comprende diversi tipi di *stakeholders*, tra cui:

- *Business managment*: chi fissa gli obiettivi in termini di costi;
- *Project managment*: chi gestisce le risorse del progetto;
- *Dev Team*: chi implementa il prodotto. Qua lavora il *software engeneer*;
- *Clienti*: chi commissiona il prodotto;
- *End User* : chi usa il prodotto.

1.4 Progetto

Un progetto software è un insieme di attività di produzione di software. E' permeato dal principio "*by construction*", ovvero il prodotto è creato fin dall'inizio sapendo che funzionerà. Un progetto è una serie di passi precisi:

- *Pianificazione*: definizione degli obiettivi e delle risorse necessarie;
- *Analisi dei requisiti*:comprendere il problema e definire cosa bisogna fare per risolverlo;
- *Progettazione*: definire in forma concreta come risolvere il problema;
- *Implementazione o Realizzazione*: arrivare alla soluzione con il massimo grado di efficienza ed efficacia;
- *Verifica e Validazione*: garantire il funzionamento di ciò che si è prodotto;
- *Manutenzione*: garantire che il prodotto continui a funzionare nel tempo.

1.5 Processo

Struttura metodologia, normativa e strategica che caratterizza le attività di progetto. Suddivide le attività per obiettivi definendo come esse interagiscono tra di loro.

2 Processi Software

Un processo ingegneristico è un insieme di attivitò correlate atte a trasformare uno o più elementi di input in un prodotto in output. I processi software includono una pre ed una postcondizione e decompongono le attivitò in tasks. Le attività di un processo sono ripetute in modo iterativo secondo il principio del **riuso**.

I processi software sono descritti dallo standard ISO 12207 dove sono categorizzati in:
- **primari**;
- **di supporto**
- **organizzativi**

2.1 Processi Primari

Sono messi in atto dalle parti primarie di un propgetto, come l'acquirente, il fornitore, lo sviluppatore e il manutentore.

Sono definiti come:

- **Acquisizione**: definisce le attività svolte dal committente
- **Fornitura**: dfinisce le attività svolte dal fornitore
- **Sviluppo**: definisce le attività che costituiscono lo sviluppo del software.
- **Gestione operativa**: definisce le attivit- e i compiti dell'operatore del sistema.
- **Manutenzione**: definisce le attività di manutenzione del software.

2.1.1 Processo di sviluppo

Consistono nelle seguenti attività:

- Implementazione di processo: definito il modello di ciclo di vita e mappate le attività di sviluppo.
- Analisi dei requisiti di sistema: definizione dei requisiti del sistema.
- Progettazione del sistema: stabilita un'architettura top-level del sistema

- Analisi dei requisiti software: definizione dei requisiti software.
- Progettazione architetturale: viene progettata l’architettura software.
- Progettazione dettagliata: viene progettato il dettaglio del software.
- Codifica e testing
- Integrazione
- Software qualification testing: verificato che ogni componente rispetti i requisiti di qualità
- Integrazione del sistema
- System qualification testing: verificato che il sistema rispetti i requisiti di qualità
- Installazione
- Supporto all’accettazione

Le attività possono essere eseguite in maniera iterativa e non devono seguire uno specifico ordine temporale.

2.2 Processi di supporto

Questi vengono impiegati ed eseguiti da altri quando necessario. Contribuiscono al successo e alla qualità del prodotto. Questi sono:

- Documentazione
- Gestione della configurazione
- Quality assurance
- Verifica
- Validazione

2.3 Processi Organizzativi

Sono impiegati per migliorare costantemente la struttura organizzativa.

- Gestione dei processi
- Gestione delle infrastrutture
- Miglioramento dei processi
- Formazione

2.4 Processi, aziende e progetti

Non esistono processi migliori in generale, ma solo nello specifico. I processi vanno selezionati e adattati in relazione al

progetto, al contesto aziendale e il dominio di applicazione. Questo definisce un insieme di requisiti, terminologie e funzionalità comuni a tutti i programmi relativi ad una particolare area di sviluppo.

Stessi domini condividono gli stessi processi software:

- Proceso standard: riferimento di base condiviso dalle aziende di uno stesso dominio
- Processo definito: specializzazione di un processo srtandard adattato ai particolari bisogni di un'azienda.
- Processo di progetto: istanziazione di un processo definito.

La specializzazione dei processi richiede scelte accurate in cui si devono definire gli scenari di applicazione, le attivitò e i compiti specifici tra i processi specializzati.

Tra i fattori che influiscono nella specializzazione vi sono:

- dimensione e complessità del progetto
- rischi legati al dominio applicativo e alle tecnologie
- esperienza e competenze del team
- vincoli contrattuali

2.5 Principio di Miglioramento

L'organizzazione interna dei processi segue il principio del miglioramento continuo. Il **Ciclo di Deming** (PDCA) è un modello iterativo utilizzato in un ottica a lungo raggio.

E' diviso in 4 stadi:

- **Plan**: definizione degli obiettivi di miglioramento
- **Do**: esecuzione delle attività pianificate
- **Check**: verificazione dei risultati
- **Act**: azioni correttive

3 Ciclo di vita del software

Può essere visto come una macchina a stati, dove ogni stato rappresenta un preciso ggrado di misurazione del prodotto e dove ogni transizione richiede l'esecuzione di attività specifiche. Stati e transizioni hanno precise pre e postcondizioni determinate da vincoli, regole e strategie. La durata entro unno stato o in una transizione viene detta *fase*. Le fasi mostrano l'avanzamento nel tempo di un progetto.

3.1 Modelli di ciclo di vita

I modelli enfatizzano i processi chiave da attuare e le relazioni e interdipendenze logiche e temporali tra di essi. Il modello adottato pone vincoli su pianificazione e gestione del progetto. E' indipendente dai metodi e strumenti di sviluppo e precede la loro selezione.

3.1.1 Modello a cascata (Waterfall model)

Il miodello a cascata è un modello sequenziale nel quale il processo di realizzazione è strutturato in una sequenza strettamente lineare di fasi in cui ritornare a quella precedente è proibito. Al completamento di ogni fase è prodotta la documentazione che permette al cliente di analizzare e approvare l'avanzamento del progetto. La fase successiva non può iniziare fintanto che la fase precedente non è completata. Questo modello ha origine nell'industria manufatturiera dove i cambiamenti in corso d'opera sono gravosi e costosi.

Ogni fase di questo modello è definita in termini di:

- Attività e prodotti *input* e *output*
- Struttura e contenuto documentazione
- Responsabilità e ruoli coinvolti
- Scadenze di consegna

Questo modello porta con se dei vantaggi: pre- e postcondizioni sono ben note e rispettaste rendendolo facilmente valutabile nei costi e nelle risorse necessarie. Inoltre è sempre possibile sapere con precisione in che stato si trova il progetto ad un dato istante.

Come contro, questo modello può risultare inflessibile. Inoltre, non producendo prototipi, il cliente riceve il prodotto solo alla fine del processo di sviluppo.

Una scarsa conoscenza delle tecnologie da utilizzare da parte degli *stakeholders* può causare un sostanziale cambiamento dei requisiti in corso d'opera. Per questo è consigliabile l'utilizzo del modello a cascata nel caso in cui i requisiti siano ben definiti e stabili fin dal primo momento.

3.1.2 Modello incrementale

Il modello incrementale basa lo sviluppo software su multipli e successivi rilasci, dove ogni rilascio implementa una funzionalità aggiuntiva. I requisiti sono classificati e ordinati in base alla loro importanza: i primi incrementi mirano a soddisfare i requisiti più importanti.

Questo modello ha il vantaggio di sviluppare le funzionalità essenziali entro le prime fasi. Tali funzionalità sono di aiuto nella pianificazione degli incrementi successivi e, poiché passano attraverso multiple fasi di verifica aumentano la stabilità ad ogni fase.

3.1.3 Modello evolutivo

E’ un modello iterativo che aiuta a rispondere a bisogni non prevedibili dall’inizio. Implica l’attraversamento di stessi stadi durante il ciclo di vita e richiede spesso il rilascio e il mantenimento di più versioni nello stesso momento. L’approccio evolutivo tende a sviluppare in tempi brevi una prima implementazione a partire da specifiche asratte. Il modello suddivide il ciclo di vita in piccoli cicli a cascata incrementali e alla fine di ogni sottociclo viene rilasciato il prodotto in versione beta dove gli utenti possono rilasciare dei feedback. Ogni versione eredita dalla precedente le features migliori raffinandole in base ai feedback.

3.1.4 Modello a spirale

Il modello a spirale, proposto da Barry W. Bohem nel 1988, si focalizza sulla gestione dei rischi. Il modello rappresenta il processo come una spirale: ogni ciclo nella spirale equivale al completamento di una fase. Questo modello richiede una fotte interazione tra fornitore e cliente.

Ogni ciclo della spirale è diviso in 4 settori. Più granfe è la spirale, più risorse sono state spese e più rischi sono stati gestiti. I settori sono:

- Determinazione degli obiettivi
- Valutazione e riduzione dei rischi
- Sviluppo e validazione
- Pianificazione del successivo ciclo

3.1.5 Modello a componenti

Riconsidera il problema basandosi su risorse che possono essere riusate per risolverlo. Crea il prodotto finito assemblando componenti software già esistenti.

3.1.6 Modelli Agili

Nei primi anni '90 vi era una visione diffusa secondo la quale il modo migliore per ottenere un buon prodotto fosse attraverso fasi di analisi, sviluppo e documentazioni accurate. Questo approccio, applicato a prodotti di piccole-medie dimensioni, fa salire l'overhead in maniera esponenziale. L'insoddisfazione per questo metodo ha portato alla creazione delle metodologie agili. Queste permettono ai team di sviluppo di concentrarsi sul software piuttosto che sulla progettazione e documentazione. Sono ideali quando i rquisiti di progetto cambiano frequentemente.

I metodi agili si basano su 4 principi fondamentali:

- Individui e interazioni prima di processi e strumenti
- Meglio un software funzionante che una documentazione esaustiva
- Collaborazione con il cliente piuttosto che negoziazione contrattuale
- Rispondere al cambiamento piuttosto che seguire un piano

Il concetto di ***user story*** è centrale nei modelli agili: espone un bisogno dell'utente che il prodotto deve soddisfare. E' un documento costituito da più frasi dell'utente riguardo al bisogno che deve essere soddisfatto e la strategia per soddisfarlo.

Scrum

Scrum definisce un generico framework focalizzato sulla gestione di un modello di sviluppo iterativo.

Le fasi centrali dello Scrum sono gli *sprint*. Uno sprint è un'unità di pianificazione nel quale il lavoro da effettuare viene valutato e sono selezionate le *features* da implementare. Al termine di uno sprint si implementa la funzionalità nuova e il prodotto viene consegnato al cliente. Il progresso raggiunto viene utilizzato come starting point dello sprint successivo. Ogni sprint dura 2-4 settimane. All'inizio di ogni sprint si tiene quello che viene chiamato *product backlog*, dove vengono elencati tutti i requisiti da soddisfare per lo sprint corrente. Il cliente è coinvolto in questa riunione e può introdurre nuove features.

In Scrum sono presenti 3 ruoli principali:

- *Product Owner*: responsabile della definizione dei requisiti e delle priorità
- *Scrum Master*: responsabile del processo e del team
- *Team*: responsabile dello sviluppo del prodotto

3.2 Standard di processo

Gli standard di ciclo di vita possono essere generali o settoriali.

Rappresentano:

- Modelli di azione quando usati per definire e imporre procedure
- modelli di valutazione quando usati per valutare e confrontare best practices.

4 Gestione di progetto

E’ una parte fondamentale di SwE. Nella maggior parte dei casi, gli obiettivi di un progetto sono:

- Rispettare i tempi
- Rispettare i costi
- Rispettare i requisiti
- Rispettare la qualità

SwE si differenzia dalle altre ingegnerie in modi che la rendono particolarmente impegnativa:

1. il prodotto è intangibile
2. i progetti grandi sono spesso *one-off* , quindi anche gli ingegneri con esperienza possono incontrare problemi
3. i processi software sono variabili e specifici per ogni azienda.

4.1 Gestione dei rischi

Riguarda l’anticipazione dei rischi che potrebbero intaccare la pianificazione delle attività o la qualità del prodotto.

Ci sono tre categorie di rischi:

- **Rischi di progetto:** influenzano la pianificazione o le risorse
- **Rischi di prodotto:** influenzano la qualità del prodotto
- **Rischi d’impresa:** influenzano l’azienda sviluppatrice o fornitrice.

La gestione dei rischi è particolarmente importante nei progetti data l'incertezza che li caratterizza, causata spesso dai requisiti poco chiari e variabili nel tempo.

Il processo di gestione dei rischi si costituisce di 4 fasi:

1. Identificazione dei rischi
2. Analisi dei rischi
3. pianificazione
4. Monitoraggio e controllo

E' un processo iterativo e continua durante tutto il progetto. I risultati vanno documentati nel piano di progetto, inclusa l'analisi e le linee guida su come affrontare i rischi.

4.2 Ruoli

4.2.1 Analisti e progettisti

Gli ***analisti*** conscono il dominio del problema e sono esperti, pertanto influiscono pesantemente sul successo del progetto. Raramente seguono il progetto per tutto il ciclo di vita.

I ***progettisti*** hanno conoscenza tecnica e tecnologica. Sono i responsabili degli aspetti tecnici del progetto, e seguono il progetto fino alla fine.

4.2.2 Programmatori e Verificatori

I ***programmatori*** sono responsabili dello sviluppo e manutenzione del prodotto. Hanno visione e responsabilità limitate.

I ***Verificatori*** partecipano all'intero progetto e hanno competenze tecniche e conoscono gli standard. Si occupano della qualità del prodotto.

4.2.3 Responsabile

Rappresenta l'intero progetto nei confronti del fornitore e del cliente. Centralizza la responsabilità e partecipa al progetto nella sua interezza. Ha responsabilità nella pianificazione e nella gestione delle risorse umane oltre che nel controllo e coordinazione del progetto. Deve possedere conoscenza tecnica e abilità nel comprendere e anticipare l'evoluzione del progetto.

Compie le seguenti attività:

• Pianificazione
• Comunicazione

• Gestione dei rischi
• Gestione del personale

4.2.4 Amministratore

Gestisce e controlla l’ambiente di lavoro oltre che amministrare le risorse e le infrastrutture. Gestisce la documentazione e i VCS.

4.3 Pianificazione di progetto

E’ svolta dal responsabile e consiste nel suddividere il lavoro in più task e assegnarle ai membri del dev team.

La pianificazione avviene in tre momenti del ciclo di vita:

1. Al momento della proposta al committente
2. Durante la fase di avvio del progetto
3. Periodicamente durante il progetto

4.3.1 Project scheduling

E’ il processo nel quale si decide come il lavoro verrà suddiviso in tasks e come e quando queste dovranno essere svolte. E’ necessario stimare il tempo di calendario necessario al completamento di ciascun compito oltre allo sforzo richiesto.

La rappresentazione dello schedule del progetto può servirsi di diversi strumenti. I più comuni sono:

• Diagrami di **Gantt**: mostrano chi è responsabile per ogni attività
• Program Evaluation and Review Technique (PERT),
• Work Breakdown Structure (WBS)

Le attività di progetto sono l’elemento di pianificazione di base. Ogni attività è caratterizzata da:

• Durata
• Stima del lavoro necessario
• Deadline

Diagramma di Gantt

Permette di organizzare le attività di progetto in funzione del tempo. Ogni attività è rappresentata da una barra dove la posizione e la lunghezza ne indicano l’inizio e la durata.

Mostra istantaneamente cosa deve essere fatto e quando, permettendo di capire a vista molte informazioni, tra cui le attività sovrapposte e la durata complessiva del progetto.

PERT

E' uno strumento statistico utilizzato nella gestione di progetto che permette di analizzare le attività necessarie al completamento di un progetto. Organizza e dispone le attività secondo relazioni di dipendenza temporale.

Utilizza i seguenti parametri:

- PERT event: punto di inizio o fine di un'attività
- PERT activity: rappresenta un'attività
- Float o Slack: relativo ad una task, rappresenta quanto ritardo può avere quella task

Critical Path: cammino più lungo possibile dall'evento iniziale a quello finale.

4.3.2 Allocazione risorse

Le attività vanno assegnate ai ruoli, e i ruoli alle persone. Importante è non sovrastimare (o sottostimare) la quantità di lavoro richiesto.

4.3.3 Stima costi progetto

Stimare la quantità di lavoro richiesto non è semplice. Le aziende possono avvalersi di due tecniche di stima:

- Basate sull'esperienza
- Modellazione algoritmica

Una metrica tipicamente usata per misurare il lavoro necessario in termini di tempo è il **tempo/persona**. I fattori che influenzano la stima sono:

- Dimensione del progetto
- Esperienza nel dominio applicativo
- Tecnologie adottate
- Ambiente di sviluppo
- Livello di qualità richiesto dai processi

Constructive Cost Model (CoCoMo)

E’ un modello empirico basato sulla raccolta dati nel tempo da un largo numero di progetti software. Misura le risorse necessarie in **mesi/persona**:

$$\frac{M}{P} = C \times D^S \times M$$

Dove:

- C: Fattore complessità
- D: Dimensione del prodotto stimata in KDSI (Kilo Delivered Source Instructions)
- S: Fattore di complessità
- M: Moltiplicatori di costo $\prod_i \alpha_i$ dove α_i sono attributi i cui valori cadono entro intervalli fissati.

Nella versione base, CoCoMo assume l’utilizzo di un modello sequenziale.

Si evidenziano 3 gradi di complessità:

1. Semplice: una singola persona è in grado di comprendere tutto il prodotto nel suo insieme
2. Moderata: Una persona è in grado di comprendere il prodotto isolandolo per componenti
3. Embedded: il prodotto interagisce con componenti esterni

4.4 Piano di progetto

E’ un documento che espone le risorse disponibili nel progetto, il work breakdown e il calendario delle task.

Viene costantemente aggiornato e ha lo scopo di organizzare le attività in modo da permetterne l’efficace valutazione.

Nel piano di progetto vengono stabilite le milestone e punti critici del progetto.

Solitamente segue la struttura:

- Introduzione
- Organizzazione del progetto
- Analisi dei rischi
- Risorse disponibili e necessarie
- Work Breakdown
- Calendario attività
- Controllo e rendicontazione

5 Amministrazione di progetto

L’amministrazione di progetto è l’attività svolta dall’amministratore, atta ad equipaggiare e festire l’ambiente di lavoro.

L’amministratore non efettua scelte di gestione, ma mette in pratica scelte tecnologiche.

Le sue attività comprendono:
- Redazione e manutenzione di regole e procedure
- Reperimento, organizzazione, gestione e manutenzione delle risorse

Infrastruttura: Insieme di strumenti che determinano il modus operandi

Servizio: Mezzo o strumento che permette all’utilizzatore di raggiungere un obiettivo senza occuparsi dei costi e dei rischi

5.1 Documentazione

La documentazione ha diverse funzioni:

- Comunicazione tra membri
- Repository di informazioni per i mantenitori
- Pianificazione e controllo
- Guida all’utilizzo

Documentazione di processo: Relativa ai processi di sviluppo e manutenzione del software.

Documentazione di prodotto: Descrive il prodotto e si divide in documentazione di sistema e per l’utente.

Disponibilità e diffuzione: Un documento è utile solo se è:
- sempre disponibile
- chiaramente identificato
- corretto nei contenuti
- verificato e approvato
- aggiornato, datato e versionato

La diffusione della documentazione deve essere strettamente controllata identificando chiaramente i destinatari.

Ogni documento deve avere una lista di distribuzione che viene gestita dall’amministratore.

5.2 Ambiente di lavoro

Rappresenta le persone, ruoli, procedure e infrastrutture necessarie per mettere in opera i processi di produzione. La sua qualità determina la produttività e influisce sulla qualtià.

Deve essere:

• Completo
• Ordinato
• Aggiornato

5.2.1 Infrastruttura

E’ una serie di elementi strutturali interconnessi, servizi e strimenti che forniscono un framework. Comprende hardware e software.

5.3 Gestione di configurazione

Un prodotto è composto da diverse parti separate, unite secondo regole che costituiscono la configurazione. La gestione di configurazione è l’attività che si occupa di mantenere la configurazione del prodotto.

Configuration Item: Tutto ciò che è sotto il controllo di configurazione. Ogni item ha un’identità unica e si trova in più versioni.

La gestione di configurazione identifica e controlla i configuration items definendo quali compongono il prodotto e come sono aggregati nel processo di build.

Baseline: La gestione di configurazione identifica e controlla le baseline, che sono collezioni di configuration items. Le baseline sono utili per il controllo delle modifiche e per la valutazione delle prestazioni. Sono tipicamente associate a milestone.

Le attività che compongono il processo di gestione di configurazione sono:

• Identificazione della configurazione
• Gestione dei cambiamenti
• Controllo versioni
• Build
• Release managment

Identificazione della configurazione

Si occupa di impostare e mantenere le baseline. Stabilisce e mantiene in maniera incrementale i configuration items durante il ciclo di vita. Baseline ben definite garantiscono riproducibilità, tracciabilità e analisi del processo.

Gestione dei cambiamenti

Il processo di gestione dei cambiamenti analizza costi e benefici delle proposte di modifica e approva quelle significative.

Queste proposte possono provenire da:

- Utenti (Bug Report)
- Sviluppatori
- Competizione con altre aziende

Le proposte di modifica seguono uno specifico processo di analisi e implementazione. Ogni richiesta deve essere presentata tranmite un CRF (Change Request Form) nel quale vengono memorizzate decisioni e raccomandazioni riguardanti la modifica, il costo e le date di ogni decisione.

Controllo versione

Ogni componente si trova in più versioni. Il controllo versione si occupa di tenere traccia delle differenti versioni di ogni componente del sistema e fa in modo che il lavoro del developer non interferisca con quello degli altri.

Il controllo versione riguarda la gestione di *codeline* e *baseline*.

5.4 Norme di progetto

Costituiscono le linee guida per le attività di progetto:

- Organizzazione e convenzione sugli strumenti di sviluppo
- Organizzazione della comunicazione e cooperazione
- Attività di Codifica
- Gestione cambiamenti

Possono essere regole o raccomandazioni. Il contesto definisce la portata di tale norma: troppe regole sono difficili da attuare.

6 Qualità Software

Con qualità si intende l’insieme delle caratteristiche di un’entità che ne determinano la capacità di soddisfare esigenze espresse e implicite.

Ci sono diverse visioni su cui si valuta la qualità:

- Intrinseca: conformità ai requisiti, idoneità all’uso
- Relativa: soddisfazione cliente
- Quantitativa: misura per confronto

Secondo ISO/IEC 9001 per qualità si intende «capacità di un prodotto di soddisfare i requisiti».

Vi sono due aspetti della qualità: considerazioni oggettive, indipendenti dall’uomoo e l’esperienza dell’utente.

La qualità software non rigurarda solo l’aderenza ai requisiti ma anche dai suoi attributi non funzionali.

6.1 Sistema di qualità

Struttura organizzativa, processi e risorse messe in atto per il perseguimento della qualità

6.1.1 Documentazione SGQ

La normativa prevede della documentazione per la realizzazione dei SGQ. Il manuale della qualità è un documento che definisce il sistema di gestione della qualità di un’organizzazione. Esprime una visione orizzontale ad alto livello integrandosi con le procedure aziendali.

Il piano della qualità è il documento che definisce gli elementi del SGQ e le risorse che devono essere applicate in uno specifico caso. COncretizza il manuale della qualità a livello di progetto.

Il piano di qualità accerta la disponibilità di analisi, pianificazione e risultati nonché tracciabilità di soluzioni.

6.1.2 Gestione della qualità

Sono le attività del sistema di qualità pianificate e attuate perché prodotti, servizi e processi soddisfino i requisiti. Comprende le attività di pianificazione, quality assurance e controllo della qualità.

- Pianificazione della qualità: definisce gli obiettivi di qualità e i processi necessari per raggiungerli.

- Quality assurance: sistematica misurazione e monitoraggio dei processi per prevenire l’errore.
- Controllo qualità: attività atte ad esaminare specifici prodotti per determinarne la loro conformità agli standard.
- Miglioramento dei processi: attività che cercano di migliorare l’efficacia e l’efficienza dei processi.

6.2 Standard di Qualità

Gli standard di qualità giocano un ruolo molto importante nella gestione della qualità del software:

- Catturano e rappresentano le conoscenze
- Supportano la continuità nel momento in cui il lavoro viene continuato da altri.

Un uso errato degli standard può portare effetti negativi, è importante quindi che le norme siano snelle e chiaramente comprensibili.

- Standard poco comprensibili → percepibili come irrilevanti
- Attuazione cieca → eccessi di burocrazia
- Niente supporto di strumenti informatici → tediose attività manuali

Definizione e valutazione di qualità

Gli standard forniscono modelli e metriche per la definizione e la valutazioone della qualità eliminando percezioni soggettive e convertendo le proprietà astratte e poco chiare in valori quantificabili e misurabili.

- Definizione, modello di quailità: catalogazione sistematica delle caratteristiche rilevanti
- Valutazione, metriche: definizione di metriche per la valutazione.

6.2.1 Definizione della qualità: modelli di Qualità

I modelli di qualità classificano la qualità di un software in un insieme di caratteristiche e costiuiscono un modello comune per committenti, uniformando la percezione e la valutazione della qualità. Esempi di modelli di qualità sono il modello di Bohem e l’ISO/IEC 9126:2001. Definiscono valutazioni secondo più punti di vista:

- Visione dell’utente

• Visione della produzione
• Visione della direzione

6.2.2 Valutazione della qualità: metriche

Si intende il proceso attrraverso cui vengono assegnati valori ad attributi di una entità su scala predefinita. Definiamo metrica qualsiasi tipo di misurazione riguardante un sistema, un processo o un documento. Le metriche costituiscono uno strumento di quantificazione del prodotto.

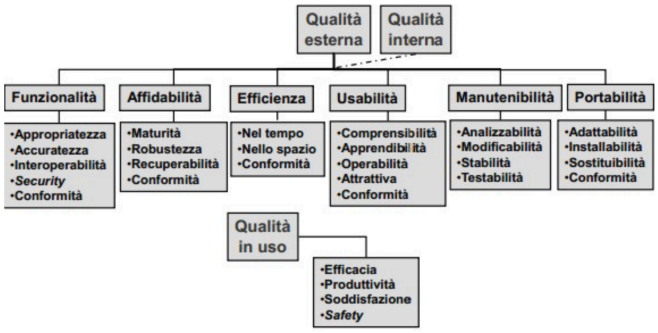


Figura 1: Caratteristiche di qualità

Attributi interni ed esterni

Le metriche misurano attributi di qualità interni del sistema ma spesso siamo interessati a quelli esterni. Questi sono difficili da misurari dato che dipendono da fattori soggettivi all'esperienza utente. Per farlo, occorre misurare gli attributi inyterni e assumere che esista una relazione con gli attributi esterni che si voglio valutare.

Per una misurazione corretta e utile alla valutazione esterna occorre che:

• l'attributo sia misurato correttamente
• il valore sia correlato con quello che si sa misurare
• la relazione sia esprimibile e validabile formalmente.

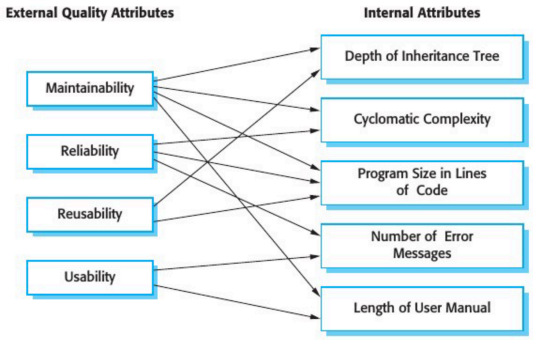


Figura 2: Attributi interni ed esterni

6.2.3 ISO 9000

ISO 9000 è una famiglia di standard di qualità cpn l'obiettivo di integrare un sistema di qualità in azienda. Divide i processi in:
- Responsabilità della direzione
- Gestione risorse
- Realizzazione prodotto
- Misura, analisi, miglioramento

6.3 Qualità di processo

E' direttamente collegata alla qualità del processo impiegato per crearlo. Per ottenere qualità di processo serve:

- Definire il processo
- Controllare il processo
- Usare buoni strumenti di valutazione

6.3.1 Strumenti di valutazione dei processi

- Software Process Assesment & Improvment (SPY): Valutazione oggettiva dei processi di organizzazione
- Capability Maturity Model Integration (CMMI): modello per la valutazione uniforme dei fornitori
- Software Process Improvment Capability dEtermination (SPICE): nato per armonizzare SPY e ISO/IEC 12207 e ISO 9000

CMMI

E' un insieme strutturato di elementi che descrivono le caratteristiche di processi efficaci. Costituisce la base concettuale per la valutazione e il miglioramento dei processi.

- Capability: misura di quanto un processo è adeguato in termini di efficacia ed efficienza
- Maturity: quanto il sistema di processi è ben definito e gestito
- Model: insieme di requisiti di Qualità
- Integration: Architettura di integrazione delle diverse attività

SPICE

E' un modello di riferimento per modelli di maturità, usato per valutare il livello generale di maturità dell'azienda.

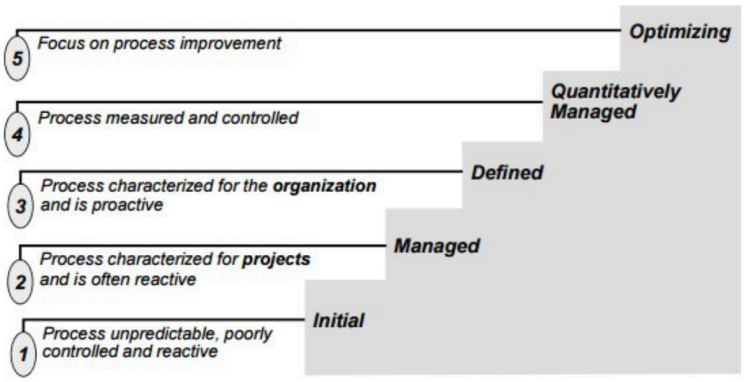


Figura 3: CMMI

Definisce una dimensione di processo e di capacità. La dimensione di processo divide i processi in 5 categorie:

1. Customer supplier
2. Engineering
3. Supporting
4. Management
5. Organization

Per ogni processo lo standard definisce un livello di capacità sulla seguente scala:

1. 0, Incomplete
2. 1, Performed
3. 2, Managed
4. 3, Established
5. 4, Predictable
6. 5, Optimizing

Lo standard specifica poi 9 attributi di processo usati per valutare la capacità di processo, oltre che una metodologia di valutazione con le seguenti operazioni:

• Identificazione portatori di interesse
• Scelta tra valutazione e miglioramento
• Definizione della portata

7 Ingegneria dei requisiti

I requisiti sono descrizioni delle funzionalità di un sistema, dei seriviz che fornisce e dei vincoli delle sue operazioni.

Il glossario IEEE offre una definizione basata sul concetto di capability secondo cui un requisito è:

- una condizione necessaria per risolvere un problema o raggiungere un obiettivo
- Una condizione che deve essere soddisfatta o posseduta da un sistema
- Una descrizione documentata di una condizione

L’insieme di attività atte a delineare, analizzare, documentare e verificare i requisiti è chiamato ingegneria dei requisiti.

7.1 Tecniche di analisi

- Analisi dei bisogni (Cosa so o leggo che devo fare) e delle fonti (da dove scaturiscono altri bisogni). I bisogni possono essere espliciti (capitolato) o impliciti (dominio applicativo).
- Interazione con il cliente: instaurare un rapporto con il cliente definito intervista.
- Studio del dominio: acquisizione delle competenze necessarie ad un’attività di analisi esaustiva
- Discussioni creative: coinvolgono tre figure:
  - Un gruppo di persone che discuta il problema
  - un facilitatore che aiuti la discussione (unbiased e non coinvolto nella discussione)
  - Una persona che tenga la minute della discussione
- Prototipazione: interna a vantaggio del foritore o esterna per arricchire il rapporto con il cliente
- Uso del glossario: raccoglie e definisce i termini del dominio
- Classificazione dei requisiti: insieme organizzato e strutturato dei requisiti
- Modellazione concettuale del sistema: decomposizione concettualr del sistema dal punto di vista esterno
- Assegnazione dei requisiti a parti distinte del sistema
- Negoziazione con il committente: per chiarire i requisiti e le aspettative

Requirements Technology baseline

Raggiunta quando vengono definiti tutti i requisiti. Determina la fine dell’attività di analisi e l’inizio della progettazione.

7.2 Classificazione dei requisiti

E’ fondamentale organizzare i requisiti in maniera strutturata suddividendoli in livelli di dettaglio per comunicare informa-

zioni sul sistema a diverse categorie di lettori. Si distingue tra **requisiti utente** e **di sistema**:

- Utente: affermazioni in linguaggio naturale e diagrammi di quali servizi il sistema offre all'utente
- Sistema: descrizioni dettagliate delle funzioni, servizi e vincoli del sistema

I requisiti di sistema si dividono in:

- funzionali: specificano le funzionalità
- non funzionali: descrivono le caratteristiche del sistema nella sua interezza.

Attributi di prodotto e di processo

- Prodotto: definiscono le caratteristiche richieste al sistema ed esprimono i requisiti funzionali
- Processo: definiscono cinxoli sulla conduzione dei processi di produzione. Esprimono i requisiti non funzionali

Alcuni requisiti derivano, direttamente o indirettamente, da attributi di prodotto o processo.

Requisiti non funzionali

Specificano criteri per giudicare le operazioni del sistema. Rispondono alla domanda «come il sistema lo fa».

Si dividono in:

- prestazionali
- di qualità
- di vincolo

possono provenire da caratteristiche richieste dal prodotto, dall'azienda o da fonti esterne:

- requisiti di prodotto: specificano o vincolano il comportamento del software
- requisiti aziendali: ad alto livello, derivati da politiche o procedure aziendali
- requisiti esterni: derivati da fattori esterni al sistema e al processo di sviluppo

Spesso i requisiti non funzionali sono più critici: un requisito non funzionale non soddisfatto non è facilmente aggirabile.

Classificazione per utilità

I requisiti hanno una diversa utilità, e vanno categorizzati di conseguenza:

- Obbligatori
- Desiderabili
- Opzionali

7.3 Documentazione

L'attività di analisi si concretizza nei documenti di **Specifica Tecnica** e **Analisi dei requisiti**. La documentazione è necessaria per rendere i requisiti discutibili e non ambigui utilizzando diagrammi e formalismi. I documenti riguardanti i requisiti sono:

- Analisi dei requisiti
- Studio della fattibilità

7.3.1 Studio fattibilità

precede l'analisi dei requisiti e si occupa di valutare la fattibilità del progetto in termini di rischi, costi e benefici.

- **Fattibilità tecnico-organizzative**: valutare se si habno le competenze e gli strumenti necessari
- Rapporto **costi / benefici**: valutare il rapporto tra costo di produzione e la redditività
- Individuazione **rischi**
- Valutazione **scadenze temporali**
- Valutazione **alternative**
  - scelte architetturali
  - strategie realizzative
  - strategie operative

Lo studio di fattibilità è un'attività preliminare che non può impiegare troppo tempo, deve essere rapida.

7.4 Software Requiurements Specification

Lo standard IEEE 830-1998 definisce la struttura e le caratteristiche del documento SRS, che ha le seguenti caratteristiche:

- **Correttezza**: ogni requisito verrà soddisfatto
- **Non Ambiguità**: Ogni requisito ha un'unica interpretazione indipendente dal contesto
- **Completezza**: un documento SRS è completo se e solo se:
  - i requisiti sono tutti e soli quelli necessari e sufficienti

  - Definisce le risposte del sistema ad ogni possibile classe di input
  - Contiene didascalie complete peer figure tabelle e diagrammi
- **Atomicità**: requisiti elementari sono facilmente tracciabili
- **Coerenza**: i requisiti non sono in conflitto tra di loro
- **Ordine** per importanza e/o stabilità
- **Verificabilità**: ogni requisito è verificabile
- **Modificabilità**: La struttura o lo stile del documento devono essere tali da permettere modifiche
- **Tracciabile**: ogni requisito deve identificare la fonte ed essere identificato da un codice univoco

Un corretto SRS limita il *range* di soluzioni progettuali ma non ne specifica alcuna.

Requisiti Verificabili

Un requisito è verificabile $\Leftrightarrow \exists$ una procedura dal costo finito che permett di verificare che il prodotto soddisfi il requisito.

Chi impone un requisito deve sapere anche come accertare il soddisfacimento.

Tipo requisito	**Modalità di verifica**
Requisiti funzionali	test, dimostrazione, revisione
Requisiti prestazionali	misurazione
Requisiti qualitativi	verifica *ad hoc*
Requisiti dichiarativi	revisione

Tabella 1: verifica requisiti

7.4.1 Struttura del documento

Lo standard IEE 830-1998 suggerisce la seguente struttura:
- Introduzione
- Descrizione generale
  - Funzioni del prodotto
  - Caratteristiche degli utenti
  - Vincoli generali
  - Assunzioni e dipendenze
- Specifica dei requisiti
  - Requisiti utenti
  - Architettura del sistema
  - Requisiti di sistema
- Appendici

7.5 Gestione e manutenzione dei reqiisiti

I requisiti cambiano nel tempo. Serve quindi un insieme di regole e procedure per gestire questi cambiamenti valutandone la fattibilità e l'impatto sul progetto. I processi di gestione della configurazione e dei cambiamenti hanno un ruolo chiave in questo contesto.

Tracciamento

I requisiti devono essere tracciati verso le fonti e le componenti architetturali con il supporto di strumenti automatizzati. Questo permette di sapere quale requisito stiamo soddisfacendo con il lavoro in atto. Rappresentando il piano di attività come un grafo direzionato e aciclico, il trcciamento comunica il motico di ciascun arco.

7.6 Stato di progresso per SEMAT

- **Conceived**: committente identificato e stakeholders vedono opportunità per il progetto
- **Bounded**: i bisogni sono chiari e i meccanismi di gestione fissati
- **Coherent**: i requisiti sono classificati e quelli essenziali sono chiari
- **Acceptable**: i requisiti definiscono un sistema soddisfacente
- **Addressed**: il prodotto soddisfa i principali requisiti a punto di poter essere rilasciato
- **Fulfilled**: il prodotto soddisfa abbastanza requisiti per la piena approvazione degli stakeholders

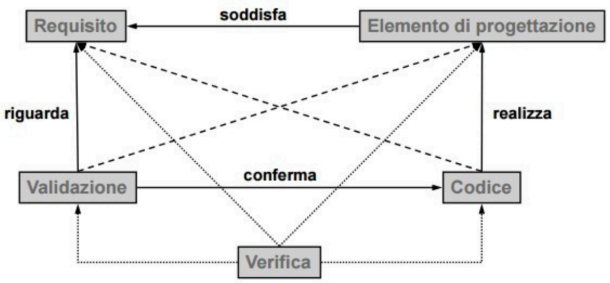


Figura 4: Tracciabilità a livello progetto

8 Progettazione software

Per perseguire la correttezza si progetta prima di produrre.

E’ necessario per:

• Governare la complessità
• Organizzare e ripartire le responsabilità
• produrre in efficienza ed efficacia

Mentre l’analisi segue un processo **investigativo**, la progettazione segue un approccio **sintetico**.

8.1 Architettura software

L’attività di progettazione fissa l’architettura del prodotto definita come:

• decomposizione del sistema in componenti
• organizzazione dei diversi componenti
• interfaccie necessarie all’interazione tra componenti
• paradigmi di composizione delle componenti

L’architettura software è importnte perché influisce sulle caratteristiche non funzionali del sistema.

Una buona architettura facilita il successo del prodotto:

• impiega componenti con specifica chiara e coesa
• realizzabile con risorse date e costi fissati
• struttura modulare che facilita cambiamenti

8.1.1 Perché l’architettura è importante

• Impone vincoli sull’implementazione
• Detta la struttura organizzativa
• Fornisce una descrizione sulla qualità del sistema
• Sono modelli riusabili e trasferibili

8.1.2 Qualità architetturali

Le seguenti sono qualità da ricercare in un’architettura:

• Sufficienza: capacità di soddisfare tutti i requisiti
• Comprensibilità: comprensibile agli stakeholders
• Modularità: suddivisione in parti chiare e distinte
• Robustezza: capacità di sopportare input diversi
• Flessibilità: permettere modifiche a costo contenuto
• Riusabilità: parti riutilizzabili
• Efficienza
• Affidabilità: funziona bene

- Disponibilità: necessita di tempo ridotto o nullo per manutenzione offline
- Security
- Safety
- Semplicità: ogni parte contiene il necessario e nulla più
  - rasoio di Occam
- Information Hiding
  - Diminuisce l'accoppiamento
  - cresce la manutenibilità
  - aumentano le opportunità di requisito
- Coesione: le parti che collaborano hanno gli stessi obiettivi
- Basso accoppiamento

Accoppiamento

E' una proprietà esterna dei componenti che indica quanto essi si usano a vicenda. Vi sono due metriche utili:

- *Structural* ***fan in***: componente utilizzato. E' indice di utilità e va ***massimizzato***
- *Structural* ***fan out***: componente che dipende. Indice di dipendenza e va ***minimizzato***

Una buona progettazione presenta alto ***fan in*** e basso ***fan out***.

8.1.3 Decomposizione architetturale

Può seguire due approcci:

- Top-down: decomposizione di problemi in sotto-problemi
- Bottom-up: costruzione di parti e assemblaggio, stile Oject-oriented

Solitamente si predilige un approccio intermedio

8.1.4 Progettazione per riuso

Il riuso impiega prodotti già esistenti minimizzando il costo realizzativo. Questo porta alcuni problemi:

- progettare *per* riuso è difficile: necessario anticipare bisogno futuri
- progettare *con* riuso non è immediato: minimizzare modifiche alle componenti

Il riuso è puro costo nel breve periodo e risparmio nel lungo.

8.2 framework

Un *franework* è un insieme integrato di componenti sofwtare prefabbricate.

- Sono *bottom-up* perché è codice già sviluppato e la progettazione avviene da una base fornita.
- Sono *top-down* perché impongono uno stile architetturale

Utili come base riusabile di diverse applicazioni in un dato dominio.

8.3 Design pattern

Offrono soluzioni progettuali a problemi ricorrenti. Sono l'equivalente architetturale degli algoritmi.

8.3.1 Pattern architetturali

Costituiscono soluzioni fattorizzate per problemi di progettazione ricorrenti. Sono determinati da:

- Descrizione del tipo di elementi dell'architettura
- Configurazione topologica degli elementi
- Insieme di vincoli semantici
- Insieme di meccanismi di interazione

Un pattern architetturale fornisce un'utile visione d'insieme del sistema. Impone inoltre vincoli utili sull'architettura e quindi sul sistema.
Un aspetto fondamentale dei pattern è che esibiscono attributi e qualità note e rappresentano soluzioni a problemi di diverse categorie.

La soluzione fornita deve:

- riflettere il contesto
- essere credibile

Alcuni esempi sono:
- modello client-server
- modello event-driven
- architettura MVC (Model View Controller)

8.4 Ricetta generale

Durante la progettazione è importante:

- Dominare la complessità del sistema
- Riconoscere le componenti terminali

- Creare bilanciamento

8.5 Progettazione di dettaglio

Riguarda le seguenti attività

- Definizione dei moduli: ogni modulo deve avere dimensione atomica e costituire un carico di lavoro assegnabile ad un singolo developer
- Specifica delle unità come insieme di moduli
- Assegnare unità a componenti
- Produrre la documentazione necessaria
- Definire gli strumenti necessari per i test di unità

8.6 Documentazione: Software Design Document

Lo standard IEEE 1016-1998 definisce la struttura e le caratteristiche del documento SDD.

8.6.1 Stato progresso per SEMAT

- Architecture Selected: selezione architettura tecnicamente adatta
- Demonstrable: dimostrazioni delle caratteristiche principali dell’architettura
- Usable: il sistema è utilizzabile con le caratteristiche desiderate
- Ready: la documentazione per l’utente è pronta

9 Verifica e validazione

• **Verifica**: Did i build the right system?
• **Validazione**: Did i build the system right?

La risposta ad entrambe richiede una prova oggettiva e non deve provenire da percezioni soggettive. Una delle caratteristiche di SwE è la quantificabilità.

Verifica

Fornisce prove oggettive che i prodotti in *output* da una particolare fase del ciclo di vita raggiungono tutti i requisiti specificati per quella fase.
La verifica valuta la corerenza, completezza e correttezza del software e la relativa documentazione e fornisce supporto per la successiva validazione.

Pone attenzione sul metodo di lavoro: ha a che fare con le regole, norme e strumenti. Lo svolgimento di attività secondo regole prestabilite elimina (o quantomeno riduce) la possibilità di introdurre errori.

Il verificatore non deve svolgere lavoro già eseguito da altri.

Validazione

Conferma che il software è conforme ai bisogni dell'utente e che i requisiti sono soddisfatti.

Si può attuare solo a fine profetto quando è disponibile un sistema da validare. E' supportata dalla verifica e dice che la risposta è sensata, ma assicurare che la validazione sia in grado di mappare la risposta e la domanda è compito della verifica.

9.1 Analisi statica

Non richiede esecuzione di codice, ma piuttosto studia caratteristiche del codice sorgente e della documentazione associata.

Viene utilizzata in cerifica quando non è disponibile un prodotto finito. Molti sistemi incorporano funzionalità critiche sulla sicurezza, in particolare:
• Safety: prevenzione di errori verso persone o coesa
• Security: prevenzione di intrusioni

Tali sistemi devono possedere tutte le caratteristiche funzionali e non previste dai requisiti. Nessun linguaggio garantisce a

priori la completa verificabilità dei programmi scritti in esso, in ogni linguaggio c’è compromesso tra potere espressivo e costo di verifica.

9.1.1 Forme di analisi statica

Non richiedendo esecuzione di codice, si applica a qualunque prodotto di processo.

• Metodi di lettura (Desk check)
• Metodi Formali basati sulla prova assistita di proprietà

Metodi di lettura

Sono *walkthrough* e *inspection*. Pratici e basati sulla lettura del documento. Dipendono dall’esperienza del verificatore.

L’inspection consiste nel rilevare la presenza di difetti eseguendo una lettura mirata della documentazione.

Viene svolta da verificatori separati e si suddivide in:

• Pianificazione
• Definizione della lista di controllo
• Lettura
• Correzione dei difetti
• Redazione di documentazione

Il walkthrough consiste nel rilevare la presenza di difetti eseguendo una lettura a largo spettro della documentazione.

Si articola in:

• Pianificazione
• Lettura
• Discussione
• Correzione degli errori
• Redazione di documentazione

Una possibile strategia è al percorrenza simulando l’esecuzione del codice. E’ il metodo più semplice ma più oneroso e meno utile. Richiede maggiore attezione e tempo mentre l’inspection è più rapida e appare più ragionevole.

9.1.2 Tecniche di verifica

Tracciamento

Dimostra completezza ed economicità della soluzione.

Ha luogo:

- tra requisiti software e utente
- tra procedure di verifica e requisiti
- tra codice sorgente e codice oggetto

Alcuni stili di verifica facilitano il tracciamento. L'obiettivo è assegnare singoli requisti elementari a singoli moduli per ottenere una sola procedura di prova per ottenere una più semplice corrispondenza. MAggiore è l'astrazione maggiore la quantità di codice oggetto generato e maggiore l'onere di dimosttazione di corrispondenza.

Revisioni (joint review / audit)

Sono uno strumento essenziale del processo di verifica. Condotte su analisi, progettazione, codice, procedure e risultati di verifica. Richiedono l'interazione tra individui e quindi non sono automatizzabili.
Possono essere **formali** o **informali**.

9.1.3 Tipi di analisi statica

- **Analisi di flusso di controllo**:
  - Accerta che il codice esegua nella sequenza specificata
  - Localizza il codice non raggiungibile o loop infiniti
- **Analisi di flusso dei dati**
  - Accerta che nessun cammino di esecuzione acceda a variabili senza valore
  - Rileva anomalie
  - Rileva problemi legati a variabili globali
- **Analisi di flusso di informazione**
  - Determina dipendenze tra input e output.
  - Dal singolo modulo all'intero sistema
- **Esecuzione simbolica**
  - Analizza quali input causano l'esecuzione delle diverse parti di un programma
  - Permette di verificare le proprietà del programma mediante manipolazione algebrica del codice sorgente
- **Verifica formale del codice**
  - Prova la correttezza del codice rispetto alla specifica algebrica dei reuquisiti.
  - Condizioni di verifica espresse come teoremi la cui verità implica pre-condizioni in input e post-condizioni in output
- **Analisi di limite**

  - Verifica che i dati rimangano enrtro i limiti di tipo e di precisione desiderati
    - Overflow e underflow
    - Errori di arrotondamento
    - Range checking
    - Analisi di limite delle strutture

  Linguaggi evoluti assegnano limiti statici a tipi discreti consentendo verifiche automatiche
- **Analisi dello stack**
  - determina la massima domanda di stack richiesto da un'esecuzione.
- **Analisi temporale**
  - Studia le proprietà temporali richieste dalle dipendenze e output del programma
- **Analisi di interferenza**
  - Mostra l'assenza di effetti di interferenza tra parti separate del sistema
- **Analisi del codice oggetto**
  - assicura che il codice oggetto sia una traduzione corretta di quello sorgente e che nessun errore sia stato introdotto dal compilatore

9.1.4 Programmi Verificabili

Serve coerenzatra standard di codifica e costrutti del linguaggio e metodi di verifica. E' importante anche sviluppare tenendo a mente le esigenze di verifica. La verifica retrospettiva è spesso inadeguata e il costo di correzione è tanto maggiore più avanza lo stato di sviluppo.

Eseguire dei cicli di revisione-verifica dopo ogni rilevazione di errore è inefficacie e oneroso. E' bene avere norme di progetto che prediligano o poibiscano l'uso di particolari costrutti per:

- Assicurare comportamento predicibile
- Consentire analisi di sistema
- Facilitare le prove

Comportamento predicibile

Il codice non deve presentare ambiguità, in particolare deve:

- Fare attenzione su effect sulle funzioni
- Disordine in elaborazione e inizializzazione
- Modalità di passaggio parametri

Analizzabilità di sistema

L’analisi statica costruisce modelli astratti di software in esame come grafo diretto e ne studia i cammini. La presenza di flussi di eccezione e di dispatching complica la struttura del grafo.

Facilità di prova

Possibili strategie:

• Investigativa, informale senza obiettivi
• Formale, regolata da norme

Alcuni costrutti complicano le prove:

• Il dispatching complica quelle di copertura
• Il casting quelle di analisi di identità dei dati
• Le eccezioni quelle di copertura

9.1.5 Criteri di programazione

La buona programmazione deve:

• Riflettere l’architettura del codice
• Separare interfacce da implementazioni
• Massimizzare incapsulazione e information hiding
• Usare tipi specializzati per specificare i dati

9.1.6 Considerazioni pragmatiche

L’effcacia del metodo di analisi è funzione della qualità di strutturazione del codice. La verifica di un programma relaziona frammenti i codice con frammenti di specifica. Una buona architettura facilita la verifica.

9.2 Analisi dinamica

E’ sia un metodo di verifica che di validazione e richiede l’esecuzione del codice. Si svolge tramite test. La verifica avviene su un insieme finito di casi di prova nel dominio delle esecuzioni. Ciascun caso specifica i valori di input e lo stato iniziale del sistema, e deve produrre un esito decidibile. Ogni caso è verificato rispetto ad un comportamento atteso.

La ripetibilità è requisito essenziale e riguarda:

• Ambiente
• Specifica

• Procedure

Gli strumenti a supporto sono:

• Driver
• Stub
• Logger

9.2.1 Faults, errors and failures

Un ***failure*** avviene quando il comportamento del sistema devia da quanto specificato per esso. Sono il risultato di problemi interni al sistema che si manifestano nel suo comportamento esterno. Tali problemi sono chiamati ***errors*** e il le loro cause ***faults***.

9.2.2 Fattori da bilanciare

Nella definizione di strategia di prova occorre un bilanciamento tra

• quantità minima di casi di prova
• quantità massima di sforzo

9.2.3 Criteri di guida

L'**oggetto** della prova può essere:

• Il sistema nel suo intero
• Parti di esso in relazione funzionale
• Singole unità

L'**obiettivo** della prova può essere:

• Specificato per ogni caso
• In termini precisi e quantitativi
• Variabile al variare dell'oggetto

Il piano di qualità risponde alla domanda «quali e quante prove», e la «provabilità» va assicurata a monte, raffinando la progettazione. I test devono essere ripetibili e una prova non basta. Inoltre, hanno limiti e problemi.

• Non esiste un algoritmo che generi per esso un test finito ideale
• Il test di un programma può rilevare la presenza di errori ma non la loro assenza (tesi di Dijkstra)

9.2.4 Principi del software testing (B. Meyer)

- Testare un programma è cercare di farlo fallire
- I test non sostituiscono le specifiche
- Da ogni esecuzione fallita deve essere ricavato un caso di prova da includere permanentemente
- Gli oracoli dobrebbero essere parte del codice
- Ogni strategia di testing dovrebbe includere un processo riproducibile
- La qualità piùimportante di una strategia è il numero di guasti rilevati in funzione del tempo

9.2.5 Elementi di una prova

- **Test case**: tripla input, output, ambiente.
- **Test suite**: procedimento per eseguire, registrare, analizzare e valutare i risultati delle prove
- **Prova**: procedura
- **Oracolo**: Metodo per generare a priori i risultati attesi e convalidare quelli ottenuti

Il problema dell'oracolo è spesso più complesso di quanto si pensi. Alcuni oracoli sono basati su:

- Postcondizioni delle funzioni
- Specifiche e documentazione
- Altri prodotti utilizzanti algoritmi diversi da quello sotto test
- Euristiche
- Valutazione umana

9.2.6 Strategie di integrazione

- Assemblare parti in modo incrementale
- Assemblare produttori prima dei consumatori
- Assemblare in modo che ogni passo sia reversibile

La pianificazione della codifica deve essere tale da rendere disponibile le componenti da integrare in modo ordinato.

Bottom-up: si sviluppano e si integrano prima le parti con minor dipendenza funzionale e maggior utilità. Questo riduce il numero di stub necessari.

Top-down: Si sviluppano prima le parti più esterne e poi si scende. Comporta l'uso di molti stub ma integra a partire dalla funzionalità di livello più alto.

9.2.7 Classi di equivalenza

Dato il costo elevato dei test è importante cercare di ridurre al minimo la quantità dei casi di prova. Questo è possibile individuando le **classi di equivalenza** per valori di input.
Una classe di equivalenza è un insieme rappresentabile da un singolo valore che ne descriva le caratteristiche. L'idea è di raggruppare astrattamemte calori che a fine di test sono indistinguibili.
Il dominio dei valori per ogni parametro è determinato dal tipo del parametro stesso.

Su questo dominio si individuano diverse classi di equivalenza che tipicamente sono:

• Valori nominali interni al dominio
• Valori legali di limite
• Valori illegali

Si considera come esempio una variabile che rappresenta l'indice di un array: i valori illegali sono gli indici inferiori al lower bound e superiori all'upper bound, con quelli legali interni a questo intervallo. Si individuano 5 classi di equivalenza.

9.2.8 Test di unità

L'oggetto del test di unità è l'elemento più piccolo e indivisibile del sistema. Unità e moduli sono determinati in fase di progettazione di dettaglio e di conseguenza il loro piano di test. Il testing di unità termina quando tutte le unità sono verificate. Sono i test che hanno costo maggiore.

Test black box

Fa riferimento alla specifica dell'unità e valuta i risultati in output.
Ciascun insieme di dati in input che producono un dato comportamento funzionale costituiscono un test case. Il test funzionalr da solo non acerta correttezza e completezza e va necessariamente integrato con un test strutturale.

Test white box

verifica la logica interna del codice di unità cercando massima copertura. Ciascuna prova deve essere progettata per attivare ogni cammino di esecuzione.